# OPTIMIZATION METHODS IN HIGH-LEVEL SYNTHESIS

# YÜKSEK SEVİYEDE SENTEZLEMEDE ENİYİLEME YÖNTEMLERİ

**SELMA DİLEK**

**PROF. DR. SÜLEYMAN TOSUN**
**Supervisor**

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Doctor of Philosophy
in Computer Engineering

June 2022

*To my beloved brother Semir Pilav, and my family.*

## ABSTRACT

## OPTIMIZATION METHODS IN HIGH-LEVEL SYNTHESIS

**Selma DİLEK**
**Doctor of Philosophy, Computer Engineering**
**Supervisor: Prof. Dr. Süleyman TOSUN**
**June 2022, 191 pages**

Continuous decrease in the transistor technology sizes has enabled much denser packaging of electronic components on chips, which has resulted in integrated circuits with more functionalities and lower costs. However, it has also given rise to new issues and challenges in the integrated circuit design process, including higher vulnerability to soft errors. Modular hardware redundancy is a popular method for improving the reliability of a system against errors at the cost of increasing area and energy consumption. Voltage scaling methods can be employed to tackle high energy costs; however, these approaches also negatively affect a circuit's reliability and performance. Therefore, designing circuits with all these conflicting parameters is a very challenging task. In this study, we employ two optimization approaches: mathematical programming and metaheuristic algorithms for designing integrated circuits with several conflicting parameters. Mathematical programming approaches guarantee the generation of the optimal solutions; however, they are usually highly impractical for complex real-life problems due to their high computational complexity and unrealistically long running times. Nevertheless, the optimal solutions obtained for relatively small problem sizes are useful for testing the performance of other (meta)heuristic methods that solve the same problem in much faster execution times, although without any guarantees about solution optimality.

In this thesis, we propose integer linear programming (ILP)-based and simulated annealing (SA)-based high-level synthesis (HLS) methods to optimize both reliability and energy of the final circuit designs under the area and latency constraints. Our models employ full and partial resource duplication (modular redundancy) to improve the system reliability as long as the area constraint permits. They also utilize voltage islands as the energy reduction method of choice. This problem is even more interesting and complex because our resource library is characterized under multiple supply voltages. We use different versions of the same resources with different area, latency, reliability, and energy values. Although this affects the execution time of the proposed methods, it also gives us more design options. We compared and showed the effectiveness of our methods against a genetic algorithm (GA)-based method on several HLS benchmarks. The ILP-based methods return the optimum results for smaller problem sizes and most of the time for larger problem sizes under the given time limits. In contrast, the SA-based methods outperform the GA-based methods and generate optimal or acceptably near-optimal results for all benchmarks in much faster running times.

**Keywords:** High-level synthesis, application specific integrated circuits, soft errors, reliability, energy, voltage islands, optimization, metaheuristic algorithms, integer linear programming, simulated annealing.

# ÖZET

## YÜKSEK SEVİYEDE SENTEZLEMEDE ENİYİLEME YÖNTEMLERİ

**Selma DİLEK**
**Doktora, Bilgisayar Mühendisliği**
**Danışman: Prof. Dr. Süleyman TOSUN**
**Haziran 2022, 191 sayfa**

Transistor boyutlarındaki sürekli azalma, elektronik bileşenlerin yongalar üzerinde çok daha yoğun bir şekilde paketlenmesini sağlarken, daha fazla fonksiyon içeren ve maliyeti daha düşük tümleşik devrelerin geliştirilmesine yol açmaktadır. Bununla birlikte, teknolojideki bu gelişmeler tümleşik devre tasarımı sürecinde geçici hatalara karşı daha korumasız olunması gibi yeni sorunlar ve zorluklar da doğurmuştur. Bileşenlerin yedeklenmesi bir sistemin hatalara karşı dayanıklılığını artırmak için popüler bir yöntem olsa da alan ve enerji tüketimi artışına neden olmaktadır. Artan enerji tüketiminin üstesinden gelmek için gerilim ölçeklendirme yöntemi kullanılabilir, ancak bu yöntem devrenin güvenilirliğini ve performansını da olumsuz etkilemektedir. Bu nedenle, tüm bu çelişen değişkenlerle devre tasarlamak çok zorlu bir iştir. Bu çalışmada devre tasarlanırken birbirleriyle çelişen parametreleri göz önüne alan matematiksel programlama ve metasezgisel algoritmalar olmak üzere iki optimizasyon yaklaşımı kullanılmıştır. Matematiksel programlama yaklaşımları, optimum çözümlerin üretilmesini garanti eder. Fakat yüksek hesaplama karmaşıklıkları ve gerçekçi olmayan uzun çalışma süreleri nedeniyle, genellikle gerçek hayattaki karmaşık problemler için pratik değildirler. Bununla birlikte, nispeten küçük problem boyutları için elde edilen optimum çözümler, aynı problemi çok daha hızlı yürütme sürelerinde çözen diğer metasezgisel yöntemlerin performansını ölçmek için faydalıdır. Bunun yanında metasezgisel yöntemler çözümün en iyi çözüm olup olmadığı hakkında herhangi bir garanti vermezler.

Bu çalışmada, alan ve gecikme kısıtlamaları altında nihai devre tasarımlarının hem güvenilirliğini hem de enerjisini optimize etmek için tam sayı doğrusal programlama (ILP) tabanlı ve benzetimli tavlama (SA) tabanlı yüksek seviyede sentezleme (HLS) yöntemleri sunulmuştur. Modellerimizde sistem güvenilirliğini geliştirmek için alan kısıtlaması izin verdiği sürece tam ve kısmi kaynak çoğaltılması (modül yedekleme) kullanılmaktadır. Ayrıca, tercih edilen enerji azaltma yöntemi olarak gerilim adaları kullanılmaktadır. Bu sorunu daha da ilginç ve karmaşık yapan şey, çoklu besleme gerilimi altında kaynak kütüphanesinin tanımlanmasıdır. Kütüphanede aynı donanım kaynaklarının farklı alan, gecikme, güvenilirlik ve enerji değerlerine sahip farklı versiyonları kullanılmaktadır. Bu, önerilen yöntemlerin uygulama süresini etkilese de, daha fazla tasarım seçeneği de sunmaktadır. Sunulan yöntemlerin etkinliğini ölçmek için bazı HLS denektaşları kullanarak genetik algoritmaya (GA) dayalı bir yöntemle karşılaştırılmıştır. ILP tabanlı yöntemler küçük boyutlu problemler için optimum sonuçları verirken, çoğu zaman daha büyük çizge boyutları için verilen zaman sınırlarının altında optimum çözümleri bulabilmiştir. SA tabanlı yöntemler GA tabanlı yöntemlerden daha iyi performans göstermiş ve optimum veya optimuma yakın sonuçları çok daha hızlı bir şekilde elde etmiştir.

**Anahtar kelimeler:** Yüksek seviyede sentezleme, uygulamaya yönelik tümleşik devreler, geçici hatalar, güvenilirlik, enerji, gerilim adaları, eniyileme, metasezgisel algoritmalar, tam sayı doğrusal programlama, benzetimli tavlama.

# ACKNOWLEDGEMENTS

In the name of God, the Most Gracious, the Most Merciful.

I would like to convey my deep appreciation to my advisor **Prof. Dr. Süleyman Tosun** for his invaluable guidance, wisdom, patience, and particularly his understanding and support, without which this endeavor would not have been possible. Words cannot express my gratitude to all my professors, the members of my thesis committee, the chair **Prof. Dr. Suat Özdemir** and the member of the committee **Prof. Dr. Özcan Öztürk**, for their invaluable feedback and generously provided knowledge and expertise.

Furthermore, this journey would not have been possible without the love and support of my beloved family, my mother **Amra Bjelanović**, my father **Husein Pilav** who gave me my first computer Commodore 64 that set me on this path, and my brothers **Semir** and **Omer**. I also thank **Dr. Alma Osmanović Salman**, **Amra Erdić**, and **Asma Aiouez** for being the sisters I never had, and my source of strength, as well as my true 'brother-from-another-mother' friends **Adnan** and **Tarik Kraljić**. Whenever I hit a rough patch throughout this journey, the thought of my late grandparents and uncle, who were such a great source of motivation and people to look up to, kept me going. **Mr. Ramo Krivdić** has been a loving and supportive 'grandpa' in their stead. Their love and boundless support have kept my spirits and motivation high during this process.

I would also like to extend my sincere thanks to my colleagues at Hacettepe Computer Engineering Department for their support and understanding, especially during the trying times of pandemics. I am deeply indebted to research assitants **Necva Bölücü**, **Alperen Çakın**, **Burçak Asal**, **Burcu Yalçıner**, **Dr. Tuğba Gürgen Erdoğan**, **Zeynep Bala**, **Bahar Gezici**, **Merve Özdeş**, **Feyza Nur Kılıçaslan**, **Nebi Yılmaz**, **Asst. Prof. Dr. Alaettin Uçan**, **Asst. Prof. Dr. Pelin Canbay**, and all other research assistants at our department for their invaluable support throughout this journey.

Special thanks are due to my dear professors and role models **Dr. Douglas Van Wieren**, **Prof. Dr. Ahmed Moosajee Patel**, **Assoc. Prof. Dr. Željko Jurić**, and **Mr. Nedim Mašić** who instilled in me appreciation and love for computer science and academia, and showed me what it means to be an excellent educator, and to my doctors **Prof. Dr. Murat Tuncer**, **Prof. Dr. Alper Demirbaş**, and **Prof. Dr. Sabri Tekin**.

Finally, I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting this research in part under project number 116E095, and Fair Isaac Corporation (FICO) for granting us an academic license for FICO Xpress optimizer that we used to run our ILP models.

I dedicate this work to my hero, my selfless brother **Semir Pilav**, who risked his life to save mine without forethought. I owe him my health and everything I have accomplished since he donated his kidney to save my life.

# Contents

# TABLES

# FIGURES

**Page**

# ABBREVIATIONS

| | | |
|---|---|---|
| **ACO** | : | **A**nt-**C**olony **O**ptimization |
| **ADRS** | : | **A**verage **D**istance to **R**eference **S**et |
| **AR** | : | **A**uto-**R**egressive Filter |
| **ASAP** | : | **A**s **S**oon **A**s **P**ossible |
| **ASIC** | : | **A**pplication-**S**pecific **I**ntegrated **C**ircuit |
| **CAD** | : | **C**omputer **A**ided **D**esign |
| **CMOS** | : | **C**omplementary **M**etal-**O**xide **S**emiconductor |
| **CPU** | : | **C**entral **P**rocessing **U**nit |
| **DAG** | : | **D**irected **A**cyclic **G**raph |
| **DES** | : | **D**ifferential **E**quation **S**olver |
| **DMR** | : | **D**ual **M**odular **R**edundancy |
| **DSE** | : | **D**esign **S**pace **E**xploration |
| **DSP** | : | **D**igital **S**ignal **P**rocessing |
| **DVS** | : | **D**ynamic **V**oltage **S**caling |
| **EWF** | : | **E**lliptic **W**ave **F**ilter |
| **FCCM** | : | **F**PGA **C**ustom **C**omputing **M**achine |
| **FIR** | : | **F**inite **I**mpulse **R**esponse Filter |
| **FPGA** | : | **F**ield **P**rogrammable **G**ate **A**rray |
| **FU** | : | **F**unctional **U**nit |
| **GA** | : | **G**enetic **A**lgorithm |
| **HDL** | : | **H**ardware **D**escription **L**anguage |
| **HLS** | : | **H**igh-**L**evel **S**ynthesis |
| **HW** | : | **H**ard**w**are |
| **IC** | : | **I**ntegrated **C**ircuit |
| **ILP** | : | **I**nteger **L**inear **P**rogramming |
| **MHSP** | : | **M**eta-**H**euristic **S**pecific **P**arameters |

| | | |
|---|---|---|
| **MILP** | : | **M**ixed-**I**nteger **L**inear **P**rogramming |
| **ML** | : | **M**achine **L**earning |
| **MOOP** | : | **M**ulti-**O**bjective **O**ptimization **P**roblem |
| **MTBF** | : | **M**ean **T**ime **B**etween **F**ailures |
| **MUX** | : | **Mu**ltiple**x**er |
| **nm** | : | **N**ano**m**eter |
| **NOP** | : | **N**o **Op**erations |
| **NP** | : | **N**ondeterministic **P**olynomial |
| **RTL** | : | **R**egister-**T**ransfer **L**evel |
| **SA** | : | **S**imulated **A**nnealing |
| **SE** | : | **S**oft **E**rror |
| **SER** | : | **S**oft **E**rror **R**ate |
| **SRAM** | : | **S**tatic **R**andom **A**ccess **M**emory |
| **SSA** | : | **S**purious **S**witching **A**ctivity |
| **SW** | : | **S**oft**w**are |
| **TMR** | : | **T**riple **M**odular **R**edundancy |
| **VI** | : | **V**oltage **I**sland |
| **VLSI** | : | **V**ery **L**arge **S**cale **I**ntegration |
| **WCET** | : | **W**orst-**C**ase **E**xecution **T**ime |

# 1. INTRODUCTION

Continuous decrease in the transistor technology sizes due to ever-increasing demands for higher performance of computer applications has facilitated packing a considerably higher number of electronic components on chips. While 7-nm and 5-nm technology sizes are the current industry standard, the shift towards 3-nm technology has already been set into motion [1, 2]. The resulting increase in circuit densities has caused a decrease in the integrated circuit costs and area. However, at the same time, it has brought about new challenges in the process of integrated circuit design, such as higher vulnerability to transient (soft) errors due to radiation effects and lower supply and threshold voltage levels [3]. This increase in soft error rates (SER) is particularly evident in combinational circuits, which necessitates novel reliability-oriented design methods. There are certain error detection techniques for combinational circuits and memory elements; however, they usually increase the circuit area and cost (e.g., redundancy-based error detection and error correction codes). Furthermore, energy-aware designs face yet another negative effect on their reliability caused by energy reduction methods such as dynamic voltage scaling (DVS) or voltage islands (VIs) because a decrease in the supply voltage also decreases a circuit's reliability [4, 5]. Modular redundancy, i.e., replication of the system components, can improve the reliability of a system, but at the cost of increasing the resulting area and energy consumption.

Several parameters and system requirements may need to be considered during hardware design, such as area, performance, energy consumption and reliability. High-level synthesis (HLS) has multiple advantages compared to traditional register-transfer level (RTL)-based hardware design, including raising the abstraction level, accelerated verification, faster design space exploration, portability to new platforms, and accessibility to software engineers [6]. A major benefit of HLS is enabling the exploration of unique trade-offs for the generation of diverse microarchitectures that stem from the same behavioral specification but are obtained through different synthesis options, also called knobs [7, 8]. During the high-level synthesis (HLS) step of the application-specific integrated circuit (ASIC) design, it is possible to consider multiple system constraints (e.g., area and latency) and optimization parameters (e.g., reliability and energy consumption) at a higher level of abstraction, and unify them to alleviate the burdensome design process that must consider all system requirements and constraints simultaneously. Automation tools that would facilitate the design process are particularly crucial in the design of integrated circuits with a large number of components.

In HLS design space exploration, several design parameters often opposing each other are considered for optimization (e.g., minimizing energy consumption negatively affects reliability). Thus, we can categorize it as a multi-objective optimization problem (MOOP) in which the goal is to look for a set of solution designs that are Pareto-optimal, meaning that any improvement in one of the parameters would cause a deterioration in another. Having several Pareto-optimal designs allows designers to choose the ones that fit best to the project's requirements by controlling the HLS synthesis process through synthesis options settings. The major issue with this is that the number of synthesis options superlinearly affects the growth of the search space [8]. Therefore, optimization methods for efficiently searching the space need to be proposed.

## 1.1. Scope of the Thesis

Traditional HLS methods usually consider only area and latency along with either energy or reliability. It is evident that there is a need for new systematic design methods that will consider all these requirements on a higher level of abstraction. This research has explored novel HLS optimization methods that can integrate all system requirements on a higher level of abstraction and relieve integrated circuit designers from lower-level design burdens.

Furthermore, there has been an exponential growth in the complexity of very-large-scale integration (VLSI) systems, which poses a great productivity bottleneck for the processes of design and verification in RTL as the prevalent method that has been used for decades to characterize VLSI systems. The reason is that RTL tools have not improved in proportion to the increase in VLSI system complexity. HLS contributes greatly to the hardware design process with many benefits it provides over the traditional RTL-based approach, as discussed above. Nevertheless, there is still a non-trivial productivity gap between these two design flows, which is an important current issue in hardware design. Modern HLS tools still trail behind the RTL design flows in terms of results' quality, and should this productivity and quality gap close, HLS would become the new standard approach for hardware design [9]. Hence, another goal of this research was to propose novel HLS optimization methods that will contribute to closing that gap.

To the best of our knowledge, no prior research has proposed an optimization method that considers both area and latency as constraints while considering energy consumption and reliability as multi-objective optimization parameters. Moreover, the previous studies have neglected to thoroughly investigate the effect of multiple supply voltage levels on reliability and energy efficiency.

## 1.2. Contributions

In this research, we address the mentioned deficiencies and propose novel and efficient ILP and SA-based HLS methods that also employ dual modular redundancy (DMR) for ASIC design with the objectives of minimizing energy consumption and maximizing reliability under the given area and latency constraints. In our study, we use different versions of the same resources in terms of varying area, performance, energy, and reliability characteristics, depending on the supply voltages at which they operate. Since we have two parameters in our optimization function, we blend the energy and reliability values by assigning weights to each of them to handle our multi-optimization problem. For the mapping and scheduling steps of the HLS, we use the ILP and SA-based optimization methods. The main contributions of this research can be summarized as follows:

- We propose bi-objective ILP-based HLS methods that employ dual modular redundancy for increasing reliability in ASIC design, with the objectives of minimizing energy consumption and maximizing reliability under the given area and latency constraints.

- We formulate the mapping and scheduling steps of our duplication-based HLS design flow with two ILP-based methods (partial and full duplication) that obtain the optimum results in short times for smaller-scale benchmark applications due to their computational complexity. ILP-based methods are unsuitable for problems with many variables due to their undesirable CPU times. Nevertheless, they are important as they provide optimal results that can be used for testing other heuristic or metaheuristic methods designed to solve the same problems for larger-scale applications within more acceptable running times.

- We present an extended resource library with the varying area, delay, energy, and reliability parameters based on multiple supply voltage levels. We believe our resource library will also benefit future studies in HLS.

- We test and discuss the effects of utilizing voltage islands (VIs) on reliability and energy consumption in circuit design that employs modular redundancy.

- We illustrate the effectiveness of our ILP duplication models over the genetic algorithm (GA)-based selective duplication method in terms of energy and reliability results on several benchmarks by conducting a thorough experimental analysis.

- We also propose SA-based metaheuristic methods that tackle the same HLS problem and obtain optimal or near-optimal solutions in acceptable polynomial time, as opposed to the ILP-based methods whose running times increase exponentially as the application size increases.

- By conducting a thorough experimental analysis, we demonstrate the effectiveness of our SA-based methods compared to the ILP-based models and their superiority over the genetic algorithm (GA)-based methods in terms of energy and reliability results on several benchmarks.

## 1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 presents our motivation, contributions and the scope of the thesis.
- Chapter 2 provides background information on relevant topics, including optimization, high-level synthesis, soft errors, and the effects of modular hardware redundancy and multi-supply voltages on integrated circuits' reliability, latency, and energy consumption.
- Chapter 3 presents an overview, classification, and a comparative summary of the relevant related work.
- Chapter 4 gives a detailed problem definition.
- Chapter 5 introduces the proposed HLS optimization methodologies. First, we present the ILP-based mathematical optimization models. Then, we explain our metaheuristic SA-based methods.
- Chapter 6 demonstrates the effectiveness of the proposed methods by presenting and discussing the experimental results.
- Chapter 7 concludes this study with a summary of the thesis and possible future directions.

# 2. BACKGROUND OVERVIEW

In this section, we discuss some fundamental concepts about optimization, reliability, latency, and energy-related topics and issues in digital electronic systems, which are relevant to this research.

## 2.1. Optimization Fundamentals

The conventional engineering design process is generally an exhausting endeavor of iterations between the conceptual, preliminary, and detailed design steps that are usually also iterative processes within themselves, before reaching a desirable final design that meets all initial design requirements and specifications. Optimization has emerged as an alternative mechanism that can speed up the design cycle and generate better results while lowering the total design cost. Optimization entails finding the best solution from a set of possible alternative solutions for an optimization problem, which can be obtained via changing the controllable variables that define the problem based on some criteria, while usually subject to certain constraints [10, Chapter 1]. Optimization generally implies minimizing or maximizing a cost function or a set of functions, also known as the objective functions, which depends on selecting the appropriate input values for the function variables. Optimization goals may involve anything from maximizing metrics such as efficiency, profit, reliability, or performance, to minimizing metrics such as cost, delay, or energy consumption.

Optimization problems may be either linear or nonlinear depending on whether all constraints and objective functions can be formulated using linear functions or not. If a linear formulation of the constraints and objective functions is impossible, it leads to a much more complex nonconvex decision space. For nonlinear problems whose search space usually contains multiple locally optimal solutions, the main challenge is how to apply an approach that will avoid getting trapped in local minima or maxima while performing the search. Getting trapped in local minima or maxima would impede finding the globally optimum solution [11].

The optimization process necessitates that designers formulate a mathematical model of the problem at hand, which correctly describes the problem, defines the decision variables and the optimization objective, and specifies the constraints if any. Martins and Ning (2021) and Yang (2014) presented a general mathematical formulation that can be used as a common definition for a majority of continuous optimization problems [10, 12]. Let $\vec{x}$ be a design vector defined in (1), where $x_i$ elements represent $d$ decision variables of a problem.

$$\vec{x} = (x_1, x_2, ..., x_d) \tag{1}$$

Let $f_i(\vec{x})$, $h_j(\vec{x})$, and $g_k(\vec{x})$ be functions of the design vector $\vec{x}$, and $\mathcal{R}^d$ the search space within the scope of the decision variables. Then, an optimization problem can be defined as formulated in (2), where $f_i(\vec{x})$ represents $M$ objective functions, while $h_j(\vec{x})$ and $g_k(\vec{x})$ represent $J$ and $K$ equality and inequality constraint functions, respectively.

$$\begin{aligned} \min_{\vec{x} \in \mathcal{R}^d} \quad & f_i(\vec{x}), \quad (i = 1, 2, ..., M), \\ \text{by varying} \quad & \underline{x}_i \leq x_i \leq \overline{x}_i, \quad (i = 1, ..., d), \\ \text{such that} \quad & h_j(\vec{x}) = 0, \quad (j = 1, 2, ..., J), \\ & g_k(\vec{x}) \leq 0, \quad (k = 1, 2, ..., K) \end{aligned} \tag{2}$$

The formulation given in (2) defines a minimization problem. However, this formulation can also be used to define a maximization problem, given that it can easily be transformed into a minimization problem as given in (3). Furthermore, the inequality constraints can also be defined using $\geq 0$ inequality if needed.

$$\max[f(x)] = -\min[-f(x)] \tag{3}$$

Nevertheless, an inequality constraint should never be expressed as a strict inequality since, in that case, the solution would not be properly mathematically defined as it would be allowed to choose among solutions arbitrarily close to the equality. This consideration is particularly important for optimization attempts using computers as they use finite-precision arithmetic. Additionally, Martins and Ning (2021) argue that the optimization problem formulation should include as many independent decision variables as possible, although it may be desirable to begin with a smaller set that can later be broadened [10, Chapter 1].

Some problems may aim to optimize a single variable; hence, they have a single objective function. However, many problems in real life involve optimizing multiple metrics. Such problems are called multi-objective optimization problems (MOOPs). MOOPs frequently do not have an optimal solution that optimizes all of the objectives simultaneously, especially since some of those objectives may oppose each other and require designers to make trade-offs in search of an optimal solution based on the requirements. Such is the case with the problem of reliability and energy-oriented HLS addressed in this research. More reliable resources mostly require more power, while energy reduction techniques negatively affect the reliability of circuits. Therefore, when searching for the best solutions to MOOPs, designers need to make compromises that will produce Pareto-optimal results in the context of a conflicting multi-objective nature of the problem. A Pareto-optimal design is an efficient solution with the optimal objective function (no other feasible solution dominates it) such that the achieved trade-off between all objectives is optimal. *I.e.*, improving any objective function would come at the cost of worsening at least one other objective [13].

This compromise among different and often competing objectives may be achieved by reformulating the objective function for the given problem. One of the most commonly applied approaches is the weighted scalarization method which allows designers to assign a preference value to each objective function. The goal of weighted scalarization is to formulate a single scalar-valued objective function (*i.e.*, a utility function) $\mathcal{F}$ that incorporates all of the objectives simultaneously [14]. In Equation (4), the weighted scalarization of $M$ objective functions $f_i$ is formulated, where weights are assigned to each objective function through coefficients $w_i$.

$$\mathcal{F}\left(f_1(\vec{x}), f_2(\vec{x}), ..., f_M(\vec{x})\right) = \sum_{i=1}^{M} w_i \cdot f_i(\vec{x}) \tag{4}$$

## 2.2. Optimization Algorithms

There exist many optimization algorithms that can be categorized based on several characteristics, including stochasticity (random or deterministic), time dependence (static or dynamic), order (zeroth, first, second), type of search performed (local or global), how the objective function is evaluated (directly or via a surrogate model), or depending on the algorithm nature (mathematical or heuristic) [10, Chapter 1]. Mathematical algorithms (e.g., linear programming) generate solutions based on the mathematical model of the problem, and they can provide optimal results. The major drawback is their inefficient complexity, especially for complex optimization problems and problems with many decision variables. Heuristic and metaheuristic algorithms (e.g., genetic algorithm, simulated annealing) are preferred in such scenarios as they can generate optimal or near-optimal (approximate but close enough) results in practical running times. However, they cannot guarantee the optimality of the solutions as they are not based on strict mathematical formulations.

None of the existing optimization algorithms is efficient for all optimization problems; hence, they should be chosen carefully based on the nature of the problem at hand. In this research, we have employed both mathematical and metaheuristic algorithms to optimize the problem at hand. The following sections briefly discuss the nature of the algorithms employed in this study and our reasoning for choosing them.

### 2.2.1. Linear Programming

Linear programming is a subset of mathematical optimization and one of the most widely used approaches for the optimization of convex optimization problems that can be formulated using linear objective and constraint functions [10, Chapter 11]. It is a popular approach in allocation problems optimization, which is one of the reasons it was our method of choice in this research because, in HLS, hardware resource allocation is one of the major tasks.

As the main focus of this research is the optimization of digital circuit design, our optimization problem consists of decision variables that fall under the integral domain and, more specifically, under the modular number system. Such problems can be modeled as integer linear programming problems, a subset of linear programming and the approach we adopted in this study. Scheduling and resource allocation are examples of discrete optimization problems. Optimization variables in discrete optimization can be binary, integer, and discrete, and all of them can be represented with integer values [10, Chapter 8].

One of the first approaches considered in linear programming optimization, and one of the most efficient ones in practice, is the simplex method, developed by George Dantzig in 1947, which operates by first discovering a simple, feasible solution such that all the constraints are met. Mathematically speaking, such a feasible solution is theoretically an extreme point on the edge of the feasible region, i.e., a convex polyhedron that spans the hyperspace $\mathbb{R}^n$ bounded by $n$ constraints expressed as linear functions. The simplex method then explores along the polyhedron edges in search of better objective function values to find the optimal solution. Nevertheless, the major drawback of this approach is its exponential worst-case complexity [15].

In this study, the primary approach used by the optimizer is a revised (primal) simplex method that is more computationally efficient than the original simplex method. This efficiency is achieved by constraining the simplex pivot operations to the inverse matrix only, hence, avoiding the execution of unnecessary tableau updates. As it is out of the scope of this study, we leave out the details of these methods. An interested reader can refer to [16] for detailed explanations of the inner workings of these methods. Another method employed by the optimizer used in this study, in addition to the simplex method, was the Newton Barrier method. Contrary to the simplex method, the Newton Barrier method searches through the interior region of the feasible region, looking for a close estimation of an optimal solution.

It usually takes a uniform number of iterations to complete, irrespective of the size of the optimization problem.

Discrete optimization is nondeterministic polynomial-time complete [10, Chapter 8], which means that there are no known efficient approaches that compute optimal solutions in polynomial time. ILP problems, in particular, are known to be polynomial-time hard [15]. We employed an ILP-based approach to obtain the optimal solutions for relatively small applications, which are then used for evaluating the performance of our heuristic methods that tackle the same optimization problem in practical running times.

### 2.2.2. Metaheuristic Approaches for Discrete Optimization

A heuristic (from the Greek word ‘heuriskō’ for “I find, discover”) is an algorithm in mathematical optimization which uses a trial and error approach to generate adequate (optimal or close enough) solutions for a complex optimization problem in acceptable running times, for which classical optimization methods cannot produce optimal results most often due to the huge computational complexity of those approaches. For some problems, the optimality and accuracy of the solution can be traded for speed if the goal is to obtain good enough (maybe even optimal but without any guarantees) solutions in reasonable running times. For such optimization problems, heuristic algorithms are a very efficient and sometimes the only practical alternative [12].

Metaheuristic algorithms emerged as more advanced heuristic approaches that can explore the search space more efficiently through higher-level considerations and trade-offs between local search versus randomization. These trade-offs enable metaheuristic approaches to break away from local search to a more global scale, and hence, to avoid getting stuck in local optima [12, 17]. Randomization is important as it enables more efficient exploration of the search space, specifically by either intensifying the search or diversifying it via random walks, while also using some deterministic process. Random walks guarantee that the candidate solutions generated along the way are diversified in their distribution over the search space on the global scale [18].

One of the popular metaheuristic optimization approaches developed for discrete optimization is a nature-inspired method called simulated annealing (SA). Simulated annealing was initially proposed in the early 80's as an optimization method analogous to the physical process of the annealing of solids. It was first proposed by Kirkpatrick et al. (1983) in [19], who coined the name and applied the approach to solving the traveling salesman problem, while Černý (1985) in [20] independently had the same idea of applying statistical thermodynamics principles to solving the same problem. These two works were pioneers in demonstrating how the analogy with statistical thermodynamics which states that large systems will spontaneously reach so-called state of equilibrium at a given temperature, can be applied as a generic method to solving large combinatorial optimization problems [21].

A strong feature of simulated annealing is its aptitude to avoid getting stuck in local optima as opposed to other deterministic approaches, as it can converge to the global optimum when its randomness and cooling schedule are properly arranged. The fundamental concept of SA is the use of random search as a Markov chain in such a way that better solutions are always accepted, while some candidate solutions that have a worse cost in terms of the objective function may still get accepted based on a transition probability [22].

The transition probability $p$ is defined in Equation (5), in which $\Delta f$ represents the change in the cost of the objective function $f(x_j) - f(x_i)$ where $x_i$ is the current best solution and $x_j$ the candidate solution, $k_B$ represents the Boltzmann constant, and $T$ represents the current temperature that manages the annealing operation. The calculation of the transition probability can be adjusted (e.g., taking the Boltzmann constant to be 1). However, it must be in correlation with the annealing temperature and the objective function costs of the current and candidate solutions.

$$p = \exp\left(-\frac{\Delta f}{K_b T}\right) \tag{5}$$

In SA, a worse candidate will be kept if $p > r$, where $r$ is a random real number selected from a uniform distribution in the range $[0, 1]$. Temperature plays a crucial role in efficient search space exploration as the desired outcome is slowly decreasing the probability of a worse solution getting accepted as the process cools down. A higher probability of accepting worse solutions at the beginning of the annealing process is desirable as it allows for more random exploration of the search space and guarantees that the algorithm will not get trapped in local optima. However, as the annealing process nears its end, this probability should become lower to enable intensified search for the optimum solution.

The most important concepts in SA-based optimization include (i) selection of good initial temperature, (ii) an efficient annealing (cooling) schedule that gradually decreases the temperature, (iii) a random neighboring design that ensures that all neighboring states (solutions) are not equally probable, and (iv) obtaining an acceptance probability contingent on a probability function that is positively correlated with the temperature. Pseudocode given in Alg. 1 shows the basic steps of the SA process. The decisions about the implementation details, such as selection of the starting state, initial temperature, neighbor states, and the appropriate cooling schedule, must be made for each specific problem.

**Algorithm 1** A General Simulated Annealing Pseudocode

```
Inputs: $s_0$ starting state, $T_0$ initial temperature, $numIterations$.
Output: $s$ the final optimal state.
  $s \leftarrow s_0$                          ▷ Pick the starting state $s_0$ as the current solution $s$.
  $T \leftarrow T_0$                          ▷ Start with the initial temperature.
  for $i = 1...$ numIterations do
     Pick a neighboring candidate state $s^*$ .
     if $f(s^*) > f(s)$ then
        $s \leftarrow s^*$                    ▷ Accept the better candidate.
     else if $P\left(f(s), f(s^*), T\right) \geq rand(0,1)$ then
        $s \leftarrow s^*$                    ▷ Accept the worse candidate anyway.
     end if
     $T \leftarrow coolingSchedule(T)$        ▷ Reduce the temperature.
  end for
  return $s$
```

Several different methods for computing the initial temperature have been proposed in the literature [19, 21, 23–26], as well as several cooling (annealing) schedule approaches [19, 21, 27, 28]. Cooling rate selection should be performed carefully to match the problem complexity as a slower cooling rate will guarantee more optimal search space exploration [29]. Moreover, theoretical studies show that the number of iterations required to guarantee that the obtained results are close enough to the optimum solutions is exponential based on the size of the problem; hence, starting and stopping criteria should be thoughtfully selected [27]. The starting conditions and cooling schedule adopted in this study are presented in detail in Section 5..

## 2.3. Reliability, Energy, and Latency Considerations for Optimization in Digital Electronic System Design

In this section, we discuss reliability, latency, and energy-related topics and issues in digital electronic systems design, which have been taken into consideration during this research. Since we employ VIs as an energy reduction technique in our study, we specifically focus on the effect of multi-supply voltages on each of these metrics.

### 2.3.1. Soft Errors, Reliability, and Modular Redundancy

Transient or soft errors (SEs) in digital systems are non-persistent errors that do not cause permanent or fatal damage to a system. A system or a part of it will be affected only while the fault is present and its value is used. Thus, a digital system can resume its normal operation once it is restarted or when the erroneous value is rewritten. SEs can be caused by several external factors but mainly by radiation rays, either cosmic or resulting from the product packaging, which may cause a bit flip in semiconductor fabrics. Generally, the incident of SEs happens when the energy accumulated in a transistor surpasses its critical energy. Figure 2.1 illustrates the occurrence of a soft error in the silicon view and the transistor view for an n-type CMOS transistor.

The reliability of an electronic circuit is essentially the probability that the circuit will operate (perform the required function) correctly for a specified time interval. The failure rate, also known as soft error rate (SER), is the most commonly used metric for estimating the reliability of electronic circuits [30]. It is obtained using Equation (6), where $MTBF$

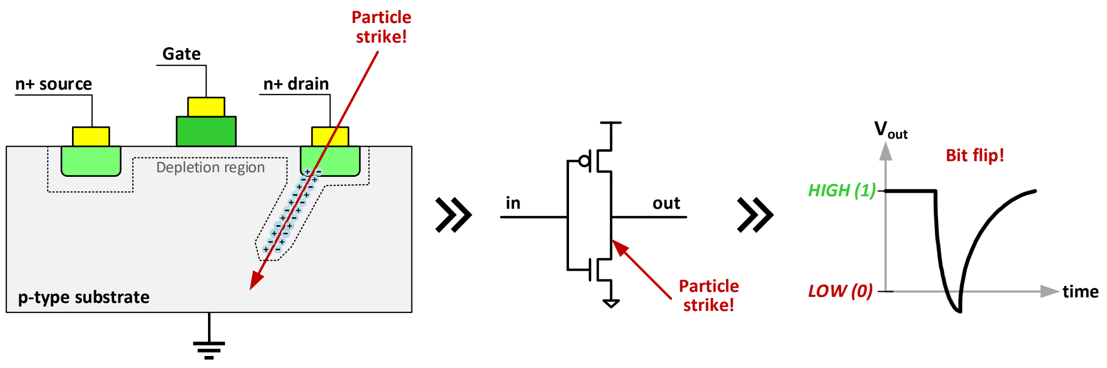


Figure 2.1 Occurrence of a soft error in an n-type CMOS transistor.

represents the mean time between failures (MTBF) which can be defined as the mean operating (up) time between recoverable failures in a circuit.

$$\lambda = \frac{1}{MTBF} \tag{6}$$

A high SER of a system has a negative effect on the system's reliability. The reliability of a system can be obtained using Equation (7), where $\lambda$ represents the SER of the system, while $t$ represents the operating time. From Equation (7), we can observe that the reliability of a system is inversely proportional to its SER: the higher the SER, the lower the reliability.

$$R(t) = e^{-\lambda t} \tag{7}$$

The resulting reliability of a circuit depends on the reliability, number, and placement of the individual elements in the circuit. Arrangement of the circuit elements can be in series, parallel, or a combination of both [31]. For example, when operations execute in series in a circuit, the overall reliability will be calculated as the product of all individual reliability values. Hence, the low reliability of one individual element will negatively affect the overall circuit reliability.

One of the popular techniques to increase a system's reliability is to use modular redundancy (i.e., duplicate components and make their backups). When a component is duplicated in parallel, its new boosted reliability value can be obtained through Equation (8), where $R_c$ represents the total reliability of the component after duplication, while $R_i$ and $R_{i'}$ represent the individual reliability values of the original instance of the system component and its duplicated backup, respectively.

$$R_c = R_i + R_{i'} - R_i R_{i'} \tag{8}$$

Dual and triple modular redundancy with checker circuits (e.g., a majority voting circuitry in TMR) are popular hardware redundancy approaches. Nonetheless, in general, a parallel n-modular redundancy solution for a fault-tolerant circuit can be used (see Figure 2.2) [32].

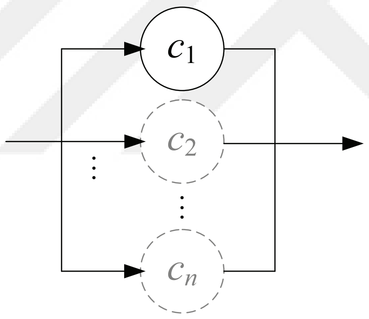


Figure 2.2 An n-modular redundancy illustration for a fault tolerant component.

The overall improved reliability of such a design can be obtained with Equation (9) where $R_{c_i}$ represents the reliability of the $i^{th}$ replica of the component $c$.

$$R_c = 1 - \prod_{i=1..n} (1 - R_{c_i}) \tag{9}$$

Several points should be considered when deciding which components to duplicate (full or partial duplication) and which resources to use for modular redundancy. In terms of complying with the area and latency constraints, a system with duplicated components should still meet the given area and latency requirements. Usually, backups are created using the same circuit elements used for the original components. However, sometimes using different implementations of those circuit elements may result in a more optimal solution because the behavior against SEs manifests differently in varying implementations of the same circuit element (e.g., a slow circuit element may demonstrate higher reliability than a fast one) [33]. Finally, modular redundancy will inevitably add to the system's energy consumption, which is a crucial design consideration in energy-aware systems.

### 2.3.2. Effects of Multi-Supply Voltages on Energy and Latency

While reliability and/or performance (application runtime or latency) are the first and foremost design considerations in safety-critical systems, efficient energy consumption is usually the main design consideration in battery-powered systems. Therefore, varying integrated systems may require different design trade-offs to achieve the desired system requirements. In this study, we employ one of the popular energy reduction approaches for digital systems; namely voltage islands [34]. In digital systems that use voltage islands, different parts of the circuit may operate under different supply voltages to lower the system's overall energy consumption.

Energy reduction techniques are beneficial because lowering the supply voltage does not affect the worst-case execution time (WCET) in the same proportion it affects the overall energy consumption of a system. A change in energy consumption is directly proportional to the square of the change in the supply voltage. This is formulated in Equation (10), where $E_{v_h}$ and $E_{v_l}$ are the energy consumption amounts of the circuit at a high and low voltage levels $v_h$ and $v_l$, respectively.

$$E_{v_l} = E_{v_h}(\frac{v_l}{v_h})^2 \tag{10}$$

On the other hand, a reduction in the supply voltage will only result in a proportional increase of the worst-case execution time (WCET) of a system. This is formulated in Equation (11), where $v_l$ is a threshold voltage, while $t_{v_h}$ and $t_{v_l}$ are the WCETs of a circuit at a high and low voltage levels, respectively.

$$t_{v_l} = t_{v_h}(\frac{v_l}{v_h})(\frac{v_h - v_t}{v_l - v_t})^2 \tag{11}$$

Therefore, having a part of a circuit whose operation at a lower voltage will not cause an unacceptable increase in the circuit's WCET, while coincidentally resulting in a notable decrease in the overall energy consumption of the circuit may be a desirable design objective.

### 2.3.3. Effects of Multi-Supply Voltages on Reliability

As discussed in the previous subsection, energy reduction techniques in general, and VIs in particular, may provide an efficient way to reduce energy consumption in a system without introducing too much extra latency. Nevertheless, another crucial consideration when using multi-supply voltages in a circuit is the effect of this approach on the circuit's reliability.

As already mentioned, SEs will inevitably occur when the energy accumulated in a transistor surpasses its critical energy. When a circuit operates under lower voltages, the critical energy can be much more easily exceeded. Hence, lowering the supply voltage will also negatively affect a system's reliability. This relation is formulated in Equation (12), where $\lambda$ refers to the SER, $d > 0$ is a constant, and $\lambda_0$ refers to the average error rate at frequency $f$ [35].

$$\lambda(f) = \lambda_0 10^{\frac{d(1-f)}{(1-f_{min})}} \tag{12}$$

# 3. RELATED WORK

In this section, we categorize the related work based on four approaches employed in this research: (i) ILP-based HLS solutions; (ii) SA-based HLS solutions; (iii) HLS with modular redundancy for improved reliability; and (iv) HLS solutions that employ multi-supply voltages for energy-oriented designs.

## 3.1. ILP-Based HLS Studies

Several studies have employed ILP-based methods for tackling the HLS problem with different objectives, but mainly to increase reliability. In [36], the authors proposed a 0-1 ILP formulation for reliability-oriented HLS under the given area and latency constraints. This study did not consider energy awareness during the HLS process.

In [37], the authors focused on the aging of the functional units (FUs) to maximize the reliability and lifetime of both data-flow intensive and control-flow intensive designs. They proposed an ILP-based scheduling technique that employs task chaining and multicycling to mitigate aging traits of the FUs to achieve more than a double lifetime with an acceptable added latency overhead. This study considered control-flow intensive designs as well, as opposed to this study; however, energy considerations were completely neglected.

Another study that also focused on optimizing reliability is [38], in which authors proposed both an ILP formulation and a heuristic allocation and binding method for optimizing reliability in control-flow intensive systems under the area and latency constraints. Their methods are based on a prior vulnerability analysis of variables and tasks in the behavioral description, which was proved to enhance system reliability significantly compared to the vulnerability-unaware HLS. Although the proposed method achieves a significant improvement of up to 85% in terms of reliability without adding extra area overhead, the authors did not consider its effect on energy consumption.

The authors of [39] also studied scheduling in HLS, focusing on optimizing circuit datapath reliability under the latency and area constraints. They proposed an ILP formulation that considers tasks and their active times in scheduling for maximum reliability. However, energy considerations were again ignored in this study.

In [40], different objectives of minimizing the latency and area of the resulting integrated circuit are investigated. The authors did not focus on either reliability or energy consumption optimization; however, they proposed a novel multi-objective MILP model considering multiplexers along with FUs during the HLS design step. The experiments showed that the circuits which result from a design process that incorporates multiplexer usage considerations have more optimal latency and area values on average than the circuits when they are ignored.

Circuit area minimization was also studied in [41]. The authors proposed a novel ILP formulation and an ILP-based heuristic for the resource allocation and binding steps of HLS under the given latency constraint. They considered FUs, registers, and multiplexers for control and datapath circuits in their proposed methods. However, reliability and energy considerations were left out in this study. In [42], the authors presented an ILP model for HLS of DSP algorithms with the goal of area cost minimization under latency constraints. Energy and reliability considerations were again left out.

Another study that focused on latency minimization is [43]. The authors proposed a MILP-based approach to solving HLS with a built-in programmable duty cycle mechanism for designs that use dual-edge-triggered flip-flops as memory elements. Their approach was able to improve latency for several benchmarks. The proposed method, however, did not consider any area, reliability, or energy issues.

[44] also studied delay and performance yield optimization for scheduling and resource binding in HLS. The authors presented an ILP formulation that considers timing variations of different resources and employs successive tasks and resource chaining for improved latency and performance yield, which can be defined as the probability that the design can operate efficiently at a specific clock frequency. This method, however, also does not tackle reliability or energy considerations.

In [45], the authors proposed an ILP-based HLS approach for designs that use VIs with dual supply voltage to minimize energy consumption. Unlike this study, they do not consider the effect of energy reduction on reliability. Another study that employed multiple supply voltages approach for energy minimization is [46], in which the authors presented an ILP-based HLS method for optimizing energy but also neglected its effects on reliability.

Optimization of energy consumption was also tackled in [47]. The authors presented an ILP model for energy-aware HLS under the constraints of area and latency. Nevertheless, they also ignored the effect of energy savings on circuit reliability. Another study that focused on energy optimization under latency constraints is [48]. Similarly to our study, they used pipelined resources in their designs; however, reliability considerations were again disregarded.

In [49], the authors focused on both energy and task delay minimization under latency constraints. They proposed an ILP model that minimizes the total energy consumption through optimizing task scheduling. The reliability concerns were not discussed in the study.

In Table 3.1, we present a summary of the relevant related work that employed the ILP-based approaches to optimization in HLS. We note the optimization metrics considered, as well as design constraints, if any.

Table 3.1 A summary of related ILP-based HLS studies.

| **Study** | **Optimization Metrics** | **Constraints** | **Target Systems** |
|---|---|---|---|
| [36] | Reliability | Area, latency | ASIC |
| [37] | Reliability, circuit lifetime | Latency | Data-flow and control-flow intensive circuits |
| [38] | Reliability, area | - | Data-flow and control-flow intensive circuits |
| [39] | Reliability | Area, latency | ASIC |
| [40] | Area, latency | - | ASIC |
| [41] | Area | Latency | Data-flow and control-flow intensive circuits |
| [42] | Area | Latency | DSP algorithms with blocked schedules |
| [43] | Latency | - | Circuits with dual-edge-triggered flip-flops |
| [44] | Area, Latency | Performance yield | ASIC |
| [45] | Energy | - | Dual supply voltage circuits |
| [46] | Energy | - | Multi-supply voltage ASIC |
| [47] | Energy | Area, latency | ASIC |
| [48] | Energy | Latency | Pipelined ASIC |
| [49] | Energy, task delay | Latency | ASIC |
| This study | Reliability, energy | Area, latency | Pipelined multi-supply voltage ASIC |

Figure 3.1 presents the citation and publication statistics on ILP-based HLS studies in the literature indexed on the Web of Science over the last three decades. Furthermore, Figure 3.2 shows the percentages of the total number of ILP-based HLS studies in the literature classified according to the Web of Science categories.

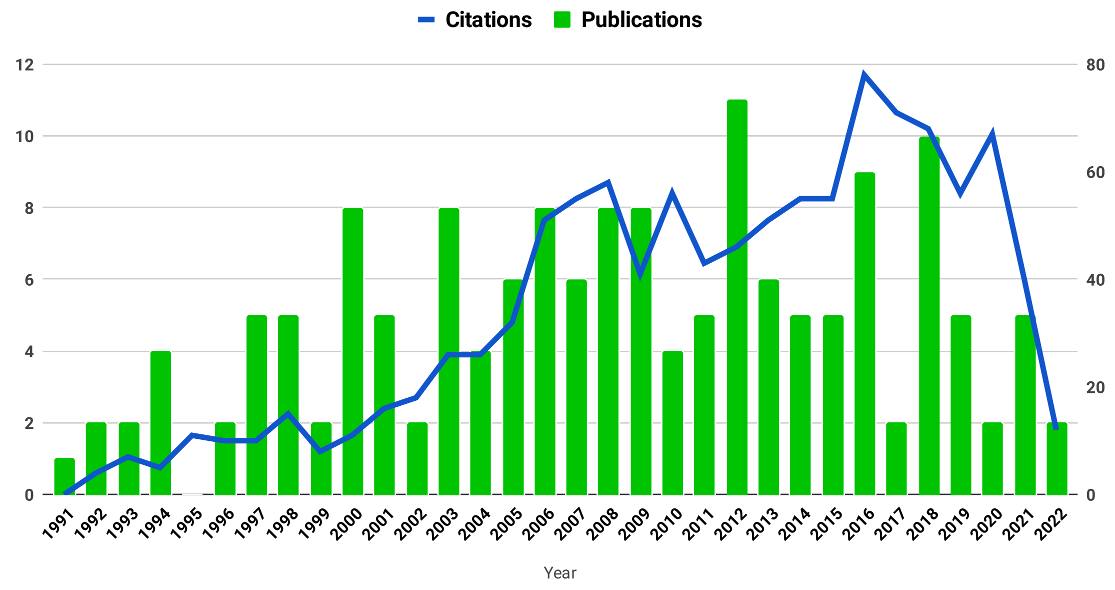


Figure 3.1 ILP-based studies for HLS: times cited and publications over time on Web of Science.

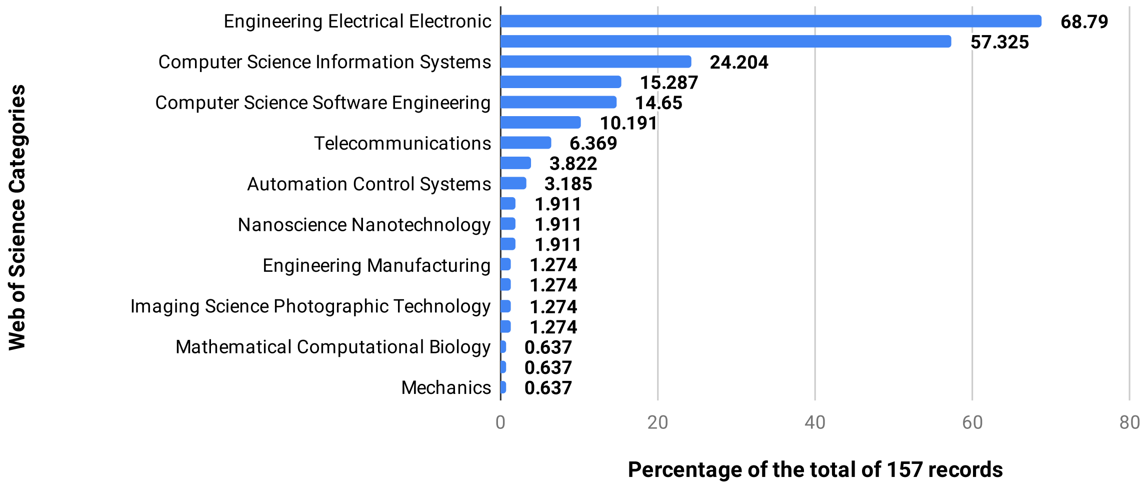


Figure 3.2 ILP-based studies for HLS: literature record count percentages based on Web of Science categories.

## 3.2. SA-Based HLS Studies

Simulated annealing has been a popular metaheuristic method of choice for several HLS studies in the literature. In [50], an SA-based HLS algorithm for pipelined datapath synthesis was proposed. The algorithm optimizes the latency and resource cost and can also take those metrics as design constraints. In [51], the authors focused on optimizing the design area for the HLS process in FPGAs. Their SA-based method employs rescheduling and switching the tasks between resources to look for better solutions in terms of area. The experimental results showed that the proposed approach outperforms the conventional HLS flow in terms of optimizing the exploration of search space up to 37% and achieving more than 11% reduction in the overall design area. However, these studies did not focus on reliability or energy optimization, and no design constraints were considered in the proposed method in [51].

In [52], the authors focused on the area optimization problem in HLS of multiple word-length DSP algorithms on heterogeneous-resource FPGAs. They used different variations of resources, as we did in this study, to achieve area improvements of up to 60% compared to logic-based approaches. Their SA-based HLS method takes latency as a constraint while minimizing the area; however, no reliability or energy considerations are taken during the HLS process.

SALSA, a scheduling method in HLS for optimizing HW resource cost under the latency constraint, was proposed in [53]. It employs an SA-based approach for improving the initial schedule that satisfies the timing constraint by exploring alternative schedules to minimize resource cost. Again, the energy and reliability considerations were left out of the HLS process.

The authors in [54] focused on the area and latency optimization in performance yield-guaranteed HLS. They proposed an SA-based HLS algorithm with statistical static timing analysis. They employed a similar approach to their multi-objective problem in this study, namely the weighted scalarization method that is also employed in our study. The experimental results showed an average area reduction of 14% under the 95% performance yield constraint. This method, however, also does not tackle reliability or energy considerations.

In [55], the authors presented a unified method for HW/SW codesign, incorporating an SA-based HLS approach for multi-FPGAs. Their objective was to minimize resource utilization under the latency constraint. This study did not include reliability or energy considerations in the proposed HLS method. Another study that employed an SA-based approach to HW/SW codesign and HLS for low-power embedded systems is [56]. The authors proposed an HLS method intending to minimize overall power consumption under the given latency constraint. The experiment results showed that the proposed method achieves more effective performance trade-offs than the task-level codesign. However, this study did not investigate the effect of the power savings on the final design reliability.

Apart from the area and latency considerations, testability of the final RTL designs was also considered as an objective in [57]. The authors proposed an enhanced SA-based algorithm for HLS of digital systems, which optimizes the area, latency, and testability metrics. The experimental results showed that the proposed enhanced SA-based algorithm achieves better than the conventional SA-based approach. Nevertheless, this study did not focus on the energy or reliability of the final designs.

In [58], the authors tackled the problem of HLS for digital systems targeted for high-density FPGAs with run-time reconfigurable HW resources with pipelined execution. They proposed an SA-based optimization algorithm that optimizes area and latency but disregards the reliability or energy considerations.

In [59], the authors proposed a method based on three metaheuristic approaches (SA, GA, and ACO) for improved HLS design space exploration (DSE). Their objective was to optimize the design area subject to latency and area constraints. They compared the performance of the proposed method to the traditional approach with a weighted-sum cost function and showed that the proposed method results in up to ten times better Average Distance to Reference Set (ADRS) metric than the traditional approach, although with some increase in running times. This method can be applied to ASIC design; however, the researchers did not focus on the reliability or energy aspects of the final designs.

In [60], the authors tackled the same problem of optimizing HLS design space exploration using the same three metaheuristic approaches. However, they added a machine learning (ML)-based phase to their method flow, which generates predictive models that result in Meta-Heuristic Specific Parameters (MHSP) used in the following phase of metaheuristic HLS DSE. Additionally, they proposed a combined DSE method that uses all three metaheuristics simultaneously. The experimental results showed that their proposed method achieved about 2x better ADRS on average than the default approach with similar execution times. Another study that employed ML to improve SA-based HLS DSE is [61]. The authors used an ML-based technique to obtain a decision tree using a standard SA approach, which is then used to update the synthesis directives for improved optimization of the objective function cost. They achieved up to 48% faster execution than the traditional metaheuristic approach with similar performance. Although these methods can be employed for multi-objective optimization in HLS, they do not consider any design constraints.

In [62], and then in [63], the authors proposed LOPASS, a low-power architectural synthesis system for FPGAs that includes SA-based HLS. The proposed tool takes either latency constraint or resource constraint and optimizes the energy consumption and interconnections in the datapath via multiplexer optimization. The experimental results demonstrated a significant improvement in power reduction compared to a few other existing academic or commercial synthesis tools. The proposed system takes only one metric as a constraint and does not consider the resulting reliability.

In low-power digital circuits that employ power reduction techniques, sub-threshold leakage currents play a great role in total power dissipation. Efficient dual-threshold voltage techniques can alleviate standby power dissipation in combinational circuits [64]. In [65], the authors presented an efficient SA-based scheduling method for resources with dual-threshold voltage techniques to minimize the leakage power under the latency and resource constraints. Nevertheless, the reliability of the circuits was not taken into consideration.

Gate leakage current is not a trivial issue in CMOS technologies below 65 nm. In [66], the authors proposed an SA-based algorithm for minimizing gate leakage current and area overhead during HLS. The average achieved minimization of gate leakage was over 76%, while the area overhead was increased to about 17% on average. The proposed method does not consider reliability or latency during the HLS process.

Voltage islands, a power consumption reduction technique we employed in this study, can also be effective in minimizing the circuit power leakage and spurious switching activity (SSA) [67]. SSA can occur in circuits in which resources (functional units or registers) have more than one task or variable bound to it, which results in different values being output in different clock cycles. This can affect other parts of the circuit if not properly handled. In [68], the authors proposed an SA-based register binding method to alleviate SSA in HLS. The experimental results showed that the proposed method achieved a 40% reduction in SSA on average, leading to improved energy consumption overall. This study did not consider the reliability of the final designs nor any design constraints.

Thermal effects are another crucial issue in present-day VLSI circuits due to increasing power densities that lead to elevated on-chip peak temperatures. In [69], the authors presented an SA-based HLS and floorplanning method for energy and temperature-aware IC designs. The experiments showed that the proposed method could lower peak on-chip temperatures by 12% on average compared to traditional algorithms that only focus on average power optimization. However, this achievement comes at the cost of an average area overhead increase of up to 15%. No constraints were taken into consideration, nor the effects of these improvements on the circuit reliability were discussed.

In bus-based architectures, crosstalk violations are another issue that can be tackled at the HLS stage of the design. In [70], the authors presented an SA-based HLS method with the objective of minimizing the crosstalk while achieving designs with minimum possible latency and area. They managed to achieve a 75% reduction in crosstalk violations on average compared to a traditional flow. In [71], the same problem was tackled in a similar manner, achieving also an average 23% of improvement in performance. These studies do not take reliability or energy consumption into consideration in their objective functions.

In VLSI, routing congestion is also a serious issue that affects performance. The authors of [72] proposed a routing congestion-aware HLS algorithm that employs simulated annealing. The proposed method produced up to 40% better results (with less congestion) than the traditional method it was compared to. Nevertheless, the method does not take reliability or energy into consideration during the HLS process.

In [73], the authors presented a VLSI synthesis system for HLS of image processing architectures, which integrates SA-based algorithms for both datapath and control synthesis. The objective of the proposed system is to minimize energy in terms of the number of used resources (registers and MUXes). The proposed system disregards the effect of energy minimization on the reliability of the final design. COBRA-ABS is another CAD tool used for HLS of digital signal processing algorithms in FPGA custom computing machines (FCCMs), which employs a simulated annealing approach for all HLS steps [74, 75].

In Table 3.2, we present a summary of the related work that employed an SA-based approach to optimization in HLS. We note the optimization metrics considered, as well as design constraints, if any. Figure 3.3 presents the citation and publication statistics on SA-based HLS studies in the literature indexed on the Web of Science over the last three decades.

Figure 3.4 shows the percentages of the total number of SA-based HLS studies in the literature classified according to the Web of Science categories.

It is evident that simulated annealing has been a popular approach to tackling HLS optimization for a number of target systems with varying design requirements and considerations. The great majority of the existing related studies in the literature focus on optimizing area and/or latency, or energy consumption, while some of them even consider the area and/or latency as constraints. None of the existing related studies tries to optimize both reliability and energy simultaneously under the area and latency constraints. The studies that do focus on energy optimization do not investigate its effect on the reliability of the final

Table 3.2 A summary of related SA-based HLS studies.

| Study | Optimization Metrics | Constraints | Target Systems |
|---|---|---|---|
| [50] | Latency, resource cost | Latency and/or cost may be given | Pipelined datapaths |
| [51] | Area | - | FPGA |
| [52] | Area | Latency | Heterogeneous-resource FPGAs |
| [54] | Area, Latency | Performance yield | ASIC |
| [55] | Area | Latency | Multi-FPGA |
| [56] | Energy | Latency | Low-power embedded systems |
| [57] | Area, latency, testability | - | VLSI, ASIC |
| [58] | Area, latency | - | FPGAs with reconfigurable FUs |
| [59] | Area | Area, latency | ASIC, FPGA |
| [62, 63] | Energy, interconnections (MUX) | Latency or resources (but not both) | FPGA |
| [65] | Energy | Area, latency | Low-power digital circuits |
| [66] | Leakage energy, area | - | Nanoscale CMOS datapath circuits |
| [68] | Energy | - | ASIC |
| [69] | Energy, thermal on-chip effects | - | VLSI |
| [70, 71] | Crosstalk, area, latency | - | Bus-based architectures |
| [72] | Routing congestion | - | VLSI |
| [73] | Energy | - | VLSI image processing applications |
| This study | Reliability, energy | Area, latency | Pipelined multi-supply voltage ASIC |

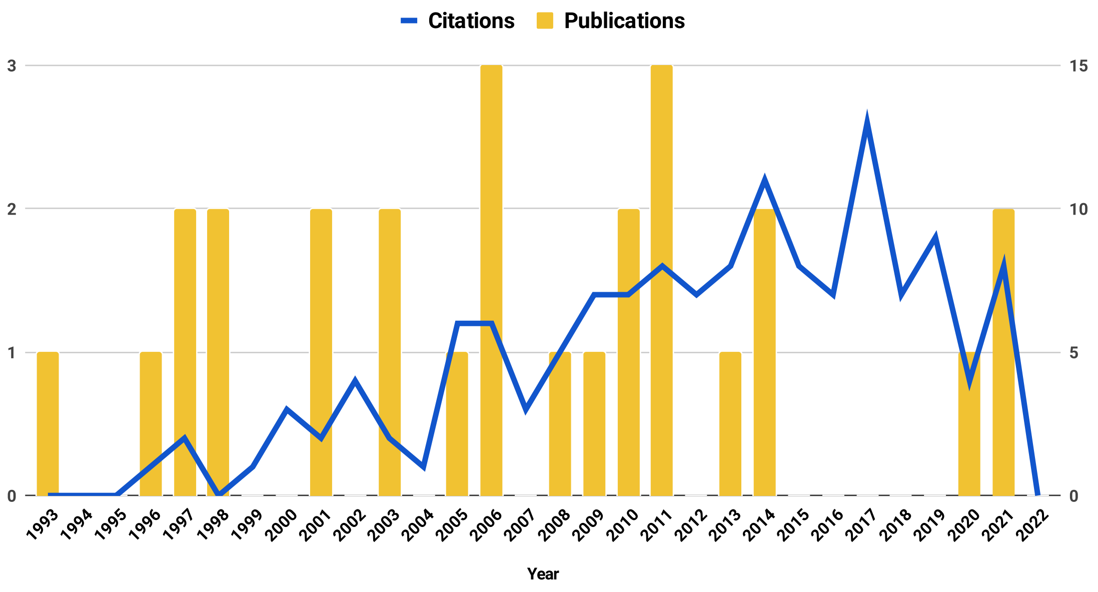


Figure 3.3 SA-based studies for HLS: times cited and publications over time on Web of Science.

designs. Hence, there is a necessity for efficient HLS methods that will take all those features into consideration.

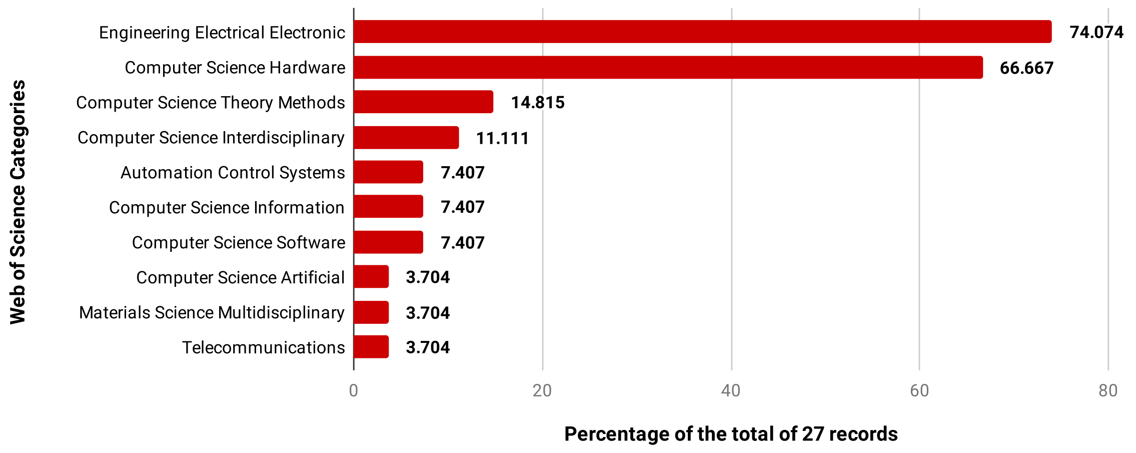


Figure 3.4 SA-based studies for HLS: literature record count percentages based on Web of Science categories.

## 3.3. HLS Studies With Modular Redundancy

Hardware redundancy is an effective method for boosting the reliability of designs. For example, [76], and [77] employed triple modular redundancy (TMR) in the HLS design flow of SRAM-based FPGA for obtaining hardened RTL designs. In [78], TMR was also used in HLS of fault-tolerant hardware accelerator systems with Pareto-optimal configurations of varying area, latency, and reliability characteristics. Similarly to this study, in [79], the authors used DMR to obtain designs that are more resilient to faults. They proposed a duplication-based HLS method for low-cost and resilient application-specific datapath processors. Although these methods lead to a significant boost in fault tolerance, the added area overhead must be considered.

In [80], and [81], the authors also integrated TMR in HLS for FPGAs, but with a focus on exploring trade-offs for balanced designs with the optimized area, latency, and redundancy. Another study that aimed to decrease the area overhead of DMR and TMR approaches is [82]. The authors proposed an HLS method based on resource sensitivity by categorizing the resources as sensitive, semi-sensitive, and insensitive. They suggested applying TMR only on the sensitive resources, while the semi-sensitive resources were treated with gate sizing.

They showed a significant decrease in area overhead and, thus, in energy cost, compared to the traditional TMR method.

## 3.4. HLS Under Multi-Supply Voltages and Other Energy-Aware Approaches

In [83], the authors focused on multi-voltage scheduling in power-efficient HLS under latency constraints with the objective of minimizing the overall area and proposed an ILP formulation further reduced to a piece-wise linear programming problem. The first step of the proposed method is to obtain predefined mobility of the tasks, which is subsequently used in dependency-free task scheduling. This piece-wise approach proved to be very efficient in terms of optimizing running time for all the benchmarks used in the experiments.

Clock gating is another energy reduction technique useful for dynamic power reduction in sequential circuits. In [84], the authors employed this approach and presented an ILP formulation for the HLS of clock control logic with the objectives of minimizing both the area and energy cost.

Similarly to the approach in this study, the authors of [45] proposed an ILP and a heuristic algorithm for HLS of designs that use VIs with dual supply voltage to minimize energy consumption. Unlike this study, they do not consider the effect of energy reduction on reliability.

In [46], the authors presented an ILP-based HLS method for designs with multi-cycling and multiple supply voltages with the goal of optimizing energy consumption. The energy optimization in the proposed approach is achieved through the minimization of both module and cycle peak powers. The experimental results showed the effectiveness of the proposed model in obtaining designs with optimal energy consumption; however, the effect on reliability was not considered in the study.

## 3.5. Other HLS-Related Studies

There have also been several other heuristic and metaheuristic attempts for optimizing reliability with different HLS approaches, such as starting with the most reliable solution and then updating the allocated resources of the longest and most area-consuming tasks until an acceptable solution is obtained [33]. In [85], the authors employed different versions of the available resources for achieving higher fault tolerance. [86] focused on HLS approaches for multi-cycle transient faults. HLS methodology for combinational circuit HLS was also tackled in [87, 88]. Contrary to this study, none of the mentioned related work which focused on maximizing the reliability incorporated the energy consumption considerations into the HLS process.

Other related work that focused on energy consumption optimization usually apply varying supply voltage techniques in their designs [89, 90]. For example, DVS has been the most popular energy optimization technique since it was proposed by [91]. In [92], several methods for low-power HLS design of CMOS circuits are presented. A few other studies that tackled power-aware HLS include [93–95].

To the best of our knowledge, none of the previous related studies incorporated both reliability and energy consumption considerations into the HLS steps under multiple constraints (latency and area), except for [96], in which modular redundancy for increased reliability was not considered in the proposed ILP formulations, and the proposed GA-based metaheuristic method was not able to generate arbitrarily close solutions to the optimum ones in all test cases. Moreover, partial selective duplication was only considered as a post-processing step in the proposed GA method and thus, did not provide optimal or acceptably near-optimal results.

A few studies focus on reliability and energy considerations for multiprocessor architectures; however, those methods cannot be readily applied to the ASIC design due to the area considerations. In this study, we propose both ILP formulations for designs without duplication and two additional DMR-based HLS methods that tackle the same problem.

These methods achieve improved design solutions in terms of optimizing both reliability and energy costs. Furthermore, we propose an SA-based metaheuristic method that outperforms the GA-based method and provides optimal or near-optimal solutions for all benchmarks.

# 4. PROBLEM DEFINITION

In this section, we discuss the basic components of the problem at hand. We briefly introduce high-level synthesis, define the problem at hand, and present an extended resource library with multiple supply voltages used in this research.

## 4.1. High-Level Synthesis

High-level synthesis is a process of converting a behavioral description of a digital system (e.g., given in the form of a directed acyclic graph) to a corresponding register transfer level (RTL) netlist under the given design constraints, if any. The goal of raising design considerations to a higher level of abstraction is to facilitate processes such as verification, power management, memory organization, and synthesis while simultaneously enabling the reuse of high-level specifications. This is especially beneficial for ASIC, and field-programmable gate array (FPGA) designs [97]. HLS can greatly benefit the process of design and production of digital circuits as it reduces the cost and time of both design and production by enabling designers to conceptualize circuit designs regardless of the target architecture or design style and by greatly improving the correctness and optimality of the final designs [6].

Modeling of the digital circuits on a high level is performed with the help of hardware description languages (HDLs) that differ from traditional programming languages in several aspects. HDLs are used to characterize either behavior or structure of a digital circuit, and they describe parallel hardware operations as opposed to programming languages that usually represent a sequential flow of software operations. Several CAD tools use HDLs for the simulation and synthesis of digital circuits. The lack of HDL standardization poses an issue in high-level modeling. HLS deals with behavioral modeling and allows designers to introduce optimization into the process of digital circuit design at a higher level of abstraction [6, 98].

The design steps of the HLS process are illustrated in Figure 4.1 that has been adopted from [97].

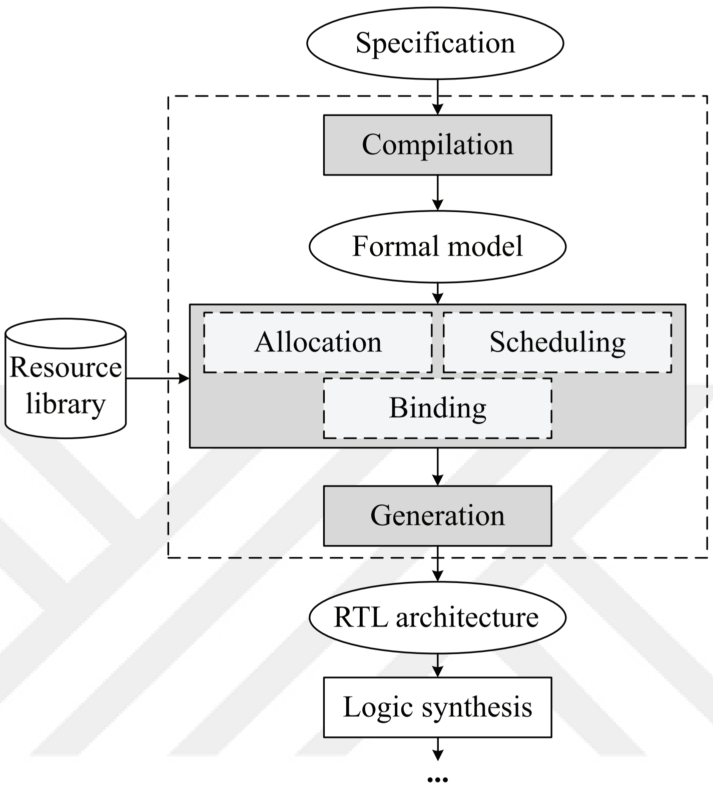


Figure 4.1 HLS design steps.

In this study, we address the resource allocation, binding, and scheduling stages of HLS for optimized designs in terms of reliability and energy. In allocation and binding, hardware resources (e.g., various FUs such as adders and multipliers, as well as storage and connectivity elements such as registers and buses) that are available from the resource library are assigned to the tasks and variables in the model based on criteria such as their types of operation and the design constraints, if any. A resource library is a collection of hardware resources that may include anything from FUs to storage elements and connectivity components, each characterized by its area, latency, and energy consumption properties [97]. Having multiple instances of a resource with the same type of functionality but varying properties complicates the allocation problem. Still, at the same time, it allows for more flexible trade-off decisions to be made when optimizing the final designs. For example, an addition operation can be performed by a fast adder (e.g., carry look-ahead) that has a large area if the area constraint will not be violated, or a smaller adder can be allocated instead (e.g., ripple carry) but with a cost of somewhat slower execution [99].

Scheduling, on the other hand, is a process of determining the start times (cycles or control steps) of each operation (task) in a design. The start time of a task dependent on at least one other predecessor operation will be subject to both the start time and delay of its predecessor tasks. The tasks that are not dependent on each other may be scheduled to run concurrently, granted there are enough resources. Scheduling and resource allocation are closely coupled processes [99]. Allocated resources impact scheduling because the latency of a task depends on the delay of the resource allocated to it. Conversely, scheduling affects allocation since the total number of necessary resources in a design depends on the maximum number of tasks scheduled at any same control step. Spreading the start times of tasks over a wider range of control steps can minimize the number of tasks scheduled in any single control step, thus leading to a reduced number of necessary resources overall. Consequently, the latency-area trade-offs can be made during HLS based on the design requirements. This is illustrated in Figure 4.2, where the schedule in (a) necessitates two adders and one multiplier, whereas the schedule in (b) uses only one adder for both additions by scheduling them in different control steps. This results in a smaller circuit area but a longer overall delay by one control step.

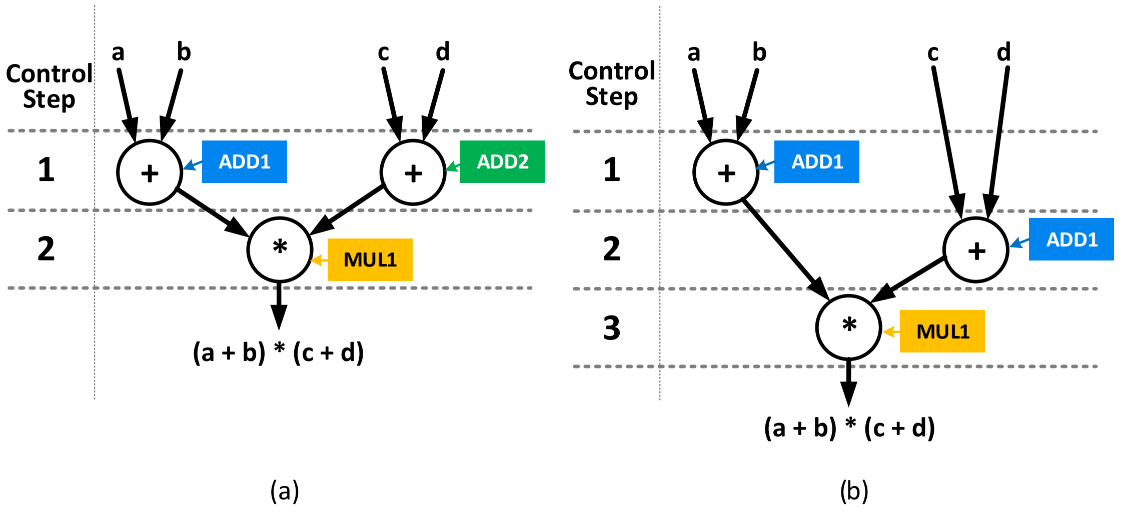


Figure 4.2 (a) A faster schedule with larger area, (b) A slower schedule with smaller area.

In some cases, moving the start time of a task may not introduce extra latency if the task is not on the critical path while simultaneously reducing the area. This is discussed in more detail in Section 5.2.1..

In this research, we focus on pipelined designs. Functional pipelining is an effective approach that increases the circuit performance by allowing circuit elements to be partitioned into stages such that more than one task can be executed concurrently on the same resource (resource sharing). In pipelined execution, each task can be in a different stage as the partial results are latched after each stage [99], and the output of a preceding stage is fed as the input to the next stage [6]. Scheduling considerations in pipelined circuits are, therefore, slightly different than concerns about scheduling in non-pipelined circuits. Although latching of the partial results may increase the task delay, it increases the overall throughput of all tasks proportionally to the number of stages, which results in increased circuit performance.

HLS is a known NP-hard problem, and it has been shown that each of its mentioned sub-problems of resource allocation and scheduling are NP-complete [74]. Therefore, there is a necessity for efficient heuristic or metaheuristic approaches to tackling this problem, which will be able to generate optimal or near-optimal solutions in practically acceptable running times, especially for larger target designs.

## 4.2. Behavioral Description of a Target Design

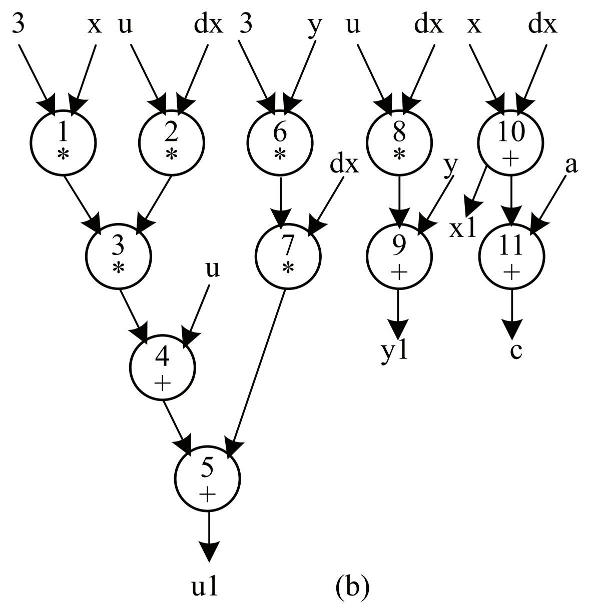
This study focuses on proposing methods for the resource allocation, binding, and scheduling steps of the HLS design with the goal of maximizing the reliability and minimize the energy consumption of the final design under the given latency and area constraints.

The fundamental features of a target design can be captured as abstract behavioral models so that their behavioral representation may be independent of any specific language. Graph representations are widely used in several different approaches to modeling digital circuit behavior (e.g., data-flow graph) [6]. The inputs to our proposed methods include such an abstract behavioral description of a target system given as a data-flow representation in a form of a directed acyclic graph (DAG) whose vertices represent operations (tasks) and edges represent dependencies (precedence constraints) between those tasks, and the available resource library with clearly defined types of the resources, along with their respective area, latency, energy, and reliability values under the available supply voltage levels. In our study, we consider only data-flow intensive operations and do not tackle memory-related parts of the target design since they exhibit different characteristics [100]; hence, we leave memory considerations as future work.

An example behavioral description of a target system (differential equation solver) represented as a DAG is given in Figure 4.3, adopted from [101]. The added *source* and *sink* nodes are dummy nodes taken as “No Operations” (NOP), which merely facilitate the implementation of the scheduling process. The symbols inside the nodes represent the operation type.

```
x1 = x + dx;
u1 = u - (3*x*u*dx) - (3*y*dx);
y1 = y + u*dx;
c = x1 < a;
```

(a)



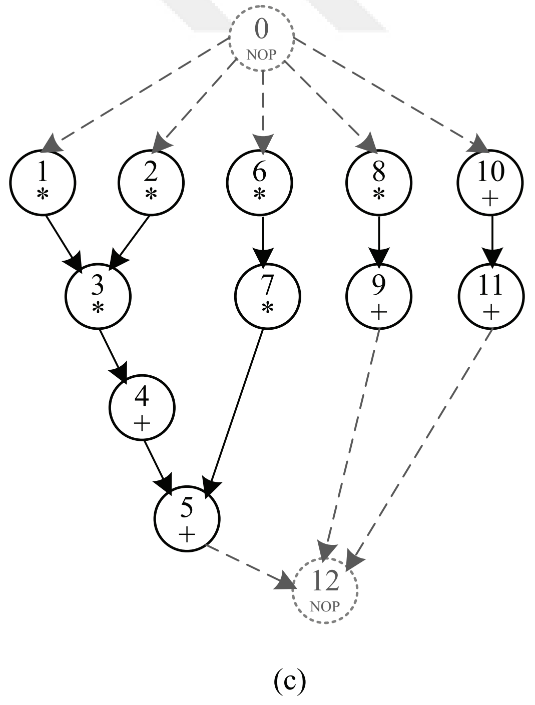


Figure 4.3 (a) An example design specification for differential equation solver, (b) Data-flow representation with dependencies (precedence constraints), and (c) DAG of the design specification.

In a DAG representation, node $n_i$ is said to be a predecessor of node $n_j$ if there is an edge (or a path) from $n_i$ to $n_j$. Analogously, $n_j$ is said to be a successor of $n_i$, and that it depends on $n_i$. Hence, the edges define the dependency (precedence constraints) among the tasks.

## 4.3. Resource Library Under Multi-Supply Voltages

In our previous study [96], we proposed a resource library including three adders and two multipliers under two voltage levels (high voltage level $v_h$ of 1.2 V, and low voltage level $v_l$ of 1.0 V). For the purposes of this research, particularly to be able to demonstrate the effects of VIs on reliability and energy consumption in circuit design, we present the extended resource library to three voltage levels in Table 4.1.

Table 4.1 Resource library used in this study.

| **Type** | **Resource Name** | **A** | $\mathbf{L}_h$ $\mathbf{L}_m$ $\mathbf{L}_l$ | $\mathbf{R}_h$ $\mathbf{R}_m$ $\mathbf{R}_l$ | $\mathbf{E}_h$ $\mathbf{E}_m$ $\mathbf{E}_l$ |
|---|---|---|---|---|---|
| Adder (A1) | Ripple Carry | 2 | 5 | 0.999 | 12.00 |
| | | | 6 | 0.9986 | 10.83 |
| | | | 8 | 0.998 | 8.33 |
| Adder (A2) | Brent Kung | 3 | 3 | 0.969 | 5.00 |
| | | | 4 | 0.9545 | 4.13 |
| | | | 5 | 0.938 | 3.47 |
| Adder (A3) | Kogge Stone | 5 | 2 | 0.987 | 6.00 |
| | | | 3 | 0.9783 | 4.96 |
| | | | 3 | 0.976 | 4.17 |
| Multiplier (M1) | Carry Save | 8 | 10 | 0.999 | 80.00 |
| | | | 12 | 0.9989 | 66.12 |
| | | | 16 | 0.998 | 55.56 |
| Multiplier (M2) | Carry Lookahead | 12 | 15 | 0.969 | 160.00 |
| | | | 19 | 0.9567 | 132.23 |
| | | | 25 | 0.938 | 111.11 |

For obtaining the new latency, energy, and reliability values under the medium supply voltage level ($v_m$) of 1.1 V, we used 0.5-V threshold voltage level and Equations (10), (11), (12), and

(7), respectively. In Table 4.1, A is the area of the resource measured in unit area. $L_h$, $L_m$, and $L_l$ represent the latency values of the corresponding resources under high, medium, and low voltage respectively, and they are measured in time steps (control steps). Similarly, $R_h$, $R_m$, and $R_l$ represent the reliability values, whereas $E_h$, $E_m$, and $E_l$ represent the energy consumption under high, medium, and low voltage measured in nanojoules (nJ), respectively.

The task scheduling problem in HLS tackles the scheduling of the start times of all operations given in a DAG to clock cycles (control steps) while considering both the given latency constraint and the task dependencies. For instance, some tasks may execute in parallel if there is no data dependency between them and if enough resources are available for parallel execution. On the other hand, the resource allocation problem deals with assigning available hardware resources from the resource library to the tasks in a DAG such that the resulting overall area of the design does not exceed the given area constraint.

## 4.4. Modular Redundancy Considerations

In duplicated designs, checker circuits can be added to check the correctness of the duplicated task's computation, as illustrated in Figure 4.4. Addition of checker circuits will introduce some additional delay to the design [102]. However, since the delay and area of checker circuits are negligible compared to the overall design of a combinational circuit, in our proposed methods, we do not consider the added delay and area of the checkers in our design process. Moreover, checker circuitry may also be susceptible to soft errors; however, we do not include this problem within the scope of our study and assume that the reliability of checkers is ensured.

The sample design solutions obtained through HLS of the target system given in Figure 4.3 under two voltage levels (high and low) and the given latency (L) and area (A) constraints, using the resource library given in Table 4.1, are illustrated in Figure 4.5. The tasks scheduled on the high voltage island are represented with white, whereas those scheduled on the high voltage island are represented with grey oval shapes.

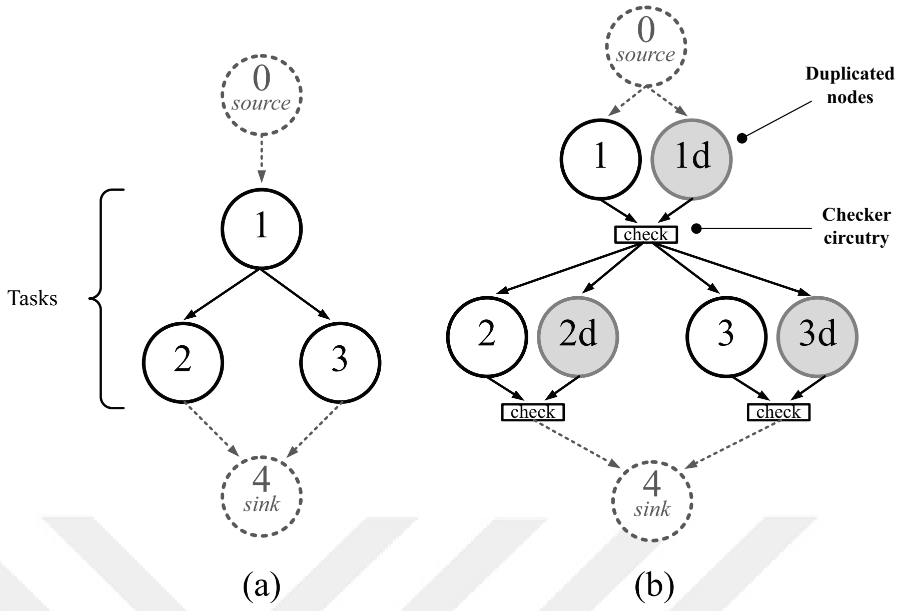


Figure 4.4 (a) An example task graph, (b) its fully duplicated version.

While performing scheduling and resource allocation, the goal is to obtain the optimum solution based on the objective function. In our work, we focus on both maximizing reliability and minimizing energy consumption, which makes the problem very challenging for the following reasons. Both scheduling and resource allocation problems are known NP-hard/complete optimization problems that occur in many different areas of life. In HLS, both problems are tackled simultaneously. Additional complexity is introduced because the available resource library has a variety of possible resources of the same type from which the optimum ones must be chosen, with varying area, latency, reliability, and energy consumption values that need to be considered simultaneously. We also employ multi-supply voltages that additionally complicate the model. Moreover, the flexible multi-objective nature of the problem at hand necessitates different trade-offs to be considered as well.

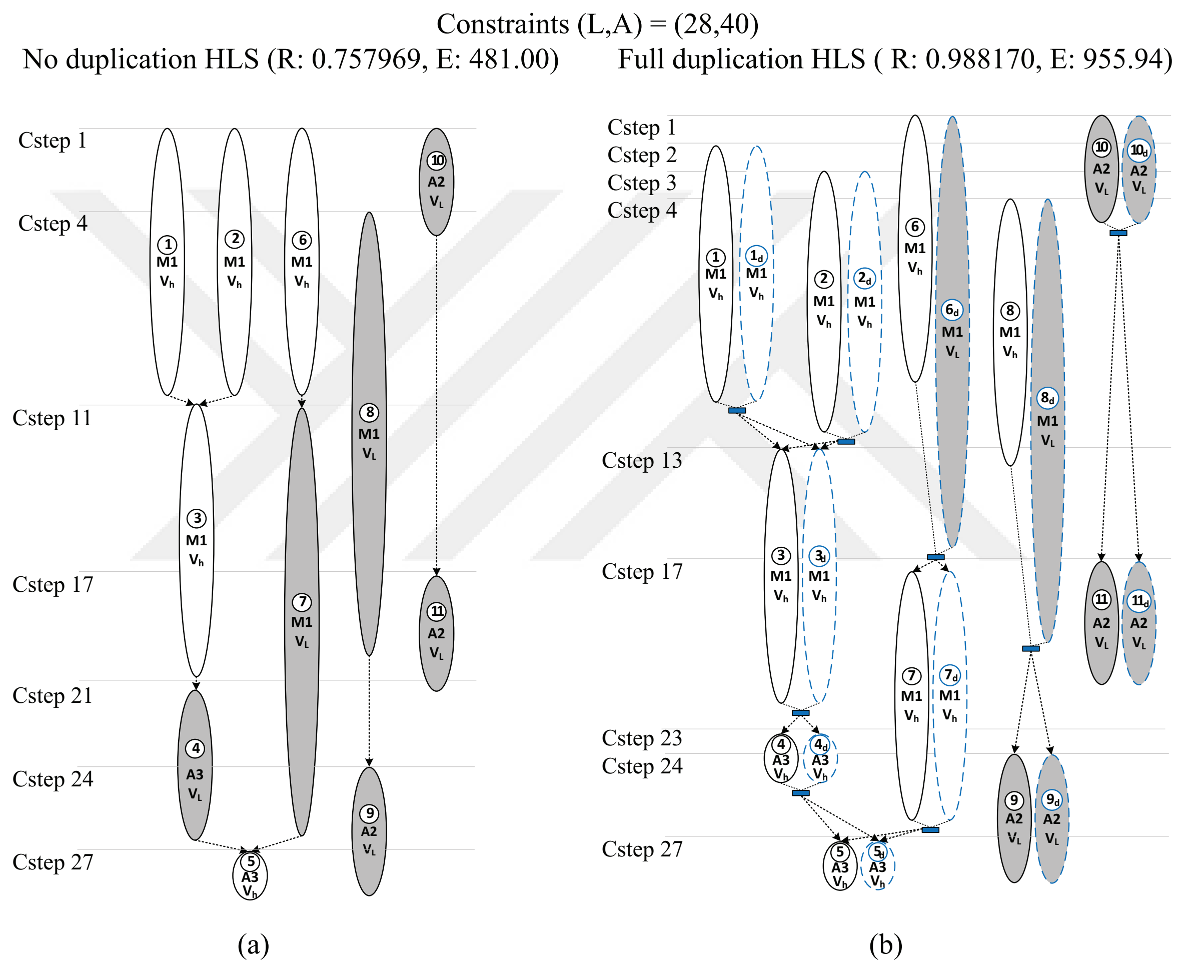

Constraints (L,A) = (28,40)
No duplication HLS (R: 0.757969, E: 481.00)
Full duplication HLS ( R: 0.988170, E: 955.94)
Cstep 1
Cstep 2
Cstep 3
Cstep 4
Cstep 11
Cstep 13
Cstep 17
Cstep 21
Cstep 23
Cstep 24
Cstep 27
M1
A2
A3
$V_h$
$V_L$
(a)
(b)

Figure 4.5 (a) HLS result without duplication, (b) HLS result with full DMR under the specified constraints.

# 5. PROPOSED METHODS

In this section, we present our proposed ILP and SA-based HLS methods, whose goal is to maximize the total reliability while minimizing the total energy consumption of the given application designs under the desired area and latency constraints.

## 5.1. Integer Linear Programming Formulations

In this section, we present our ILP formulations of the problem for the non-duplicated model, as well as both partial and full duplication methods for scheduling and resource allocation steps of HLS, whose goal is to maximize the total reliability while minimizing the total energy consumption of the given application designs under the desired area and latency constraints. We present all three models simultaneously, noting where they differ while providing their respective equations. The notations used in the ILP formulations of the problem are defined in Table 5.1.

The Boolean variable $\zeta_{i,j}$ refers to the compatibility of the task $T_i$ with the resource $R_j$ (e.g., an adder resource can only be assigned to an addition operation), and it is formulated in Equation (13).

$$\zeta_{i,j} = \begin{cases} 1 & \text{if } T\_type_i = R\_type_j \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The Boolean variable $\mathcal{A}_{i,j,v}$ specifies if the resource $R_j$ is assigned to the task $T_i$ under the voltage level $V_v$ (see Equation (14)). Similarly, the Boolean variable $\mathcal{A}\_d_{i,j,v}$ specifies if the resource $R_j$ is assigned to the duplicated instance of the task $T_i$ under the voltage level $V_v$

$$\mathcal{A}_{i,j,v} = \begin{cases} 1 & \text{if } T_i \text{ is assigned to } R_j \text{ under } V_v \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Table 5.1 ILP notations.

| | |
|---|---|
| $T = \{T_i : i = 1, ..., N\}$ | A set of N tasks (additions, multiplications, NOPs) where $T_i$ is the $i^{th}$ task in $T$ |
| $T\_type_i$ | The type of $T_i$ (addition, multiplication, NOP) |
| $R = \{R_i : i = 1, ..., M\}$ | A library of M available hardware resources where $R_i$ is the $i^{th}$ resource in $R$ |
| $V = \{V_l, V_m, V_h\}$ | A set of available voltage levels (high $V_h = 1.2$ V, medium $V_m = 1.1$ V, low $V_l = 1.0$ V) |
| $V_i$ | The voltage at the voltage level $i$ |
| $R\_type_j$ | The type of $R_j$ |
| $\zeta_{i,j}$ | The compatibility of the task $T_i$ with the resource $R_j$ |
| $\mathcal{A}_{i,j,v}$ | Denotes if $R_j$ is assigned to $T_i$ under $V_v$ |
| $\mathcal{A}\_d_{i,j,v}$ | Denotes if $R_j$ is assigned to a duplicated $T_i$ under $V_v$ |
| $Csteps$ | A set of control steps |
| $Start_{i,s}$ | Denotes if $Cstep_s$ is the start time of the task $T_i$ |
| $Start_N$ | The start time of the last $sink$ task |
| $G = (T, PREC)$ | Precedence graph where $PREC(i, j)$ means $T_i$ precedes $T_j$ |
| $Rel_{j,v}$ | The reliability of $R_j$ under $V_v$ |
| $A_j$ | The area occupied by $R_j$ |
| $L_{j,v}$ | The latency of $R_j$ under $V_v$ |
| $E_{j,v}$ | The energy consumption of $R_j$ under $V_v$ |
| $\rho_i$ | The reliability of $T_i$ |
| $\delta i$ | The delay of $T_i$ |
| $\epsilon_i$ | The energy consumed by $T_i$ |
| $\kappa_{i,s,r,v}$ | Denotes if $T_i$ starts at $Cstep_s$ and is assigned $R_r$ under $V_v$ |
| $NumR_{j,s,v}$ | The total number of instances of $R_j$ used at $Cstep_s$ under $V_v$ |
| $\Upsilon_{r,v}$ | The total number of instances of $R_j$ used within the circuit under $V_v$ |
| *obj* | The objective function |
| $\Lambda$ | Area constraint |
| $\lambda$ | Latency constraint |

In our DMR-based models, duplication of a resource is allowed for partial duplication and a requirement for full duplication. Thus, in partial duplication, one or two resources can be assigned to each task while taking the compatibility of the resources with the tasks into consideration. Similarly, only one voltage island can be assigned. This is formulated in Equation (15) and Inequality (16).

$$\begin{aligned} &\text{While } \zeta_{i,j} = 1 \\ &\forall i \in T\!: \sum_{j \in R, v \in V} \mathcal{A}_{i,j,v} = 1 \end{aligned} \tag{15}$$

$$\begin{aligned} &\text{While } \zeta_{i,j} \ = 1 \\ &\forall i \ \in T\text{:} \sum_{j \in R, v \in V} \mathcal{A}\_d_{i,j,v} \leq 1 \end{aligned} \tag{16}$$

In our full duplication model, two resources must be assigned to each task, and a single voltage island can be assigned to each of them while considering the resources' compatibility with the tasks. This is formulated in Equations (15) and (17).

$$\begin{aligned} &\text{While } \zeta_{i,j} \ = 1 \\ &\forall i \ \in T\text{:} \sum_{j \in R, v \in V} \mathcal{A}\_d_{i,j,v} = 1 \end{aligned} \tag{17}$$

The Boolean variable $Start_{i,s}$ specifies if the task $T_i$ starts at the control step $Cstep_s$. It is formulated in Equation (18).

$$Start_{i,s} = \begin{cases} 1 & \text{if } T_i \text{ scheduled at } Cstep_s \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Any task may be scheduled at only one control step, which is achieved with Equation (19).

$$\forall i \in T\text{:} \sum_{s \in Csteps} Start_{i,s} = 1 \tag{19}$$

The delay (latency) of a task depends both on the latency of the resources assigned to it and the assigned operating voltage levels. The overall latency of a task will be the maximum latency of all resources assigned to it as formulated in Equation (20).

$$\begin{aligned} &\forall i \ \in T\text{:} \\ &\delta i \ = \max_{r \in R, v \in V} \left( \sum_{r \in R, v \in V} L_{r,v} \cdot \mathcal{A}_{i,r,v} \quad , \sum_{r \in R, v \in V} L_{r,v} \cdot \mathcal{A}\_d_{i,r,v} \right) \end{aligned} \tag{20}$$

Some tasks are dependent on others; hence, a task that is dependent on other tasks cannot start before all of its predecessors have finished. These precedence constraints are given in the precedence graph $G = (T, PREC)$, and they must be satisfied. Inequality (21) guarantees that the start time of a dependant task will be greater than the maximum end time of all its precedent tasks.

$$\begin{gathered}\forall(i,j)\ \in T\text{: If } PREC(i,j) = 1\\ \sum_{s\ \in Csteps} Start_{j,s} \cdot s \geq \sum_{s \in Csteps} Start_{i,s} \cdot s + \delta i\end{gathered} \tag{21}$$

The reliability of the task $i$ depends on the reliability of its assigned resources under the applied voltage levels for both the original and duplicated resource, denoted as $\rho_i$ and $\rho\_d_i$, and formulated in Equations (22) and (23), respectively.

$$\begin{aligned}&\forall i\ \in T\text{:}\\ &\rho_i\ = \sum_{r \in R, v \in V} Rel_{r,v} \cdot \mathcal{A}_{i,r,v}\end{aligned} \tag{22}$$

$$\begin{aligned}&\forall i\ \in T\text{:}\\ &\rho\_d_i\ = \sum_{r \in R, v \in V} Rel_{r,v} \cdot \mathcal{A}\_d_{i,r,v}\end{aligned} \tag{23}$$

The Boolean variable $\kappa_{i,s,r,v}$ specifies if the task $T_i$ started at the control step $Cstep_s$ and if the resource $R_r$ has been assigned to it under the voltage level $V_v$. It is formulated in Equation (24). The Boolean variable $\kappa\_d_{i,s,r,v}$ denotes the same case for duplicated tasks, and is formulated in the same way.

$$\kappa_{i,s,r,v} = \begin{cases} 1 & \text{if } T_i \text{ scheduled at } Cstep_s \text{ and } R_r \text{ is allocated to it under } V_v \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

The start time of each task can only be scheduled at a single control step, only one resource can be allocated to each task, and each task can be executed on a single voltage island. This is formulated in Equation (25) and Inequality (26).

$$\begin{aligned}&\forall i \in T: \\ &\sum_{r \in R, s \in Csteps, v \in V} \kappa_{i,s,r,v} = 1\end{aligned} \tag{25}$$

$$\begin{aligned}&\forall (i \in T, s \in Csteps, r \in R, v \in V): \\ &\kappa_{i,s,r,v} \geq \mathcal{A}_{i,r,v} + Start_{i,s} - 1\end{aligned} \tag{26}$$

The similar equations are used to constrain $\kappa\_d_{i,s,r,v}$ in our full duplication method. For partial duplication, on the other hand, some tasks may not be duplicated, hence $\kappa\_d_{i,s,r,v}$ will be zero for an unduplicated $T_i$. This is formulated in Equation (27) and Inequality (28).

$$\begin{aligned}&\forall i \in T: \\ &\sum_{r \in R, s \in Csteps, v \in V} \kappa\_d_{i,s,r,v} \leq 1\end{aligned} \tag{27}$$

$$\begin{aligned}&\forall (i \in T, s \in Csteps, r \in R, v \in V): \\ &\kappa\_d_{i,s,r,v} \geq \mathcal{A}\_d_{i,r,v} + Start_{i,s} - 1\end{aligned} \tag{28}$$

The amount of energy consumed during a task execution depends on how much energy the resource allocated to that task consumes under the assigned supply voltage level. We formulate this in Equation (29).

$$\begin{aligned}&\forall i \in T: \\ &\epsilon_i = \sum_{r \in R, v \in V} E_{r,v} \cdot \mathcal{A}_{i,r,v} + \sum_{r \in R, v \in V} E_{r,v} \cdot \mathcal{A}\_d_{i,r,v}\end{aligned} \tag{29}$$

To calculate the area of the final design, we first count the total number of instances of each resource allocated overall. Since our datapath is pipelined, each control step should be inspected for only the tasks scheduled in that particular step across every voltage island and count the number of instances of the resources assigned to those scheduled tasks. $NumR_{r,s,v}$ represents the total number of instances of the resource $R_j$ at the control step $Cstep_s$ under the supply voltage $V_v$, and is formulated in Equation (30).

$$\begin{aligned}&\forall(r \in R, s \in Csteps, v \in V): \\ &NumR_{r,s,v} \; = \sum_{i \in Tasks} \kappa_{i,s,r,v} + \sum_{i \in Tasks} \kappa\_d_{i,s,r,v}\end{aligned} \tag{30}$$

Finally, the total number of instances of each resource assigned to each available voltage island, which are used in the overall circuit design, is represented with the decision variable $\Upsilon_{r,v}$. $\Upsilon_{r,v}$ will be the maximum of all $NumR_{r,s,v}$ at any control step as formulated in Equation (31).

$$\begin{aligned}&\forall(r \in R, v \in V): \\ &\Upsilon_{r,v} \; = \max_{s \in Csteps} NumR_{r,s,v}\end{aligned} \tag{31}$$

### 5.1.1. Constraints

To satisfy the area constraint of the overall design, the sum of the areas of all assigned resources must not exceed the given maximum allowed area constraint. We formulate this requirement with Inequality (32).

$$\sum_{r \in R, v \in V} \Upsilon_{r,v} \cdot A_r \leq \Lambda \tag{32}$$

Similarly, to satisfy the latency constraint of the overall design, we need to ensure that the last *sink* task is scheduled such that its start time, denoted as $Start_N$ and defined in Equation (33), does not exceed the given latency constraint. We formulate this with Inequality (34).

$$Start_N = \sum_{s \in Csteps} Start_{N,s} \cdot s \tag{33}$$

$$Start_N \leq \lambda \tag{34}$$

### 5.1.2. Objective Functions

The goal of our bi-objective problem is to maximize the overall reliability of the circuit denoted as $R_{total}$), while minimizing its total energy consumption denoted as $E_{total}$.

The overall reliability of a design with tasks executed in a sequence is calculated as the product of the reliability of each task. Moreover, considering the duplication in our models, we also employ the approach given in Equation (8) to calculate the overall reliability of a design as given in Equation (35).

$$R_{total} = \prod_{i \in Tasks} (\rho_i + \rho\_d_i - \rho_i \cdot \rho\_d_i) \tag{35}$$

However, in an ILP model, non-linear functions like this cannot be used. Thus, we employ the approach of using reliability summation instead of a product as the objective function in our ILP model, which accomplishes the same task of maximizing the overall design reliability. Therefore, while the final overall reliability of a design is still calculated as given in Equation (35), the maximization of the overall reliability is formulated as the objective function (36) in our ILP formulation.

$$\textbf{Maximize } R_{total} = \sum_{i \in Tasks} \rho_i + \sum_{i \in Tasks} \rho\_d_i \tag{36}$$

The minimization of the overall energy consumption is formulated as the objective function (37).

$$\textbf{Minimize } E_{total} = \sum_{i \in Tasks} \epsilon_i \tag{37}$$

We employ the scalarization technique to formulate our bi-objective problem as a single objective function given in (38), as its optimal solution will be one of the Pareto optimal solutions for the original bi-objective problem. The weighted sum of reliability and energy values are combined, while the parameter $\alpha$ is used to assign the desired weight to either objective. When $\alpha = 1.0$, the focus is on optimizing reliability only, whereas, for $\alpha = 0.0$, we disregard reliability and optimize energy consumption only. In this manner, we can assign priority to either objective if necessary or keep the balanced bi-objective function by assigning $\alpha = 0.5$. The higher the $\alpha$ value, the more priority is given to maximizing reliability over minimizing energy consumption and vice versa.

$$\textbf{Minimize } obj = \alpha \cdot (1 - R_{norm}) + (1 - \alpha) \cdot (E_{norm}) \tag{38}$$

To be able to employ the weighted scalarization method, we used normalized values of the total reliability and the total energy consumption ($R_{norm}$ and $E_{norm}$, respectively), which are normalized to the range [0,1]. Normalization of the values are performed according to Equation (39).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{39}$$

The minimum and maximum reliability values of a given circuit used for calculating the normalized values can be obtained by assigning the least (or most) reliable resources in the resource library to every task, respectively. Similarly, we can obtain the minimum and maximum energy expenditure values of a circuit.

## 5.2. Simulated Annealing-Based HLS Method

In this section, we present our SA-based HLS method. We provide details about the scheduling algorithms used in the task scheduling phase of HLS, resource allocation, initial solution generation, neighbor selection, initial temperature calculation, and cooling schedule. The notations used in formulating the initial temperature and acceptance probability calculations are given in Table 5.2.

Table 5.2 Notations used in SA-based HLS method equations.

| | |
|---|---|
| $T$ | Current temperature |
| $N_i$ | Set of the neighbors of the state $i$ |
| $E_i$ | Energy (cost) of the state $i$ |
| $max_t$ | State after a transition $t$ |
| $min_t$ | State before a transition $t$ |
| $t$ | A strictly positive state transition ($E_{\mathrm{max}_t} > E_{\mathrm{min}_t}$) |
| $\delta_t$ | Energy (cost) difference between two states |
| $\pi_i$ | Stationary distribution |
| $\mathcal{P}_t$ | Probability to generate a transition $t$ when the energy states are distributed in conformity with $\pi_i$ |
| $p_t$ | Probability of accepting a positive transition $t$ |
| $\chi(T)$ | Acceptance probability at temperature $T$ |
| $S$ | Random set of strictly positive state transitions |
| $\hat{\chi}(T)$ | Acceptance probability based on $S$ |
| $\chi_0$ | Desired starting acceptance probability |
| $P_{metropolis}$ | Metropolis acceptance probability |
| $\alpha_c$ | Cooling constant |

### 5.2.1. Task Scheduling Algorithms

The general unconstrained task scheduling problem can be defined as follows. Let $G(V, E)$ be a sequencing graph with $|V|= n$ vertices (tasks), and $E$ directed edges that represent task dependencies (precedence constraints). Also, let $L = \{l_i : i = 1..n\}$ represent the latency (delay) of the tasks in $G$, where $l_i$ is the delay of task $v_i \in V$. Scheduling of tasks in $G$ is the problem of determining the start time $s_i$ of each task $v_i \in V$ such that the precedence constraints are satisfied. The overall design latency can be obtained as $(s_n + l_n) - s_1$.

The simplest scheduling problems are unconstrained scheduling problems that do not consider any resource constraints and for which there exist efficient algorithms that generate solutions in polynomial time. Scheduling without resource constraints is useful as it can provide lower or upper bounds on latency that can be used to simplify finding solutions to constrained problems [6]. In our proposed SA-based method, we use two such algorithms, namely As Soon As Possible (ASAP) and As Late As Possible (ALAP) scheduling algorithms [101].

ASAP is useful to obtain a task schedule such that all tasks are scheduled as early as possible, resulting in the minimum possible latency of the overall design. This is necessary for our proposed method because the initial solution generation and neighbor selection are performed randomly. ASAP scheduling provides the lower bound on the design latency, which tells us if the obtained candidate solution meets the desired latency constraint or not.

Our ASAP scheduling algorithm is presented in Algorithm 2. It starts by assigning Task 0 (NOP) at the control step 0 and proceeds to schedule other tasks in topological order to ensure that at each step, all predecessors of the task to be scheduled have already been scheduled. It is necessary to find the latest control step at which any of a task's predecessors end to obtain its ASAP start time.

On the other hand, ALAP scheduling is used to obtain a schedule such that each task's start time is set to the latest possible control step based on the given latency constraint. This scheduling is useful as it enables us to compute if a specific task is on the critical path or if it is flexible such that we can explore a range of possible start times for that particular task without violating the latency constraint of the final design. This flexibility is referred to as task mobility (or slack), and it is a useful feature that enables different design decisions to be made for more optimal solutions. For instance, moving the start time of a mobile task up could lead to a design with a smaller area while still preserving the overall latency constraint of the final solution. This is possible because a resource that has already been utilized in the design can be allocated to the mobile task as a shared resource in some other control step when other tasks are not using it, and thus, eliminate the necessity for adding another

**Algorithm 2** As Soon As Possible Scheduling Algorithm

**Inputs:** $s$ current solution, $t_{ord}$ topological ordering of the tasks, $n$ number of the tasks.
**Output:** $l_{ASAP}$ ASAP scheduling of $s$.

```
procedure GETASAPSCHEDULE($s, t_{ord}, n$)
    $l_{ASAP}[0] \leftarrow 0$                    ▷ Schedule the start task in control step 0.
    $scheduled[0] \leftarrow True$
    for i = 1..(n-1) do
        $scheduled[i] \leftarrow False$
    end for
    $j \leftarrow 1$
    while $scheduled[n-1] == False$ do
        $id \leftarrow t_{ord}[j]$                ▷ Get the ID of the next task in topological order.
        $l_{ASAP}[id] \leftarrow \max\{l_{ASAP}[k] + s[k].latency\}, \forall s[k]$ predecessor of $s[id]$
        $scheduled[id] \leftarrow True$
        $j \leftarrow j + 1$
    end while
    return $l_{ASAP}$
end procedure
```

resource with the same functionality that would be needed if the mobile task were to start sooner.

The difference between ASAL and ALAP scheduling is illustrated in Figure 5.1, where we show ASAP and ALAP schedules for differential equation solver DAG presented in Figure 4.3. Note that we assume a delay of one control step for each operation for the purpose of simplicity.

Tasks 1 through 5 are tasks with zero mobility and are on the critical path. Consequently, their ALAP and ASAP schedules are the same as moving the start time of any critical task up would result in the increased overall delay of the circuit. On the contrary, tasks 6 through 11 have some mobility. For example, tasks 8 and 10 can be scheduled anywhere between control steps 1 and 3. This flexibility is utilized in the List scheduling algorithm for obtaining more optimal schedules with respect to the design area.

In our ALAP scheduling algorithm presented in Algorithm 3, the latency constraint is obtained from ALAP scheduling and taken as the start time of the final dummy *sink* task. It starts with scheduling the *sink* task (NOP) first at the control step $l_{ASAP}[n-1]$ and proceeds

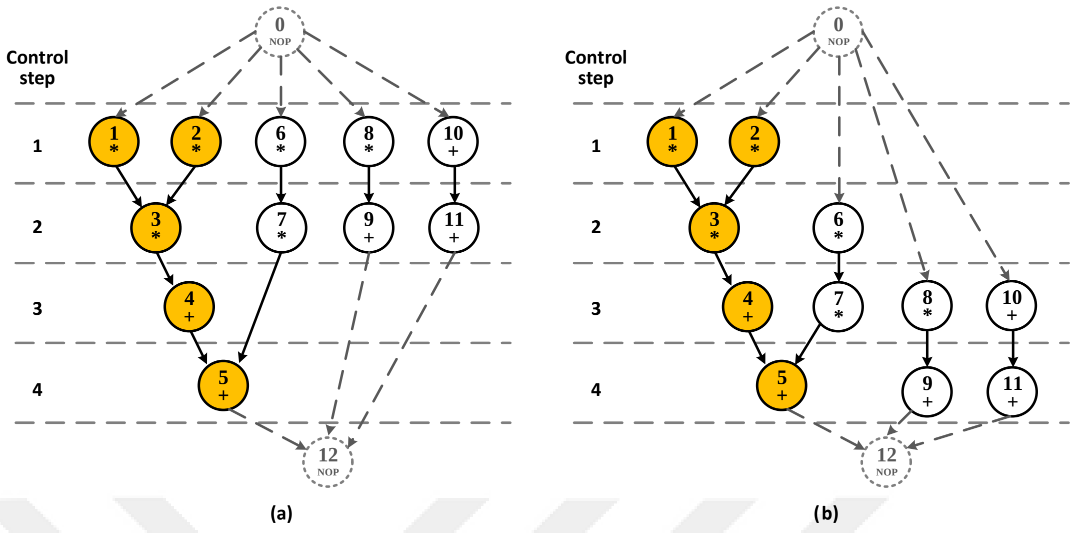


Figure 5.1 (a) ASAP schedule of DES, (b) ALAP schedule of DES.

to schedule other tasks in reverse topological order to ensure that at each step, all successors of the task to be scheduled have already been scheduled. To obtain a task's ALAP start time, it is necessary to find a control step at which it can be scheduled so that its earliest successor will start immediately after its completion.

Contrary to the unconstrained algorithms presented above, scheduling under resource constraints is an NP-complete problem. In our HLS problem, both latency and area are given as constraints. The ASAP scheduling can tell us if our design can satisfy the latency constraint, but to try and meet the area constraint, we also need to perform area-aware scheduling. The ASAP and ALAP scheduling algorithms do not give any guarantees about the final design area. In fact, in the given example of the ASAP and ALAP schedules in Figure 5.1, we observe that ASAP scheduling for that particular circuit results in the need for usage of four multipliers simply because tasks 1, 2, 6, and 8 are all scheduled in the same control step. Similarly, ALAP scheduling for the same sample circuit necessitates three adders because tasks 5, 9, and 11 are scheduled in the same control step.

A more optimal schedule in terms of area is possible if the mobile tasks are scheduled to minimize resource usage in any given control step. One such alternative schedule is

**Algorithm 3** As Late As Possible Scheduling Algorithm

**Inputs:** $s$ current solution, $t_{ord}$ topological ordering of the tasks, $l_{ASAP}$ ASAP scheduling of $s$, $n$ number of the tasks.

**Output:** $l_{ALAP}$ ALAP scheduling of $s$.

```
procedure GETALAPSCHEDULE($s, t_{ord}, l_{ASAP}, n$)
    $l_{ALAP}[n-1] \leftarrow l_{ASAP}[n-1]$    ▷ Schedule the sink task at the minimum latency from ASAP.
    $scheduled[n-1] \leftarrow True$
    for i = 0..(n-2) do
        $scheduled[i] \leftarrow False$
    end for
    $j \leftarrow n-2$
    while $scheduled[0] == False$ do
        $id \leftarrow t_{ord}[j]$    ▷ Get the ID of the previous task in topological order.
        $l_{ALAP}[id] \leftarrow \min\{l_{ALAP}[k] - s[id].latency\}, \forall s[k]$ successor of $s[id]$
        $scheduled[id] \leftarrow True$
        $j \leftarrow j-1$
    end while
    return $l_{ALAP}$
end procedure
```

illustrated in Figure 5.2, in which, at most two adders and two multipliers are enough to satisfy the resource needs.

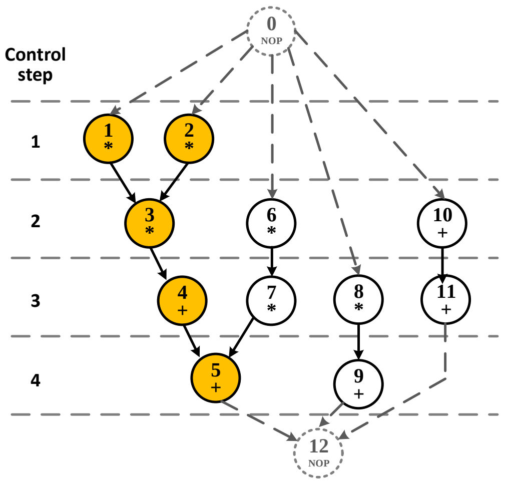


Figure 5.2 An alternative schedule of DES with smaller area.

Mathematical optimization of the constrained scheduling problem, such are our ILP formulations, is not practical in real life for large problems (e.g., VLSI) as the problem is intractable [68]. Therefore, heuristic approaches are generally used to overcome this obstacle. A popular heuristic approach to constrained scheduling is List scheduling, which comes in two forms; either scheduling for minimum latency under the area constraint or scheduling for the minimum area under the latency constraint.

Our SA-based HLS method employs List scheduling for the minimum area under the latency constraint. Our approach is presented in Algorithm 4.

Our List scheduling algorithm performs the final scheduling and resource binding as follows. The ALAP schedule is checked to ensure the obtained schedule produced a feasible solution. Next, the first task to be scheduled is the *start* NOP task. All other tasks are tagged as unscheduled and not ready. The scheduling is performed one control step at a time, starting from control step 1. At each step, each task's remaining mobility (slack) is calculated. Furthermore, the readiness of each task to be scheduled is checked. If any predecessor task is unscheduled or unfinished, the task is marked as not ready. At this point, we schedule all ready tasks with the current mobility of 0 as postponing their scheduling would increase the overall latency. After all critical tasks have been scheduled, we check if any resources that have already been allocated in the design are free at the current control step. If any of the resources are available, we can also schedule some of the other ready but still mobile tasks that can be satisfied with the available resources without needing additional ones. This procedure is performed for each control step until the last sink task is scheduled.

### 5.2.2. Initial and Neighbor Solution Generation with Resource Allocation

To generate the initial solution used as the starting state in the SA process of finding the optimal solution, we randomly assign resources to each application task based on their types. For example, a task that performs an addition operation is assigned a random adder among all adders in the resource library. The supply voltage level for each task (the voltage island it will be placed on) is also assigned randomly. We also make sure that the initial solution is an acceptable state, i.e., that it meets the given area and latency constraints. Still, we disregard

**Algorithm 4** Latency-Constrained List Scheduling Algorithm For Minimum Area

**Inputs:** $s$ current solution, $t_{ord}$ topological ordering of the tasks, $l_{ASAP}$ ASAP scheduling of $s$, $l_{ALAP}$ ALAP scheduling of $s$, $n$ number of the tasks.

**Output:** $l_{LIST}$ List scheduling of $s$.

```
procedure GETLISTSCHEDULE($s, t_{ord}, l_{ASAP}, l_{ALAP}$)
    if $l_{ALAP}[0] < 0$ then                                  ▷ Checking if ALAP scheduling failed.
        return -1
    end if
    $l_{LIST}[0] \leftarrow 0$                                  ▷ Schedule the start task in control step 0.
    $scheduled[0] \leftarrow True$
    $ready[0] \leftarrow True$
    for i = 1..(n-1) do
        $scheduled[i] \leftarrow False$
        $ready[i] \leftarrow False$
    end for
    $cStep \leftarrow 1$
    while $scheduled[n-1] == False$ do
        for i = 1..(n-1) do
            $mobility[i] \leftarrow l_{ALAP}[i] - cStep$            ▷ Computing tasks' remaining mobility.
            $ready[i] \leftarrow True$                              ▷ Assume it is ready to be scheduled.
            if $!scheduled[k]$ s.t. $s[k]$ precedes $s[i]$ in $t_{ord}$ then
                $ready[i] \leftarrow False$                 ▷ Revert if an unscheduled precedent task is found.
            end if
            if $scheduled[k]$ and $(l_{LIST}[k] + s[k].latency - 1) > cStep$ s.t. $s[k]$ precedes $s[i]$ in $t_{ord}$ then
                $ready[i] \leftarrow False$                 ▷ Revert if an unfinished precedent task is found.
            end if
        end for
        for i = 1..(n-1) do
            if ! $scheduled[i]$ and $ready[i]$ and $mobility[i] == 0$ then
                $l_{LIST}[i] \leftarrow cStep$                 ▷ Schedule tasks with 0 mobility immediately.
                $scheduled[i] \leftarrow True$
            end if
        end for
        $R_{free} \leftarrow$ resources already used in the design but available at this $cStep$
        for i = 1..(n-1) do
            if ! $scheduled[i]$ and $ready[i]$ and $s[i].type == r.type$ s.t. $(r \in R_{free})$ then
                $l_{LIST}[i] \leftarrow cStep$                          ▷ Schedule another ready task that
                $scheduled[i] \leftarrow True$                           ▷ does not need extra resources.
                $R_{free} = R_{free}\ r$                                        ▷ Update $R_{free}$.
            end if
        end for
        $cStep \leftarrow cStep + 1$                                 ▷ Move to the next control step.
    end while
    return $l_{LIST}$
end procedure
```

any considerations about its optimality at this stage. The pseudocode for this procedure is given in Algorithm 5.

After the random assignment of resources and voltage islands, we perform the scheduling to

**Algorithm 5** Generating a Random Solution

**Inputs:** $DAG$ design specification, $\mathcal{R}$ resource library, $V$ number of voltage islands.
**Output:** $s$ a solution.

```
procedure GENERATERANDOMSOLUTION($DAG, \mathcal{R}, V, \Lambda, \lambda$)
    for each $task \in DAG$ do
        $s[task].assignedResource \leftarrow getRandomResource(\mathcal{R}, task.type)$
        $s[task].assignedVoltage \leftarrow getRandomVoltageLevel(V)$
    end for
    return $s$
end procedure
```

obtain the minimum latency and area of the generated design solution because it is necessary to check if the solution meets the given constraints. The ASAP schedule provides us with the minimum latency necessary for the obtained allocation, while the List scheduling step ensures that the final design is scheduled optimally with respect to the overall area based on the allocated resources. It is discarded if the initial random allocation fails to meet the latency constraint. However, suppose the area constraint is violated, but the latency of the design permits further relaxation. In that case, there may still be a way to adjust the obtained solution to meet both constraints.

This scenario may occur when area constraints are tight, and the generated random solutions cannot satisfy them after a predetermined number of iterations. In such a scenario, first, a heuristic area reducing algorithm via schedule modification is applied to try and fit the area of the generated random solution within the allowed limit if the design latency allows it. If this attempt also fails, which can happen for the edge cases in which we are trying to find a solution with an area close to the minimum necessary area for a specific design, a different initial solution generation approach is employed. We name this approach area and latency-aware random solution generation.

The pseudocode for our heuristic area reducing algorithm is given in Algorithm 6, and it works as follows. First, each control step is examined to find the step in which the scheduled tasks contribute to the maximum area coverage of the design. The idea is to shift the start time of one of the tasks to reduce the number of resources necessary at that particular control step. When such a control step is identified, the tasks scheduled in it are examined to find

**Algorithm 6** Area Reduction Through Schedule Modification Algorithm

**Inputs:** $s$ current solution, $DAG$ design specification, $t_{ord}$ topological ordering of the tasks, $l_{ASAP}$ ASAP scheduling of $s$, $l_{ALAP}$ ALAP scheduling of $s$, $l_{LIST}$ List scheduling of $s$.
**Output:** $s$ modified current solution with smaller area.

```
procedure REDUCEAREA($s, DAG, t_{ord}, l_{ASAP}, l_{ALAP}, l_{LIST}$)
    $cSteps \leftarrow s[|DAG|-1].startTime$
    $maxArea \leftarrow 0$
    $cStepMax \leftarrow -1$
    for $i = 1...cSteps$ do
        $areaAtcStep \leftarrow 0$
        for $j \in 0...|DAG|-1$ do
            if $s[j].startTime == i$ then
                $areaAtcStep \leftarrow areaAtcStep + s[j].assignedResource \rightarrow area$
            end if
        end for
        if $areaAtcStep > maxArea$ then
            $maxArea \leftarrow areaAtcStep$
            $cStepMax \leftarrow i$                ▷ Note the cStep that affects the are the most.
        end if
    end for
    $taskToMove \leftarrow -1$
    $maxSlack \leftarrow -1$
    for $j \in 0...|DAG|-1$ do
        if $s[j].startTime == cStepMax$ then
            if $(l_{ALAP}[j] - l_{ASAP}[j]) > maxSlack$ then
                $taskToMove \leftarrow j$    ▷ Find the task in the cStep with maximum mobility.
                $maxSlack \leftarrow l_{ALAP}[j] - l_{ASAP}[j]$
            end if
        end if
    end for
    $s \leftarrow recursivelyShiftTasks(taskToMove, s, t_{ord}, l_{LIST}, DAG)$
    return $s$
end procedure
```

the task with the highest remaining mobility, hoping that shifting such a task may help us avoid the negative effect on the overall latency. If no such task exists, one of the critical tasks will be rescheduled; however, it does not pose an issue as this algorithm is executed only if the latency constraint allows for such schedule relaxation. We must assure that the latency constraint allows rescheduling, because finding a mobile task does not guarantee that its successors will not have to be rescheduled.

Once the task to be rescheduled is identified, a recursive function is called to recursively move the chosen task's start time and its successors. We present the pseudocode for this function in Algorithm 7. The start time of the task to reschedule is moved up by one control step, followed by recursively rescheduling all of its successors (other tasks dependent on its completion) if their start times are now conflicting with its new shifted end time.

**Algorithm 7** Recursive Rescheduling of Dependent Tasks

**Inputs:** $taskID$ ID of the task to shift, $s$ current solution, $t_{ord}$ topological ordering of the tasks, $l_{LIST}$ List scheduling of $s$, $DAG$ design specification.
**Output:** $s$ current solution with modified schedule.

```
procedure RECURSIVELYSHIFTTASKS($taskID, s, t_{ord}, l_{LIST}, DAG$)
    if Task $taskID$ has not been rescheduled already then
        $s[taskID].startTime++$
        $l_{LIST}[taskID]++$
        Mark Task $taskID$ as rescheduled.
        for $i \in 0...|DAG|-1$ do
            $endTime \leftarrow s[taskID].startTime + s[taskID].delay$
            if $DAG[taskID][i] == 1$ and $endTime > s[i].startTime$ then
                $s \leftarrow recursivelyShiftTasks(i, s, t_{ord}, l_{LIST}, DAG)$
            end if
        end for
    end if
    return $s$
end procedure
```

The area reducing procedure is called until the desired area is achieved and as long as the latency constraint allows it. If the area that satisfies the constraint is achieved before the latency constraint is violated, the solution is accepted for the further annealing process. Otherwise, the generated solution is marked as unfeasible and dismissed.

Failing to generate a viable starting solution using completely random generation in a predetermined number of iterations will trigger the second approach of initial solution generation. This approach also has random elements, even though it uses a heuristic that will lead to the generation of more area and latency-aware solutions compared to the completely random method. We present the pseudocode for the area and latency-aware random solution generation in Algorithm 8.

**Algorithm 8** Generating an Area and Latency-Aware Random Solution

```
Inputs: $DAG$ design specification, $\mathcal{R}$ resource library, $V$ number of voltage islands.
Output: $s$ a solution.
  procedure GENERATEALAWARERANDOMSOLUTION($DAG, \mathcal{R}, V, \Lambda, \lambda$)
      for each $task \in DAG$ do
          $coinFlip \leftarrow randInt(0, 1)$
          if $coinFlip == 0$ then                ▷ Assign the resource with the minimum area.
              $task.assignedResource \leftarrow getMinAreaResource(\mathcal{R}, task.type)$
          else                                   ▷ Assign the resource with the minimum latency.
              $task.assignedResource \leftarrow getMinLatencyResource(\mathcal{R}, task.type)$
          end if
          $task.assignedVoltage \leftarrow HIGH$
      end for
      return $s$
  end procedure
```

Area and latency-aware random solution generation is performed such that each task is randomly assigned either the fastest resource of its type or the resource with the minimum area. They are all assigned to the same HIGH voltage island to ensure the fast operation of the initial solution. This predetermined VI allocation is not a problem for finding different allocations during the annealing process since the generation of a neighbor solution may change the assigned voltage island.

Generation of a neighbor state (a candidate solution) is also performed randomly. In systems with many variables, any change in one of the design variables can be considered as a neighboring state. Hence, we randomly pick a task to be changed and the neighbor selection criteria, as we can change either the assigned resource or the supply voltage island. In Algorithm 9, we present the pseudocode for this procedure.

The scheduling algorithms will again be executed to obtain the minimum latency and area of the candidate design because it is necessary to ensure that also the candidate solution meets the given latency and area constraints. Otherwise, it cannot be considered a viable candidate. The same area reduction approach mentioned above is used if necessary and possible to try and fit the candidate solution within the given constraints before it is discarded as not viable.

**Algorithm 9** Generating a Candidate Neighbor Solution

**Inputs:** $s$ current solution, $DAG$ design specification, $\mathcal{R}$ resource library, $V$ number of voltage islands.

**Output:** $s^*$ a candidate neighbor solution.

```
procedure GETRANDOMNEIGHBOR($s, DAG, \mathcal{R}, V, \Lambda, \lambda$)
    $s^* \leftarrow s$
    $idx \leftarrow getRandInt(1, |DAG|)$                    ▷ Randomly pick a task to change.
    $whatToChange \leftarrow getRandInt(1, 2)$               ▷ 1 for resource, 2 for voltage.
    if $whatToChange == 1$ then
        $s^*[task_{idx}].assignedResource \leftarrow getRandomResource(\mathcal{R}, s^*[task_{idx}].type)$
    else if $whatToChange == 2$ then
        $s^*[task_{idx}].assignedVoltage \leftarrow getRandomVoltageLevel(V)$
    end if
    return $s^*$
end procedure
```

### 5.2.3. Computing the Initial Temperature

The initial temperature should be selected such that it results in a desired starting acceptance probability. At the beginning of the simulated annealing process, it is often desirable to choose a relatively high starting acceptance probability, which ensures that some worse solutions will be accepted, in order to achieve more random exploration of the search space and guarantee that the algorithm will not get stuck in local optima. In this study, we have chosen the starting acceptance probability of 0.8.

We adopt an iterative method of calculating the corresponding initial temperature proposed in [26]. In the following formulations, a state simply refers to a solution to an optimization problem, whereas energy refers to the cost of its objective function. Stationary distribution $\pi_i$ is formulated in Equation (40), under the assumption that the generation probability of a state $j$ is $1/|N(i)|$ if $j \in N(i)$, and 0 otherwise.

$$\pi_i = \frac{|N(i)|\text{exp}(-E_i/T)}{\sum_j |N(j)|\text{exp}(-E_j/T)} \tag{40}$$

The cost difference between two states $\delta_t$ is formulated in Equation (41). The probability to generate a transition $t$ when the energy states are distributed in conformity with $\pi_i$, namely $\mathcal{P}_t$, is formulated in Equation (42).

$$\delta_t = E_{\max_t} - E_{\min_t} \tag{41}$$

$$\mathcal{P}_t = \pi_{\min_t} \frac{1}{|N(\min_t)|} \tag{42}$$

Probability of accepting a positive transition $p_t$ is formulated in Equation (43).

$$p_t = \exp(-\delta_t/T) \tag{43}$$

Then, the acceptance probability $\chi(T)$ can be estimated based on a random set $S$ of strictly positive transitions as $\hat{\chi}(T)$, formulated in Equations (44) and (45). To generate a random set of positive transitions $S$, it is enough to randomly generate some states (solutions) and their neighbors (one neighbor for each state), and save the costs (energies) of their corresponding objective functions as $E_{\max_t}$ and $E_{\min_t}$.

$$\begin{aligned} \hat{\chi}(T) &= \frac{\sum_{t \in S} \mathcal{P}_t \cdot p_t}{\sum_{t \in S} \mathcal{P}_t} && (44) \\ &= \frac{\sum_{t \in S} \exp(-E_{\max_t}/T)}{\sum_{t \in S} \exp(-E_{\min_t}/T)}. && (45) \end{aligned}$$

The goal is to find an initial temperature $T_0$ such that the acceptance probability will match the desired acceptance probability; i.e., $\chi(T_0) = \chi_0$ ($\chi_0 \in [0, 1]$). $T_0$ can be computed recursively using Formula (46), where $p$ is a real number $\geq 1$. The stopping criteria for the recursive call is when $\hat{\chi}(T_n)$ reaches the value of the desired acceptance probability $\chi_0$. The obtained $T_n$ can be taken as an adequate approximation of the desired initial temperature $T_0$.

$$T_{n+1} = T_n \frac{\ln(\hat{\chi}(T_n))}{\ln(\chi_0)}^{1/p} \tag{46}$$

We use experimental analysis to take $p = 1$ in the SA-based HLS method proposed in this study. A random set of positive transitions $S$ is generated for each application benchmark prior to the calculation of the initial temperature that satisfies the desired starting acceptance probability.

We present the pseudocodes of our calculations of the initial temperature in Algorithms (10) and (11).

**Algorithm 10** Calculating the Acceptance Probability

```
Inputs: $T$ temperature, $S$ random transition set.
Output: $p_{accept}$ acceptance probability.
  procedure GETACCEPTANCEPROBABILITY($T, S$)
      $divident \leftarrow 0.0$
      $divisor \leftarrow 0.0$
      for $i = 1...|S|$ do
          $divident \leftarrow divident + \exp\left(-\frac{max_{S[i]}}{T}\right)$
          $divisor \leftarrow divisor + \exp\left(-\frac{min_{S[i]}}{T}\right)$
      end for
      $p_{accept} \leftarrow \frac{divident}{divisor}$
      return $p_{accept}$
  end procedure
```

### 5.2.4. Annealing Schedule

Several different annealing (cooling) schedules have been proposed in the literature [19, 21, 103–105]. The convergence of an SA-based algorithm towards the optimal solution is significantly affected by choice of the annealing schedule. Statistical analysis is important if we wish to adopt the optimal cooling strategy for the problem at hand [103].

The cooling schedule adopted in this study is a frequently used cooling approach known as the geometric schedule that originates from [19]. It is formulated in Equation (47), where $\alpha_c$ is usually chosen to be a positive constant smaller than but close to 1 ($0.8 \geq \alpha_c \leq 0.99$).

**Algorithm 11** Calculating the Initial Temperature

**Inputs:** $T$ temperature, $S$ random transition set, $\chi_0$ desired starting acceptance probability, $\epsilon_{rr}$ acceptance probability error.
**Output:** $T_0$ initial temperature.

```
procedure GETINITIALTEMPERATURE($S, \chi_0, \epsilon_{rr}$)
    $P_{currentEstimate} \leftarrow getAcceptanceProbability(T, S)$
    $p \leftarrow 1.0$                                              ▷ $p$ real number $\geq 1$.
    $T_{new} \leftarrow T \times \left(\frac{\ln(P_{currentEstimate})}{\ln(\chi_0)}\right)^{\frac{1}{p}}$
    if $(P_{currentEstimate} - \chi_0) < \epsilon_{rr}$ then
        $T_0 \leftarrow T_{new}$
    else
        $T_0 \leftarrow getInitialTemperature(T_{new}, S, \chi_0, \epsilon_{rr})$
    end if
    return $T_0$
end procedure
```

$$T_{n+1} = T_n \times \alpha_c \quad (n = 0, 1, ...) \tag{47}$$

For calculating the acceptance probability to decide if a worse candidate solution will be accepted anyway, we adopt the approach presented in [106]. The Metropolis acceptance probability $P_{metropolis}$ is defined as formulated in Equation (48), where $i$ is the current and $j$ is the candidate neighbor state (solution).

$$P^{i,j}_{metropolis} = \begin{cases} \exp\left(-\frac{E_j - E_i}{T}\right) & \text{if } E_j > E_i \\ 1 & \text{otherwise} \end{cases} \tag{48}$$

To choose the most suitable cooling constant $\alpha_c$ for our problem, we tried different values of 0.8, 0.95, and 0.99 to analyze the evolution of the probability of accepting a worse candidate solution as the temperature cools and the annealing process reaches its end. Based on the obtained results, we adopted $\alpha_c = 0.95$ as it showed the most desired behavior for our problem. The value of 0.99 resulted in too slow cooling and high acceptance probability even for very low temperatures, while the value of 0.8 resulted in a longer convergence time.

To ensure the temperature never reaches the exact value of 0, which would lead to a division by 0 when calculating the Metropolis acceptance probability, we use the following cooling schedule as presented in Algorithm 12.

**Algorithm 12** Temperature Cooling

```
Inputs: $T$ temperature, $\alpha_c$ cooling constant.
Output: $T$ new temperature.
  procedure GETNEWTEMPERATURE($T, \alpha_c$)
      if $T \leq 0.00001$ then
          $T \leftarrow 0.00001$
      else
          $T \leftarrow T \times \alpha_c$
      end if
      return $T$
  end procedure
```

### 5.2.5. Additional Considerations for SA-Based HLS With Partial DMR

In our SA-based HLS method that employs partial duplication of resources for improved reliability of the final designs, some considerations necessitated a few modifications to the algorithms described in previous sections. For the brevity of this section, we only discuss these modifications without presenting the same algorithms in their entirety for the second time.

At the initial solution generation stage, it is necessary to allow each task to be assigned either one or two resources. We use a duplication percentage constant $\varphi_{dup}$ to tune the amount of duplication introduced at this stage. If we set $\varphi_{dup} = 0.5$, approximately half of the tasks will be duplicated. The pseudocode for this approach is given in Algorithm 13.

For tight area constraints, starting with less duplication is better to more easily find a starting solution. The same approach is employed for the area and latency-aware random solution generation in case random generation fails to produce a viable initial solution. It is important to note that this approach does not limit the amount of duplication in the final design, as it can change during the SA process of finding better solutions through an adequate neighbor generation.

**Algorithm 13** Generating a Random Solution With Partial DMR

**Inputs:** $DAG$ design specification, $\mathcal{R}$ resource library, $V$ number of voltage islands, $\varphi_{dup}$ duplication percentage constant.

**Output:** $s$ a solution.

```
procedure GENERATERANDOMPDSOLUTION($DAG, \mathcal{R}, V, \Lambda, \lambda, \varphi_{dup}$)
    for each $task \in DAG$ do
        $s[task].assignedResource \leftarrow getRandomResource(\mathcal{R}, task.type)$
        $s[task].assignedVoltage \leftarrow getRandomVoltageLevel(V)$
        if $getRandDouble(0.0, 1.0) < \varphi_{dup}$ then
            $s[task].assignedDuplicateResource \leftarrow getRandomResource(\mathcal{R}, task.type)$
            $s[task].assignedDuplicateVoltage \leftarrow getRandomVoltageLevel(V)$
        else
            $s[task].assignedDuplicateResource \leftarrow$ NULL
        end if
    end for
    return $s$
end procedure
```

In addition to selecting which task to change and if either the assigned resource or voltage island should be changed to obtain a neighbor, we also have some additional options for a candidate solution selection in the partial duplication method. When a task to be changed is selected, we can choose between changing the main allocated resource or voltage island of the task or changing the duplicated ones. Furthermore, if the duplicate is to be changed, but the task has not been assigned a duplicate resource, we can opt to assign a duplicate resource and obtain a neighbor in that way. Similarly, we could choose to remove the duplicate resource. This approach ensures that the amount of duplication at any stage of the annealing process can change if it results in getting closer to the optimal solution.

As far as the scheduling algorithms are concerned, duplicated tasks must also be considered since any dependent task must wait for all of its predecessors, including their duplicates, to finish before it can be scheduled. Hence, when looking for the latest end time among all predecessor tasks, the start times and latency of the duplicated predecessors must also be inspected. Additionally, in List scheduling, duplicated tasks should also be considered when trying to optimize the area.

Similar considerations are taken in our area reduction through the schedule modification algorithm. When looking for the control step, the highest area concentration caused by the tasks scheduled in that control step, we also check the contribution of the duplicated tasks, if any. Similarly, when employing recursive rescheduling of dependent tasks, we inspect the duplicated successors when checking if the start time conflicts with the new end time of the rescheduled task before deciding if they should be rescheduled as well or not.

The initial temperature calculations are performed using the same approach described in Section 5.2.3.. Finally, we use the same annealing schedule with a difference of experimentally selecting the cooling constant $\alpha_c = 0.99$ to achieve a slightly slower cooling rate shown in Figure 5.3.

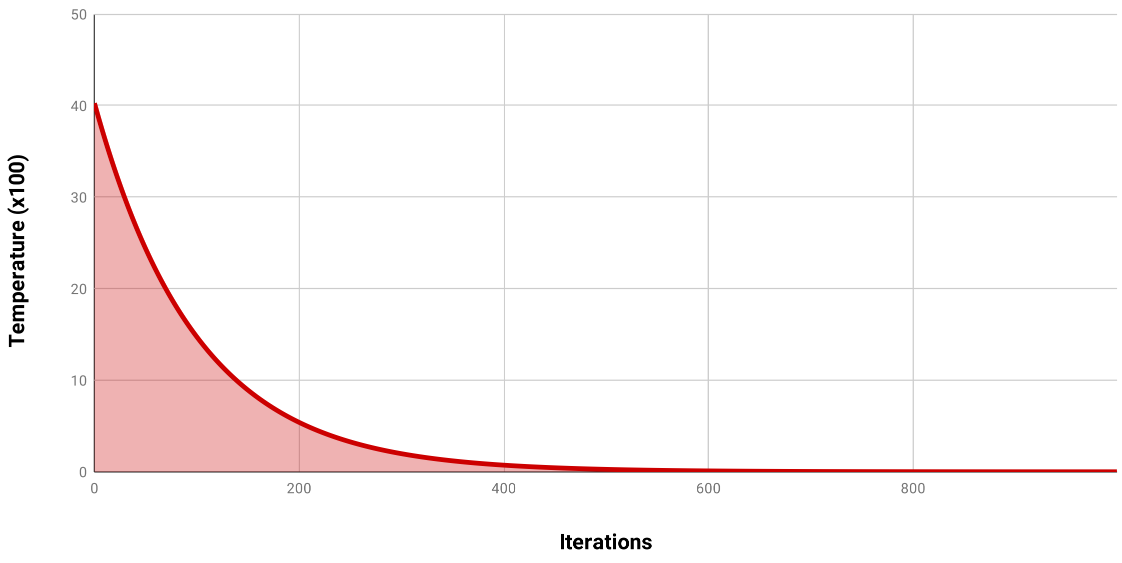


Figure 5.3 Temperature cooling rate for $\alpha_c = 0.99$.

### 5.2.6. SA-Based HLS Algorithm

We present the pseudocode for the proposed SA-based reliability and energy-oriented HLS method in Algorithm 14. Before starting the simulated annealing optimization process, we first generate a random initial solution and ensure that it meets the given area and latency constraints. We do not make any assumptions regarding its optimality at this point. Given different constraints, it may be easier or harder to randomly generate a viable solution;

**Algorithm 14** SA-Based HLS Method Pseudocode

**Inputs:** $DAG$ design specification, $\mathcal{R}$ resource library, $S$ random transition set, $\chi_0$ desired starting acceptance probability, $\Lambda$ area constraint, $\lambda$ latency constraint, $V$ number of voltage islands, $\epsilon_{rr}$ acceptance probability error, $nIter$ number of SA iterations, $\alpha_c$ cooling constant, $k$ maximum number of allowed iterations for initial solution generation.
**Output:** $s$ the final solution.

```
1: $s \leftarrow$ NULL                                                    ▷ To return NULL if solution is infeasible.
2: $count \leftarrow 0$
3: while $(getArea(s, \mathcal{R}, V) > \Lambda$ or $getLatency(s) > \lambda)$ and $count < k$ do
4:     $s \leftarrow generateRandomSolution(DAG, \mathcal{R}, V, \Lambda, \lambda)$
5:     $t_{ord} \leftarrow getTopologicalOrdering(DAG)$
6:     $l_{ASAP} \leftarrow getASAPSchedule(s, t_{ord}, |DAG|)$            ▷ Get the shortest possible latency.
7:     $l_{ALAP} \leftarrow getALAPSchedule(s, t_{ord}, l_{ASAP}, |DAG|)$   ▷ Needed for the task mobility.
8:     $l_{LIST} \leftarrow getListSchedule(s, t_{ord}, l_{ASAP}, l_{ALAP}, |DAG|)$   ▷ Reduce the area.
9:     if $getArea(s, \mathcal{R}, V) > \Lambda$ or $getLatency(s) < \lambda$ then
10:        while $getArea(s, \mathcal{R}, V) > \Lambda$ and $getLatency(s) < \lambda$ do
11:            $s \leftarrow reduceArea(s, DAG, t_{ord}, l_{ASAP}, l_{ALAP}, l_{LIST})$   ▷ Reduce the area further.
12:        end while
13:    end if
14: end while
15: if $count \geq k$ then                                     ▷ The completely random generation failed.
16:    $count \leftarrow 0$
17:    while $(getArea(s) > \Lambda$ or $getLatency(s) > \lambda)$ and $count < k$ do
18:        $s \leftarrow generateALAwareRandomSolution(DAG, \mathcal{R}, V, \Lambda, \lambda)$
19:        $l_{ASAP} \leftarrow getASAPSchedule(s, t_{ord}, |DAG|)$            ▷ Get the shortest possible latency.
20:        $l_{ALAP} \leftarrow getALAPSchedule(s, t_{ord}, l_{ASAP}, |DAG|)$   ▷ Needed for the task mobility.
21:        $l_{LIST} \leftarrow getListSchedule(s, t_{ord}, l_{ASAP}, l_{ALAP}, |DAG|)$   ▷ Reduce the area.
22:        if $getArea(s, \mathcal{R}, V) > \Lambda$ or $getLatency(s) < \lambda$ then
23:            while $getArea(s) > \Lambda$ and $getLatency(s) < \lambda$ do
24:                $s \leftarrow reduceArea(s, DAG, t_{ord}, l_{ASAP}, l_{ALAP}, l_{LIST})$   ▷ Reduce the area further.
25:            end while
26:        end if
27:    end while
28: end if
29: if $count < k$ then                          ▷ A viable initial solution was successfully generated.
30:    $T_{start} \leftarrow 100.0$                                 ▷ Some arbitrarily high start temperature.
31:    $T \leftarrow getInitialTemperature(T_{start}, S, \chi_0, \epsilon_{rr})$
32:    for $i = 1...nIter$ do
33:        $s^* \leftarrow getRandomNeighbor(s, DAG, \mathcal{R}, \Lambda, \lambda)$
34:        $l^*_{ASAP} \leftarrow getASAPSchedule(s^*, t_{ord}, |DAG|)$          ▷ Get the shortest possible latency.
35:        $l^*_{ALAP} \leftarrow getALAPSchedule(s^*, t_{ord}, l^*_{ASAP}, |DAG|)$   ▷ Needed for the task mobility.
36:        $l^*_{LIST} \leftarrow getListSchedule(s^*, t_{ord}, l^*_{ASAP}, l^*_{ALAP}, |DAG|)$   ▷ Reduce the area.
37:        if $getArea(s^*, \mathcal{R}, V) > \Lambda$ or $getLatency(s^*) < \lambda$ then
38:            while $getArea(s^*, \mathcal{R}, V) > \Lambda$ and $getLatency(s^*) < \lambda$ do
39:                $s^* \leftarrow reduceArea(s^*, DAG, t_{ord}, l^*_{ASAP}, l^*_{ALAP}, l^*_{LIST})$   ▷ Reduce the area further.
40:            end while
41:        end if
42:        if $getArea(s^*, \mathcal{R}, V) \leq \Lambda$ or $getLatency(s^*) \leq \lambda$ then   ▷ If candidate is viable.
43:            $E_{current} \leftarrow getObjectiveFunctionCost(s)$
44:            $E_{new} \leftarrow getObjectiveFunctionCost(s^*)$
45:            if $E_(new) > E_(current)$ then
46:                $s \leftarrow s^*$                               ▷ Accept the better candidate solution.
47:            else if $getMetropolisAcceptanceProbability\ (E_{current}, E_{new}, T) \geq rand(0, 1)$ then
48:                $s \leftarrow s^*$                        ▷ Accept the worse candidate solution anyway.
49:            end if
50:        end if
51:        $T \leftarrow getNewTemperature(T, \alpha_c)$                      ▷ Apply the cooling schedule.
52:    end for
53: end if
54: return $s$
```

hence, we assign an arbitrary number of allowed generation trials before calling the solution infeasible. In our experimental setting, we allowed for 1000 iterations.

Once a viable starting solution is found, the SA process may start looking for better candidates. First, the initial temperature is calculated based on the desired acceptance probability and the set of random positive transitions. The SA process is repeated the predetermined number of iterations. In each iteration, a candidate neighbor solution is generated, and if it meets the given constraints, it is considered for acceptance. Better candidates are always accepted, whereas worse candidates are accepted based on the Metropolis probability affected by the current system temperature. As the temperature cools, the probability of accepting a worse candidate decreases.

The overall latency of a design is easily obtained by looking at the start time of the last `sink` task. On the other hand, to calculate the area of a design, we use the following approach presented in Algorithm 15. Since we assume a pipelined datapath, we consider each control step for the tasks scheduled in it and look for the maximum number of used resources at each control step per each voltage island (as it has been formulated in Equation (31) in Section 5.1.). The maximum sum of the area of all resources scheduled at any control step under all supply voltage levels gives us the overall design area.

We use the same objective function given in (38) that optimizes both reliability and energy according to the parameter $\alpha$ used to assign the desired priority to either objective. The method `getObjectiveFunctionCost(<solution>)` returns the weighted sum of reliability and energy values, and is used to search for Pareto optimal solutions for the bi-objective problem at hand.

**Algorithm 15** Area Calculation Algorithm

**Inputs:** $s$ current solution, $\mathcal{R}$ resource library, $V$ number of voltage islands.

**Output:** $maxArea$ area of the solution $s$.

```
procedure GETAREA($s, \mathcal{R}, V$)
    $cSteps \leftarrow s[|s|-1].startTime$
    $resourceCountAtCStepPerVdd[|\mathcal{R}|][V][cSteps] \leftarrow 0$
    for $i = 1...cSteps$ do
        for $j = 1...|\mathcal{R}|$ do
            if $s[j].startTime == i$ then
                $resourceCountAtCStepPerVdd[j][s[j].assignedVoltage][i] + +$
            end if
        end for
    end for
    $maxArea \leftarrow 0$
    $maxCountOfResourcesAtVdd[|\mathcal{R}|][V] \leftarrow 0$
    for $i = 1...|\mathcal{R}|$ do
        for $j = 1...V$ do
            $maxCountOfResourceiAtVdd[j] \leftarrow 0$
            for $k = 1...cSteps$ do
                if $maxCountOfResourceiAtVdd[k] < resourceCountAtCStepPerVdd[i][j][k]$ then
                    $maxCountOfResourceiAtVdd[k] \leftarrow resourceCountAtCStepPerVdd[i][j][k]$
                end if
            end for
            $maxCountOfResourcesAtVdd[i][j] \leftarrow maxCountOfResourceiAtVdd[j]$
        end for
    end for
    for $i = 1...|\mathcal{R}|$ do
        for $j = 1...V$ do
            $maxArea = maxArea + maxCountOfResourcesAtVdd[i][j] \times \mathcal{R}[i].area$
        end for
    end for
    return $s$
end procedure
```

# 6. EXPERIMENTAL RESULTS AND DISCUSSION

We tested the performance of our proposed methods through several exhaustive experiments. We implemented our ILP models in Mosel modeling language and conducted the experiments with FICO Xpress optimizer [107]. SA-based HLS method was implemented in the C programming language. Mosel and C codes are given in APPENDIX A. We used the four most commonly used benchmarks in literature: Differential Equation Solver (DES), Finite Impulse Response Filter (FIR), Auto-Regressive (AR) filter, and Elliptic Wave Filter (EWF). The summary of each benchmark is presented in Table 6.1. Interested readers may find the dataflow graphs with more detailed specifications of the used benchmarks in APPENDIX B.

Table 6.1 Benchmark specifications: the number of nodes, edges and, types of operations in their respective dataflow graphs.

| Benchmark | Nodes | Edges | Addition Operations | Multiplication Operations |
|---|---|---|---|---|
| DES | 11 | 8 | 5 | 6 |
| FIR | 23 | 22 | 15 | 8 |
| EWF | 26 | 40 | 26 | 0 |
| AR | 28 | 30 | 12 | 16 |

The resource library used in the experiments is given in Table 4.1, where each resource's area, latency, reliability, and energy consumption values are listed under low, medium, and high voltage levels. The latency is measured in control time steps (e.g., clock cycles), the area is measured in units, and the energy consumption is measured in nanojoules (nJ).

To set the basis for our experiments, we first obtained the minimum area and latency values that can be given as the constraints to ensure that at least one feasible solution exists for the given constraints. To obtain the minimum area constraints for each benchmark, we assigned the resources with the smallest area to each task. Later, these constraints were gradually relaxed to allow the design solutions with faster and more reliable resources with a bigger area. On the other hand, we used As Soon As Possible (ASAP) scheduling algorithm to obtain the minimum performance constraints. This scheduling algorithm assigns the earliest possible start time for each task based on the task dependencies; thus, it provides the upper bound for the latency of an application [6]. Latency constraints are also gradually increased throughout the experiments to allow for the allocation of different and slower but more reliable resources, as it can improve the overall design reliability. Furthermore, it will enable slower execution of the same resources when operating under lower supply voltages, which can result in more energy-efficient designs.

The experimental setup includes comprehensive tests for the varying area and performance constraints, varying numbers of supply voltages, and different $\alpha$ values through which we can assign weighted priority to either of the objective functions if desired. Our formulation of the bi-objective function as the weighted sum of the normalized reliability and energy values given in Equation (38) allows us to assign different weights (priorities) to either optimization goal as desired by changing the parameter $\alpha$.

In our exhaustive experiments performed over all the benchmarks with the varying latency and area constraints, we conduct tests with the $\alpha$ values of 1.0 to prioritize reliability optimization above everything else, 0.5 to look for the Pareto-optimal solutions where both objectives are equally considered, and 0.0 to optimize energy consumption only, without any reliability considerations. More fine-grained weight changes were also performed on a single benchmark to show how fine-tuning the objective function affects the reliability and energy of the final designs, especially in the presence of multi-supply voltages (see Subsection 6.4.).

The experiments were performed on a desktop computer with the following configuration: Intel Core(TM)2 Duo CPU E8500, at 3.16 GHz, with two cores, two logical processors, and a total physical memory of 6.00 GB.

## 6.1. Comparison of ILP and GA-Based Methods With DMR

In our first set of experiments, we compare our partial and full DMR ILP models to the GA-based selective DMR method proposed in [96]. We use the following abbreviations to denote the methods that are being compared:

- ILP-PD: Partial DMR ILP model,

- ILP-PD-C: Adaptive partial DMR ILP model with either the energy constraint for $\alpha = 1$ or reliability constraint for $\alpha = 0$ from GA-based partial duplication method results for the same experiment configuration (same area and latency constraints),
- ILP-FD: Full DMR ILP model,
- GA-SD: GA-based selective duplication from [96].

For the purpose of a fair comparison, the set of experiments in this subsection was carried out over two voltage supply levels: high and low because we compare the results with the results of the GA-SD method that were obtained under only two voltage levels.

In the first column of the result tables given in the following subsections, the value of the parameter $\alpha$ used for that specific set of experiments is given. The second column defines the values of the latency (L) and area (A) constraints used in those particular test cases. The subsequent columns denoted by the method abbreviations present the resulting reliability or energy consumption values for each test case, respectively. Finally, the columns denoted by delta ($\Delta$) indicate the percentage change of the ILP results relative to the GA results. We are interested in percentage change increase for the reliability values since higher fault tolerance indicates a more reliable solution. Similarly, a decrease in percentage change for the energy consumption values is desirable, as lower energy expenditure indicates a better solution. For some experiment configurations, fully duplicated designs are not feasible and are marked with '-'.

### 6.1.1. Reliability Optimization Results Discussion

In the first set of experiments, we assign the parameter $\alpha = 1.0$ to focus only on reliability optimization, disregarding energy costs completely. Comparison of the reliability results of ILP and GA duplication methods for all benchmarks are given in Table 6.2.

From the given table that presents reliability results for $\alpha = 1.0$ where the objective is to maximize the reliability without any consideration about the energy consumption, it is

Table 6.2 Comparison of the reliability results of ILP and GA duplication methods for all benchmarks when $\alpha = 1.0$.

| (L, A) | ILP-PD | ILP-PD-C | ILP-FD | GA-SD | ILP-PD/GA-SD | ILP-PD-C/GA-SD | ILP-FD/GA-SD |
|---|---|---|---|---|---|---|---|
| | | **DES Reliability Results** | | | | $\Delta$ (%) | |
| (28,20) | 0.993795 | 0.993795 | 0.999871 | 0.982875 | 1.11 | 1.11 | 1.73 |
| (28,30) | 0.999821 | 0.993840 | 0.999821 | 0.982875 | 1.72 | 1.12 | 1.72 |
| (28,40) | 0.999821 | 0.995805 | 0.999821 | 0.982875 | 1.72 | 1.32 | 1.72 |
| (25,20) | 0.993640 | 0.993640 | - | 0.971069 | 2.32 | 2.32 | - |
| (25,30) | 0.999653 | 0.994655 | 0.999653 | 0.971069 | 2.94 | 2.43 | 2.94 |
| (25,40) | 0.999653 | 0.994667 | 0.999653 | 0.971069 | 2.94 | 2.43 | 2.94 |
| | | | | **Average $\Delta$ (%):** | **1.60** | **1.34** | **1.58** |
| | | **EWF Reliability Results** | | | | $\Delta$ (%) | |
| (30,10) | 0.995615 | 0.885586 | 0.995615 | 0.826462 | 20.47 | 7.15 | 20.47 |
| (30,20) | 0.997625 | 0.989771 | 0.997625 | 0.826462 | 20.71 | 19.76 | 20.71 |
| (30,30) | 0.997625 | 0.989771 | 0.997625 | 0.826462 | 20.71 | 19.76 | 20.71 |
| (40,10) | 0.995615 | 0.995615 | 0.995615 | 0.973232 | 2.30 | 2.30 | 2.30 |
| (40,20) | 0.998966 | 0.995330 | 0.998966 | 0.952262 | 4.90 | 4.52 | 4.90 |
| (40,30) | 0.998966 | 0.995758 | 0.998966 | 0.973232 | 2.64 | 2.31 | 2.64 |
| (50,10) | 0.997067 | 0.996963 | 0.997097 | 0.988461 | 0.87 | 0.86 | 0.87 |
| (50,20) | 0.999638 | 0.996963 | 0.999638 | 0.988461 | 1.13 | 0.86 | 1.13 |
| | | | | **Average $\Delta$ (%):** | **8.19** | **6.39** | **8.19** |
| | | **FIR Reliability Results** | | | | $\Delta$ (%) | |
| (30,20) | 0.990752 | 0.990752 | - | 0.841248 | 17.77 | 17.77 | - |
| (35,20) | 0.990907 | 0.990907 | - | 0.930428 | 6.50 | 6.50 | - |
| (40,20) | 0.991061 | 0.991061 | - | 0.947032 | 4.65 | 4.65 | - |
| (50,20) | 0.995987 | 0.991370 | - | 0.977758 | 1.86 | 1.39 | - |
| (35,30) | 0.998969 | 0.994278 | 0.998802 | 0.937610 | 6.54 | 6.04 | 6.53 |
| (40,30) | 0.999305 | 0.990316 | 0.998802 | 0.972032 | 2.81 | 1.88 | 2.75 |
| (45,30) | 0.999641 | 0.986848 | 0.998802 | 0.985077 | 1.48 | 0.18 | 1.39 |
| (50,30) | 0.999809 | 0.994143 | 0.998802 | 0.977758 | 2.26 | 1.68 | 2.15 |
| | | | | **Average $\Delta$ (%):** | **5.48** | **5.01** | **3.21** |
| | | **AR Reliability Results** | | | | $\Delta$ (%) | |
| (65,20) | 0.999972 | 0.984852 | 0.999972 | 0.984852 | 1.54 | 0.00 | 1.54 |
| (50,30) | 0.998797 | 0.981463 | 0.998797 | 0.964155 | 3.59 | 1.80 | 3.59 |
| (55,30) | 0.999300 | 0.994155 | 0.999300 | 0.994155 | 0.52 | 0.00 | 0.52 |
| (60,30) | 0.999804 | 0.989493 | 0.999804 | 0.989493 | 1.04 | 0.00 | 1.04 |
| (50,40) | 0.998797 | 0.990321 | 0.998797 | 0.976516 | 2.28 | 1.41 | 2.28 |
| (55,40) | 0.999300 | 0.989124 | 0.999300 | 0.989124 | 1.03 | 0.00 | 1.03 |
| | | | | **Average $\Delta$ (%):** | **1.67** | **0.53** | **1.67** |

evident that all of the proposed ILP methods perform better than the GA-based method, offering an average increase in reliability up to 8.19%. This improvement goes up to more than 20% for some individual designs with the full duplication-based and partial duplication without constraints ILP models, while it can reach up to over 17% with ILP-PD-C.

At the same time, if we consider Table 6.3 that presents only the average change in energy consumption results for the same experiment setups, we observe that even though full DMR can increase the energy consumption up to 70%, the adaptive partial DMR can even achieve an average decrease in energy expenditure of up to more than 5%. For the sake of clarity in this section, the full comparison of the energy results for this set of experiments is presented in Table C.1 in APPENDIX C.

Table 6.3 Average percentage change in the energy results of ILP-based models compared to GA duplication method for all benchmarks when $\alpha = 1.0$.

| | **Average $\Delta$ (%) in Energy Consumption** | | |
|---|---|---|---|
| **Benchmark** | **ILP-PD/GA-SD** | **ILP-PD-C/GA-SD** | **ILP-FD/GA-SD** |
| DES | 42.97 | -4.46 | 69.67 |
| EWF | 59.40 | -1.61 | 59.66 |
| FIR | 34.19 | -5.52 | 59.63 |
| AR | 71.50 | -0.27 | 71.50 |

In summary, when compared to GA-SD, our adaptive ILP-PD-C model that can also take the energy and reliability constraints generates solutions with an average increase in reliability of up to 6.39% while also decreasing the energy consumption up to more than 5% on average, even when the only objective is to maximize reliability, while energy considerations are entirely ignored.

### 6.1.2. Energy Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 0.0$ to focus only on energy optimization, disregarding reliability completely. Comparison of the energy results of ILP and GA duplication methods for all benchmarks are given in Table 6.4.

From the given tables that present energy consumption results for $\alpha = 0.0$, where the objective is to minimize the energy cost without any consideration about the reliability, it is evident that our ILP-PD and ILP-PD-C models perform better than GA-SD for every benchmark, offering an average decrease of energy consumption of up to 69%.

At the same time, if we consider Table 6.5 that presents only the average change in reliability results for the same experiment setups, we can observe that our ILP-PD-C model generates

Table 6.4 Comparison of the energy results of ILP and GA duplication methods for all benchmarks when $\alpha = 0.0$.

| (L, A) | ILP-PD | ILP-PD-C | ILP-FD | GA-SD | ILP-PD/GA-SD | ILP-PD-C/GA-SD | ILP-FD/GA-SD |
|---|---|---|---|---|---|---|---|
| | | **DES Energy Results** | | | | Δ (%) | |
| (28,20) | 505.69 | 505.69 | - | 917.67 | -44.89 | -44.89 | - |
| (28,30) | 457.94 | 461.41 | 985.56 | 1004.73 | -54.42 | -54.08 | -1.91 |
| (28,40) | 457.94 | 461.41 | 955.94 | 1004.73 | -54.42 | -54.08 | -4.86 |
| (25,20) | 509.35 | 512.82 | - | 993.00 | -48.71 | -48.36 | - |
| (25,30) | 484.91 | 507.97 | 1012.41 | 993.00 | -51.17 | -48.84 | 1.95 |
| (25,40) | 483.08 | 492.55 | 980.38 | 993.00 | -51.35 | -50.40 | -1.27 |
| | | | | **Average Δ (%):** | **-50.83** | **-50.11** | **-1.52** |
| | | **EWF Energy Results** | | | | Δ (%) | |
| (30,10) | 125.40 | 125.40 | 312.00 | 231.00 | -45.71 | -45.71 | 35.06 |
| (30,20) | 117.78 | 121.95 | 238.80 | 230.00 | -48.79 | -46.98 | 3.83 |
| (30,30) | 116.65 | 129.86 | 221.68 | 241.00 | -51.60 | -46.12 | -8.02 |
| (40,10) | 106.96 | 115.30 | 216.84 | 175.72 | -39.13 | -34.38 | 23.40 |
| (40,20) | 105.56 | 112.50 | 197.24 | 168.08 | -37.20 | -33.07 | 17.35 |
| (40,30) | 105.56 | 113.20 | 197.24 | 160.44 | -34.21 | -29.44 | 22.94 |
| (50,10) | 100.66 | 127.78 | 216.84 | 191.00 | -47.30 | -33.10 | 13.53 |
| (50,20) | 99.96 | 116.61 | 184.64 | 167.38 | -40.28 | -30.33 | 10.31 |
| | | | | **Average Δ (%):** | **-43.03** | **-37.39** | **14.80** |
| | | **FIR Energy Results** | | | | Δ (%) | |
| (30,20) | 716.70 | 734.05 | - | 1843.00 | -61.11 | -60.17 | - |
| (35,20) | 523.71 | 541.06 | - | 1219.14 | -57.04 | -55.62 | - |
| (40,20) | 514.86 | 539.47 | - | 1187.14 | -56.63 | -54.56 | - |
| (50,20) | 506.27 | 527.09 | - | 1278.50 | -60.40 | -58.77 | - |
| (35,30) | 520.75 | 538.80 | 1051.25 | 1193.14 | -56.35 | -54.84 | -11.89 |
| (40,30) | 510.60 | 529.65 | 1046.19 | 1292.47 | -60.49 | -59.02 | -19.05 |
| (45,30) | 507.67 | 527.82 | 1006.44 | 1173.81 | -56.75 | -55.03 | -14.26 |
| (50,30) | 505.57 | 526.39 | 1001.98 | 1277.80 | -60.43 | -58.80 | -21.59 |
| | | | | **Average Δ (%):** | **-58.65** | **-57.10** | **-16.70** |
| | | **AR Energy Results** | | | | Δ (%) | |
| (65,20) | 955.98 | 960.31 | 2848.00 | 2804.72 | -65.92 | -65.76 | 1.54 |
| (50,30) | 1151.93 | 1156.10 | 2691.35 | 3344.00 | -65.55 | -65.43 | -19.52 |
| (55,30) | 1076.78 | 1077.48 | 2508.92 | 3961.00 | -72.82 | -72.80 | -36.66 |
| (60,30) | 961.90 | 967.90 | 2435.60 | 3945.00 | -75.62 | -75.47 | -38.26 |
| (50,40) | 1145.74 | 1164.52 | 2522.80 | 3179.00 | -63.96 | -63.37 | -20.64 |
| (55,40) | 1051.64 | 1056.51 | 2279.00 | 3494.00 | -69.90 | -69.76 | -34.77 |
| | | | | **Average Δ (%):** | **-68.96** | **-68.76** | **-24.72** |

solutions with higher reliability for every benchmark. The improvement in reliability can reach up to 24.58% percent on average, even though reliability optimization is not an objective in this set of experiments. For the sake of clarity in this section, the full comparison of the reliability results for this set of experiments is presented in Table C.2 in APPENDIX C.

Table 6.5 Average change in the reliability results of ILP-based models compared to GA duplication method for all benchmarks with $\alpha = 0.0$.

| | Average $\Delta$ (%) in reliability | | |
|---|---|---|---|
| **Benchmark** | **ILP-PD/GA-SD** | **ILP-PD-C/GA-SD** | **ILP-FD/GA-SD** |
| DES | -4.56 | 1.32 | 7.09 |
| EWF | 9.22 | 7.19 | 9.22 |
| FIR | -7.88 | 24.58 | 64.78 |
| AR | -4.71 | 0.57 | 21.33 |

ILP-FD also performs well, obtaining design solutions with better reliability values for all benchmarks than GA-SD, even though the only consideration is minimizing energy consumption in this set of experiments. This increase in the overall reliability varies from 7.09% to 64.78% on average for different benchmarks, while at the same time, the induced extra energy cost does not exceed 14.80% on average. For all benchmarks except EWF, the obtained design solutions even showed less energy consumption than their GA-SD counterparts, with energy savings of up to 24.72% percent.

In summary, compared to GA-SD, our adaptive ILP-PD-C model, which can be customized by adding the reliability constraint, generates solutions with an average decrease in energy consumption of up to 69%. At the same time, their reliability is also boosted up to an average of 24% even when the only objective is to minimize the energy cost, while reliability considerations are completely ignored.

### 6.1.3. Joint Reliability and Energy Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 0.5$ to optimize both reliability and energy costs simultaneously. Comparison of the reliability and energy results of the bi-objective partial DMR-based ILP and GA duplication methods for all benchmarks are given in Table 6.6.

The results demonstrate that our ILP-PD method generates better solutions than the GA-SD method for all benchmarks. The solution designs' reliability is improved from 1.63% for the DES benchmark to 27.43% for the EWF benchmark on average. At the same time, the

Table 6.6 Comparison of the reliability and energy results of ILP and GA-based partial duplication methods for all benchmarks with $\alpha = 0.5$.

| | **Reliability Results** | | | **Energy Results** | | | | **Reliability Results** | | | **Energy Results** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **DES** | | | | | | | **AR** | | | |
| **(L, A)** | **ILP-PD** | **GA-SD** | **Δ (%)** | **ILP-PD** | **GA-SD** | **Δ (%)** | **(L, A)** | **ILP-PD** | **GA-SD** | **Δ (%)** | **ILP-PD** | **GA-SD** | **Δ (%)** |
| (28,20) | 0.993175 | 0.992875 | 0.03 | 540.00 | 1,029.00 | -47.52 | (65,20) | 0.999972 | 0.998018 | 0.20 | 2,848.00 | 4,026.00 | -29.26 |
| (28,30) | 0.995197 | 0.963088 | 3.33 | 985.56 | 999.90 | -1.43 | (50,30) | 0.998797 | 0.881482 | 13.31 | 2,764.00 | 3,678.00 | -24.85 |
| (28,40) | 0.998576 | 0.963088 | 3.68 | 963.80 | 999.90 | -3.61 | (55,30) | 0.999300 | 0.987646 | 1.18 | 2,800.00 | 4,020.00 | -30.35 |
| (25,20) | 0.993175 | 0.991069 | 0.21 | 540.00 | 1,023.00 | -47.21 | (60,30) | 0.999804 | 0.989493 | 1.04 | 2,836.00 | 2,444.00 | 16.04 |
| (25,30) | 0.998151 | 0.986044 | 1.23 | 940.00 | 1,001.00 | -6.09 | (50,40) | 0.998280 | 0.884995 | 12.80 | 2,542.24 | 3,513.00 | -27.63 |
| (25,40) | 0.998720 | 0.986044 | 1.29 | 990.07 | 1,001.00 | -1.09 | (55,40) | 0.998277 | 0.948816 | 5.21 | 2,468.92 | 3,205.00 | -22.97 |
| | | **Average Δ (%):** | **1.63** | | | **-17.83** | | | **Average Δ (%):** | **5.62** | | | **-19.84** |
| | | | **EWF** | | | | | | | **FIR** | | | |
| **(L, A)** | **ILP-PD** | **GA-SD** | **Δ (%)** | **ILP-PD** | **GA-SD** | **Δ (%)** | **(L, A)** | **ILP-PD** | **GA-SD** | **Δ (%)** | **ILP-PD** | **GA-SD** | **Δ (%)** |
| (30,10) | 0.945483 | 0.754100 | 25.38 | 288.00 | 246.00 | 17.07 | (35,15) | 0.977824 | 0.928654 | 5.29 | 790.00 | 1,753.00 | -54.93 |
| (30,20) | 0.987541 | 0.754100 | 30.96 | 238.80 | 246.00 | -2.93 | (40,15) | 0.977824 | 0.862414 | 13.38 | 790.00 | 1,898.00 | -58.38 |
| (30,30) | 0.986335 | 0.754100 | 30.80 | 227.82 | 246.00 | -7.39 | (35,20) | 0.981620 | 0.790357 | 24.20 | 624.48 | 1,319.25 | -52.66 |
| (40,10) | 0.985131 | 0.754100 | 30.64 | 216.84 | 246.00 | -11.85 | (40,20) | 0.981620 | 0.792690 | 23.83 | 624.48 | 1,314.41 | -52.49 |
| (40,20) | 0.940980 | 0.754100 | 24.78 | 197.24 | 246.00 | -19.82 | (50,20) | 0.975643 | 0.872780 | 11.79 | 569.58 | 1,219.12 | -53.28 |
| (40,30) | 0.940980 | 0.754100 | 24.78 | 197.24 | 246.00 | -19.82 | (35,30) | 0.997436 | 0.949710 | 5.03 | 1,068.96 | 1,902.00 | -43.80 |
| (50,10) | 0.985131 | 0.754100 | 30.64 | 216.84 | 246.00 | -11.85 | (45,30) | 0.991363 | 0.935407 | 5.98 | 1,014.06 | 1,275.14 | -20.47 |
| (50,20) | 0.915808 | 0.754100 | 21.44 | 185.34 | 246.00 | -24.66 | (50,30) | 0.991363 | 0.961537 | 3.10 | 1,014.06 | 1,281.14 | -20.85 |
| | | **Average Δ (%):** | **27.43** | | | **-10.16** | | | **Average Δ (%):** | **12.32** | | | **-40.59** |

energy cost is significantly reduced for all benchmarks. The energy savings can reach up to 40% on average.

Comparison of the reliability and energy results of the bi-objective full DMR-based ILP and partial GA duplication methods for all benchmarks are given in Table 6.7.

Table 6.7 Comparison of the reliability and energy results of ILP full duplication method with GA-based partial duplication method for all benchmarks with $\alpha = 0.5$.

| | **Reliability Results** | | | **Energy Results** | | | | **Reliability Results** | | | **Energy Results** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **DES** | | | | | | | **AR** | | | |
| **(L, A)** | **ILP-FD** | **GA-SD** | **Δ (%)** | **ILP-FD** | **GA-SD** | **Δ (%)** | **(L, A)** | **ILP-FD** | **GA-SD** | **Δ (%)** | **ILP-FD** | **GA-SD** | **Δ (%)** |
| (31,30) | 0.999985 | 0.985354 | 1.48 | 1044.55 | 1,091.64 | -4.31 | (65,20) | 0.999972 | 0.998018 | 0.20 | 2,848.00 | 4,026.00 | -29.26 |
| (28,30) | 0.999644 | 0.963088 | 3.80 | 1033.98 | 999.90 | 3.41 | (50,30) | 0.998797 | 0.881482 | 13.31 | 2,764.00 | 3,678.00 | -24.85 |
| (28,40) | 0.999642 | 0.963088 | 3.80 | 985.10 | 999.90 | -1.48 | (55,30) | 0.999300 | 0.987646 | 1.18 | 2,800.00 | 4,020.00 | -30.35 |
| (25,30) | 0.999644 | 0.986044 | 1.38 | 1033.98 | 1,001.00 | 3.29 | (50,40) | 0.998280 | 0.884995 | 12.80 | 2,542.24 | 3,513.00 | -27.63 |
| (25,40) | 0.999650 | 0.986044 | 1.38 | 1024.22 | 1,001.00 | 2.32 | (55,40) | 0.998277 | 0.948816 | 5.21 | 2,468.92 | 3,205.00 | -22.97 |
| | | **Average Δ (%):** | **2.37** | | | **0.65** | | | **Average Δ (%):** | **6.54** | | | **-27.01** |
| | | | **EWF** | | | | | | | **FIR** | | | |
| | Reliability Results | | | Energy Results | | | | Reliability Results | | | Energy Results | | |
| **(L, A)** | **ILP-FD** | **GA-SD** | **Δ (%)** | **ILP-FD** | **GA-SD** | **Δ (%)** | **(L, A)** | **ILP-FD** | **GA-SD** | **Δ (%)** | **ILP-FD** | **GA-SD** | **Δ (%)** |
| (30,30) | 0.986335 | 0.754100 | 30.80 | 227.82 | 246.00 | -7.39 | (35,30) | 0.998259 | 0.949710 | 5.11 | 1092.26 | 1,902.00 | -42.57 |
| (40,30) | 0.985131 | 0.754100 | 30.64 | 216.84 | 246.00 | -11.85 | (40,30) | 0.998754 | 0.771518 | 29.45 | 1106.24 | 1,319.25 | -16.15 |
| (50,10) | 0.985131 | 0.754100 | 30.64 | 216.84 | 246.00 | -11.85 | (45,30) | 0.998918 | 0.935407 | 6.79 | 1110.90 | 1,275.14 | -12.88 |
| (50,20) | 0.985131 | 0.754100 | 30.64 | 216.84 | 246.00 | -11.85 | (50,30) | 0.998918 | 0.961537 | 3.89 | 1110.90 | 1,281.14 | -13.29 |
| | | **Average Δ (%):** | **30.68** | | | **-10.74** | | | **Average Δ (%):** | **11.31** | | | **-21.22** |

The results presented in Table 6.7 demonstrate how using full instead of partial modular redundancy when the constraints can be met will yield even more reliable final designs. The reliability of the solution designs is improved as expected since a full DMR-based method is being compared to a partial DMR-based method. The reliability improvement ranges from 2.37% for the DES benchmark to 30.68% for the EWF benchmark on average. Furthermore, the results demonstrate that our ILP-FD method also generates more energy-aware solutions overall when compared to the GA-SD method. The energy cost is not significantly affected for the smallest DES benchmark; however, the energy cost is notably reduced for all other benchmarks. The energy savings can even reach up to 27% on average.

In summary, we observe that both partial duplication-based and full DMR-based models have their advantages in some cases. If the most significant design concern is to obtain designs with maximum reliability without any energy concerns, our ILP-FD model will be a good choice as it will generate the designs with the highest possible reliability for the given area and latency constraints. On the other hand, if the goal is to obtain more energy-aware designs, the ILP-PD-C model generates the most desirable solutions overall, especially when the objective is to optimize one of the parameters disregarding the other. Finally, when optimizing both reliability and energy consumption simultaneously, the ILP-PD model generates far superior solutions to other proposed methods. Sample solutions without and with partial DMR obtained from the ILP models for the DES benchmark for varying $\alpha$ values are presented in Figures C.1, C.2, and C.3 in APPENDIX C.

## 6.2. Comparison of SA-Based Method With ILP and GA-Based HLS Methods

In this set of experiments, we compare our SA-based HLS method with the ILP model and GA-based HLS method proposed in [96]. We use the following abbreviations to denote the methods that are being compared:

- ILP-ND: ILP model without duplication,

- GA-ND: GA-based HLS method without duplication from [96],
- SA-ND: The proposed SA-based HLS method.

For the purpose of a fair comparison, the set of experiments in this subsection was also carried out over DES, FIR, EWF, and AR benchmarks under two voltage supply levels: high and low, because we compare the results of the proposed SA-ND method with the results of the ILP-ND and GA-ND methods that were obtained under only two voltage levels for those specific benchmarks.

### 6.2.1. Reliability Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 1.0$ to focus only on reliability optimization, disregarding energy costs completely. Comparison of the reliability results of SA-ND with the results obtained from ILP-ND and GA-ND methods for all four considered benchmarks are given in Table 6.8.

From the given table that presents reliability results for $\alpha = 1.0$ where the objective is to maximize the reliability without any consideration about the energy consumption, it can be observed that the proposed SA-ND method generates the optimal solutions for almost all test cases. Only for the AR benchmark slightly worse solutions were generated in a couple of test cases compared to the results of the ILP-ND method. Nevertheless, the average percentage change is negligible.

On the other hand, the proposed SA-ND method outperformed the other metaheuristic GA-ND method in all test cases, except for one for which it produced the same result, for which the GA-based HLS method produced worse results compared to the ILP-ND. The average improvement of SA-ND over GA-ND in final design reliability reached up to 2.31% for the FIR benchmark.

At the same time, if we consider Table 6.9 that presents only the average change in energy consumption results for the same experiment setups in which energy considerations have

Table 6.8 Comparison of the reliability results of SA-based method with ILP and GA-based methods for all benchmarks when $\alpha = 1.0$.

| (L, A) | SA-ND | ILP-ND | GA-ND | SA-ND/ILP-ND | SA-ND/GA-ND |
|---|---|---|---|---|---|
| **DES Reliability Results** | | | | $\Delta$ (%) | |
| (31,20) | 0.989055 | 0.989055 | 0.989055 | 0.00 | 0.00 |
| (31,30) | 0.989055 | 0.989055 | 0.989055 | 0.00 | 0.00 |
| (28,20) | 0.977174 | 0.977174 | 0.977174 | 0.00 | 0.00 |
| (28,30) | 0.977174 | 0.977174 | 0.977174 | 0.00 | 0.00 |
| (28,40) | 0.977174 | 0.977174 | 0.977174 | 0.00 | 0.00 |
| (25,20) | 0.965436 | 0.965436 | 0.965436 | 0.00 | 0.00 |
| (25,30) | 0.965436 | 0.965436 | 0.965436 | 0.00 | 0.00 |
| (25,40) | 0.965436 | 0.965436 | 0.965436 | 0.00 | 0.00 |
| | | | **Average $\Delta$ (%)** | **0.00** | **0.00** |
| **EWF Reliability Results** | | | | $\Delta$ (%) | |
| (30,10) | 0.822671 | 0.822671 | 0.793380 | 0.00 | 3.69 |
| (30,20) | 0.822671 | 0.822671 | 0.793380 | 0.00 | 3.69 |
| (30,30) | 0.822671 | 0.822671 | 0.793380 | 0.00 | 3.69 |
| (40,10) | 0.906176 | 0.906176 | 0.906176 | 0.00 | 0.00 |
| (40,20) | 0.906176 | 0.906176 | 0.895291 | 0.00 | 1.22 |
| (40,30) | 0.906176 | 0.906176 | 0.906176 | 0.00 | 0.00 |
| (50,10) | 0.951056 | 0.951056 | 0.951056 | 0.00 | 0.00 |
| (50,20) | 0.951056 | 0.951056 | 0.951056 | 0.00 | 0.00 |
| (50,30) | 0.951056 | 0.951056 | 0.951056 | 0.00 | 0.00 |
| | | | **Average $\Delta$ (%)** | **0.00** | **1.37** |
| **FIR Reliability Results** | | | | $\Delta$ (%) | |
| (30,20) | 0.897983 | 0.897983 | 0.790310 | 0.00 | 13.62 |
| (35,20) | 0.908900 | 0.908900 | 0.908900 | 0.00 | 0.00 |
| (35,30) | 0.919951 | 0.919951 | 0.897983 | 0.00 | 2.45 |
| (40,20) | 0.931136 | 0.931136 | 0.931136 | 0.00 | 0.00 |
| (40,30) | 0.931136 | 0.931136 | 0.931136 | 0.00 | 0.00 |
| (45,30) | 0.953915 | 0.953915 | 0.953915 | 0.00 | 0.00 |
| (50,20) | 0.977251 | 0.977251 | 0.965512 | 0.00 | 1.22 |
| (50,30) | 0.977251 | 0.977251 | 0.965512 | 0.00 | 1.22 |
| | | | **Average $\Delta$ (%)** | **0.00** | **2.31** |
| **AR Reliability Results** | | | | $\Delta$ (%) | |
| (55,20) | 0.926489 | 0.926489 | 0.926489 | 0.00 | 0.00 |
| (55,30) | 0.949155 | 0.949155 | 0.937753 | 0.00 | 1.22 |
| (55,40) | 0.972375 | 0.972375 | 0.960695 | 0.00 | 1.22 |
| (60,20) | 0.949155 | 0.972375 | 0.949155 | -2.39 | 0.00 |
| (60,30) | 0.972375 | 0.972375 | 0.972375 | 0.00 | 0.00 |
| (65,15) | 0.972375 | 0.972375 | 0.949155 | 0.00 | 2.45 |
| (65,20) | 0.971401 | 0.972375 | 0.949155 | -0.10 | 2.45 |
| | | | **Average $\Delta$ (%)** | **-0.36** | **1.03** |

Table 6.9 Average percentage change in the energy results of SA-based method compared to ILP and GA-based methods for all benchmarks when $\alpha = 1.0$.

| | **Average $\Delta$ (%) in Energy Consumption** | |
|---|---|---|
| **Benchmark** | **SA-ND/ILP-ND** | **SA-ND/GA-ND** |
| DES | 0.00 | 0.00 |
| EWF | 0.00 | 3.10 |
| FIR | 0.00 | 1.04 |
| AR | -0.24 | 0.24 |

completely been disregarded in the optimization, we observe that our proposed SA-ND method does not introduce any extra energy consumption overhead compared to the ILP-ND method while obtaining the optimal or near-optimal solutions. The average percentage change compared to the GA-ND method shows a negligible increase; however, considering that energy minimization is not an objective in this set of experiments, this outcome is expected and irrelevant since the proposed method obtains better results in terms of reliability for those test cases. For the sake of brevity in this section, the full comparison of the energy results for this set of experiments is presented in Table C.3 in APPENDIX C.

### 6.2.2. Energy Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 0.0$ to focus only on energy optimization, disregarding reliability completely. Comparison of the energy results of SA-ND with the results obtained from ILP-ND and GA-ND methods for all four considered benchmarks are given in Table 6.10.

From the given table that presents energy results for $\alpha = 0.0$ where the objective is to minimize the energy consumption without any consideration about the circuit reliability, it can be observed that the proposed SA-ND method generated the optimal or near-optimal solutions for most of the test cases, except for AR benchmark for which the obtained solutions exhibit about 7.69% more energy consumption compared to the results generated with ILP-ND method. Nevertheless, the proposed SA-ND method outperformed the other metaheuristic GA-ND method in almost all test cases, except for a couple of cases for which

Table 6.10 Comparison of the energy results of SA-based method with ILP and GA-based methods for all benchmarks when $\alpha = 0.0$.

| **(L, A)** | **SA-ND** | **ILP-ND** | **GA-ND** | **SA-ND/ILP-ND** | **SA-ND/GA-ND** |
|---|---|---|---|---|---|
| | **DES Energy Results** | | | $\Delta$ (%) | |
| (31,20) | 448.47 | 448.47 | 475.44 | 0.00 | -5.67 |
| (31,30) | 427.26 | 404.65 | 436.62 | 5.59 | -2.14 |
| (28,20) | 480.56 | 480.56 | 481.56 | 0.00 | -0.21 |
| (28,30) | 451.00 | 451.00 | 452.10 | 0.00 | -0.24 |
| (28,40) | 451.00 | 451.00 | 452.10 | 0.00 | -0.24 |
| (25,20) | 502.41 | 502.41 | 508.00 | 0.00 | -1.10 |
| (25,30) | 477.97 | 477.97 | 508.00 | 0.00 | -5.91 |
| (25,40) | 476.14 | 476.14 | 508.00 | 0.00 | -6.27 |
| | | | **Average $\Delta$ (%)** | **0.70** | **-2.72** |
| | **EWF Energy Results** | | | $\Delta$ (%) | |
| (30,10) | 119.40 | 119.40 | 137.00 | 0.00 | -12.85 |
| (30,20) | 116.16 | 109.71 | 135.00 | 5.88 | -13.96 |
| (30,30) | 116.16 | 109.71 | 136.00 | 5.88 | -14.59 |
| (40,10) | 99.88 | 98.62 | 100.02 | 1.28 | -0.14 |
| (40,20) | 99.05 | 98.62 | 99.32 | 0.44 | -0.27 |
| (40,30) | 99.05 | 98.62 | 101.42 | 0.44 | -2.34 |
| (50,10) | 93.02 | 92.32 | 93.72 | 0.76 | -0.75 |
| (50,20) | 93.02 | 92.32 | 94.42 | 0.76 | -1.48 |
| (50,30) | 92.32 | 92.32 | 92.32 | 0.00 | 0.00 |
| | | | **Average $\Delta$ (%)** | **1.71** | **-5.15** |
| | **FIR Energy Results** | | | $\Delta$ (%) | |
| (30,20) | 709.76 | 709.76 | 723.00 | 0.00 | -1.83 |
| (35,20) | 515.77 | 515.77 | 542.48 | 0.00 | -4.92 |
| (35,30) | 513.81 | 513.81 | 526.48 | 0.00 | -2.41 |
| (40,15) | 534.48 | 534.48 | 748.00 | 0.00 | -28.55 |
| (40,20) | 508.86 | 508.86 | 520.48 | 0.00 | -2.23 |
| (40,30) | 505.92 | 502.83 | 514.70 | 0.61 | -1.71 |
| (45,30) | 501.86 | 500.73 | 507.15 | 0.23 | -1.04 |
| (50,15) | 523.07 | 502.65 | 532.32 | 4.06 | -1.74 |
| (50,20) | 499.46 | 499.33 | 500.73 | 0.03 | -0.25 |
| (50,30) | 498.63 | 498.63 | 500.03 | 0.00 | -0.28 |
| | | | **Average $\Delta$ (%)** | **0.49** | **-4.50** |
| | **AR Energy Results** | | | $\Delta$ (%) | |
| (55,20) | 1144.48 | 1144.48 | 1343.00 | 0.00 | -14.78 |
| (55,30) | 1151.50 | 1058.72 | 1346.00 | 8.76 | -14.45 |
| (55,40) | 1095.05 | 1048.17 | 1370.00 | 4.47 | -20.07 |
| (60,20) | 1336.35 | 1071.16 | 1340.68 | 24.76 | -0.32 |
| (60,30) | 1095.15 | 955.90 | 1351.00 | 14.57 | -18.94 |
| (65,15) | 960.96 | 960.96 | 1375.00 | 0.00 | -30.11 |
| (65,20) | 961.96 | 949.98 | 961.96 | 1.26 | 0.00 |
| | | | **Average $\Delta$ (%)** | **7.69** | **-14.10** |

Table 6.11 Average percentage change in the reliability results of SA-based method compared to ILP and GA-based methods for all benchmarks when $\alpha = 0.0$.

| | **Average $\Delta$ (%) in Reliability** | |
|---|---|---|
| **Benchmark** | **SA-ND/ILP-ND** | **SA-ND/GA-ND** |
| DES | -0.13 | -8.19 |
| EWF | -1.82 | -22.10 |
| FIR | -1.46 | -15.44 |
| AR | -0.41 | 0.58 |

it produced the same results. The bigger the circuit, the higher average energy savings are obtained, reaching up to over 14% more energy-efficient solutions on average compared to the GA-ND method.

At the same time, if we consider Table 6.11 that presents only the average change in reliability results for the same experiment setups in which reliability considerations have completely been disregarded in the optimization, we observe that our proposed SA-ND method does not produce solutions with significantly deteriorated reliability compared to the optimal results obtained from ILP-ND method. For the sake of brevity in this section, the full comparison of the reliability results for this set of experiments is presented in Table C.4 in APPENDIX C.

Overall, SA-ND generates solutions much closer to the optimal ones when optimizing the energy consumption is the only objective, which can be observed from the relatively higher average percentage changes for both the energy and reliability results. Hence, the proposed SA-ND method is much more efficient than GA-ND when optimizing energy only.

### 6.2.3. Joint Reliability and Energy Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 0.5$ to optimize both reliability and energy costs simultaneously. Comparison of the reliability and energy results of the bi-objective SA-ND with the results obtained from the corresponding ILP-ND and GA-ND methods for all four considered benchmarks are given in Table 6.12.

Table 6.12 Comparison of the reliability and energy results of SA-based method with ILP and GA-based methods for all benchmarks when $\alpha = 0.5$.

| | Reliability Results | | | Δ (%) | | Energy Results | | | Δ (%) | |
|---|---|---|---|---|---|---|---|---|---|---|
| **(L, A)** | **SA-ND** | **ILP-ND** | **GA-ND** | **SA-ND/ILP-ND** | **SA-ND/GA-ND** | **SA-ND** | **ILP-ND** | **GA-ND** | **SA-ND/ILP-ND** | **SA-ND/GA-ND** |
| | | | | | **DES** | | | | | |
| (31,20) | 0.984115 | 0.984115 | 0.988065 | 0.00 | -0.40 | 480.11 | 480.11 | 515.56 | 0.00 | -6.88 |
| (31,30) | 0.958692 | 0.958692 | 0.941322 | 0.00 | 1.85 | 419.23 | 419.23 | 492.06 | 0.00 | -14.80 |
| (28,20) | 0.972293 | 0.974243 | 0.977174 | -0.20 | -0.50 | 474.11 | 522.99 | 534.00 | -9.35 | -11.22 |
| (28,30) | 0.972293 | 0.972293 | 0.933635 | 0.00 | 4.14 | 474.11 | 474.11 | 447.68 | 0.00 | 5.90 |
| (28,40) | 0.972293 | 0.972293 | 0.933635 | 0.00 | 4.14 | 474.11 | 474.11 | 447.68 | 0.00 | 5.90 |
| (25,20) | 0.962540 | 0.962540 | 0.965436 | 0.00 | -0.30 | 496.22 | 516.99 | 528.00 | -4.02 | -6.02 |
| (25,30) | 0.961577 | 0.961577 | 0.942382 | 0.00 | 2.04 | 492.55 | 492.55 | 516.00 | 0.00 | -4.54 |
| (25,40) | 0.961577 | 0.961577 | 0.942382 | 0.00 | 2.04 | 492.55 | 492.55 | 516.00 | 0.00 | -4.54 |
| | | | **Average Δ (%)** | **-0.03** | **1.63** | | | **Average Δ (%)** | **-1.67** | **-4.52** |
| | | | | | **EWF** | | | | | |
| (30,10) | 0.562383 | 0.568721 | 0.711616 | -1.11 | -20.97 | 117.57 | 119.40 | 156.00 | -1.53 | -24.63 |
| (30,20) | 0.556115 | 0.549917 | 0.711616 | 1.13 | -21.85 | 115.74 | 113.91 | 156.00 | 1.61 | -25.81 |
| (30,30) | 0.556115 | 0.549917 | 0.711616 | 1.13 | -21.85 | 115.74 | 113.91 | 156.00 | 1.61 | -25.81 |
| (40,10) | 0.537728 | 0.531735 | 0.711616 | 1.13 | -24.44 | 110.25 | 108.42 | 156.00 | 1.69 | -29.33 |
| (40,20) | 0.531735 | 0.531735 | 0.711616 | 0.00 | -25.28 | 108.42 | 108.42 | 156.00 | 0.00 | -30.50 |
| (40,30) | 0.531735 | 0.531735 | 0.711616 | 0.00 | -25.28 | 108.42 | 108.42 | 156.00 | 0.00 | -30.50 |
| (50,10) | 0.537728 | 0.531735 | 0.711616 | 1.13 | -24.44 | 110.25 | 108.42 | 156.00 | 1.69 | -29.33 |
| (50,20) | 0.531735 | 0.531735 | 0.711616 | 0.00 | -25.28 | 108.42 | 108.42 | 156.00 | 0.00 | -30.50 |
| (50,30) | 0.531735 | 0.531735 | 0.711616 | 0.00 | -25.28 | 108.42 | 108.42 | 156.00 | 0.00 | -30.50 |
| | | | **Average Δ (%)** | **0.38** | **-23.85** | | | **Average Δ (%)** | **0.56** | **-28.54** |
| | | | | | **FIR** | | | | | |
| (35,15) | 0.886367 | 0.854808 | 0.876539 | 3.69 | 1.12 | 605.67 | 546.13 | 766.00 | 10.90 | -20.93 |
| (35,20) | 0.886367 | 0.875720 | 0.760460 | 1.22 | 16.56 | 564.13 | 558.13 | 536.48 | 1.08 | 5.15 |
| (35,30) | 0.886367 | 0.886367 | 0.887196 | 0.00 | -0.09 | 564.13 | 564.13 | 772.00 | 0.00 | -26.93 |
| (40,15) | 0.884593 | 0.883708 | 0.813872 | 0.10 | 8.69 | 577.56 | 553.12 | 778.00 | 4.42 | -25.76 |
| (40,20) | 0.883708 | 0.883708 | 0.782338 | 0.00 | 12.96 | 553.12 | 553.12 | 536.64 | 0.00 | 3.07 |
| (40,30) | 0.883708 | 0.883708 | 0.741703 | 0.00 | 19.15 | 553.12 | 553.12 | 541.48 | 0.00 | 2.15 |
| (45,30) | 0.893557 | 0.893557 | 0.901647 | 0.00 | -0.90 | 555.45 | 555.45 | 588.48 | 0.00 | -5.61 |
| (50,15) | 0.904421 | 0.893557 | 0.901647 | 1.22 | 0.31 | 582.22 | 555.45 | 588.48 | 4.82 | -1.06 |
| (50,20) | 0.903515 | 0.904421 | 0.843225 | -0.10 | 7.15 | 557.78 | 561.45 | 552.46 | -0.65 | 0.96 |
| (50,30) | 0.903515 | 0.903515 | 0.912609 | 0.00 | -1.00 | 557.78 | 557.78 | 594.48 | 0.00 | -6.17 |
| | | | **Average Δ (%)** | **0.61** | **6.39** | | | **Average Δ (%)** | **2.06** | **-7.51** |
| | | | | | **AR** | | | | | |
| (55,20) | 0.924635 | 0.924635 | 0.920652 | 0.00 | 0.43 | 1392.66 | 1392.66 | 1405.00 | 0.00 | -0.88 |
| (55,30) | 0.874848 | 0.831064 | 0.926489 | 5.27 | -5.57 | 1176.81 | 1058.72 | 1400.00 | 11.15 | -15.94 |
| (55,40) | 0.873097 | 0.849691 | 0.920652 | 2.75 | -5.17 | 1107.16 | 1063.38 | 1405.00 | 4.12 | -21.20 |
| (60,20) | 0.969458 | 0.968487 | 0.931845 | 0.10 | 4.04 | 1412.99 | 1409.32 | 1411.00 | 0.26 | 0.14 |
| (60,30) | 0.854603 | 0.846294 | 0.972375 | 0.98 | -12.11 | 1095.15 | 965.62 | 1424.00 | 13.41 | -23.09 |
| (65,15) | 0.882769 | 0.858301 | 0.949155 | 2.85 | -6.99 | 1376.00 | 978.96 | 1412.00 | 40.56 | -2.55 |
| | | | **Average Δ (%)** | **1.99** | **-4.23** | | | **Average Δ (%)** | **11.58** | **-10.59** |

From the given table that presents both reliability and energy results for $\alpha = 0.5$ where the objective is to both maximize the reliability and minimize the energy consumption with equal weight given to both objectives, it can be observed that the proposed SA-ND method generates Pareto-optimal solutions for some cases and acceptably near-Pareto-optimal solutions in other while showing more consistency than GA-ND in generating closer Pareto-optimal solutions overall to those obtained from ILP-ND. The worst performance for the proposed SA-ND method is apparent for the AR benchmark, for which it generates about

11% less energy-oriented solutions than ILP-ND on average. However, that additional energy overhead is compensated for with an overall reliability increase of up to 2% on average.

All results considered, although the proposed SA-ND method may still need some tuning, especially for some edge cases with the tight area and latency constraints, if it is to generate closer Pareto-optimal solutions to those obtained with ILP-ND in bi-objective optimization tests, it can generate optimal solutions for many cases unlike GA-ND and is more consistent than GA-ND in generating close enough solutions in other cases.

Furthermore, the proposed SA-ND method outperforms GA-ND for all benchmarks when optimizing one objective at a time. The deviation of the reliability values for the design solutions obtained by SA-ND from the optimal values is under 2% for all experimental setups, including the cases in which only energy optimization is being considered with complete disregard for a possible impact on reliability. Similarly, the deviation of the energy results from the optimal ones is also around 2% for all benchmarks except AR, for which the average percentage change in energy values reaches up to more than 7% when optimizing energy only, and 11% for bi-objective optimization due to some edge cases. A possible explanation for this could be the test execution setup that was designed to be as close as possible to the experimental setup used for obtaining GA-ND results in order to have a fair comparison. The GA-ND results were obtained through a GA-based process of 1000 iterations for all benchmarks regardless of their size. We used the same number of iterations for the simulated annealing process for the purpose of fair comparison.

## 6.3. Comparison of SA-Based Partial DMR HLS Method With the Corresponding ILP and GA-Based Methods

In this set of experiments, we compare our SA-based Partial DMR HLS method with the corresponding ILP model (ILP-PD) and GA-based HLS method with selective duplication (GA-SD) proposed in [96]. We use the following abbreviations to denote the methods that are being compared:

- ILP-PD: Partial DMR ILP model,
- GA-SD: GA-based selective duplication method from [96],
- SA-PD: The proposed SA-based HLS method with partial duplication.

For the purpose of a fair comparison, the set of experiments in this subsection was also carried out over DES, FIR, EWF, and AR benchmarks under two voltage supply levels: high and low, because we compare the results of the proposed SA-PD method with the results of the ILP-PD and GA-SD methods that were obtained under only two voltage levels for those specific benchmarks.

### 6.3.1. Reliability Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 1.0$ to focus only on reliability optimization, disregarding energy costs completely. Comparison of the reliability results of SA-PD with the results obtained from ILP-PD and GA-SD methods for all four considered benchmarks are given in Table 6.13.

From the given table that presents reliability results for $\alpha = 1.0$ where the objective is to maximize the reliability without any consideration about the energy consumption, it can be observed that the proposed SA-ND method generated the optimal or near-optimal solutions for most of the test cases. The average percentage change for the reliability results when our SA-based partial duplication method is compared to the ILP-PD results is less than 2% for all benchmarks. Moreover, the proposed SA-PD method outperformed the other metaheuristic GA-SD method in almost all test cases, with the average improvement in the final design reliability for all benchmarks and up to 5% for the EWF benchmark.

### 6.3.2. Energy Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 0.0$ to focus only on energy optimization, disregarding reliability completely. Comparison of the energy results of

Table 6.13 Comparison of the reliability results of SA-based partial DMR method with the corresponding ILP and GA-based methods for all benchmarks when $\alpha = 1.0$.

| **(L, A)** | **SA-PD** | **ILP-PD** | **GA-SD** | **SA-PD/ILP-PD** | **SA-PD/GA-SD** |
|---|---|---|---|---|---|
| **DES Reliability Results** | | | | $\Delta$ **(%)** | |
| (31,20) | 0.995004 | 0.999989 | 0.994825 | -0.50 | 0.02 |
| (31,30) | 0.995004 | 0.999989 | 0.994825 | -0.50 | 0.02 |
| (28,20) | 0.993843 | 0.993795 | 0.982875 | 0.00 | 1.12 |
| (28,30) | 0.994837 | 0.999821 | 0.982875 | -0.50 | 1.22 |
| (28,40) | 0.995832 | 0.999821 | 0.982875 | -0.40 | 1.32 |
| (25,20) | 0.980924 | 0.993640 | 0.971069 | -1.28 | 1.01 |
| (25,30) | 0.981905 | 0.999653 | 0.971069 | -1.78 | 1.12 |
| (25,40) | 0.981905 | 0.999653 | 0.971069 | -1.78 | 1.12 |
| | | | **Average $\Delta$ (%)** | **-0.84** | **0.87** |
| **EWF Reliability Results** | | | | $\Delta$ **(%)** | |
| (30,20) | 0.996200 | 0.997625 | 0.826462 | -0.14 | 20.54 |
| (30,30) | 0.997625 | 0.997625 | 0.826462 | 0.00 | 20.71 |
| (40,10) | 0.906176 | 0.995615 | 0.973232 | -8.98 | -6.89 |
| (40,20) | 0.998148 | 0.998966 | 0.952262 | -0.08 | 4.82 |
| (40,30) | 0.998966 | 0.998966 | 0.973232 | 0.00 | 2.64 |
| (50,10) | 0.951056 | 0.997067 | 0.988461 | -4.61 | -3.78 |
| (50,20) | 0.998536 | 0.999638 | 0.988461 | -0.11 | 1.02 |
| (50,30) | 0.999638 | 0.999638 | 0.988461 | 0.00 | 1.13 |
| | | | **Average $\Delta$ (%)** | **-1.74** | **5.02** |
| **FIR Reliability Results** | | | | $\Delta$ **(%)** | |
| (35,20) | 0.961080 | 0.990907 | 0.930428 | -3.01 | 3.29 |
| (35,30) | 0.980394 | 0.998969 | 0.937610 | -1.86 | 4.56 |
| (40,20) | 0.961080 | 0.991061 | 0.947032 | -3.03 | 1.48 |
| (40,30) | 0.997066 | 0.999305 | 0.972032 | -0.22 | 2.58 |
| (45,30) | 0.997597 | 0.999641 | 0.985077 | -0.20 | 1.27 |
| (50,20) | 0.977758 | 0.995987 | 0.977758 | -1.83 | 0.00 |
| (50,30) | 0.997764 | 0.999809 | 0.977758 | -0.20 | 2.05 |
| | | | **Average $\Delta$ (%)** | **-1.48** | **2.18** |
| **AR Reliability Results** | | | | $\Delta$ **(%)** | |
| (50,30) | 0.998797 | 0.998797 | 0.964155 | 0.00 | 3.59 |
| (50,40) | 0.998629 | 0.998797 | 0.976516 | -0.02 | 2.26 |
| (55,30) | 0.999132 | 0.999300 | 0.994155 | -0.02 | 0.50 |
| (55,40) | 0.999132 | 0.999300 | 0.989124 | -0.02 | 1.01 |
| (60,20) | 0.997975 | 0.997975 | 0.997389 | 0.00 | 0.06 |
| (60,30) | 0.999132 | 0.999804 | 0.989493 | -0.07 | 0.97 |
| | | | **Average $\Delta$ (%)** | **-0.02** | **1.40** |

SA-PD with the results obtained from ILP-PD and GA-SD methods for all four considered benchmarks are given in Table 6.14.

From the given table that presents energy results for $\alpha = 0.0$ where the objective is to minimize the energy consumption without any consideration about the circuit reliability, it can be observed that the proposed SA-PD method generates optimal or near-optimal results only for some test cases. The average percentage change in the final circuit energy consumption results ranges from 3.65% for the DES benchmark to 31.77% for the FIR benchmark. These results show that the proposed SA-PD method is not as efficient in obtaining energy-aware results as it is in optimizing reliability.

Nevertheless, the proposed SA-PD method significantly outperforms the other metaheuristic GA-SD method in all test cases and for all benchmarks, generating 35% to 65% more energy-saving solutions on average.

### 6.3.3. Joint Reliability and Energy Optimization Results Discussion

In this set of experiments, we assign the parameter $\alpha = 0.5$ to optimize both reliability and energy costs simultaneously. Comparison of the reliability and energy results of the bi-objective SA-PD with the results obtained from the corresponding ILP-PD and GA-SD methods for all four considered benchmarks are given in Table 6.15.

From the given table that presents both reliability and energy results for $\alpha = 0.5$ where the objective is to both maximize the reliability and minimize the energy consumption with equal weight given to both objectives, the following conclusions can be made. When we look at the SA-PD/ILP-PD columns for both reliability and energy, we observe that the proposed SA-PD method does not generate Pareto-optimal or near-Pareto-optimal solutions as consistently as the SA-based method for HLS without resource duplication.

For the DES benchmark, SA-PD generates about 24% more energy-saving solutions on average at the expense of an average decrease in reliability of about 1%. However, for all other benchmarks, SA-PD generates solutions that have slightly worse both reliability and

Table 6.14 Comparison of the energy results of SA-based partial DMR method with the corresponding ILP and GA-based methods for all benchmarks when $\alpha = 0.0$.

| **(L, A)** | **SA-PD** | **ILP-PD** | **GA-SD** | **SA-PD/ILP-PD** | **SA-PD/GA-SD** |
|---|---|---|---|---|---|
| **DES Energy Results** | | | | **Δ (%)** | |
| (31,20) | 411.29 | 504.11 | 906.55 | -18.41 | -54.63 |
| (31,30) | 411.29 | 411.59 | 978.84 | -0.07 | -57.98 |
| (28,20) | 510.03 | 505.69 | 917.67 | 0.86 | -44.42 |
| (28,30) | 457.94 | 457.94 | 1004.73 | 0.00 | -54.42 |
| (28,40) | 457.94 | 457.94 | 1004.73 | 0.00 | -54.42 |
| (25,20) | 598.07 | 509.35 | 993.00 | 17.42 | -39.77 |
| (25,30) | 598.07 | 484.91 | 993.00 | 23.34 | -39.77 |
| (25,40) | 512.52 | 483.08 | 993.00 | 6.09 | -48.39 |
| | | | **Average Δ (%)** | **3.65** | **-49.23** |
| **EWF Energy Results** | | | | **Δ (%)** | |
| (30,20) | 143.00 | 117.78 | 230.00 | 21.41 | -37.83 |
| (30,30) | 143.00 | 116.65 | 241.00 | 22.59 | -40.66 |
| (40,10) | 143.00 | 106.96 | 175.72 | 33.69 | -18.62 |
| (40,20) | 105.70 | 105.56 | 168.08 | 0.13 | -37.11 |
| (40,30) | 105.70 | 105.56 | 160.44 | 0.13 | -34.12 |
| (50,10) | 143.00 | 100.66 | 191.00 | 42.06 | -25.13 |
| (50,20) | 99.96 | 99.96 | 167.38 | 0.00 | -40.28 |
| (50,30) | 99.96 | 99.96 | 181.96 | 0.00 | -45.06 |
| | | | **Average Δ (%)** | **15.00** | **-34.85** |
| **FIR Energy Results** | | | | **Δ (%)** | |
| (35,20) | 742.00 | 523.71 | 1219.14 | 41.68 | -39.14 |
| (35,30) | 648.50 | 520.75 | 1193.14 | 24.53 | -45.65 |
| (40,20) | 726.00 | 514.86 | 1187.14 | 41.01 | -38.84 |
| (40,30) | 648.50 | 510.60 | 1292.47 | 27.01 | -49.82 |
| (45,30) | 645.42 | 507.67 | 1173.81 | 27.13 | -45.01 |
| (50,20) | 726.00 | 506.27 | 1278.50 | 43.40 | -43.21 |
| (50,30) | 594.55 | 505.57 | 1277.80 | 17.60 | -53.47 |
| | | | **Average Δ (%)** | **31.77** | **-45.02** |
| **AR Energy Results** | | | | **Δ (%)** | |
| (50,30) | 1361.00 | 1151.93 | 3344.00 | 18.15 | -59.30 |
| (50,40) | 1364.00 | 1145.74 | 3179.00 | 19.05 | -57.09 |
| (55,30) | 1361.00 | 1076.78 | 3961.00 | 26.40 | -65.64 |
| (55,40) | 1242.35 | 1051.64 | 3494.00 | 18.13 | -64.44 |
| (60,20) | 1326.54 | 1151.16 | 3943.24 | 15.24 | -66.36 |
| (60,30) | 1095.15 | 961.90 | 3945.00 | 13.85 | -72.24 |
| | | | **Average Δ (%)** | **18.47** | **-64.18** |

Table 6.15 Comparison of the reliability and energy results of SA-based partial DMR method with the corresponding ILP and GA-based methods for all benchmarks when $\alpha = 0.5$.

| **Reliability Results** | | | | **Δ (%)** | | **Energy Results** | | | **Δ (%)** | |
|---|---|---|---|---|---|---|---|---|---|---|
| **(L, A)** | **SA-PD** | **ILP-PD** | **GA-SD** | **SA-PD/ILP-PD** | **SA-PD/GA-SD** | **SA-PD** | **ILP-PD** | **GA-SD** | **SA-PD/ILP-PD** | **SA-PD/GA-SD** |
| | | | | | **DES** | | | | | |
| (31,20) | 0.993702 | 0.999989 | 0.994732 | -0.63 | -0.10 | 743.93 | 1080.00 | 961.67 | -31.12 | -22.64 |
| (31,30) | 0.989172 | 0.995196 | 0.985354 | -0.61 | 0.39 | 472.82 | 961.12 | 1091.64 | -50.81 | -56.69 |
| (28,20) | 0.981768 | 0.993175 | 0.992875 | -1.15 | -1.12 | 661.60 | 540.00 | 1029.00 | 22.52 | -35.70 |
| (28,30) | 0.981768 | 0.995197 | 0.963088 | -1.35 | 1.94 | 661.60 | 985.56 | 999.90 | -32.87 | -33.83 |
| (28,40) | 0.981768 | 0.998576 | 0.963088 | -1.68 | 1.94 | 661.60 | 963.80 | 999.90 | -31.36 | -33.83 |
| (25,20) | 0.980790 | 0.993175 | 0.991069 | -1.25 | -1.04 | 630.48 | 540.00 | 1023.00 | 16.76 | -38.37 |
| (25,30) | 0.980790 | 0.999149 | 0.986044 | -1.84 | -0.53 | 630.48 | 1020.00 | 1001.00 | -38.19 | -37.01 |
| (25,40) | 0.988150 | 0.998720 | 0.986044 | -1.06 | 0.21 | 529.83 | 990.07 | 1001.00 | -46.49 | -47.07 |
| | | | **Average Δ (%)** | **-1.19** | **0.21** | | | **Average Δ (%)** | **-23.94** | **-38.14** |
| | | | | | **EWF** | | | | | |
| (30,20) | 0.985183 | 0.987541 | 0.754100 | -0.24 | 30.64 | 258.23 | 238.80 | 246.00 | 8.14 | 4.97 |
| (30,30) | 0.945293 | 0.986335 | 0.754100 | -4.16 | 25.35 | 227.74 | 227.82 | 246.00 | -0.04 | -7.42 |
| (40,10) | 0.793380 | 0.985131 | 0.754100 | -19.46 | 5.21 | 210.00 | 216.84 | 246.00 | -3.15 | -14.63 |
| (40,20) | 0.954473 | 0.940980 | 0.754100 | 1.43 | 26.57 | 215.09 | 197.24 | 246.00 | 9.05 | -12.57 |
| (40,30) | 0.945118 | 0.940980 | 0.754100 | 0.44 | 25.33 | 200.30 | 197.24 | 246.00 | 1.55 | -18.58 |
| (50,10) | 0.823493 | 0.985131 | 0.754100 | -16.41 | 9.20 | 240.00 | 216.84 | 246.00 | 10.68 | -2.44 |
| (50,20) | 0.937518 | 0.915808 | 0.754100 | 2.37 | 24.32 | 197.19 | 185.34 | 246.00 | 6.39 | -19.84 |
| (50,30) | 0.927536 | 0.915808 | 0.754100 | 1.28 | 23.00 | 189.54 | 185.34 | 246.00 | 2.27 | -22.95 |
| | | | **Average Δ (%)** | **-4.34** | **21.20** | | | **Average Δ (%)** | **4.36** | **-11.68** |
| | | | | | **FIR** | | | | | |
| (35,20) | 0.912687 | 0.981620 | 0.790357 | -7.02 | 15.48 | 784.000 | 624.4800 | 1,319.2500 | 25.54 | -40.57 |
| (35,30) | 0.982287 | 0.997436 | 0.949710 | -1.52 | 3.43 | 1134.57 | 1068.96 | 1902.00 | 6.14 | -40.35 |
| (40,20) | 0.947958 | 0.981620 | 0.792690 | -3.43 | 19.59 | 802.00 | 624.48 | 1314.41 | 28.43 | -38.98 |
| (40,30) | 0.993873 | 0.997436 | 0.771518 | -0.36 | 28.82 | 1303.98 | 1068.96 | 1319.25 | 21.99 | -1.16 |
| (45,30) | 0.982220 | 0.991363 | 0.935407 | -0.92 | 5.00 | 1181.96 | 1014.06 | 1275.14 | 16.56 | -7.31 |
| (50,30) | 0.996160 | 0.991363 | 0.961537 | 0.48 | 3.60 | 1197.78 | 1014.06 | 1281.14 | 18.12 | -6.51 |
| | | | **Average Δ (%)** | **-2.13** | **12.65** | | | **Average Δ (%)** | **19.46** | **-22.48** |
| | | | | | **AR** | | | | | |
| (50,30) | 0.998461 | 0.997958 | 0.881482 | 0.05 | 13.27 | 2740.00 | 2704.00 | 3678.00 | 1.33 | -25.50 |
| (50,40) | 0.996467 | 0.996954 | 0.884995 | -0.05 | 12.60 | 2580.00 | 2568.99 | 3513.00 | 0.43 | -26.56 |
| (55,30) | 0.996308 | 0.996246 | 0.987646 | 0.01 | 0.88 | 2696.00 | 2682.04 | 4020.00 | 0.52 | -32.94 |
| (55,40) | 0.971797 | 0.976736 | 0.948816 | -0.51 | 2.42 | 2002.00 | 2015.16 | 3205.00 | -0.65 | -37.54 |
| (60,30) | 0.994319 | 0.993964 | 0.989493 | 0.04 | 0.49 | 2536.00 | 2090.72 | 2444.00 | 21.30 | 3.76 |
| | | | **Average Δ (%)** | **-0.09** | **5.93** | | | **Average Δ (%)** | **4.59** | **-23.75** |

energy values on average. These results demonstrate that more tuning is necessary for the SA-based HLS method with partial DMR.

Nonetheless, when compared to the other metaheuristic GA-based method (as shown in the SA-PD/GA-SD columns for both reliability and energy values), we observe that SA-PD again outperforms GA-SD for all benchmarks by generating solutions that are, on average, closer to optimal in both reliability and energy values of the final designs. For EWF, SA-PD generates up to 21% more reliable solutions on average while simultaneously achieving an average reduction in energy consumption of about 11%. The highest energy reduction is observed for the DES benchmark of about 38% on average while preserving the final designs' reliability.

All results considered, although the proposed SA-PD still requires additional tuning to generate closer Pareto-optimal solutions to those obtained with ILP-PD in bi-objective optimization tests, it outperforms the GA-based selective duplication method for all benchmarks. For the purpose of fair comparison with GA-SD, our SA-PD was tested using the same experimental setup of 1000 iterations for all benchmarks regardless of its size. Tuning the cooling schedule and number of iterations to the problem size may result in even more optimal solutions when compared to the ILP-PD model.

## 6.4. Effects of Multiple Supply Voltages on Reliability and Energy Consumption

In this set of experiments, we investigate the effects of using multiple supply voltage levels in integrated circuits on their overall reliability and energy consumption. We employ VIs technique in our designs with three voltage levels (high voltage level $v_h$ of 1.2 V, medium voltage level $v_m$ of 1.1 V, and low voltage level $v_l$ of 1.0 V). The experiments for different latency and area constraints are set for each benchmark, and the tests are carried out under different $\alpha$ values using our ILP-FD model.

The reliability and energy results for the DES and FIR benchmarks of full DMR-based solutions for a different number of supply voltages are given in Table 6.16 and Table 6.17, respectively. For the sake of clarity of this section, the tables that show the results for the EWF and AR benchmarks are given in APPENDIX C as Table C.5 and Table C.6, respectively.

From the reliability results, we can observe that the multi-supply voltages do not seem to affect the final design reliability much. For the cases where the only objective is to maximize reliability ($\alpha = 1.0$), it is evident that using multi-supply voltages does not result in any notable increase or decrease in reliability values. For some benchmarks, an increase is

Table 6.16 DES benchmark reliability and energy results of full DMR solutions for a different number of supply voltages.

| | | DES Full DMR Reliability Results | | | | | DES Full DMR Energy Results | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Supply Voltage Levels Used | | | Δ (%) | | Supply Voltage Levels Used | | | Δ (%) | |
| Alpha | (L, A) | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI |
| | (31,20) | 0.999989 | 0.999989 | 0.999989 | 0.00 | 0.00 | 1080.00 | 1080.00 | 1080.00 | 0.00 | 0.00 |
| | (31,30) | 0.999989 | 0.999989 | 0.999989 | 0.00 | 0.00 | 1080.00 | 1080.00 | 1080.00 | 0.00 | 0.00 |
| | (28,30) | 0.999821 | 0.999821 | 0.999821 | 0.00 | 0.00 | 1068.00 | 1068.00 | 1068.00 | 0.00 | 0.00 |
| 1.0 | (28,40) | 0.999821 | 0.999821 | 0.999821 | 0.00 | 0.00 | 1068.00 | 1068.00 | 1068.00 | 0.00 | 0.00 |
| | (25,30) | 0.999653 | 0.999653 | 0.999653 | 0.00 | 0.00 | 1056.00 | 1056.00 | 1056.00 | 0.00 | 0.00 |
| | (25,40) | 0.999653 | 0.999653 | 0.999653 | 0.00 | 0.00 | 1056.00 | 1056.00 | 1056.00 | 0.00 | 0.00 |
| | | | | **Average Δ (%)** | **0.00** | **0.00** | | | **Average Δ (%)** | **0.00** | **0.00** |
| | (31,20) | 0.999989 | 0.999989 | 0.999989 | 0.00 | 0.00 | 1080.00 | 1080.00 | 1080.00 | 0.00 | 0.00 |
| | (31,30) | 0.999989 | 0.999985 | 0.999643 | 0.00 | -0.03 | 1080.00 | 1044.55 | 867.42 | -3.28 | -19.68 |
| | (28,30) | 0.999821 | 0.999644 | 0.999644 | -0.02 | -0.02 | 1068.00 | 1033.98 | 1033.98 | -3.19 | -3.19 |
| 0.5 | (28,40) | 0.999821 | 0.999642 | 0.999644 | -0.02 | -0.02 | 1068.00 | 985.10 | 964.58 | -7.76 | -9.68 |
| | (25,30) | 0.999653 | 0.999644 | 0.999644 | 0.00 | 0.00 | 1056.00 | 1033.98 | 1033.98 | -2.09 | -2.09 |
| | (25,40) | 0.999653 | 0.999650 | 0.999647 | 0.00 | 0.00 | 1056.00 | 1024.22 | 1019.54 | -3.01 | -3.45 |
| | | | | **Average Δ (%)** | **-0.01** | **-0.01** | | | **Average Δ (%)** | **-3.22** | **-6.35** |
| | (31,20) | 0.999989 | 0.999989 | 0.999989 | 0.00 | 0.00 | 1080.00 | 1080.00 | 1080.00 | 0.00 | 0.00 |
| | (31,30) | 0.995198 | 0.995196 | 0.987035 | 0.00 | -0.82 | 1010.00 | 961.12 | 832.60 | -4.84 | -17.56 |
| | (28,30) | 0.995198 | 0.995197 | 0.995198 | 0.00 | 0.00 | 1010.00 | 985.56 | 954.48 | -2.42 | -5.50 |
| 0.0 | (28,40) | 0.995198 | 0.988170 | 0.996279 | -0.71 | 0.11 | 1010.00 | 955.94 | 894.81 | -5.35 | -11.40 |
| | (25,30) | 0.998448 | 0.997241 | 0.997241 | -0.12 | -0.12 | 1017.00 | 1012.41 | 1012.41 | -0.45 | -0.45 |
| | (25,40) | 0.996777 | 0.988171 | 0.988171 | -0.86 | -0.86 | 1014.00 | 980.38 | 980.38 | -3.32 | -3.32 |
| | | | | **Average Δ (%)** | **-0.28** | **-0.28** | | | **Average Δ (%)** | **-2.73** | **-6.37** |

Table 6.17 FIR benchmark reliability and energy results of full DMR solutions for a different number of supply voltages.

| | | FIR Full DMR Reliability Results | | | | | FIR Full DMR Energy Results | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Supply Voltage Levels Used | | | Δ (%) | | Supply Voltage Levels Used | | | Δ (%) | |
| Alpha | (L, A) | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI |
| | (35,30) | 0.998802 | 0.998969 | 0.998969 | 0.02 | 0.02 | 1556.00 | 1568.00 | 1568.00 | 0.77 | 0.77 |
| | (40,30) | 0.998802 | 0.999305 | 0.999305 | 0.05 | 0.05 | 1556.00 | 1592.00 | 1592.00 | 2.31 | 2.31 |
| 1.0 | (45,30) | 0.998802 | 0.999641 | 0.999641 | 0.08 | 0.08 | 1556.00 | 1616.00 | 1616.00 | 3.86 | 3.86 |
| | (50,30) | 0.998802 | 0.999809 | 0.999809 | 0.10 | 0.10 | 1556.00 | 1628.00 | 1628.00 | 4.63 | 4.63 |
| | | | | **Average Δ (%)** | **0.06** | **0.06** | | | **Average Δ (%)** | **2.89** | **2.89** |
| | (35,30) | 0.998969 | 0.998259 | 0.998259 | -0.07 | -0.07 | 1568.00 | 1092.26 | 1092.26 | -30.34 | -30.34 |
| | (40,30) | 0.999305 | 0.998754 | 0.998754 | -0.06 | -0.06 | 1592.00 | 1106.24 | 1106.24 | -30.51 | -30.51 |
| 0.5 | (45,30) | 0.999641 | 0.998918 | 0.998918 | -0.07 | -0.07 | 1616.00 | 1110.90 | 1110.90 | -31.26 | -31.26 |
| | (50,30) | 0.999809 | 0.998918 | 0.998918 | -0.09 | -0.09 | 1628.00 | 1110.90 | 1110.90 | -31.76 | -31.76 |
| | | | | **Average Δ (%)** | **-0.07** | **-0.07** | | | **Average Δ (%)** | **-30.97** | **-30.97** |
| | (35,30) | 0.985674 | 0.992996 | 0.992996 | 0.74 | 0.74 | 1430.00 | 1051.25 | 1051.25 | -26.49 | -26.49 |
| | (40,30) | 0.985674 | 0.991731 | 0.991731 | 0.61 | 0.61 | 1430.00 | 1046.19 | 1046.19 | -26.84 | -26.84 |
| 0.0 | (45,30) | 0.985674 | 0.961597 | 0.961597 | -2.44 | -2.44 | 1430.00 | 1006.44 | 1006.44 | -29.62 | -29.62 |
| | (50,30) | 0.985674 | 0.955639 | 0.955639 | -3.05 | -3.05 | 1430.00 | 1001.98 | 1001.98 | -29.93 | -29.93 |
| | | | | **Average Δ (%)** | **-1.03** | **-1.03** | | | **Average Δ (%)** | **-28.22** | **-28.22** |

present but negligible (e.g., 0.06% for FIR), but that increase in reliability also induces a slight overhead in energy cost (about 2-3%).

Nonetheless, the situation changes when our objective also starts considering energy optimization. For $\alpha = 0.5$, where the objective is to simultaneously optimize both reliability and energy, as well as for $\alpha = 0.0$ when we are only concerned with minimizing energy consumption in the resulting designs, we can observe that having multiple supply voltages (i.e., voltage islands in our case) significantly improves the energy efficiency of the designs with a negligible effect on their reliability. The energy savings can even reach up to 30% on average for the FIR benchmark, while the maximum average decrease in reliability does not exceed $1\%$. For the benchmarks with a large number of nodes and edges, even some small increase in reliability can be observed (e.g., 0.31$\%$ for AR), although the objective was only to minimize energy consumption.

In the final set of experiments on the effect of multi-supply voltages on reliability and energy costs, we changed the values of the parameter $\alpha$ in steps of 0.1, through which we assigned varying weights to the optimization of either reliability or energy costs, and carried out the tests under a varying number of supply voltages: (i) only one supply voltage level (high), (ii) two supply voltage levels (high and low), and (iii) three supply voltage levels (high, medium and low).

In Figures 6.1 and 6.2, the resulting reliability and energy changes are shown for the DES benchmark, respectively.

In Figures 6.3 and 6.4, the resulting reliability and energy changes are shown for the FIR benchmark, respectively.

For the sake of clarity of this section, the figures that show the results for the EWF and AR benchmarks are given in APPENDIX C. Figures C.1 and C.2 show the resulting reliability and energy changes for the EWF benchmark, respectively. Similarly, Figures C.3 and C.4 show the resulting reliability and energy changes for the AR benchmark, respectively.

As the values of the parameter $\alpha$ increase, maximizing reliability is given higher and higher priority. It can be observed from the (a) charts in the given figures that the number of the supply voltage levels does not appear to inhibit the fast convergence towards the optimum solution with the maximized reliability. On the other hand, from the (b) charts, it is evident

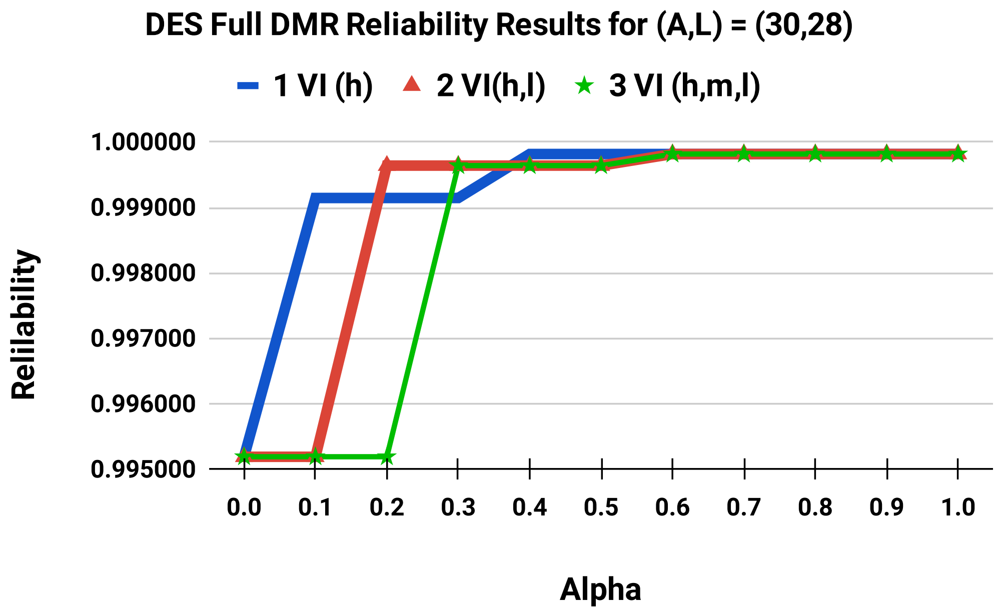


Figure 6.1 Changes in reliability over different $\alpha$ values for DES benchmark ($A = 30, L = 28$) under different numbers of supply voltages.

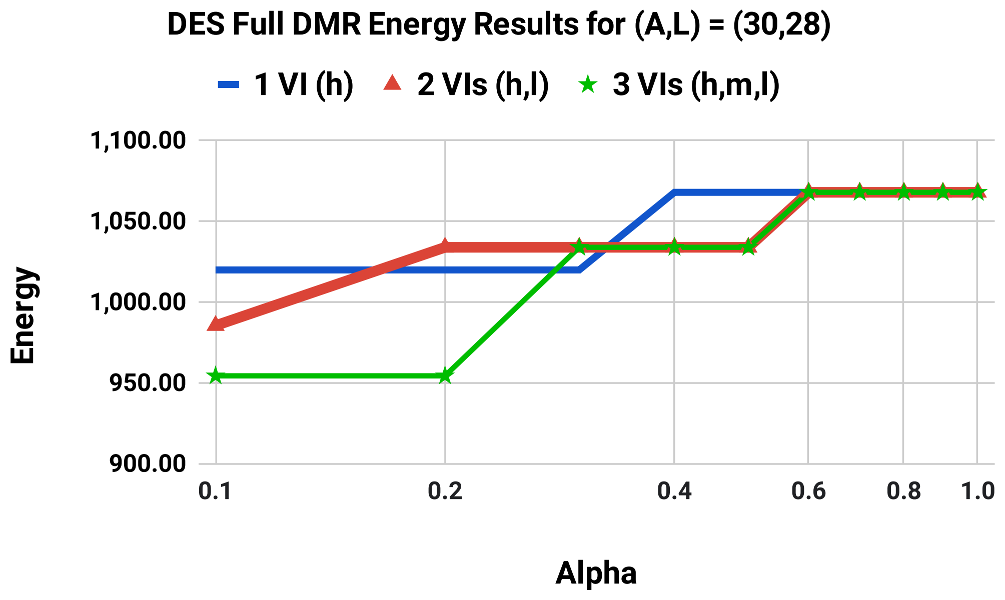


Figure 6.2 Changes in energy over different $\alpha$ values for DES benchmark ($A = 30, L = 28$) under different numbers of supply voltages.

that the more the supply voltage levels are used in a circuit design, the more notable energy savings are possible. Especially for the case when $\alpha = 0.5$ when the equal weight is given to

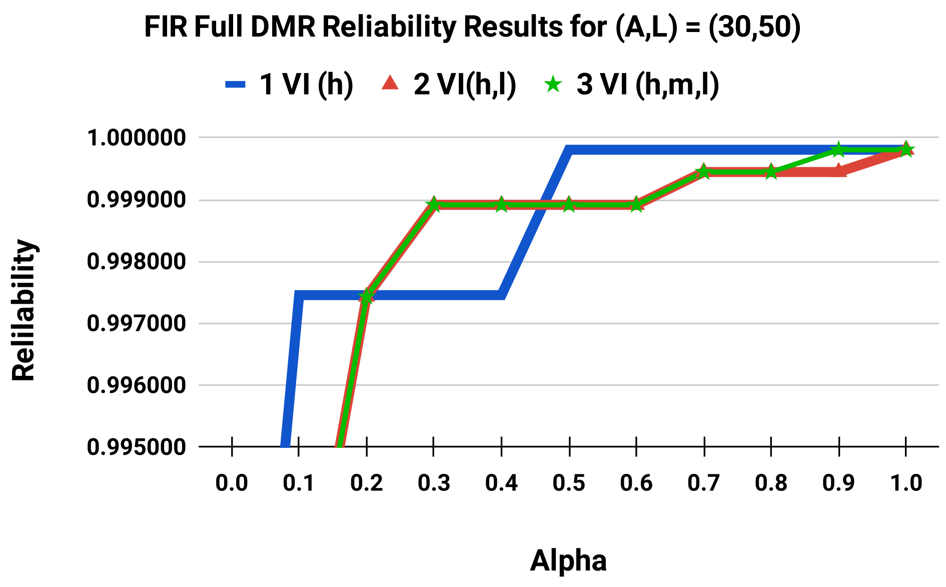


Figure 6.3 Changes in reliability over different $\alpha$ values for FIR benchmark $(A = 30, L = 50)$ under different numbers of supply voltages.

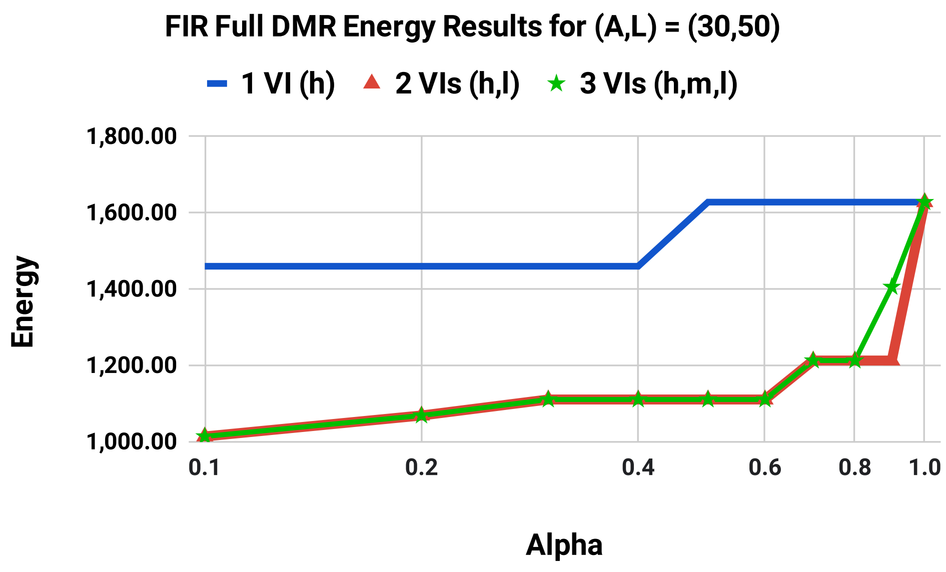


Figure 6.4 Changes in energy over different $\alpha$ values for FIR benchmark $(A = 30, L = 50)$ under different numbers of supply voltages.

the optimization of both reliability and energy, we observe that the loss in terms of reliability is negligible, while the gain in energy savings is meaningful.

These results demonstrate the effectiveness of multi-supply voltage techniques for reliability and energy-oriented designs. Even using only two different supply voltage levels is enough to achieve a significant reduction in energy costs with a negligible negative effect on the system reliability.

## 6.5. Execution Time Analysis

ILP-based optimization methods have a high computational complexity resulting in long execution times for problems with a large solution space, such as task scheduling and resource allocation, especially if those problems must be solved under specific constraints. Such problems are known as NP-hard problems. For complex applications whose DAGs consist of a large number of tasks, ILP models with constraints usually take too long to produce the optimum solution, and they are computationally impractical, as their running times increase exponentially with the increase in the number of variables. In our proposed models, we introduce further complexity by employing multiple supply voltages, which additionally expands the solution space. Nonetheless, ILP-based problem formulations provide optimal results that can be used for testing other heuristic or metaheuristic methods designed to solve the same problems but within more acceptable running times.

If we take $n$ to represent the number of nodes in a given dataflow graph, $r$ the number of resources in the resource library, and $v$ the number of supply voltage levels, to search through the entire solution space, it is necessary to explore $(rv)^n$ possible allocations for designs that do not involve any duplication alone, because any resource can be used as many times as necessary, and because all of them have different latency, energy consumption and latency properties under varying supply voltages (i.e., practically, there are $r \times v$ resources in total). The solution space increases even further when DMR is employed because, in full DMR, all tasks must also be duplicated, which effectively increases the number of active tasks to be scheduled and assigned a resource to $2n$.

Figure 6.5 presents the comparison between the average execution times of ILP, GA, and SA-based HLS methods without duplication for a varying number of benchmark nodes under two voltage levels. It can be observed that the execution time of the ILP-ND method starts growing exponentially for the benchmarks with over 20 nodes, while the average execution times of the GA and SA-based methods increase only linearly with respect to the benchmark size. This demonstrates how metaheuristic methods are much more practical approaches for designing complex circuits with a large number of nodes.

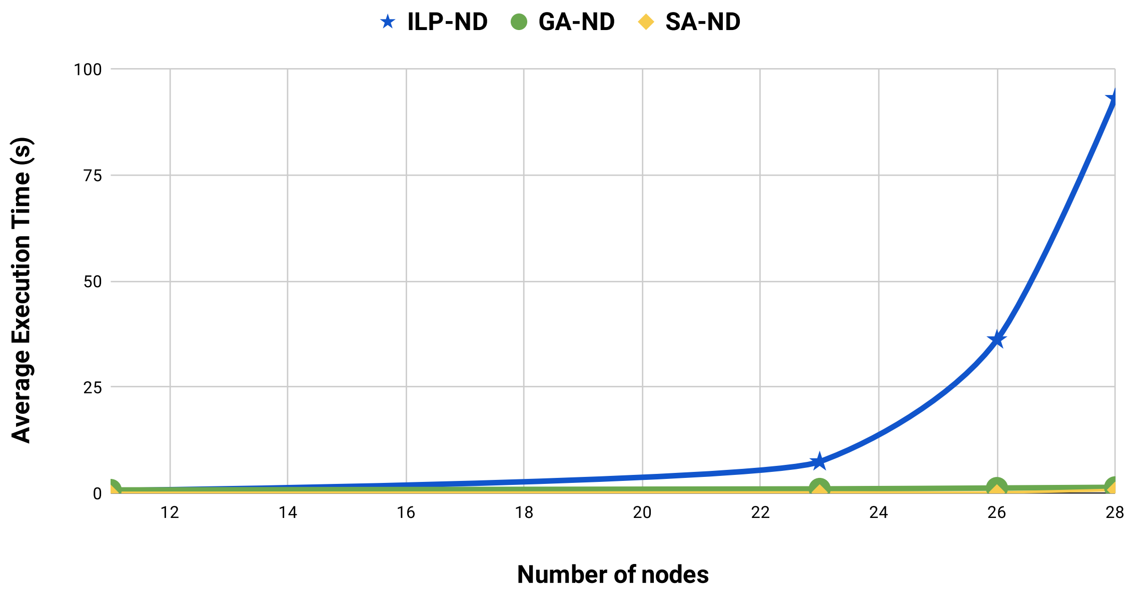


Figure 6.5 Average execution times of ILP, GA, and SA-based HLS methods without duplication for varying number of benchmark nodes.

Figure 6.6 presents the comparison between the average execution times of only GA and SA-based HLS methods without duplication for a varying number of benchmark nodes under two voltage levels to show more clearly how the SA-based method outperforms the GA-based HLS method when no duplication is used.

The increase in solution search space complexity when DMR is employed can be observed from Figure 6.7 in which we compare the average running times of our ILP models against the performance of the no-duplication ILP model (denoted as ILP-ND) and each other, for different benchmarks with the increasing varying number of nodes and edges.

It is important to note that some of our tests for the benchmarks with a large number of nodes and edges (AR and EWF) were running too long without providing the 100% optimal solutions with the ILP-PD-C model where additional constraints of either reliability or

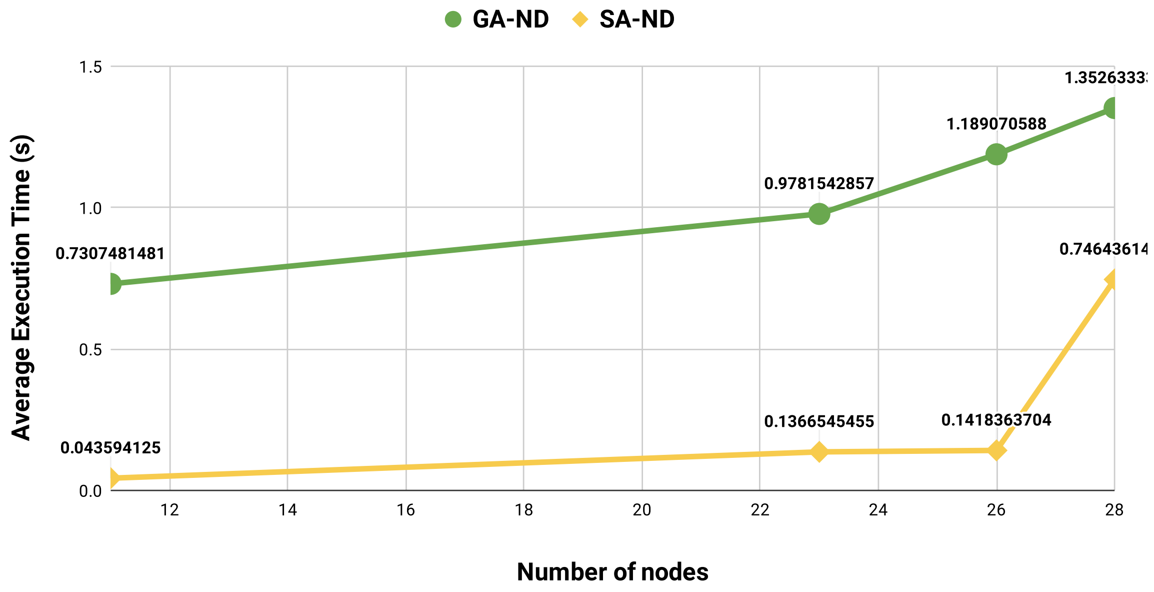


Figure 6.6 Average execution times of GA and SA-based HLS methods without duplication for varying number of benchmark nodes.

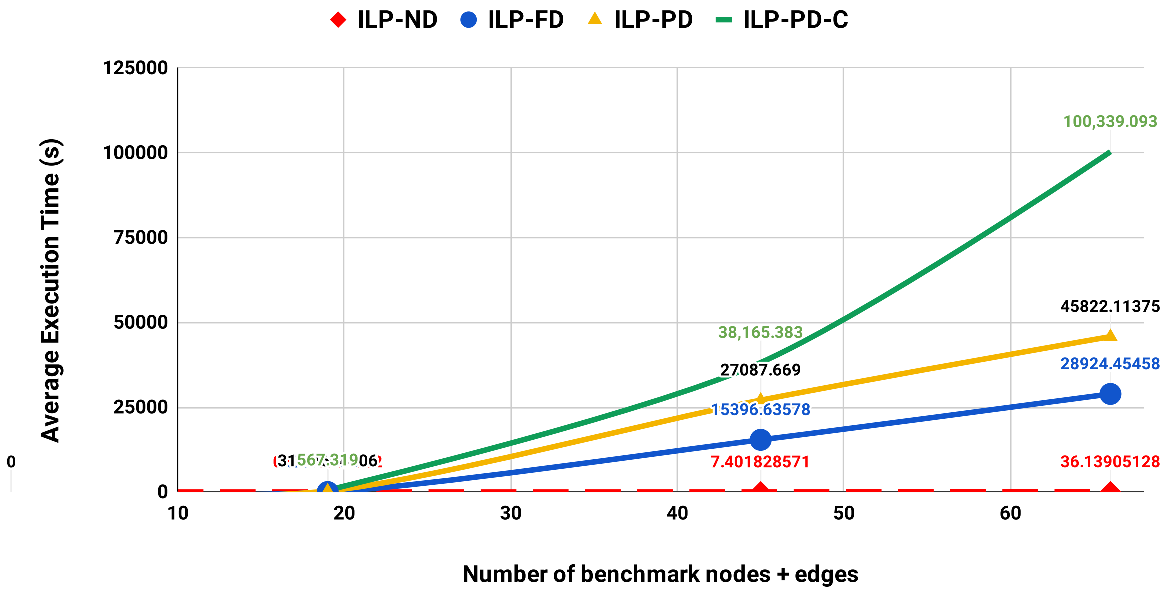


Figure 6.7 Average execution times of ILP models for varying number of benchmark nodes and edges.

energy were added. Such test cases were terminated after eight hours of execution, and the current best solution was taken as the final result. Even so, the running times of partial duplication-based models surpass the running time of the full duplication-based model. The reason for this is that partial DMR expands the solution space by far compared to the full DMR approach because, in partial DMR, any resource may or may not be duplicated. This situation exponentially complicates the solution space since, for the applications with $n$ tasks in their DAG, there are $2^n - 1$ possible configurations of a partially-duplicated solution design.

Figure 6.8 presents the comparison between the average execution times of SA-ND and SA-PD HLS methods for a varying number of benchmark nodes under two voltage levels. The aim is to show how the execution time of the SA-based partial duplication method does not start growing exponentially compared to no duplication approach as is the case with ILP-based models.

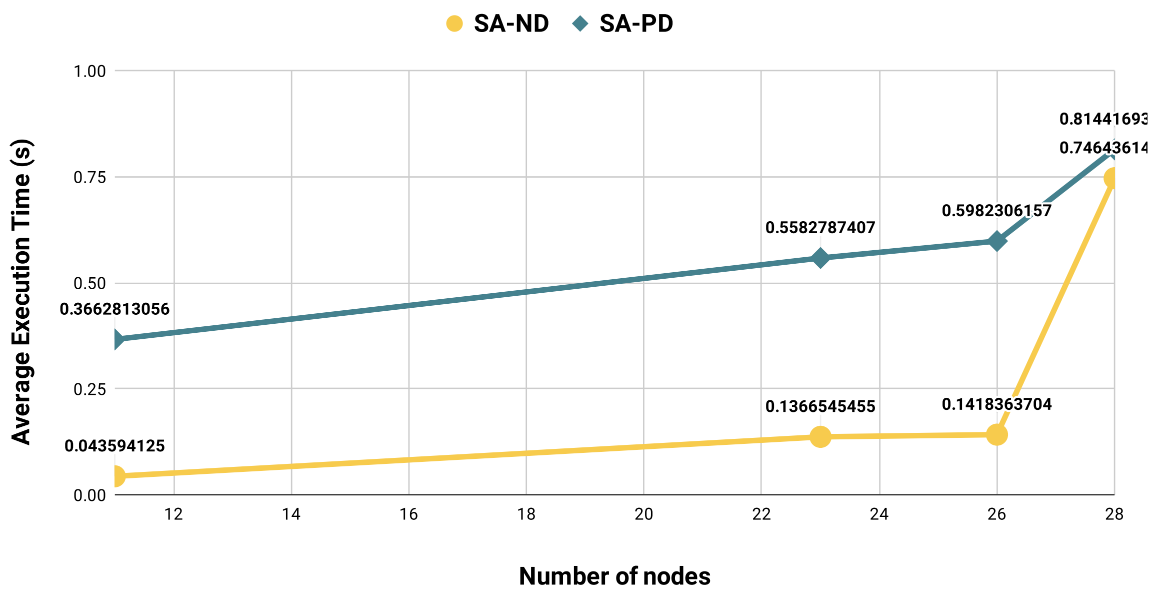


Figure 6.8 Average execution times of the SA-ND and SA-PD methods for varying number of benchmark nodes.

The impractical running times of the ILP-based HLS models demonstrate the necessity for other heuristic and/or metaheuristic methods that will provide optimal or near-optimal solutions in practical running times. Nevertheless, these models are necessary as they provide the optimum solutions we can use to test the performance of other methods that tackle the same problems. Therefore, in this study, we proposed a metaheuristic HLS method that produces optimal or near-optimal solutions in much shorter and more practical running times.

# 7. CONCLUSION

Continuously decreasing transistor technology sizes have enabled much denser packaging of electronic components on chips. This increase in circuit densities has positively affected the costs and area of integrated circuits. However, it has consequently given rise to new issues and challenges in the integrated circuit design process, including higher vulnerability to soft errors due to unavoidable radiation effects, lower supply and threshold voltage levels, etc. Thus, novel reliability-oriented design solutions have become a necessity. Modular hardware redundancy is a popular method for boosting the reliability of a system, but it comes at the cost of increasing the overall area and energy expenditure. Energy reduction methods such as DVS and VIs may be employed to tackle high energy costs; nevertheless, reduction in energy costs, in turn, will also negatively affect a circuit's reliability and latency.

This study considers both energy and reliability optimization metrics during the HLS design process for integrated systems with additional modular redundancy while satisfying both area and latency requirements. We employ integer linear programming as our mathematical optimization approach of choice to propose an ILP-based model without modular redundancy and two ILP-based model formulations for systems with both partially and fully duplicated components to achieve improved reliability with a minimum increase in the resulting energy consumption, execution time, and area. VIs are employed as the energy reduction method of choice, and their effect on the system performance, reliability, and energy costs is discussed in detail.

ILP-based optimization methods have a high computational complexity resulting in long execution times since they perform the search over the entire solution space. These methods usually take too long to produce the optimal solution for complex applications with a large number of operations. Nevertheless, they give optimal results which can be used for testing the performance of other heuristic or metaheuristic methods designed to solve the same problems. Therefore, we also propose two metaheuristic methods based on a simulated annealing technique that produces optimal or near-optimal solutions in much shorter and more reasonable running times.

The proposed ILP models generate optimal results and outperform other metaheuristic methods for the relatively smaller-sized benchmarks for which they could finish execution in practical running times. When the main objective is to maximize reliability, the proposed full DMR-based ILP model is a good choice as it generates the designs with the highest possible reliability for the given area and latency constraints. When the goal is to obtain more energy-aware designs, the proposed partial DMR-based ILP model with constraints generates the most desirable solutions overall, especially when the objective is to optimize one of the parameters disregarding the other. Finally, when optimizing both reliability and energy consumption at the same time, the proposed partial DMR-based ILP model generates far superior solutions to other proposed methods.

The results obtained from the ILP models were used to test the performance of the proposed metaheuristic SA-based methods that tackle the same problem of optimized HLS. All results considered, the proposed SA-based HLS methods could generate better solutions for all benchmarks compared to the other GA-based HLS methods. The proposed SA-based HLS method without duplication was able to generate optimal or near-optimal solutions in almost all cases, except for some edge cases with the tight area and latency constraints for which it needed additional tuning. Moreover, the proposed SA-based HLS method that employs partial DMR for improved reliability also outperforms the GA-based selective duplication method for all benchmarks. Nevertheless, in bi-objective optimization tests, it could not find closely Pareto-optimal solutions for all test cases, which may be caused by the experimental setup chosen for the purpose of fair comparison with the GA-based method.

Furthermore, the experiments performed on the effects of multiple supply voltages on the reliability and energy consumption of digital circuits demonstrated the effectiveness of the approach. Using multiple supply voltages facilitated a significant reduction in energy costs with a negligible negative effect on the circuit reliability.

Overall, the results show the necessity for more research on this problem to propose even more efficient metaheuristic HLS methods that employ modular redundancy for designs that operate under multiple supply voltages. Furthermore, in this study, we only focused on data-flow intensive operations while disregarding memory considerations. Incorporating memory-related design considerations into multi-objective HLS methods under multiple constraints for designs operated under multiple supply voltages is yet another area for future research. The ILP formulations presented in this study can be used for different optimization problems that consider area, latency, energy consumption, and reliability. For example, optimizing the area for a fully duplicated circuit under latency constraints can be easily incorporated into our ILP model.

This research aimed to propose efficient HLS methods for reliability and energy-oriented designs, which will hopefully contribute to closing the productivity and quality gap between HLS and RTL design flows. Moreover, considering the importance of reliability considerations for mission-critical and safety-critical systems, we hope the outcomes of this study will also contribute to the domain of reliable and energy-efficient hardware design.

# REFERENCES

[1] M.H. Na, D. Jang, R. Baert, S. Sarkar, S. Patli, O. Zografos, B. Chehab, A. Spessot, G. Sisto, P. Schuddinck, H. Mertens, Y. Oniki, G. Hellings, E. Dentoni Litta, J. Ryckaert, and N. Horiguchi. Disruptive technology elements, and rapid and accurate block-level performance evaluation for 3nm and beyond. In *2021 5th IEEE Electron Devices Technology Manufacturing Conference (EDTM)*, pages 1–3. **2021**. doi:10.1109/EDTM50988.2021.9420975.

[2] Meng Wang, Yabin Sun, Xiaojin Li, Yanling Shi, Shaojian Hu, Enming Shang, and Shoumian Chen. Design technology co-optimization for 3 nm gate-all-around nanosheet fets. In *2020 IEEE 15th International Conference on Solid-State Integrated Circuit Technology (ICSICT)*, pages 1–3. **2020**. doi:10.1109/ICSICT49897.2020.9278197.

[3] A. Dixit and A. Wood. The impact of new technology on soft error rates. In *2011 International Reliability Physics Symposium*, pages 5B.4.1–5B.4.7. **2011**. ISSN 1541-7026. doi:10.1109/IRPS.2011.5784522.

[4] Vikas Chandra and Robert Aitken. Impact of voltage scaling on nanoscale sram reliability. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE ’09, page 387–392. European Design and Automation Association, Leuven, BEL, **2009**. ISBN 9783981080155. doi:10.5555/1874620.1874713.

[5] F. Dabiri, N. Amini, M. Rofouei, and M. Sarrafzadeh. Reliability-aware optimization for dvs-enabled real-time embedded systems. In *9th International Symposium on Quality Electronic Design (isqed 2008)*, pages 780–783. **2008**. ISSN 1948-3295. doi:10.1109/ISQED.2008.4479837.

[6] Giovanni De Micheli. High-level synthesis of digital circuits. volume 37 of *Advances in Computers*, pages 207–283. Elsevier, **1993**. doi:https://doi.org/10.1016/S0065-2458(08)60406-4.

[7] Zhiru Zhang, Deming Chen, Steve Dai, and Keith Campbell. High-level synthesis for low-power design. *IPSJ Transactions on System LSI Design Methodology*, 8:12–25, **2015**.

[8] Benjamin Carrion Schafer and Zi Wang. High-level synthesis design space exploration: Past, present, and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2628–2639, **2020**. doi:10.1109/TCAD.2019.2943570.

[9] Sakari Lahti, Panu Sjövall, Jarno Vanne, and Timo D. Hämäläinen. Are we there yet? a study on the state of high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(5):898–911, **2019**. doi:10.1109/TCAD.2018.2834439.

[10] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, **2021**. doi:10.1017/9781108980647.

[11] Zaineb Chelly Dagdia and Miroslav Mirchev. Chapter 15 - when evolutionary computing meets astro- and geoinformatics. In Petr Škoda and Fathalrahman Adam, editors, *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pages 283–306. Elsevier, **2020**. ISBN 978-0-12-819154-5. doi:https://doi.org/10.1016/B978-0-12-819154-5.00026-6.

[12] Xin-She Yang. Chapter 1 - introduction to algorithms. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 1–21. Elsevier, Oxford, **2014**. ISBN 978-0-12-416743-8. doi:https://doi.org/10.1016/B978-0-12-416743-8.00001-4.

[13] S. Brisset and F. Gillon. 4 - approaches for multi-objective optimization in the ecodesign of electric systems. In Jean-Luc Bessède, editor, *Eco-Friendly*

*Innovation in Electricity Transmission and Distribution Networks*, pages 83–97. Woodhead Publishing, Oxford, **2015**. ISBN 978-1-78242-010-1. doi:https://doi.org/10.1016/B978-1-78242-010-1.00004-5.

[14] Xin-She Yang. Chapter 14 - multi-objective optimization. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 197–211. Elsevier, Oxford, **2014**. ISBN 978-0-12-416743-8. doi:https://doi.org/10.1016/B978-0-12-416743-8.00014-2.

[15] Chung-Yang (Ric) Huang, Chao-Yue Lai, and Kwang-Ting (Tim) Cheng. Chapter 4 - fundamentals of algorithms. In Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting (Tim) Cheng, editors, *Electronic Design Automation*, pages 173–234. Morgan Kaufmann, Boston, **2009**. ISBN 978-0-12-374364-0. doi:https://doi.org/10.1016/B978-0-12-374364-0.50011-4.

[16] Katta G. Murty. *Linear Programming*. Wiley, **1983**.

[17] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, **2003**. ISSN 0360-0300. doi:10.1145/937503.937505.

[18] Xin-She Yang. Chapter 3 - random walks and optimization. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 45–65. Elsevier, Oxford, **2014**. ISBN 978-0-12-416743-8. doi:https://doi.org/10.1016/B978-0-12-416743-8.00003-8.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, **1983**. doi:10.1126/science.220.4598.671.

[20] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, **1985**. ISSN 1573-2878. doi:10.1007/BF00940812.

[21] Emile Aarts, Jan Korst, and Wil Michiels. *Simulated Annealing*, pages 187–210. Springer US, Boston, MA, **2005**. ISBN 978-0-387-28356-2. doi:10.1007/0-387-28356-0_7.

[22] Xin-She Yang. Chapter 4 - simulated annealing. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 67–75. Elsevier, Oxford, **2014**. ISBN 978-0-12-416743-8. doi:https://doi.org/10.1016/B978-0-12-416743-8.00004-X.

[23] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations Research*, 37(6):865–892, **1989**. doi:10.1287/opre.37.6.865.

[24] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, **1991**. ISSN 0030364X, 15265463.

[25] Steve R. White. Concepts of scale in simulated annealing. *AIP Conference Proceedings*, 122(1):261–270, **1984**. doi:10.1063/1.34823.

[26] Walid Ben-Ameur. Computing the initial temperature of simulated annealing. *Computational Optimization and Applications*, 29(3):369–385, **2004**. ISSN 1573-2894. doi:10.1023/B:COAP.0000044187.23143.bd.

[27] Kathryn A. Dowsland and Jonathan M. Thompson. *Simulated Annealing*, pages 1623–1655. Springer Berlin Heidelberg, Berlin, Heidelberg, **2012**. ISBN 978-3-540-92910-9. doi:10.1007/978-3-540-92910-9_49.

[28] M. Locatelli. Simulated annealing algorithms for continuous global optimization: Convergence conditions. *Journal of Optimization Theory and Applications*, 104(1):121–133, **2000**. ISSN 1573-2878. doi:10.1023/A:1004680806815.

[29] Peter Salamon, Paolo Sibani, and Richard Frost. *Selecting the Schedule*, chapter 13, pages 89–97. Society for Industrial and Applied Mathematics, Philadelphia, Pa., **2002**. ISBN 978-0-89871-508-8. doi:10.1137/1.9780898718300.ch13.

[30] Matheus Ferreira Pontes, Clayton Farias, Rafael Schvittz, Paulo Butzen, and Leomar da Rosa Jr. Survey on reliability estimation in digital circuits. *Journal of Integrated Circuits and Systems*, 16(3):1–11, **2021**. doi:10.29292/jics.v16i3.568.

[31] Jaroslav Menčík. Reliability of systems. In Jaroslav Mencik, editor, *Concise Reliability for Engineers*, chapter 5. IntechOpen, Rijeka, **2016**. doi:10.5772/62358.

[32] Zhen Li, Junfeng Tian, and Pengyuan Zhao. Software reliability estimate with duplicated components based on connection structure. *Cybernetics and Information Technologies*, 14(3):3–13, **2014**. doi:10.2478/cait-2014-0028.

[33] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Yuan Xie. Reliability-centric high-level synthesis. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, DATE '05, page 1258–1263. IEEE Computer Society, USA, **2005**. ISBN 0769522882. doi:10.1109/DATE.2005.258.

[34] D.E. Lackey, P.S. Zuchowski, T.R. Bednar, D.W. Stout, S.W. Gould, and J.M. Cohn. Managing power and performance for system-on-chip designs using voltage islands. In *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.*, pages 195–202. **2002**. doi:10.1109/ICCAD.2002.1167534.

[35] Farshad Firouzi, Mostafa E Salehi, Fan Wang, and Sied Mehdi Fakhraie. An accurate model for soft error rate estimation considering dynamic voltage and

frequency scaling effects. *Microelectronics Reliability*, 51(2):460–467, **2011**. ISSN 0026-2714. doi:10.1016/j.microrel.2010.08.016.

[36] S. Tosun, O. Ozturk, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W.-L. Hung. An ilp formulation for reliability-oriented high-level synthesis. In *Sixth international symposium on quality electronic design (isqed'05)*, pages 364–369. **2005**. doi:10.1109/ISQED.2005.15.

[37] Siavash Es'haghi and Mohammad Eshghi. Lifetime-aware scheduling in high level synthesis. *Microelectronics Reliability*, 91:86–97, **2018**. ISSN 0026-2714. doi:10.1016/j.microrel.2018.06.016.

[38] Liang Chen, Mojtaba Ebrahimi, and Mehdi B. Tahoori. Reliability-aware resource allocation and binding in high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 21(2), **2016**. ISSN 1084-4309. doi:10.1145/2839300.

[39] Yuko Hara-Azumi and Hiroyuki Tomiyama. Cost-efficient scheduling in high-level synthesis for soft-error vulnerability mitigation. In *International Symposium on Quality Electronic Design (ISQED)*, pages 502–507. IEEE, **2013**. doi:10.1109/ISQED.2013.6523658.

[40] Necati Aras and Arda Yurdakul. A new multi-objective mathematical model for the high-level synthesis of integrated circuits. *Applied Mathematical Modelling*, 40(3):2274–2290, **2016**. ISSN 0307-904X. doi:10.1016/j.apm.2015.09.061.

[41] Yuko Hara-Azumit and Hiroyuki Tomiyama. Clock-constrained simultaneous allocation and binding for multiplexer optimization in high-level synthesis. In *17th Asia and South Pacific Design Automation Conference*, pages 251–256. **2012**. doi:10.1109/ASPDAC.2012.6164954.

[42] K. Ito, L.E. Lucke, and K.K. Parhi. Ilp-based cost-optimal dsp synthesis with module selection and data format conversion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):582–594, **1998**. doi:10.1109/92.736132.

[43] Keisuke INOUE and Mineo KANEKO. Dual-edge-triggered flip-flop-based high-level synthesis with programmable duty cycle. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E96.A(12):2689–2697, **2013**. doi:10.1587/transfun.E96.A.2689.

[44] Kartikey Mittal, Arpit Joshi, and Madhu Mutyam. Timing variation-aware scheduling and resource binding in high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 16(4), **2011**. ISSN 1084-4309. doi:10.1145/2003695.2003700.

[45] Insup Shin, Seungwhun Paik, Dongwan Shin, and Youngsoo Shin. Hls-dv: A high-level synthesis framework for dual-vdd architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(4):593–604, **2012**. doi:10.1109/TVLSI.2011.2122310.

[46] Zhen Zhao, Jinian Bian, Zhipeng Liu, Yunfeng Wang, and Kang Zhao. High level synthesis with multiple supply voltages for energy and combined peak power minimization. In *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems*, pages 864–867. **2006**. doi:10.1109/APCCAS.2006.342178.

[47] Shih-Hsu Huang and Chun-Hua Cheng. An ilp approach to the simultaneous application of operation scheduling and power management. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E91.A(1):375–382, **2008**. doi:10.1093/ietfec/e91-a.1.375.

[48] Wen-Tsong Shiue. High level synthesis for peak power minimization using ilp. In *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 103–112. **2000**. doi:10.1109/ASAP.2000.862382.

[49] Zong-Han Xie, Shih-Hsu Huang, and Chun-Hua Cheng. Utilizing power management and timing slack for low power in high-level synthesis. In *2018*

*IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pages 1–2. **2018**. doi:10.1109/ICCE-China.2018.8448584.

[50] S. Devadas and A.R. Newton. Algorithms for hardware allocation in data path synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(7):768–781, **1989**. doi:10.1109/43.31534.

[51] Maria Abi Saad and Iyad Ouaiss. Priority-driven area optimization in high-level synthesis. *Journal of Circuits, Systems and Computers*, 20(06):1131–1163, **2011**. doi:10.1142/S0218126611007803.

[52] Gabriel Caffarena, Juan A. Lopez, Carlos Carreras, and Octavio Nieto-Taladriz. High-level synthesis of multiple word-length dsp algorithms using heterogeneous-resource fpgas. In *2006 International Conference on Field Programmable Logic and Applications*, pages 1–4. **2006**. doi:10.1109/FPL.2006.311288.

[53] J.A. Nestor and G. Krishnamoorthy. Salsa: a new approach to scheduling with timing constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(8):1107–1122, **1993**. doi:10.1109/43.238604.

[54] W.-L. Hung, Xiaoxia Wu, and Yuan Xie. Guaranteeing performance yield in high-level synthesis. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD ’06, page 303–309. Association for Computing Machinery, New York, NY, USA, **2006**. ISBN 1595933891. doi:10.1145/1233501.1233561.

[55] M Akil. High-level synthesis based upon dependence graph for multi-fpga. *INFORMACIJE MIDEM-JOURNAL OF MICROELECTRONICS ELECTRONIC COMPONENTS AND MATERIALS*, 33(4):267–275, **2003**. ISSN 0352-9045.

[56] A. Doboli. Integrated hardware-software co-synthesis and high-level synthesis for design of embedded systems under power and latency constraints. In

*Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, pages 612–619. **2001**. doi:10.1109/DATE.2001.915087.

[57] C.P. Ravikumar, S. Gupta, and A. Jajoo. Synthesis of testable rtl designs. In *Proceedings Eleventh International Conference on VLSI Design*, pages 187–192. **1998**. doi:10.1109/ICVD.1998.646600.

[58] J.C. Alves and J.S. Matos. A simulated annealing approach for high-level synthesis with reconfigurable functional units. In *38th Midwest Symposium on Circuits and Systems. Proceedings*, volume 1, pages 314–317 vol.1. **1995**. doi:10.1109/MWSCAS.1995.504440.

[59] Yiheng Gao and Benjamin Carrion Schafer. Effective high-level synthesis design space exploration through a novel cost function formulation. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. **2021**. doi:10.1109/ISCAS51556.2021.9401684.

[60] Zi Wang and Benjamin Carrion Schafer. Machine leaming to set meta-heuristic specific parameters for high-level synthesis design space exploration. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. **2020**. doi:10.1109/DAC18072.2020.9218674.

[61] Anushree Mahapatra and Benjamin Carrion Schafer. Machine-learning based simulated annealer method for high level synthesis design space exploration. In *Proceedings of the 2014 Electronic System Level Synthesis Conference (ESLsyn)*, pages 1–6. **2014**. doi:10.1109/ESLsyn.2014.6850383.

[62] Deming Chen, J. Cong, and Yiping Fan. Low-power high-level synthesis for fpga architectures. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED ’03.*, pages 134–139. **2003**. doi:10.1109/LPE.2003.1231849.

[63] Deming Chen, Jason Cong, Yiping Fan, and Lu Wan. Lopass: A low-power architectural synthesis system for fpgas with interconnect estimation and

optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(4):564–577, **2010**. doi:10.1109/TVLSI.2009.2013353.

[64] J.T. Kao and A.P. Chandrakasan. Dual-threshold voltage techniques for low-power digital circuits. *IEEE Journal of Solid-State Circuits*, 35(7):1009–1018, **2000**. doi:10.1109/4.848210.

[65] Nan WANG, Song CHEN, Wei ZHONG, Nan LIU, and Takeshi YOSHIMURA. Mobility overlap-removal-based leakage power and register-aware scheduling in high-level synthesis. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E97.A(8):1709–1719, **2014**. doi:10.1587/transfun.E97.A.1709.

[66] S.P. Mohanty, Ramakrishna Velagapudi, and E. Kougianos. Physical-aware simulated annealing optimization of gate leakage in nanoscale datapath circuits. In *Proceedings of the Design Automation & Test in Europe Conference*, volume 1, pages 6 pp.–. **2006**. doi:10.1109/DATE.2006.244046.

[67] D. Dal, D. Kutagulla, A. Nunez, and N. Mansouri. Power islands: a high-level synthesis technique for reducing spurious switching activity and leakage. In *48th Midwest Symposium on Circuits and Systems, 2005.*, pages 1875–1879 Vol. 2. **2005**. doi:10.1109/MWSCAS.2005.1594490.

[68] Elie Elaaraj and Iyad Ouaiss. A novel register-binding approach to reduce spurious switching activity in high-level synthesis. *Journal of Circuits, Systems and Computers*, 20(05):943–973, **2011**. doi:10.1142/S0218126611007700.

[69] Vyas Krishnan and Srinivas Katkoori. Tabs: Temperature-aware layout-driven behavioral synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(12):1649–1659, **2010**. doi:10.1109/TVLSI.2009.2026047.

[70] Hariharan Sankaran and Srinivas Katkoori. Floorplan driven high level synthesis for crosstalk noise minimization in macro-cell based designs. In *2009 IEEE*

*Computer Society Annual Symposium on VLSI*, pages 274–279. **2009**. doi:10.1109/ISVLSI.2009.59.

[71] Hariharan Sankaran and Srinivas Katkoori. Simultaneous scheduling, allocation, binding, re-ordering, and encoding for crosstalk pattern minimization during high–level synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(2):217–226, **2011**. doi:10.1109/TVLSI.2009.2031864.

[72] Junhua Wu, Chunmei Ma, and Baogui Huang. Congestion aware high level synthesis combined with floorplanning. In *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, volume 2, pages 935–938. **2008**. doi:10.1109/PACIIA.2008.205.

[73] François S. Verdier and Bertrand Zavidovique. A high level synthesis system for vlsi image processing applications. *VLSI Design*, 7:095421, **1998**. ISSN 1065-514X. doi:10.1155/1998/95421.

[74] A.A. Duncan, D.C. Hendry, and P. Gray. An overview of the cobra-abs high level synthesis system for multi-fpga systems. In *Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No.98TB100251)*, pages 106–115. **1998**. doi:10.1109/FPGA.1998.707888.

[75] A.A. Duncan, D.C. Hendry, and P. Gray. The cobra-abs high-level synthesis system for multi-fpga custom computing machines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):218–223, **2001**. doi:10.1109/92.920837.

[76] André Flores dos Santos, Lucas Antunes Tambara, and Fernanda Lima Kastensmidt. Evaluating the efficiency of using tmr in the high-level synthesis design flow of sram-based fpga. In *2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS)*, pages 1–4. **2017**. doi:10.1109/LASCAS.2017.7948064.

[77] André Flores dos Santos, Lucas Antunes Tambara, Fabio Benevenuti, Jorge Tonfat, and Fernanda Lima Kastensmidt. Applying tmr in hardware accelerators generated by high-level synthesis design flow for mitigating multiple bit upsets in sram-based fpgas. In Stephan Wong, Antonio Carlos Beck, Koen Bertels, and Luigi Carro, editors, *Applied Reconfigurable Computing*, pages 202–213. Springer International Publishing, Cham, **2017**. ISBN 978-3-319-56258-2. doi:10.1007/978-3-319-56258-2_18.

[78] Zhiqi Zhu, Farah Naz Taher, and Benjamin Carrion Schafer. Exploring design trade-offs in fault-tolerant behavioral hardware accelerators. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, page 291–294. Association for Computing Machinery, New York, NY, USA, **2019**. ISBN 9781450362528. doi:10.1145/3299874.3318020.

[79] Deepak Kachave and Anirban Sengupta. Integrating physical level design and high level synthesis for simultaneous multi-cycle transient and multiple transient fault resiliency of application specific datapath processors. *Microelectronics Reliability*, 60(C):141–152, **2016**. doi:10.1016/j.microrel.2016.03.006.

[80] David Wilson, Aniruddha Shastri, and Greg Stitt. A high-level synthesis scheduling and binding heuristic for fpga fault tolerance. *International Journal of Reconfigurable Computing*, 2017:5419767, **2017**. ISSN 1687-7195. doi:10.1155/2017/5419767.

[81] Aniruddha Shastri, Greg Stitt, and Eduardo Riccio. A scheduling and binding heuristic for high-level synthesis of fault-tolerant fpga applications. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 202–209. **2015**. doi:10.1109/ASAP.2015.7245735.

[82] Xiang Chen, Wenhui Yang, Ming Zhao, and Jing Wang. Hls-based sensitivity-inductive soft error mitigation for satellite communication systems. In *2016 IEEE 22nd International Symposium on On-Line Testing and Robust*

*System Design (IOLTS)*, pages 143–148. **2016**. doi:10.1109/IOLTS.2016.7604688.

[83] Cong Hao, Song Chen, and Takeshi Yoshimura. Network simplex method based multiple voltage scheduling in power-efficient high-level synthesis. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 237–242. **2013**. doi:10.1109/ASPDAC.2013.6509602.

[84] Shih-Hsu Huang, Wen-Pin Tu, and Bing-Hung Li. High-level synthesis for minimum-area low-power clock gating. *Journal of Information Science and Engineering*, 28(5):971–988, **2012**. ISSN 1016-2364.

[85] Michael Glaß, Martin Lukasiewycz, Thilo Streichert, Christian Haubelt, and Jürgen Teich. Reliability-aware system synthesis. *2008 Design, Automation and Test in Europe*, 0:141–148, **2007**. doi:10.1109/DATE.2007.364626.

[86] Tomoo Inoue, Hayato Henmi, Yuki Yoshikawa, and Hideyuki Ichihara. High-level synthesis for multi-cycle transient fault tolerant datapaths. In *2011 IEEE 17th International On-Line Testing Symposium*, pages 13–18. **2011**. doi:10.1109/IOLTS.2011.5993804.

[87] Aiman H. El-Maleh and Khaled A. K. Daud. Simulation-based method for synthesizing soft error tolerant combinational circuits. *IEEE Transactions on Reliability*, 64(3):935–948, **2015**. doi:10.1109/TR.2015.2440234.

[88] Suleyman Tosun and Tohid Taghizad Gogjeh Yaran. Genetic algorithm-based reliability optimization for high-level synthesis. *Journal of Circuits, Systems and Computers*, 28(03):1950039, **2019**. doi:10.1142/S0218126619500397.

[89] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No.98TH8379)*, pages 197–202. **1998**. doi:10.1145/280756.280894.

[90] Ruibin Xu, Daniel Mossé, and Rami Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Transactions on Computer Systems (TOCS)*, 25(4):9, **2007**. ISSN 0734-2071. doi:10.1145/1314299.1314300.

[91] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, page 2–es. USENIX Association, USA, **1994**. doi:10.5555/1267638.1267640.

[92] Priyardarsan Patra, Nagarajan Kougianos, Elias Ranganathan, and Saraju P Mohanty. *Low-power high-level synthesis for nanoscale CMOS circuits*. Springer, Boston, MA, **2008**. doi:10.1007/978-0-387-76474-0.

[93] Sumit Ahuja. *High level power estimation and reduction techniques for power aware hardware design*. Ph.D. thesis, Virginia Tech, **2010**.

[94] John Hansen and Montek Singh. An energy and power-aware approach to high-level synthesis of asynchronous systems. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 269–276. **2010**. doi:10.1109/ICCAD.2010.5654169.

[95] A.K. Murugavel and N. Ranganathan. A game theoretic approach for power optimization during behavioral synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(6):1031–1043, **2003**. doi:10.1109/TVLSI.2003.819566.

[96] Selma Dilek, Rawan Smri, Suleyman Tosun, and Deniz Dal. A high-level synthesis methodology for energy and reliability-oriented designs. *IEEE Transactions on Computers*, 71(1):161–174, **2022**. doi:10.1109/TC.2020.3043885.

[97] Philippe Coussy, Daniel D. Gajski, Michael Meredith, and Andres Takach. An introduction to high-level synthesis. *IEEE Design & Test of Computers*, 26(4):8–17, **2009**. doi:10.1109/MDT.2009.69.

[98] Sarah L. Harris and David Harris. 4 - hardware description languages. In Sarah L. Harris and David Harris, editors, *Digital Design and Computer Architecture*, pages 170–235. Morgan Kaufmann, **2022**. ISBN 978-0-12-820064-3. doi:https://doi.org/10.1016/B978-0-12-820064-3.00004-0.

[99] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin. *High — Level Synthesis: Introduction to Chip and System Design*. Springer New York, US, 1 edition, **1992**. ISBN 978-1-4613-6617-1,978-1-4615-3636-9. doi:https://doi.org/10.1007/978-1-4615-3636-9.

[100] Liang Chen, Mojtaba Ebrahimi, and Mehdi B. Tahoori. Reliability-aware operation chaining in high level synthesis. In *2015 20th IEEE European Test Symposium (ETS)*, pages 1–6. **2015**. doi:10.1109/ETS.2015.7138739.

[101] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1st edition, **1994**. ISBN 0070163332. doi:https://dl.acm.org/doi/10.5555/541643.

[102] Suleyman Tosun, Nazanin Mansouri, Mahmut Kandemir, and Ozcan Ozturk. An ilp formulation for task scheduling on heterogeneous chip multiprocessors. In *Proceedings of the 21st International Conference on Computer and Information Sciences*, ISCIS'06, page 267–276. Springer-Verlag, Berlin, Heidelberg, **2006**. ISBN 3540472428. doi:10.1007/11902140_30.

[103] Gill Velleda Gonzales, Elizaldo Domingues dos Santos, Leonardo Ramos Emmendorfer, Liércio André Isoldi, Luiz Alberto Oliveira Rocha, and Emanuel da Silva Diaz Estrada. A comparative study of simulated annealing with different cooling schedules for geometric optimization of a heat

transfer problem according to constructal design. *Scientia Plena*, 11(8), **2015**. doi:10.14808/sci.plena.2015.081321.

[104] Clayton V. Deutsch and Perry W. Cockerham. Practical considerations in the application of simulated annealing to stochastic simulation. *Mathematical Geology*, 26(1):67–82, **1994**. ISSN 1573-8868. doi:10.1007/BF02065876.

[105] Sergio Ledesma, Gabriel Avi na, and Raul Sanchez. Practical considerations for simulated annealing implementation. In Cher Ming Tan, editor, *Simulated Annealing*, chapter 20. IntechOpen, Rijeka, **2008**. doi:10.5772/5560.

[106] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, **1953**. doi:10.1063/1.1699114.

[107] FICO. FICO xpress optimization. `https://www.fico.com/en/products/fico-xpress-optimization`, **2001-2022**.

# 8. APPENDICES

## APPENDIX A - Mosel and C Implementations Source Code

### A.1 Mosel Code For ILP Formulation of the HLS Problem

```
!@encoding CP1252
model hls
uses "mmxprs"; !gain access to the Xpress-Optimizer solver
uses "mmsystem";
forward procedure print_solution

! ---------- declarations ----------

declarations
        N = 13          ! diffeq (1 is the source, N is the last sink task)
        !N = 30 ! AR
        !N = 28 ! EWF
        !N = 25 ! FIR
        R = 6           ! Number of resources (the last one is dummy)
        TASKS = 1..N
        Resources = 1..R
        Voltages = 1..2

          alpha = 1.0

          AreaConstraint = 40
          LatencyConstraint = 29
          Csteps = 1..LatencyConstraint

          TaskType: array(TASKS) of integer ! Types of tasks 0-nop, 1-add, 2-mult
          ! Types of resources 0-(for dummy end task), 1-add, 2-mult
          ResourceType: array(Resources) of integer
          StartTime: array(TASKS, Csteps) of mpvar ! Start times of tasks (Xis)
          StartTimeN: mpvar ! Start time of sink

          ! If task_i is assigned to resource_j under voltage_v
          Assigned: array(TASKS, Resources, Voltages) of mpvar

          PREC: array(range,range) of integer ! Matrix of the adjacency graph PREC(i,j)

          Area: array(Resources) of real              ! Area of resources

        ! Reliability of resource_j under Voltage_v
        Reliability: array(Resources, Voltages) of real
        ! Energy consumption of resource_j under Voltage_v
        Energy: array(Resources, Voltages) of real
        ! Latency of resource_j under Voltage_v
        Latency: array(Resources, Voltages) of integer

        MinR: real
        MaxR: real
        MinE: real
        MaxE: real

        Delay: array(TASKS) of mpvar ! Delay of a node (dj)

        ! 1 if vi is scheduled at Cstep s and bound to resource r under v
        k: array(TASKS, Csteps, Resources, Voltages) of mpvar

        ! Number of instances of resource_j used per voltage level
        y: array(Resources, Voltages) of mpvar

        TaskReliability: array(TASKS) of mpvar ! Reliability of task_i

        TaskEnergy: array(TASKS) of mpvar ! Energy consumed by a task

        Reliability_value: real

        NormalizedR: mpvar
        NormalizedE: mpvar

        TotalArea: real

        sol: real    ! Solution
end-declarations

! ---------- Initializations ----------

initializations from 'diffeq.dat'
!initializations from 'fir.dat'
!initializations from 'ar.dat'
!initializations from 'ew.dat'
          PREC TaskType ResourceType Area Reliability Latency Energy MinR MaxR MinE MaxE
end-initializations

writeln("Begin running model")

starttime:= gettime ! Get the start time

! ---------- Formulations ----------

forall(i in TASKS, s in Csteps) StartTime(i, s) is_binary
forall(i in TASKS, j in Resources, v in Voltages) Assigned(i, j, v) is_binary
forall(i in TASKS, s in Csteps, j in Resources, v in Voltages) k(i, s, j, v) is_binary

forall(r in Resources, v in Voltages) y(r, v) is_integer

forall(i in TASKS) sum(s in Csteps) StartTime(i, s) = 1

! If Tasktype doesn't match resource type, it cannot be assigned
forall(i in TASKS,j in Resources, v in Voltages) do
        if ResourceType(j) <> TaskType(i) then
                Assigned(i, j, v) = 0
        end-if
end-do
```

```

! Only one resource can be assigned to any task under only one voltage level:
forall(i in TASKS) sum(j in Resources, v in Voltages | ResourceType(j) = TaskType(i))
↪  Assigned(i, j, v) = 1

! Delay of task_i
forall(i in TASKS) Delay(i) = sum(r in Resources, v in Voltages) Latency(r, v)*Assigned(i,
↪  r, v)

! Precedence conditions
forall(i,j in TASKS | PREC(i,j) = 1) sum(s in Csteps)StartTime(j, s)*s >= sum(s in
↪  Csteps)StartTime(i, s)*s + Delay(i)

forall(i in TASKS) sum(j in Resources, s in Csteps, v in Voltages) k(i,s,j,v) = 1
forall(i in TASKS, s in Csteps, j in Resources, v in Voltages) k(i, s, j, v) >= Assigned(i,
↪  j, v) + StartTime(i, s) - 1

! Task reliability calculation
forall (i in TASKS) TaskReliability(i) = sum(j in Resources, v in Voltages) Reliability(j,
↪  v)*Assigned(i,j,v)

! Task energy consumption calculation
forall (i in TASKS) TaskEnergy(i) = sum(j in Resources, v in Voltages) Energy(j,
↪  v)*Assigned(i,j,v)

! Calculating how many instances of each resource are used in total
forall(r in Resources, s in Csteps, v in Voltages) y(r,v) >= sum(i in TASKS)  k(i, s, r, v)

StartTimeN = sum(s in Csteps)StartTime(N, s)*s

! ----------- Objective function -----------

TotalReliability:= sum(i in TASKS) TaskReliability(i)

 TA:= sum(j in Resources, v in Voltages) y(j,v) * Area(j)
 TotalArea:=getsol(TA)

 TotalEnergy:= sum(i in TASKS) TaskEnergy(i)

 NormalizedR = (TotalReliability-MinR)/(MaxR-MinR)

 NormalizedE = (TotalEnergy-MinE)/(MaxE-MinE)

 objectiveF := alpha*(1-NormalizedR) + (1-alpha)*(NormalizedE)

! ----------- Constraints -----------

 StartTimeN <= LatencyConstraint ! Check this!
 TA <= AreaConstraint

! Solve the problem: maximize the total reliability, minimize energy consumption
 minimize(objectiveF)

 sol:=getobjval

 writeln("Time: ",gettime-starttime, " s") !prints the program execution time

 ! Solution printing
 print_solution

writeln("End running model")

! ----------- Helper functions -----------

 procedure print_solution
         !writeln("latency(1,1) = ", Latency(1,1))
         !writeln("Energy(5,2) = ", Energy(5,2))
         !writeln
    writeln("Total duration: ", StartTimeN.sol, " cycles")
    writeln("Total area: ", getsol(TA), " units")

     Reliability_value:=1
    forall (i in 1..N) do
    Reliability_value:=Reliability_value*TaskReliability(i).sol
    end-do

    writeln("Total reliability (mult): ",Reliability_value)

    writeln("Total reliability (sum): ", getsol(TotalReliability))
          writeln

          writeln("Total Energy: ", getsol(TotalEnergy))
          writeln
          writeln("NormalizedR: ", getsol(NormalizedR))
          writeln
          writeln("NormalizedE: ", getsol(NormalizedE))
          writeln

          writeln
        forall(r in Resources, v in Voltages) do
                writeln("y(",r,",",v,") = ", y(r,v).sol)
        end-do
        writeln
          forall(i in TASKS, j in Resources, v in Voltages) do
                if (Assigned(i, j, v).sol = 1) then
                        writeln("Assigned(",i,",",j,",",v,") = ", Assigned(i, j, v).sol)
                 end-if
          end-do
          writeln

          forall(i in TASKS, s in Csteps) do
                  if (StartTime(i, s).sol = 1) then
                          writeln("StartTime(",i,",",s,") = ", StartTime(i, s).sol)
                  end-if
          end-do
          writeln

          forall(i in TASKS) do
                  writeln("Delay(",i,") = ", Delay(i).sol)
          end-do
          !writeln

 end-procedure

end-model
```

## A.2 Mosel Code For ILP Formulation of the HLS Problem With Partial DMR

```
!@encoding CP1252
model hls
uses "mmxprs"; !gain access to the Xpress-Optimizer solver
uses "mmsystem";
forward procedure print_solution

! ---------- declarations ----------

declarations
        N = 13        ! diffeq (1 is the source, N is the last sink task)
        !N = 30 ! AR
        !N = 28 ! EWF
        !N = 25 ! FIR
        R = 6         ! Number of resources (the last one is dummy)
        TASKS = 1..N
        Resources = 1..R
        Voltages = 1..2

          alpha = 1.0

          AreaConstraint = 40
          LatencyConstraint = 29
          Csteps = 1..LatencyConstraint

          TaskType: array(TASKS) of integer ! Types of tasks 0-nop, 1-add, 2-mult
          ! Types of resources 0-(for dummy end task), 1-add, 2-mult
          ResourceType: array(Resources) of integer
          StartTime: array(TASKS, Csteps) of mpvar ! Start times of tasks (Xis)
          StartTimeN: mpvar ! Start time of sink

          ! If task_i is assigned to resource_j under voltage_v
          Assigned: array(TASKS, Resources, Voltages) of mpvar

          PREC: array(range,range) of integer ! Matrix of the adjacency graph PREC(i,j)

          Area: array(Resources) of real             ! Area of resources

        ! Reliability of resource_j under Voltage_v
        Reliability: array(Resources, Voltages) of real
        ! Energy consumption of resource_j under Voltage_v
        Energy: array(Resources, Voltages) of real
        ! Latency of resource_j under Voltage_v
        Latency: array(Resources, Voltages) of integer

        MinR: real
        MaxR: real
        MinE: real
        MaxE: real

        Delay: array(TASKS) of mpvar ! Delay of a node (dj)

        ! 1 if vi is scheduled at Cstep s and bound to resource r under v
        k: array(TASKS, Csteps, Resources, Voltages) of mpvar

        ! Number of instances of resource_j used per voltage level
        y: array(Resources, Voltages) of mpvar

        TaskReliability: array(TASKS) of mpvar ! Reliability of task_i

        TaskEnergy: array(TASKS) of mpvar ! Energy consumed by a task

        Reliability_value: real

        NormalizedR: mpvar
        NormalizedE: mpvar

        TotalArea: real

        sol: real   ! Solution
end-declarations

! ---------- Initializations ----------

initializations from 'diffeq.dat'
!initializations from 'fir.dat'
!initializations from 'ar.dat'
!initializations from 'ew.dat'
          PREC TaskType ResourceType Area Reliability Latency Energy MinR MaxR MinE MaxE
end-initializations

writeln("Begin running model")

starttime:= gettime ! Get the start time

! ---------- Formulations ----------

forall(i in TASKS, s in Csteps) StartTime(i, s) is_binary
forall(i in TASKS, j in Resources, v in Voltages) Assigned(i, j, v) is_binary
forall(i in TASKS, s in Csteps, j in Resources, v in Voltages) k(i, s, j, v) is_binary
forall(i in TASKS_d, s in Csteps) StartTime_d(i, s) is_binary
forall(i in TASKS_d, j in Resources, v in Voltages) Assigned_d(i, j, v) is_binary
forall(i in TASKS_d, s in Csteps, j in Resources, v in Voltages) k_d(i, s, j, v) is_binary
forall(i in TASKS) Duplicated(i) is_binary

forall(r in Resources, v in Voltages) y(r, v) is_integer

forall(i in TASKS) sum(s in Csteps) StartTime(i, s) = 1

! Any task can only begin in one Cstep or not at all (if not duplicated)
forall(i in TASKS_d) sum(s in Csteps) StartTime_d(i, s) <= 1

! If Tasktype doesn't match resource type, it cannot be assigned
forall(i in TASKS,j in Resources, v in Voltages) do
        if ResourceType(j) <> TaskType(i) then
                Assigned(i, j, v) = 0
        end-if
end-do

```

```
forall(i in TASKS_d,j in Resources, v in Voltages) do
        if ResourceType(j) <> TaskType(i) then
                Assigned_d(i, j, v) = 0
        end-if
end-do

! Only one resource can be assigned to any task under only one voltage level:
forall(i in TASKS) sum(j in Resources, v in Voltages | ResourceType(j) = TaskType(i))
↪  Assigned(i, j, v) = 1

! For PD - a task can be assigned a resource or not
forall(i in TASKS_d) sum(j in Resources, v in Voltages | ResourceType(j) = TaskType(i))
↪  Assigned_d(i, j, v) <= 1
forall(i in TASKS) Duplicated(i) = sum(j in Resources, v in Voltages) Assigned_d(i, j, v)

! Delay of task_i
forall(i in TASKS) Delay(i) = sum(r in Resources, v in Voltages) Latency(r, v)*Assigned(i,
↪  r, v)
forall(i in TASKS_d) Delay_d(i) = sum(r in Resources, v in Voltages) Latency(r,
↪  v)*Assigned_d(i, r, v)

! Precedence conditions
forall(i,j in TASKS | PREC(i,j) = 1) sum(s in Csteps)StartTime(j, s)*s >= sum(s in
↪  Csteps)StartTime(i, s)*s + Delay(i)

! In case Delay_d(i) is greater than Delay(i), start time of a dependent task should
↪  consider that as well:
forall(i,j in TASKS | PREC(i,j) = 1) sum(s in Csteps)StartTime(j, s)*s >= sum(s in
↪  Csteps)StartTime(i, s)*s + Delay_d(i)

forall(i in TASKS) sum(s in Csteps)StartTime(i, s)*s = sum(s in Csteps)StartTime_d(i, s)*s

forall(i in TASKS) sum(j in Resources, s in Csteps, v in Voltages) k(i,s,j,v)=1
forall(i in TASKS, s in Csteps, j in Resources, v in Voltages) k(i, s, j, v) >= Assigned(i,
↪  j, v) + StartTime(i, s) - 1

! In PD some task may not be duplicated so k_d can be 0
forall(i in TASKS_d) sum(j in Resources, s in Csteps, v in Voltages) k_d(i,s,j,v) <= 1
forall(i in TASKS_d, s in Csteps, j in Resources, v in Voltages) k_d(i, s, j, v) >=
↪  Assigned_d(i, j, v) + StartTime_d(i, s) - 1

! Task reliability calculation
forall (i in TASKS) TaskReliability(i) = sum(j in Resources, v in Voltages) Reliability(j,
↪  v)*Assigned(i,j,v)
forall (i in TASKS_d) TaskReliability_d(i) = sum(j in Resources, v in Voltages)
↪  Reliability(j, v)*Assigned_d(i,j,v)


! Task energy consumption calculation
forall (i in TASKS) TaskEnergy(i) = sum(j in Resources, v in Voltages) Energy(j,
↪  v)*Assigned(i,j,v)
forall (i in TASKS_d) TaskEnergy_d(i) = sum(j in Resources, v in Voltages) Energy(j,
↪  v)*Assigned_d(i,j,v)

! Calculating how many instances of each resource are used in total
```

```
forall(r in Resources, s in Csteps, v in Voltages) y(r,v) >= (sum(i in TASKS)  k(i, s, r, v) + sum(j in TASKS_d)  k_d(j, s, r, v))

StartTimeN = sum(s in Csteps)StartTime(N, s)*s
StartTimeN_d = sum(s in Csteps)StartTime_d(N, s)*s


! ----------- Objective function -----------

TotalReliability:= sum(i in 2..N-1) TaskReliability(i) + sum(j in 2..N-1) TaskReliability_d(j)
 dMinR := sum(i in 2..N-1) Duplicated(i)*0.938+(N-2)*0.938
 dMaxR := sum(i in 2..N-1) Duplicated(i)*0.999+(N-2)*0.999
 TA:= sum(j in Resources, v in Voltages) y(j,v) * Area(j)
 TotalArea:=getsol(TA)

 TotalEnergy:= sum(i in TASKS) TaskEnergy(i) + sum(j in TASKS_d) TaskEnergy_d(j)

 NormalizedR = (TotalReliability-MinR)/(MaxR-MinR)

 NormalizedE = (TotalEnergy-MinE)/(MaxE-MinE)

 objectiveF := alpha*(1-NormalizedR) + (1-alpha)*(NormalizedE)

! ----------- Constraints -----------

 StartTimeN <= LatencyConstraint ! Check this!
 TA <= AreaConstraint

! Solve the problem: maximize the total reliability, minimize energy consumption
 minimize(objectiveF)

 sol:=getobjval

 writeln("Time: ",gettime-starttime, " s") !prints the program execution time

 ! Solution printing
 print_solution

writeln("End running model")

! ----------- Helper functions -----------

 procedure print_solution
         !writeln("latency(1,1) = ", Latency(1,1))
         !writeln("Energy(5,2) = ", Energy(5,2))
         !writeln
    writeln("Total duration: ", StartTimeN.sol, " cycles")
    writeln("Total area: ", getsol(TA), " units")

     Reliability_value:=1
    forall (i in 1..N) do
    Reliability_value:=Reliability_value*TaskReliability(i).sol
    end-do

    writeln("Total reliability (mult): ",Reliability_value)

    writeln("Total reliability (sum): ", getsol(TotalReliability))
          writeln

          writeln("Total Energy: ", getsol(TotalEnergy))
          writeln
          writeln("NormalizedR: ", getsol(NormalizedR))
          writeln
          writeln("NormalizedE: ", getsol(NormalizedE))
          writeln

          writeln
        forall(r in Resources, v in Voltages) do
                writeln("y(",r,",",v,") = ", y(r,v).sol)
        end-do
        writeln
          forall(i in TASKS, j in Resources, v in Voltages) do
                if (Assigned(i, j, v).sol = 1) then
                        writeln("Assigned(",i,",",j,",",v,") = ", Assigned(i, j, v).sol)
                   end-if
          end-do
          writeln

          forall(i in TASKS, s in Csteps) do
                  if (StartTime(i, s).sol = 1) then
                          writeln("StartTime(",i,",",s,") = ", StartTime(i, s).sol)
                  end-if
          end-do
          writeln

          forall(i in TASKS) do
                  writeln("Delay(",i,") = ", Delay(i).sol)
          end-do
          !writeln

 end-procedure

end-model
```

## A.3 Mosel Code For ILP Formulation of the HLS Problem With Full DMR

```
!@encoding CP1252
model hls
uses "mmxprs"; !gain access to the Xpress-Optimizer solver
uses "mmsystem";
forward procedure print_solution

! ---------- declarations ----------

declarations
        N = 13        ! diffeq (1 is the source, N is the last sink task)
        !N = 30 ! AR
        !N = 28 ! EWF
        !N = 25 ! FIR
        R = 6         ! Number of resources (the last one is dummy)
        TASKS = 1..N
        Resources = 1..R
        Voltages = 1..2

          alpha = 1.0

          AreaConstraint = 40
          LatencyConstraint = 29
          Csteps = 1..LatencyConstraint

          TaskType: array(TASKS) of integer ! Types of tasks 0-nop, 1-add, 2-mult
          ! Types of resources 0-(for dummy end task), 1-add, 2-mult
          ResourceType: array(Resources) of integer
          StartTime: array(TASKS, Csteps) of mpvar ! Start times of tasks (Xis)
          StartTimeN: mpvar ! Start time of sink

          ! If task_i is assigned to resource_j under voltage_v
          Assigned: array(TASKS, Resources, Voltages) of mpvar

          PREC: array(range,range) of integer ! Matrix of the adjacency graph PREC(i,j)

          Area: array(Resources) of real             ! Area of resources

        ! Reliability of resource_j under Voltage_v
        Reliability: array(Resources, Voltages) of real
        ! Energy consumption of resource_j under Voltage_v
        Energy: array(Resources, Voltages) of real
        ! Latency of resource_j under Voltage_v
        Latency: array(Resources, Voltages) of integer

        MinR: real
        MaxR: real
        MinE: real
        MaxE: real

        Delay: array(TASKS) of mpvar ! Delay of a node (dj)

        ! 1 if vi is scheduled at Cstep s and bound to resource r under v
        k: array(TASKS, Csteps, Resources, Voltages) of mpvar

        ! Number of instances of resource_j used per voltage level
        y: array(Resources, Voltages) of mpvar

        TaskReliability: array(TASKS) of mpvar ! Reliability of task_i

        TaskEnergy: array(TASKS) of mpvar ! Energy consumed by a task

        Reliability_value: real

        NormalizedR: mpvar
        NormalizedE: mpvar

        TotalArea: real

        sol: real   ! Solution
end-declarations

! ---------- Initializations ----------

initializations from 'diffeq.dat'
!initializations from 'fir.dat'
!initializations from 'ar.dat'
!initializations from 'ew.dat'
          PREC TaskType ResourceType Area Reliability Latency Energy MinR MaxR MinE MaxE
end-initializations

writeln("Begin running model")

starttime:= gettime ! Get the start time

! ---------- Formulations ----------

forall(i in TASKS, s in Csteps) StartTime(i, s) is_binary
forall(i in TASKS, j in Resources, v in Voltages) Assigned(i, j, v) is_binary
forall(i in TASKS, s in Csteps, j in Resources, v in Voltages) k(i, s, j, v) is_binary
forall(i in TASKS_d, s in Csteps) StartTime_d(i, s) is_binary
forall(i in TASKS_d, j in Resources, v in Voltages) Assigned_d(i, j, v) is_binary
forall(i in TASKS_d, s in Csteps, j in Resources, v in Voltages) k_d(i, s, j, v) is_binary

forall(r in Resources, v in Voltages) y(r, v) is_integer

forall(i in TASKS) sum(s in Csteps) StartTime(i, s) = 1
forall(i in TASKS_d) sum(s in Csteps) StartTime_d(i, s) = 1

! If Tasktype doesn't match resource type, it cannot be assigned
forall(i in TASKS,j in Resources, v in Voltages) do
        if ResourceType(j) <> TaskType(i) then
                Assigned(i, j, v) = 0
        end-if
end-do

forall(i in TASKS_d,j in Resources, v in Voltages) do
        if ResourceType(j) <> TaskType(i) then
                Assigned_d(i, j, v) = 0
        end-if
end-do

! Only one resource can be assigned to any task under only one voltage level:
forall(i in TASKS) sum(j in Resources, v in Voltages | ResourceType(j) = TaskType(i))
↪  Assigned(i, j, v) = 1
forall(i in TASKS_d) sum(j in Resources, v in Voltages | ResourceType(j) = TaskType(i))
↪  Assigned_d(i, j, v) = 1

forall (i in TASKS) do
    Delay(i) >= sum(r in Resources, v in Voltages) Latency(r, v)*Assigned(i, r, v)
    Delay(i) >= sum(r in Resources, v in Voltages) Latency(r, v)*Assigned_d(i, r, v)
    Delay_d(i) >= sum(r in Resources, v in Voltages) Latency(r, v)*Assigned(i, r, v)
    Delay_d(i) >= sum(r in Resources, v in Voltages) Latency(r, v)*Assigned_d(i, r, v)
end-do

! Precedence conditions
forall(i,j in TASKS | PREC(i,j) = 1) sum(s in Csteps)StartTime(j, s)*s >= sum(s in
↪  Csteps)StartTime(i, s)*s + Delay(i)
forall(i,j in TASKS_d | PREC(i,j) = 1) sum(s in Csteps)StartTime_d(j, s)*s >= sum(s in
↪  Csteps)StartTime_d(i, s)*s + Delay_d(i)

forall(i in TASKS) sum(s in Csteps)StartTime(i, s)*s = sum(s in Csteps) StartTime_d(i, s)*s

forall(i in TASKS) sum(j in Resources, s in Csteps, v in Voltages) k(i,s,j,v) = 1
forall(i in TASKS, s in Csteps, j in Resources, v in Voltages) k(i, s, j, v) >= Assigned(i,
↪  j, v) + StartTime(i, s) - 1
forall(i in TASKS_d) sum(j in Resources, s in Csteps, v in Voltages) k_d(i,s,j,v) = 1
forall(i in TASKS_d, s in Csteps, j in Resources, v in Voltages) k_d(i, s, j, v) >=
↪  Assigned_d(i, j, v) + StartTime_d(i, s) - 1

! Task reliability calculation
forall (i in TASKS) TaskReliability(i) = sum(j in Resources, v in Voltages) Reliability(j,
↪  v)*Assigned(i,j,v)
forall (i in TASKS_d) TaskReliability_d(i) = sum(j in Resources, v in Voltages)
↪  Reliability(j, v)*Assigned_d(i,j,v)

! Task energy consumption calculation
forall (i in TASKS) TaskEnergy(i) = sum(j in Resources, v in Voltages) Energy(j,
↪  v)*Assigned(i,j,v)
forall (i in TASKS_d) TaskEnergy_d(i) = sum(j in Resources, v in Voltages) Energy(j,
↪  v)*Assigned_d(i,j,v)

! Calculating how many instances of each resource are used in total
forall(r in Resources, s in Csteps, v in Voltages) y(r,v) >= (sum(i in TASKS)  k(i, s, r, v)
↪  + sum(j in TASKS_d)  k_d(j, s, r, v))

StartTimeN = sum(s in Csteps)StartTime(N, s)*s
StartTimeN_d = sum(s in Csteps)StartTime_d(N, s)*s

! ----------- Objective function -----------
TotalReliability:= sum(i in TASKS) TaskReliability(i)
                 + sum(j in TASKS_d) TaskReliability_d(j)

 TA:= sum(j in Resources, v in Voltages) y(j,v) * Area(j)
 TotalArea:=getsol(TA)

 TotalEnergy:= sum(i in TASKS) TaskEnergy(i) + sum(j in TASKS_d) TaskEnergy_d(j)

 NormalizedR = (TotalReliability-MinR)/(MaxR-MinR)

 NormalizedE = (TotalEnergy-MinE)/(MaxE-MinE)

 objectiveF := alpha*(1-NormalizedR) + (1-alpha)*(NormalizedE)

! ----------- Constraints -----------

 StartTimeN <= LatencyConstraint ! Check this!
 TA <= AreaConstraint

! Solve the problem: maximize the total reliability, minimize energy consumption
minimize(objectiveF)

 sol:=getobjval

 writeln("Time: ",gettime-starttime, " s") !prints the program execution time

 ! Solution printing
 print_solution

writeln("End running model")

! ----------- Helper functions -----------

 procedure print_solution
         !writeln("latency(1,1) = ", Latency(1,1))
         !writeln("Energy(5,2) = ", Energy(5,2))
         !writeln
    writeln("Total duration: ", StartTimeN.sol, " cycles")
    writeln("Total area: ", getsol(TA), " units")

     Reliability_value:=1
    forall (i in 1..N) do
    Reliability_value:=Reliability_value*TaskReliability(i).sol
    end-do

    writeln("Total reliability (mult): ",Reliability_value)

    writeln("Total reliability (sum): ", getsol(TotalReliability))
          writeln

          writeln("Total Energy: ", getsol(TotalEnergy))
          writeln
          writeln("NormalizedR: ", getsol(NormalizedR))
          writeln
          writeln("NormalizedE: ", getsol(NormalizedE))
          writeln

          writeln
        forall(r in Resources, v in Voltages) do
                writeln("y(",r,",",v,") = ", y(r,v).sol)
        end-do
        writeln
          forall(i in TASKS, j in Resources, v in Voltages) do
                if (Assigned(i, j, v).sol = 1) then
                        writeln("Assigned(",i,",",j,",",v,") = ", Assigned(i, j, v).sol)
                   end-if
          end-do
          writeln

          forall(i in TASKS, s in Csteps) do
                  if (StartTime(i, s).sol = 1) then
                          writeln("StartTime(",i,",",s,") = ", StartTime(i, s).sol)
                  end-if
          end-do
          writeln

          forall(i in TASKS) do
                  writeln("Delay(",i,") = ", Delay(i).sol)
          end-do
          !writeln

 end-procedure

end-model
```

## A.4 C Source Code For SA-Based HLS Method

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <sys/time.h>
#include <setjmp.h>
#include "helper_functions.h"
#define V_LEVELS 2 // Change for the number of available voltage levels
#define NUM_TASKS 30 // Change for the number of tasks in the TG of the benchmark
#define NUM_RESOURCES 6 // Change for the number of resources
#define ALPHA 0.5 // 1 for max R, 0 for min E
#define LATENCY_CONSTRAINT 55 // Change for the desired latency constraint
#define AREA_CONSTRAINT 30 // Change for the desired area constraint
#define DESIRED_ACCEPTANCE_PROBABILITY 0.8 // Desired starting acceptance probability for SA
#define NUM_TRANSITIONS 10000// number of transitions in the transition set for calculation
↪  of the initial temp in SA
#define TRY do{ jmp_buf ex_buf__; if( !setjmp(ex_buf__) ){
#define CATCH } else {
#define ENDTRY } }while(0)
#define MAX_NODE_SIZE 100
#define MAXINT 2147483647
#define COOLING_SCHEDULE 0.95
#define ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE 10000
#define DATFILE = "data/des.dat"; // Specify the benchmark dat file
#define LIBRARYFILE = "data/resource_library.dat"; // Specify the resource library dat file
#define TRANSITIONFILE = "data/des_transitions.dat"; // Specify the file containing sample
↪  transitions for the given benchmark to calculate the initial temp

//---------------- ENUMS -----------------
enum task_type {NOP = 0, ADD = 1, MUL = 2};
enum voltage_level {HIGH = 1, LOW = 2};

//---------------- STRUCTS -----------------
typedef struct resource{
  int id;
  int type;
  double reliability[V_LEVELS];
  double energy[V_LEVELS];
  int latency[V_LEVELS];
  int area;
}resource;
typedef struct task{
  int id;
  int type;
  resource* assigned_resource;
  int assigned_voltage;
  int start_time;
  resource* assigned_duplicate_resource;
  int assigned_duplicate_voltage;
}task;

```

```

//---------------- FUNCTION DEACLARATIONS -----------------
double get_objective(task* solution, double MaxR, double MinR, double MaxE, double MinE);
int get_area(task* solution, resource* resource_library);
void reduce_area_by_rescheduling(task* solution, resource* resource_library, int*
↪  topological_ordering, int* ASAP_scheduling, int* ALAP_scheduling, int* list_scheduling,
↪  int number_of_tasks, int (*graph)[number_of_tasks]);
void recursively_shift_dependent_tasks(int task_to_move, int* moved, task* solution,
↪  resource* resource_library, int* topological_ordering, int* ASAP_scheduling, int*
↪  ALAP_scheduling, int* list_scheduling, int number_of_tasks, int
↪  (*graph)[number_of_tasks]);
int read_library(resource* resource_library, int* no_of_mult_resources, const char*
↪  filename);
void print_resources(resource* resource_library, int num_res);
void print_solution(task* solution);
void random_solution_generator(task* solution, int no_of_mult_tasks, int* mult_nodes_list,
↪  resource* resource_library, resource* multipliers, resource* adders, int
↪  no_of_mult_resources);
void optAL_solution_generator(task* solution, int no_of_mult_tasks, int* mult_nodes_list,
↪  resource* resource_library, resource* multipliers, resource* adders, int
↪  no_of_mult_resources);
int get_ASAP_scheduling(task* solution, int number_of_tasks, int (*graph)[number_of_tasks],
↪  int* ASAP_scheduling, int* topological_ordering);
int get_ALAP_scheduling(task* solution, int number_of_tasks, int (*graph)[number_of_tasks],
↪  int* ALAP_scheduling, int* topological_ordering, int limit_latency);
int list_scheduling_min_resource_usage(task* solution, resource* resource_library, int
↪  number_of_tasks, int (*graph)[number_of_tasks], int* ALAP_scheduling, int*
↪  list_scheduling, int* topological_ordering, int limit_latency);
void get_preceding_tasks(int number_of_tasks, int (*graph)[number_of_tasks], int**
↪  predecessor_tasks_id_list, int* number_of_predecessors);
void get_topological_ordering(int number_of_tasks, int (*graph)[number_of_tasks], int*
↪  topological_ordering);
double get_temp(double prev_temp, double p, double acceptable_error, double
↪  (*set_of_transitions)[2]);
void random_candidate_generator(task* solution, task* candidate_solution, int
↪  no_of_mult_tasks, int* mult_nodes_list, resource* resource_library, resource*
↪  multipliers, resource* adders, int no_of_mult_resources);
double print_objective(task* solution, double MaxR, double MinR, double MaxE, double MinE);

//---------------- MAIN -----------------
int main(int argc, char** argv){

  task current_solution[NUM_TASKS];
  task candidate_solution[NUM_TASKS];
  int precedence_graph[NUM_TASKS][NUM_TASKS];
  int** predecessor_tasks_id_list = (int**)malloc(NUM_TASKS * sizeof(int*));// Keeping the
  ↪  IDs of predecessor tasks for each task
  double set_of_transitions[NUM_TRANSITIONS][2]; // MIN, MAX - min should be higher than max
  ↪  as we are minimizing (before cost, after cost) - strictly positive transitions!
  int number_of_predecessors[NUM_TASKS]; // Size of the list of predecessor tasks for each
  ↪  task
  int ASAP_scheduling[NUM_TASKS], ASAP_scheduling_candidate[NUM_TASKS]; // ASAP scheduling
  ↪  of the tasks
  int ALAP_scheduling[NUM_TASKS], ALAP_scheduling_candidate[NUM_TASKS]; // ALAP scheduling
  ↪  of the tasks
```

```
int list_scheduling[NUM_TASKS], list_scheduling_candidate[NUM_TASKS]; // List scheduling
↪ of the tasks
int mobility[NUM_TASKS]; // Mobility of the tasks
int no_of_mult_tasks, no_of_mult_resources, lat;
int mult_nodes_list[MAX_NODE_SIZE];
int topological_ordering[NUM_TASKS];
resource resource_library[NUM_RESOURCES];
resource *adders, *multipliers;
double MaxR, MinR, MaxE, MinE;

// Read input files
TRY
  {
    printf("Reading benchmark graph...\n");
    if(!read_benchmark(NUM_TASKS, NUM_TASKS, &MaxR, &MinR, &MaxE, &MinE,
    ↪ &no_of_mult_tasks, mult_nodes_list, precedence_graph, DATFILE)) // Read the input
    ↪ graph of the benchmark
      printf("Could not open input dat file!\n");
    else printf("Input dat file reading success!\n\n");

    printf("Reading resource library...\n");
    if(!read_library(resource_library, &no_of_mult_resources, LIBRARYFILE)) // Reading
    ↪ resource library input file
      printf("Could not open resource library dat file!\n");
    else {
      printf("Resource library dat file reading success!\n\n");
      //Putting adders and multipliers in separate lists
      multipliers = (resource*)malloc(sizeof(resource)*(no_of_mult_resources));
      adders = (resource*)malloc(sizeof(resource)*(NUM_RESOURCES - no_of_mult_resources -
      ↪ 1));
      int mul_i = 0, add_i = 0;
      for(int idx = 0; idx<NUM_RESOURCES; idx++){
        if(resource_library[idx].type == MUL){
          //add multipliers;
          multipliers[mul_i++] = resource_library[idx];
        }
        else if (resource_library[idx].type == ADD){
          // Add adders;
          adders[add_i++] = resource_library[idx];
        }
      }
    }
    printf("Reading transition set...\n");
    if(!read_double_matrix(NUM_TRANSITIONS, 2, set_of_transitions, TRANSITIONFILE)) //
    ↪ Reading resource library input file
      printf("Could not open transition set dat file!\n");
    else {
      printf("Resource library dat file reading success!\n\n");
    }

  }
CATCH
  {
    printf("Error reading input files!\n");
  }
ENDTRY;
```

```

int looping_time = 0;

clock_t begin = clock();
int area_from_ls;

// Populate the initial solution randomly so it meets the area criteria.
do{
  random_solution_generator(current_solution, no_of_mult_tasks, mult_nodes_list,
  ↪  resource_library, multipliers, adders,  no_of_mult_resources);
  get_preceding_tasks(NUM_TASKS, precedence_graph, predecessor_tasks_id_list,
  ↪  number_of_predecessors);
  get_topological_ordering(NUM_TASKS, precedence_graph, topological_ordering); // Get the
  ↪  topological ordering of the input tasks
  lat = get_ASAP_scheduling(current_solution, NUM_TASKS, precedence_graph,
  ↪  ASAP_scheduling, topological_ordering);
  get_ALAP_scheduling(current_solution, NUM_TASKS, precedence_graph, ALAP_scheduling,
  ↪  topological_ordering, lat+1);
  area_from_ls = list_scheduling_min_resource_usage(current_solution, resource_library,
  ↪  NUM_TASKS, precedence_graph, ALAP_scheduling, list_scheduling, topological_ordering,
  ↪  LATENCY_CONSTRAINT+1);
  looping_time += 1;
  if (get_area(current_solution, resource_library)>AREA_CONSTRAINT &&
  ↪  current_solution[NUM_TASKS-1].start_time<LATENCY_CONSTRAINT+1) {
    do{
      reduce_area_by_rescheduling(current_solution, resource_library,
      ↪  topological_ordering, ASAP_scheduling, ALAP_scheduling, list_scheduling,
      ↪  NUM_TASKS, precedence_graph);
    }while (get_area(current_solution, resource_library)>AREA_CONSTRAINT &&
    ↪  current_solution[NUM_TASKS-1].start_time<LATENCY_CONSTRAINT+1);
  }
}while ((get_area(current_solution, resource_library)>AREA_CONSTRAINT ||
↪  current_solution[NUM_TASKS-1].start_time>LATENCY_CONSTRAINT+1) &&
↪  looping_time<ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE);

if(looping_time>=ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE){
  looping_time = 0;
  do{
    optAL_solution_generator(current_solution, no_of_mult_tasks, mult_nodes_list,
    ↪  resource_library, multipliers, adders,  no_of_mult_resources);
    get_preceding_tasks(NUM_TASKS, precedence_graph, predecessor_tasks_id_list,
    ↪  number_of_predecessors);
    get_topological_ordering(NUM_TASKS, precedence_graph, topological_ordering); // Get
    ↪  the topological ordering of the input tasks
    lat = get_ASAP_scheduling(current_solution, NUM_TASKS, precedence_graph,
    ↪  ASAP_scheduling, topological_ordering);
    get_ALAP_scheduling(current_solution, NUM_TASKS, precedence_graph, ALAP_scheduling,
    ↪  topological_ordering, lat+1);
    area_from_ls = list_scheduling_min_resource_usage(current_solution, resource_library,
    ↪  NUM_TASKS, precedence_graph, ALAP_scheduling, list_scheduling,
    ↪  topological_ordering, LATENCY_CONSTRAINT+1);
    looping_time += 1;
    if (get_area(current_solution, resource_library)>AREA_CONSTRAINT &&
    ↪  current_solution[NUM_TASKS-1].start_time<LATENCY_CONSTRAINT+1) {
      do{
    reduce_area_by_rescheduling(current_solution, resource_library,
    ↪  topological_ordering, ASAP_scheduling, ALAP_scheduling, list_scheduling,
    ↪  NUM_TASKS, precedence_graph);
      }while (get_area(current_solution, resource_library)>AREA_CONSTRAINT &&
      ↪  current_solution[NUM_TASKS-1].start_time<LATENCY_CONSTRAINT+1);
    }
  }while ((get_area(current_solution, resource_library)>AREA_CONSTRAINT ||
  ↪  current_solution[NUM_TASKS-1].start_time>LATENCY_CONSTRAINT+1) &&
  ↪  looping_time<ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE);
}

if(looping_time>=ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE){
  print_solution(current_solution);
  printf("Solution is infeasible\n");
  return 1;
}

printf("Area of the initial solution is %d\n", get_area(current_solution,
↪  resource_library));
printf("Latency of the Initial solution is %d\n",
↪  current_solution[NUM_TASKS-1].start_time-1);
print_objective(current_solution, MaxR, MinR, MaxE, MinE);

double p = 1.0, acceptable_error = 0.000001, prev_temp = 100.0;
double temperature = get_temp(prev_temp, p, acceptable_error, set_of_transitions);

int n_iterations = 1000;

// SIMULATED ANNEALING BELOW
for(int i = 0; i < n_iterations; i++){
  int n_lat;
  int legitimate_candidate = 0;
  // Get a legitimate candidate solution
  looping_time = 0;
  do{
    random_candidate_generator(current_solution, candidate_solution, no_of_mult_tasks,
    ↪  mult_nodes_list, resource_library, multipliers, adders,  no_of_mult_resources);
    n_lat = get_ASAP_scheduling(candidate_solution, NUM_TASKS, precedence_graph,
    ↪  ASAP_scheduling_candidate, topological_ordering);
    get_ALAP_scheduling(candidate_solution, NUM_TASKS, precedence_graph,
    ↪  ALAP_scheduling_candidate, topological_ordering, n_lat+1);
    int area_from_ls_candidate = list_scheduling_min_resource_usage(candidate_solution,
    ↪  resource_library, NUM_TASKS, precedence_graph, ALAP_scheduling_candidate,
    ↪  list_scheduling_candidate, topological_ordering, n_lat+1);

    if (get_area(candidate_solution, resource_library)>AREA_CONSTRAINT &&
    ↪  candidate_solution[NUM_TASKS-1].start_time<LATENCY_CONSTRAINT+1) {
      do{
    reduce_area_by_rescheduling(candidate_solution, resource_library,
    ↪  topological_ordering, ASAP_scheduling_candidate, ALAP_scheduling_candidate,
    ↪  list_scheduling_candidate, NUM_TASKS, precedence_graph);
      }while (get_area(candidate_solution, resource_library)>AREA_CONSTRAINT &&
      ↪  candidate_solution[NUM_TASKS-1].start_time<LATENCY_CONSTRAINT+1);
    }

```

```
        if(n_lat<=LATENCY_CONSTRAINT && get_area(candidate_solution,
        ↪  resource_library)<=AREA_CONSTRAINT){
          legitimate_candidate = 1;
        }
        looping_time += 1;
      }while(!legitimate_candidate &&
      ↪  looping_time<ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE);

      if(looping_time>=ALLOWED_LOOPING_TIME_BEFORE_CALLING_INFEASIBLE){
        break;
      }

      if(get_objective(candidate_solution, MaxR, MinR, MaxE, MinE) <
      ↪  get_objective(current_solution, MaxR, MinR, MaxE, MinE) && legitimate_candidate==1){
        memcpy(&current_solution, &candidate_solution, sizeof(candidate_solution)); // accept
        ↪  candidate if better
      }
      else{
        double metropolis_acceptance_probability = get_metropolis_acceptance(temperature,
        ↪  get_objective(current_solution, MaxR, MinR, MaxE, MinE),
        ↪  get_objective(candidate_solution, MaxR, MinR, MaxE, MinE));
        double random_prob = get_rand_float_in_range(0.0, 1.0);
        if(metropolis_acceptance_probability >= random_prob  && legitimate_candidate == 1){
          memcpy(&current_solution, &candidate_solution, sizeof(candidate_solution)); //
          ↪  accept a worse candidate anyway
        }
      }

      if (temperature < 0.00001){
        temperature = 0.00001;
      }
      else{
        temperature = get_reduced_temperature_GEO(temperature, COOLING_SCHEDULE);
      }
    }


    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

    print_solution(current_solution);

    if(get_area(current_solution, resource_library)>AREA_CONSTRAINT){
      printf("\nThis solution is larger than accepted!\n");
    }
    if(current_solution[NUM_TASKS-1].start_time-1>LATENCY_CONSTRAINT){
      printf("\nThis solution is slower than accepted!\n");
    }
    printf("Area of the solution is %d\n", get_area(current_solution, resource_library));
    printf("Latency of the solution is %d\n", current_solution[NUM_TASKS-1].start_time-1);
    print_objective(current_solution, MaxR, MinR, MaxE, MinE);
    printf("Total running time: %lf\n", time_spent);

    // Free malloced stuff
    free(adders);
    free(multipliers);
```

```
  free(predecessor_tasks_id_list);

  return 0;
}

//---------------- FUNCTIONS -----------------

// Recursive function for finding the initial temp from Ben Ameur
double get_temp(double prev_temp, double p, double acceptable_error, double
↪  (*set_of_transitions)[2]){
  double new_temp;
  double current_estimate_acceptance = get_acceptance_probability(prev_temp,
  ↪  NUM_TRANSITIONS, set_of_transitions);
  double power = 1.0 / p;
  new_temp = prev_temp * (pow( (log(current_estimate_acceptance) /
  ↪  log(DESIRED_ACCEPTANCE_PROBABILITY)), power));
  if((current_estimate_acceptance - DESIRED_ACCEPTANCE_PROBABILITY) < acceptable_error)
    return new_temp;
  else
    return get_temp(new_temp, p, acceptable_error, set_of_transitions);
}

// Generates a random solution
void random_solution_generator(task* solution, int no_of_mult_tasks, int* mult_nodes_list,
↪  resource* resource_library, resource* multipliers, resource* adders, int
↪  no_of_mult_resources){
  for(int i=0; i<NUM_TASKS; i++){
    solution[i].id = i; // Assigning ID
    // Assigning type according to the benchmark inputs and Randomly assign a fitting
    ↪  resource
    if((i==0) || (i==(NUM_TASKS-1))){ // Assigning start and sink nodes NOP
      solution[i].type = NOP;
      solution[i].assigned_resource = &resource_library[NUM_RESOURCES-1];
    }
    else{
      int multype = 0;
      for (int j=0; j< no_of_mult_tasks; j++){
        if(i==mult_nodes_list[j]){
          solution[i].type = MUL;
          // Get a random multiplier
          solution[i].assigned_resource = &multipliers[get_rand_int_in_range(1,
          ↪  no_of_mult_resources) - 1];
          multype = 1; // Flag that the task is MUL
        }
      }
      if(!multype){
        solution[i].type = ADD;
        // Get a random adder
        solution[i].assigned_resource = &adders[get_rand_int_in_range(1, NUM_RESOURCES -
        ↪  no_of_mult_resources - 1) - 1];
      }
    }

    //Randomly assign a voltage:
    solution[i].assigned_voltage = get_rand_int_in_range(1, V_LEVELS);
    // Assigning start time
    solution[i].start_time = 0;

    // Changed for duplication:
    solution[i].assigned_duplicate_resource = NULL;
    solution[i].assigned_duplicate_voltage = -1;

  }
}

// Generates area and latency-aware random solution
void optAL_solution_generator(task* solution, int no_of_mult_tasks, int* mult_nodes_list,
↪  resource* resource_library, resource* multipliers, resource* adders, int
↪  no_of_mult_resources){
  for(int i=0; i<NUM_TASKS; i++){
    solution[i].id = i; // Assigning ID
    // Assigning type according to the benchmark inputs and Randomly assign a fitting
    ↪  resource
    if((i==0) || (i==(NUM_TASKS-1))){ // Assigning start and sink nodes NOP
      solution[i].type = NOP;
      solution[i].assigned_resource = &resource_library[NUM_RESOURCES-1];
    }
    else{
      int multype = 0;
      for (int j=0; j< no_of_mult_tasks; j++){
        if(i==mult_nodes_list[j]){
          solution[i].type = MUL;
          // Get THE min_d multiplier
          double min_d = MAXINT;
          int min_d_id;
          // Find the minA multiplier
          double min_area = MAXINT;
          int min_a_id;
          for (int k = 0; k < no_of_mult_resources; k++) {
            if (min_area > multipliers[k].area) {
              min_area = multipliers[k].area;
              min_a_id = k;
            }
            if (min_d > multipliers[k].latency[HIGH-1]) {
              min_d = multipliers[k].latency[HIGH-1];
              min_d_id = k;
            }
          }
          // pick randomly to add either of those 2
          int coin_flip = get_rand_int_in_range(1, 2);
          if(coin_flip==1){
            solution[i].assigned_resource = &multipliers[min_a_id];
          }
          else{
            solution[i].assigned_resource = &multipliers[min_d_id];
          }
          multype = 1; // Flag that the task is MUL
        }
      }
      if(!multype){
        solution[i].type = ADD;

        // Get a minD adder
        double min_d = MAXINT;
        int min_d_id;
        // Get a minA adder
        double min_area = MAXINT;
        int min_a_id;
        for (int k = 0; k < (NUM_RESOURCES - no_of_mult_resources - 1); k++) {
          if (min_area > adders[k].area) {
            min_area = adders[k].area;
            min_a_id = k;
          }
          if (min_d > adders[k].latency[HIGH-1]) {
            min_d = adders[k].latency[HIGH-1];
            min_d_id = k;
          }
        }
        //TODO: pick randomly to add either of those 2
        int coin_flip = get_rand_int_in_range(1, 2);
        if(coin_flip==1){
          solution[i].assigned_resource = &adders[min_a_id];
        }
        else{
          solution[i].assigned_resource = &adders[min_d_id];
        }
      }
    }

    solution[i].assigned_voltage = HIGH;
    solution[i].start_time = 0;

    // Changed for duplication:
    solution[i].assigned_duplicate_resource = NULL;
    solution[i].assigned_duplicate_voltage = -1;

  }
}

// Returns the objective function value of a solution
double get_objective(task* solution, double MaxR, double MinR, double MaxE, double MinE){
  double reliability = 0.0, energy = 0.0, mult_reliability = 1.0;
  for (int i = 0; i<NUM_TASKS; i++){
    reliability = reliability +
    ↪   solution[i].assigned_resource->reliability[solution[i].assigned_voltage-1];
    mult_reliability = mult_reliability *
    ↪   solution[i].assigned_resource->reliability[solution[i].assigned_voltage-1];
    energy = energy + solution[i].assigned_resource->
    ↪   energy[solution[i].assigned_voltage-1];
  }
  reliability = (reliability-MinR) / (MaxR - MinR);
  energy = (energy-MinE) / (MaxE - MinE);
  return ALPHA*(1-reliability)+(1-ALPHA)*energy;
}

// Prints the objective function value of a solution
double print_objective(task* solution, double MaxR, double MinR, double MaxE, double MinE){
  double reliability = 0.0, energy = 0.0, mult_reliability = 1.0;
  for (int i = 0; i<NUM_TASKS; i++){
    reliability = reliability +
    ↪  solution[i].assigned_resource->reliability[solution[i].assigned_voltage-1];
    mult_reliability = mult_reliability *
    ↪  solution[i].assigned_resource->reliability[solution[i].assigned_voltage-1];
    energy = energy + solution[i].assigned_resource->energy[solution[i].assigned_voltage-1];
  }
  printf("Reliability MULT: %lf, Energy FULL: %lf\n", mult_reliability, energy);
  reliability = (reliability-MinR) / (MaxR - MinR);
  energy = (energy-MinE) / (MaxE - MinE);
  printf("normR = %lf, normE = %lf\n", reliability, energy);
  printf("Objective of the candidate solution: %lf\n",
  ↪  ALPHA*(1-reliability)+(1-ALPHA)*energy);
  return ALPHA*(1-reliability)+(1-ALPHA)*energy;
}

// Returns the area of the solution based on the pipelined approach
int get_area(task* solution, resource* resource_library){
  int area = 0;
  int c_steps = solution[NUM_TASKS-1].start_time;
  int count_of_resources_at_cStep[NUM_RESOURCES][V_LEVELS][c_steps+1];
  for (int i = 0; i < NUM_RESOURCES; i++) {
    for (int j = 0; j < V_LEVELS; j++) {
      for (int k = 0; k < c_steps+1; k++) {
        count_of_resources_at_cStep[i][j][k] = 0;
      }
    }
  }

  //Counting how many of each resource are used in every Cstep (different voltage levels
  ↪  require different resource even if it's the same one)
  for (int i = 0; i<c_steps+1; i++){
    for (int j = 0; j < NUM_TASKS; j++) {
      if(solution[j].start_time == i){
        count_of_resources_at_cStep[(solution[j].assigned_resource->id)-1]
        ↪  [(solution[j].assigned_voltage)-1][i]++; //Have to add -1 because ids and
        ↪  voltage levels start from 1
        if (solution[j].assigned_duplicate_resource != NULL) { //CHECKING FOR DUPLICATIED
        ↪  RESOURCES
            count_of_resources_at_cStep[(solution[j].assigned_duplicate_resource->id)-1]
            ↪  [(solution[j].assigned_duplicate_voltage)-1][i]++;
        }
      }
    }
  }

  // Check each control step and find the one which induces the max area: because pipelined
  ↪  approach is considered.
  int max_area = 0;
  int max_count_of_any_R_at_v[NUM_RESOURCES][V_LEVELS];
  for (int i = 0; i < NUM_RESOURCES; i++) {
    for (int j = 0; j < V_LEVELS; j++) {
      max_count_of_any_R_at_v[i][j] = 0;
    }
  }

  for (int i = 0; i < NUM_RESOURCES; i++) {
    int max_count_for_resource_HIGH = 0;
    int max_count_for_resource_LOW = 0;
    for (int cs = 0; cs < c_steps; cs++) {
      if (count_of_resources_at_cStep[i][HIGH-1][cs] > max_count_for_resource_HIGH) {
        max_count_for_resource_HIGH = count_of_resources_at_cStep[i][HIGH-1][cs];
      }
      if (count_of_resources_at_cStep[i][LOW-1][cs] > max_count_for_resource_LOW) {
        max_count_for_resource_LOW = count_of_resources_at_cStep[i][LOW-1][cs];
      }
    }
    max_count_of_any_R_at_v[i][HIGH-1] = max_count_for_resource_HIGH;
    max_count_of_any_R_at_v[i][LOW-1] = max_count_for_resource_LOW;
  }

  for (int i = 0; i < NUM_RESOURCES; i++) {
    for (int j = 0; j < V_LEVELS; j++) {
      max_area = max_area + max_count_of_any_R_at_v[i][j]*resource_library[i].area;
    }
  }

  return max_area;
}

// Reduces area of the design by rescheduling from crowded control step
void reduce_area_by_rescheduling(task* solution, resource* resource_library, int*
↪ topological_ordering, int* ASAP_scheduling, int* ALAP_scheduling, int* list_scheduling,
↪ int number_of_tasks, int (*graph)[number_of_tasks]){

  int c_steps = solution[NUM_TASKS-1].start_time;
  int scheduled_tasks_count_at_cStep[c_steps];

  for (int k = 0; k < c_steps+1; k++) {
    scheduled_tasks_count_at_cStep[k] = 0;
  }
  // Count the number of shceduled tasks at each cStep
  for (int k = 0; k < c_steps+1; k++) {
    for (int j = 0; j < NUM_TASKS; j++) {
      if (solution[j].start_time == k) {
        scheduled_tasks_count_at_cStep[k]++;
      }
    }
  }

  // Find the most max_crowded_cStep NOT USED!!!
  int max_crowded_cStep = -1;
  int crowd = -1;
  for (int k = 0; k < c_steps+1; k++) {
    if (crowd<scheduled_tasks_count_at_cStep[k]) {
      max_crowded_cStep = k;
      crowd = scheduled_tasks_count_at_cStep[k];
    }
  }

  // Find the most max_area_cStep
  int max_area_cStep = -1;
  int max_area = 0;
  for (int k = 0; k < c_steps+1; k++) {
    int area = 0;
    for (size_t j = 0; j < NUM_TASKS; j++) {
      if (solution[j].start_time == k) {
        area = area + resource_library[solution[j].assigned_resource->id].area;
        // Check for duplication
        if (solution[j].assigned_duplicate_resource!=NULL) {
          area = area + resource_library[solution[j].assigned_duplicate_resource->id].area;
        }
      }
    }
    if(area>max_area){
      max_area = area;
      max_area_cStep = k;
    }
  }

  int moved[NUM_TASKS];
  int task_to_move = -1;
  int slack = -1;
  for (int j = 0; j < NUM_TASKS; j++) {
    moved[j] = 0;
    if (solution[j].start_time == max_area_cStep) {
      if ((ALAP_scheduling[j]-ASAP_scheduling[j])>slack) {
        task_to_move = j;
        slack = ALAP_scheduling[j]-ASAP_scheduling[j];
      }
    }
  }

  // move task_to_move and all dependent tasks recursively!
  recursively_shift_dependent_tasks(task_to_move, moved, solution, resource_library,
  ↪  topological_ordering, ASAP_scheduling, ALAP_scheduling, list_scheduling,
  ↪  number_of_tasks, graph);
}

// Recursively shifts rescheduled task and the affected successors
void recursively_shift_dependent_tasks(int task_to_move, int* moved, task* solution,
↪  resource* resource_library, int* topological_ordering, int* ASAP_scheduling, int*
↪  ALAP_scheduling, int* list_scheduling, int number_of_tasks, int
↪  (*graph)[number_of_tasks]){
  if (moved[task_to_move]==0) {
    solution[task_to_move].start_time++;
    list_scheduling[task_to_move]++;
    moved[task_to_move] = 1;
    for (int i = 0; i < NUM_TASKS; i++) {
      if (graph[task_to_move][i]==1) {
        recursively_shift_dependent_tasks(i, moved, solution, resource_library,
        ↪  topological_ordering, ASAP_scheduling, ALAP_scheduling, list_scheduling,
        ↪  number_of_tasks, graph);
      }
    }
  }
}

```

```
// Reading resource library input file
int read_library(resource* resource_library, int* no_of_mult_resources, const char*
↪  filename){
  FILE *pf;
  pf = fopen (filename, "r");
  if (pf == NULL){
    return 0;
  }

  char buf[100];
  fscanf(pf,"%s",buf);
  *no_of_mult_resources = 0;
  for(int i=0; i<NUM_RESOURCES; i++){
    resource_library[i].id = i+1;
    fscanf(pf, "%d", &resource_library[i].type);
    if(resource_library[i].type == MUL){
      *no_of_mult_resources = *no_of_mult_resources + 1;
    }
  }
  fscanf(pf,"%s",buf);
  for(int i=0; i<NUM_RESOURCES; i++){
    fscanf(pf, "%d", &resource_library[i].area);
  }
  fscanf(pf,"%s",buf);
  //printf("%s\n", buf);
  for(int i=0; i<NUM_RESOURCES; i++){
    for(int j=0; j<V_LEVELS; j++){
      fscanf(pf, "%lf", &resource_library[i].reliability[j]);
    }
  }
  fscanf(pf,"%s",buf);
  for(int i=0; i<NUM_RESOURCES; i++){
    for(int j=0; j<V_LEVELS; j++){
      fscanf(pf, "%lf", &resource_library[i].energy[j]);
    }
  }
  fscanf(pf,"%s",buf);
  for(int i=0; i<NUM_RESOURCES; i++){
    for(int j=0; j<V_LEVELS; j++){
      fscanf(pf, "%d", &resource_library[i].latency[j]);
    }
  }
  fclose (pf);
  return 1;
}

//prints solution
void print_solution(task* solution){
  for(int i=0; i<NUM_TASKS; i++){
    printf("Task id: %d, Type: %d, assigned_resource: %d, assigned_voltage: %d, delay %d,
    ↪  start_time: %d\n", solution[i].id, solution[i].type,
    ↪  solution[i].assigned_resource->id, solution[i].assigned_voltage,
    ↪  solution[i].assigned_resource->latency[solution[i].assigned_voltage-1],
    ↪  solution[i].start_time);
    if(solution[i].assigned_duplicate_resource!=NULL) printf("assigned_duplicate_resource:
    ↪  %d, assigned_duplicate_voltage: %d, delay: %d\n",
    ↪  solution[i].assigned_duplicate_resource->id, solution[i].assigned_duplicate_voltage,
    ↪  solution[i].assigned_duplicate_resource->
    ↪  latency[solution[i].assigned_duplicate_voltage-1]);
  }
}

// Prints the resources
void print_resources(resource* resource_library, int num_res){
  for(int i=0; i< num_res; i++){
    printf("ID: %d, Type: %d, area: %d\n", resource_library[i].id, resource_library[i].type,
    ↪  resource_library[i].area);
    for (int j=0; j<V_LEVELS; j++){
      printf("voltage_level (1=HIGH, 2=LOW): %d, R: %lf, E: %lf, L: %d\n", j+1,
      ↪  resource_library[i].reliability[j], resource_library[i].energy[j],
      ↪  resource_library[i].latency[j]);
    }
    printf("\n");
  }
}

// Get precedent tasks
void get_preceding_tasks(int number_of_tasks, int (*graph)[number_of_tasks], int**
↪  predecessor_tasks_id_list, int* number_of_predecessors){
  for(int j = 0; j < number_of_tasks; j++){
    number_of_predecessors[j] = 0;
    for (int i = 0; i < number_of_tasks; i++) {
      if (graph[i][j]) {
        number_of_predecessors[j]++;
        if (number_of_predecessors[j]==1) {
          predecessor_tasks_id_list[j] = (int*)malloc(sizeof(int));
          predecessor_tasks_id_list[j][number_of_predecessors[j]-1] = i;
        }
        else{
          predecessor_tasks_id_list[j] = (int*)realloc(predecessor_tasks_id_list[j],
          ↪  number_of_predecessors[j] * sizeof(int));
          predecessor_tasks_id_list[j][number_of_predecessors[j]-1] = i;
        }
      }
    }
  }
}

// Get ASAP Scheduling of a solution and return the latency
int get_ASAP_scheduling(task* solution, int number_of_tasks, int (*graph)[number_of_tasks],
↪  int* ASAP_scheduling, int* topological_ordering){
  int last_scheduled_idx = 0, idx_to_be_scheduled = 1;
  int scheduled[NUM_TASKS];
  scheduled[0] = 1; // Set start note as scheduled because it's start time is already 0
  ASAP_scheduling[0] = 0;
  for(int i=1; i<NUM_TASKS; i++){
    scheduled[i] = 0;
  }

  while (scheduled[NUM_TASKS-1] == 0) {
    // Scheduling
    int idx_task_to_be_scheduled = topological_ordering[idx_to_be_scheduled];
    int max_Cstep_so_far = 0;

    for(int i = 0; i < NUM_TASKS; i++){
      if (graph[i][idx_task_to_be_scheduled]){
        int predecessor_delay =
        ↪  solution[i].assigned_resource->latency[solution[i].assigned_voltage-1];
        if(solution[i].assigned_duplicate_resource != NULL){
          int duplicate_predecessor_delay = solution[i].assigned_duplicate_resource->
          ↪  latency[solution[i].assigned_duplicate_voltage-1];
          if (duplicate_predecessor_delay > predecessor_delay) {
            predecessor_delay = duplicate_predecessor_delay;
          }
        }
        predecessor_delay = predecessor_delay + solution[i].start_time;
        if(predecessor_delay > max_Cstep_so_far){
          max_Cstep_so_far = predecessor_delay;
        }
      }
    }

    if (max_Cstep_so_far == 0) {
      solution[idx_task_to_be_scheduled].start_time = max_Cstep_so_far + 1;
      ASAP_scheduling[idx_task_to_be_scheduled] = max_Cstep_so_far + 1;
    }
    else{
      solution[idx_task_to_be_scheduled].start_time = max_Cstep_so_far;
      ASAP_scheduling[idx_task_to_be_scheduled] = max_Cstep_so_far;
    }

    scheduled[idx_task_to_be_scheduled] = 1;
    idx_to_be_scheduled++;
  }
  return solution[topological_ordering[NUM_TASKS-1]].start_time - 1; // Return the resulting
  ↪  latency
}

// Get ALAP Scheduling of a solution
int get_ALAP_scheduling(task* solution, int number_of_tasks, int (*graph)[number_of_tasks],
↪  int* ALAP_scheduling, int* topological_ordering, int limit_latency){
  int last_scheduled_idx = NUM_TASKS-1, idx_to_be_scheduled = last_scheduled_idx - 1;
  int scheduled[NUM_TASKS];
  scheduled[NUM_TASKS-1] = 1; // Set sink  note as scheduled because it's start time is the
  ↪  given limit latency calculated from ASAP
  ALAP_scheduling[NUM_TASKS-1] = limit_latency;
  for(int i=0; i<NUM_TASKS-1; i++){
    scheduled[i] = 0;
  }

  while (scheduled[0] == 0) {
    // Scheduling
    int idx_task_to_be_scheduled = topological_ordering[idx_to_be_scheduled]; // Going in
    ↪  reverse topological ordering
    int min_start_time = limit_latency;

    for(int i = 0; i < NUM_TASKS; i++){
      if (graph[idx_task_to_be_scheduled][i]) {
        if (solution[i].start_time < min_start_time){
          min_start_time = solution[i].start_time;
        }
      }
    }

    int delay_to_subtract = solution[idx_task_to_be_scheduled].assigned_resource->
    ↪  latency[solution[idx_task_to_be_scheduled].assigned_voltage-1];
    if(solution[idx_task_to_be_scheduled].assigned_duplicate_resource != NULL){
      int duplicate_delay_to_subtract =
      ↪  solution[idx_task_to_be_scheduled].assigned_duplicate_resource->
      ↪  latency[solution[idx_task_to_be_scheduled].assigned_duplicate_voltage-1];
      if (duplicate_delay_to_subtract > delay_to_subtract) {
        delay_to_subtract = duplicate_delay_to_subtract;
      }
    }

    if ((min_start_time - delay_to_subtract) == limit_latency) {
      solution[idx_task_to_be_scheduled].start_time = min_start_time - delay_to_subtract -
      ↪  1;
      ALAP_scheduling[idx_task_to_be_scheduled] = min_start_time - delay_to_subtract - 1;
    }
    else{
      solution[idx_task_to_be_scheduled].start_time = min_start_time - delay_to_subtract;
      ALAP_scheduling[idx_task_to_be_scheduled] = min_start_time - delay_to_subtract;
    }
    if (idx_task_to_be_scheduled == 0) {
      ALAP_scheduling[idx_task_to_be_scheduled]--;
    }

    scheduled[idx_task_to_be_scheduled] = 1;
    idx_to_be_scheduled--;
  }
  return -1;
}

// Get list Scheduling for minimum resource usage of a solution given the latency
↪  constraint, returns -1 if solution is not feasable
int list_scheduling_min_resource_usage(task* solution, resource* resource_library, int
↪  number_of_tasks, int (*graph)[number_of_tasks], int* ALAP_scheduling, int*
↪  list_scheduling, int* topological_ordering, int limit_latency){
  // Check that ALAP provided feasable solution:
  if (ALAP_scheduling[0] < 0) {
    return -1;
  }
  int area = 0, num_scheduled_tasks = 1;
  int scheduled[NUM_TASKS], ready[NUM_TASKS];
  scheduled[0] = 1; // Set sink  note as scheduled
  ready[0] = 1;
  list_scheduling[0] = 0;
  solution[0].start_time = 0;
  int c_step = 1; // Start at cStep 1 (disregarding starter node already considered)
  for(int i=1; i<NUM_TASKS-1; i++){
    scheduled[i] = 0;
    ready[i] = 0;
  }

  // Setting up for area calculation
  int used_resources_count_per_voltage_level[NUM_RESOURCES][V_LEVELS];
  for (int i = 0; i < NUM_RESOURCES; i++) {
    for (int j = 0; j < V_LEVELS; j++) {
      used_resources_count_per_voltage_level[i][j] = 0;
    }
  }

  // list_scheduling algorithm
  while(num_scheduled_tasks != NUM_TASKS-1){
    for (int i = 1; i < NUM_TASKS; i++) {
      int unscheduled_predecessor_found = 0;
      for (int j = 0; j < NUM_TASKS; j++) {
        if ((graph[j][i]) && (scheduled[j]==0)){
          unscheduled_predecessor_found = 1;
        }
        int pred_end_time = solution[j].start_time +
        ↪  solution[j].assigned_resource->latency[solution[j].assigned_voltage-1] - 1;
        if (solution[j].assigned_duplicate_resource != NULL) {
          int pred_end_time_d = solution[j].start_time +
          ↪  solution[j].assigned_duplicate_resource->
          ↪  latency[solution[j].assigned_duplicate_voltage-1] - 1;
          if(pred_end_time_d > pred_end_time){
            pred_end_time = pred_end_time_d;
          }
        }
        if ((graph[j][i]) && (scheduled[j]==1 && pred_end_time>c_step)){
          unscheduled_predecessor_found = 1;
        }
      }
      if (!unscheduled_predecessor_found) {
        ready[i] = 1;
      }
    }

    // Need to be reset in every cStep:
    int usedResCount_perVdd_perCStep[NUM_RESOURCES][V_LEVELS];
    int count_of_available_resources[NUM_RESOURCES][V_LEVELS];
    for (int i = 0; i < NUM_RESOURCES; i++) {
      for (int j = 0; j < V_LEVELS; j++) {
        usedResCount_perVdd_perCStep[i][j] = 0; // No scheduled task yet in this cStep
        count_of_available_resources[i][j] = used_resources_count_per_voltage_level[i][j];
        ↪  // Set all of them available at the beginning of the new Cstep
      }
    }

    for (int i = 1; i < NUM_TASKS; i++){
      if (ready[i] && (scheduled[i] == 0) && ((solution[i].start_time - c_step) == 0)) {
        solution[i].start_time = c_step;
        list_scheduling[i] = c_step;
        scheduled[i] = 1;
        num_scheduled_tasks++;
        usedResCount_perVdd_perCStep[(solution[i].assigned_resource->id)-1]
        ↪  [(solution[i].assigned_voltage)-1]++;
        if(usedResCount_perVdd_perCStep[(solution[i].assigned_resource->id)-1]
        ↪  [(solution[i].assigned_voltage)-1] >
        ↪  count_of_available_resources[(solution[i].assigned_resource->id)-1]
        ↪  [(solution[i].assigned_voltage)-1]) {
            used_resources_count_per_voltage_level[(solution[i].assigned_resource->id)-1]
            ↪  [(solution[i].assigned_voltage)-1]++; //Update the count of used resources
        }
        else{
            count_of_available_resources[(solution[i].assigned_resource->id)-1]
            ↪  [(solution[i].assigned_voltage)-1]--;
        }

        if(solution[i].assigned_duplicate_resource != NULL){
            usedResCount_perVdd_perCStep[(solution[i].assigned_duplicate_resource->id)-1]
            ↪  [(solution[i].assigned_duplicate_voltage)-1]++;
            if(usedResCount_perVdd_perCStep[(solution[i].assigned_duplicate_resource->id)-1]
            ↪  [(solution[i].assigned_duplicate_voltage)-1] >
            ↪  count_of_available_resources[(solution[i].assigned_duplicate_resource->id)-1]
            ↪  [(solution[i].assigned_duplicate_voltage)-1]) {

                used_resources_count_per_voltage_level
                ↪  [(solution[i].assigned_duplicate_resource->id)-1]
                ↪  [(solution[i].assigned_duplicate_voltage)-1]++; //Update the count of
                ↪  used resources
          }
          else{
               count_of_available_resources[(solution[i].assigned_duplicate_resource->id)-1]
               ↪  [(solution[i].assigned_duplicate_voltage)-1]--;
          }
        }
      }
    }

    //Schedule the candidate tasks that don't need additional resources.
    for (int i = 0; i < NUM_RESOURCES; i++) {
      for (int j = 0; j < V_LEVELS; j++) {
        if (count_of_available_resources[i][j] > 0) {
          int min_slack_so_far = limit_latency;
          int min_slack_task_to_schedule = -1;
          int type_of_task_to_schedule;

          for (int k = 1; k < NUM_TASKS; k++){
            if (ready[k] && (scheduled[k] == 0) && ((solution[k].assigned_voltage-1) == j))
            ↪  {
              // CHECKING IF ALL PREDECESSORS HAVE FINISHED EXECUTION
              int all_predecessors_finished = 1;
              for (int l = 0; l < NUM_TASKS; l++) {
                if ((graph[l][k])){
                  if((solution[l].start_time +
                  ↪  solution[l].assigned_resource->latency[solution[l].assigned_voltage -
                  ↪  1] - 1) >= c_step){
                    all_predecessors_finished = 0; // An unfinished predecessor found so we
                    ↪  cannot schedule k at this c_step
                  }
```

```c
                  if(solution[l].assigned_duplicate_resource != NULL){ // Checking for the duplicate too
                    if((solution[l].start_time + solution[l].assigned_duplicate_resource->latency[solution[l].assigned_duplicate_voltage-1]-1) >= c_step){
                      all_predecessors_finished = 0; // An unfinished predecessor found so we cannot schedule k at this c_step
                    }
                  }
                }
              }
              if (all_predecessors_finished && (solution[k].start_time-c_step < min_slack_so_far)) { // find the ready and unscheduled task with min slack
                min_slack_so_far = solution[k].start_time-c_step;
                min_slack_task_to_schedule = k;
                type_of_task_to_schedule = solution[k].type;
              }
            }
          }

          if ((min_slack_task_to_schedule != -1) && (type_of_task_to_schedule == resource_library[i].type)) {
            solution[min_slack_task_to_schedule].start_time = c_step;
            list_scheduling[min_slack_task_to_schedule] = c_step;
            scheduled[min_slack_task_to_schedule] = 1;
            num_scheduled_tasks++;
            count_of_available_resources[i][j]--;
          }
        }
      }
    }

    c_step++;

  }// end while

  list_scheduling[NUM_TASKS-1] = limit_latency;
  for (int i = 0; i < NUM_RESOURCES; i++) {
    for (int j = 0; j < V_LEVELS; j++) {
      if(area < used_resources_count_per_voltage_level[i][j]*resource_library[i].area)
        area = used_resources_count_per_voltage_level[i][j]*resource_library[i].area;
    }
  }
  return area;
}

// Get topological ordering of the tasks in the benchmark
void get_topological_ordering(int number_of_tasks, int (*graph)[number_of_tasks], int* topological_ordering){
  int graph_copy[NUM_TASKS][NUM_TASKS];
  int indeg[number_of_tasks], flag[number_of_tasks];
  int count = 0, idx = 0;
  for(int i = 0; i < number_of_tasks; i++){
    indeg[i] = 0;
    flag[i] = 0;
    for(int j = 0; j < number_of_tasks; j++){
      graph_copy[i][j] = graph[i][j];
    }
  }

  for(int i = 0; i < number_of_tasks; i++){
    for(int j = 0; j < number_of_tasks; j++){
      indeg[i] = indeg[i] + graph_copy[j][i];
    }
  }

  while(count < number_of_tasks){
      for(int k = 0; k < number_of_tasks; k++){
          if((indeg[k] == 0) && (flag[k] == 0)){
            topological_ordering[idx++] = k;
            flag[k] = 1;

            for(int i = 0; i < number_of_tasks; i++){
              if(graph_copy[k][i] == 1){
                graph_copy[k][i]=0;
                indeg[i]--;
              }
            }
        }
      }
        count++;
  }
}

// Generates a random candidate by changing only one task parameters (either assigned
↪   resource or voltage)
void random_candidate_generator(task* solution, task* candidate_solution, int
    ↪   no_of_mult_tasks, int* mult_nodes_list, resource* resource_library, resource*
    ↪   multipliers, resource* adders, int no_of_mult_resources){
  int task_to_change = get_rand_int_in_range(1, NUM_TASKS-2); // NOPs not important

  int coin_flip = get_rand_int_in_range(1, 2);

  for(int i=0; i<NUM_TASKS; i++){
    candidate_solution[i].id = i;
    if((i==0) || (i==(NUM_TASKS-1))){ // Assigning start and sink nodes NOP
      candidate_solution[i].type = NOP;
      candidate_solution[i].assigned_resource = &resource_library[NUM_RESOURCES-1];
      candidate_solution[i].assigned_voltage = solution[i].assigned_voltage;
    }
    else{
      if(i == task_to_change){
        int thing_to_change = get_rand_int_in_range(1, 2); // 1 - resource, 2 - voltage
        // Assigning resource
        if(solution[i].type == MUL){
          candidate_solution[i].type = MUL;
          if(thing_to_change == 1){ // Should it be changed?
            candidate_solution[i].assigned_resource = &multipliers[get_rand_int_in_range(1,
            ↪   no_of_mult_resources) - 1];
          }
          else{
            candidate_solution[i].assigned_resource = solution[i].assigned_resource;
          }
        }
        else if(solution[i].type == ADD){
          // change add resource
          candidate_solution[i].type = ADD;
          if(thing_to_change == 1){
            candidate_solution[i].assigned_resource = &adders[get_rand_int_in_range(1,
            ↪  NUM_RESOURCES - no_of_mult_resources - 1) - 1];
          }
          else{
            candidate_solution[i].assigned_resource = solution[i].assigned_resource;
          }
        }
        // Assigning voltage
        if(thing_to_change == 2){ // Should it be changed?
          candidate_solution[i].assigned_voltage = get_rand_int_in_range(1, V_LEVELS);
        }
        else{
          candidate_solution[i].assigned_voltage = solution[i].assigned_voltage;
        }
        // Assigning start time
        candidate_solution[i].start_time = 0;
      }
      else{
        candidate_solution[i].type = solution[i].type;
        candidate_solution[i].assigned_resource = solution[i].assigned_resource;
        candidate_solution[i].assigned_voltage = solution[i].assigned_voltage;

        // Change for duplication:
        candidate_solution[i].assigned_duplicate_resource = NULL;
        candidate_solution[i].assigned_duplicate_voltage = -1;
      }
    }
    // Change for duplication:
    candidate_solution[i].assigned_duplicate_resource = NULL;
    candidate_solution[i].assigned_duplicate_voltage = -1;
  }
}

// -------------- helper_functions.h below ---------------

#include <sys/time.h>
#include <math.h>

int read_matrix(size_t rows, size_t cols, int (*a)[cols], const char* filename)
{
    FILE *pf;
    pf = fopen (filename, "r");
    if (pf == NULL)
        return 0;

    for(size_t i = 0; i < rows; ++i)
    {
        for(size_t j = 0; j < cols; ++j)
            fscanf(pf, "%d", a[i] + j);
    }
```

```c

    fclose (pf);
    return 1;
}

int read_double_matrix(size_t rows, size_t cols, double (*a)[cols], const char* filename)
{
    FILE *pf;
    pf = fopen (filename, "r");
    if (pf == NULL)
        return 0;

    for(size_t i = 0; i < rows; ++i)
    {
        for(size_t j = 0; j < cols; ++j){
            fscanf(pf, "%lf", a[i] + j);
            //printf("%lf\n", *(a[i]+j));
        }
    }
    fclose (pf);
    return 1;
}

int get_rand_int_in_range(int min, int max){
  struct timeval t1;
  gettimeofday(&t1, NULL);
  srand(t1.tv_usec * t1.tv_sec);
  return (rand() % (max - min + 1)) + min;
}

float get_rand_float_in_range( float min, float max ){
    float scale = rand() / (float) RAND_MAX;
    return min + scale * ( max - min );
}

int read_benchmark(size_t rows, size_t cols, double *MaxR, double *MinR, double *MaxE,
↪   double *MinE, int* no_of_mult_tasks, int* mult_nodes_list, int (*a)[cols], const char*
↪   filename)
{
    FILE *pf;
    pf = fopen (filename, "r");
    if (pf == NULL)
        return 0;

    fscanf(pf, "%lf", MaxR);
    fscanf(pf, "%lf", MinR);
    fscanf(pf, "%lf", MaxE);
    fscanf(pf, "%lf", MinE);
    for(size_t i = 0; i < rows; ++i)
    {
        for(size_t j = 0; j < cols; ++j)
            fscanf(pf, "%d", a[i] + j);
    }
    *no_of_mult_tasks = 0;
    while (!feof (pf))
    {
      fscanf (pf, "%d", &mult_nodes_list[*no_of_mult_tasks]);
      *no_of_mult_tasks = *no_of_mult_tasks + 1;
    }
    *no_of_mult_tasks = *no_of_mult_tasks - 1;

    fclose (pf);
    return 1;
}

void get_mobility(int* ASAP_scheduling, int* ALAP_scheduling, int* mobility, int n){
  for(int i=0; i<n; i++){
    mobility[i] = ALAP_scheduling[i] - ASAP_scheduling[i];
  }
}

double get_acceptance_probability(double temp, int num_of_transitions, double
↪  (*set_of_transitions)[2]){
  double divident = 0.0, divisor = 0.0;
  int max = 1, min = 0;
  for (int i = 0; i < num_of_transitions; i++) {
    divident += exp(0.0 - (set_of_transitions[i][max]/temp));
    divisor += exp(0.0 - (set_of_transitions[i][min]/temp));
  }
  return divident/divisor;
}

double get_Boltzmann_factor_probability(double temp, double E1, double E2){
  double power = pow(10, -23);
  double kB = 1.38065 * power;
  double P = exp(-(E2-E1)/(kB*temp));
  return P;
}

double get_metropolis_acceptance(double temp, double E1, double E2){
  double P = exp(-(E2-E1)/(temp));
  return P;
}

double get_t1(int num_of_transitions, double (*set_of_transitions)[2], double
↪  DESIRED_ACCEPTANCE_PROBABILITY){
  double t_1;
  int max = 1, min = 0;
  double sum = 0.0;
  for (int i = 0; i < num_of_transitions; i++) {
    sum = set_of_transitions[i][max] - set_of_transitions[i][min];
  }
  return t_1 = 0.0 - (sum/(num_of_transitions*log(DESIRED_ACCEPTANCE_PROBABILITY)));
}

double get_reduced_temperature_GEO(double temp, double alpha){
  return temp * alpha;
}
```

# APPENDIX B - Directed Acyclic Graphs for Benchmarks

## B.1 DAG of the Differential Equation Solver Benchmark

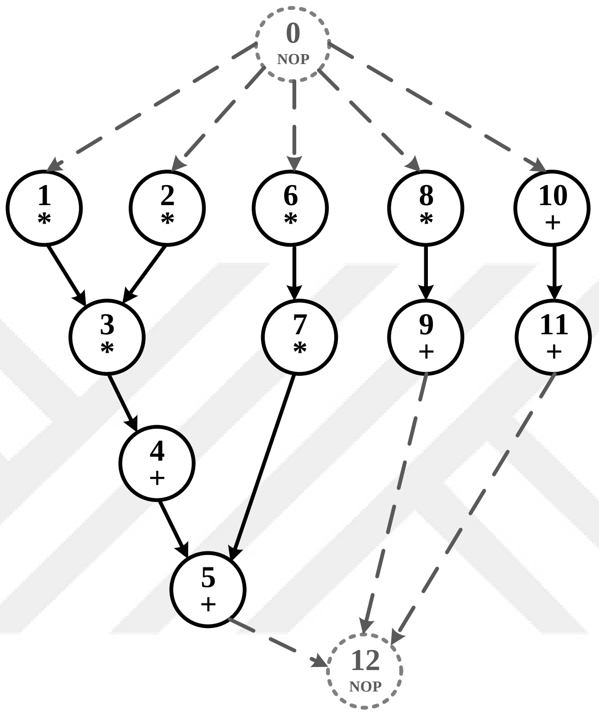


Figure B.1 DAG representation of the DES design specification.

## B.2 DAG of the Finite Impulse Response Filter Benchmark

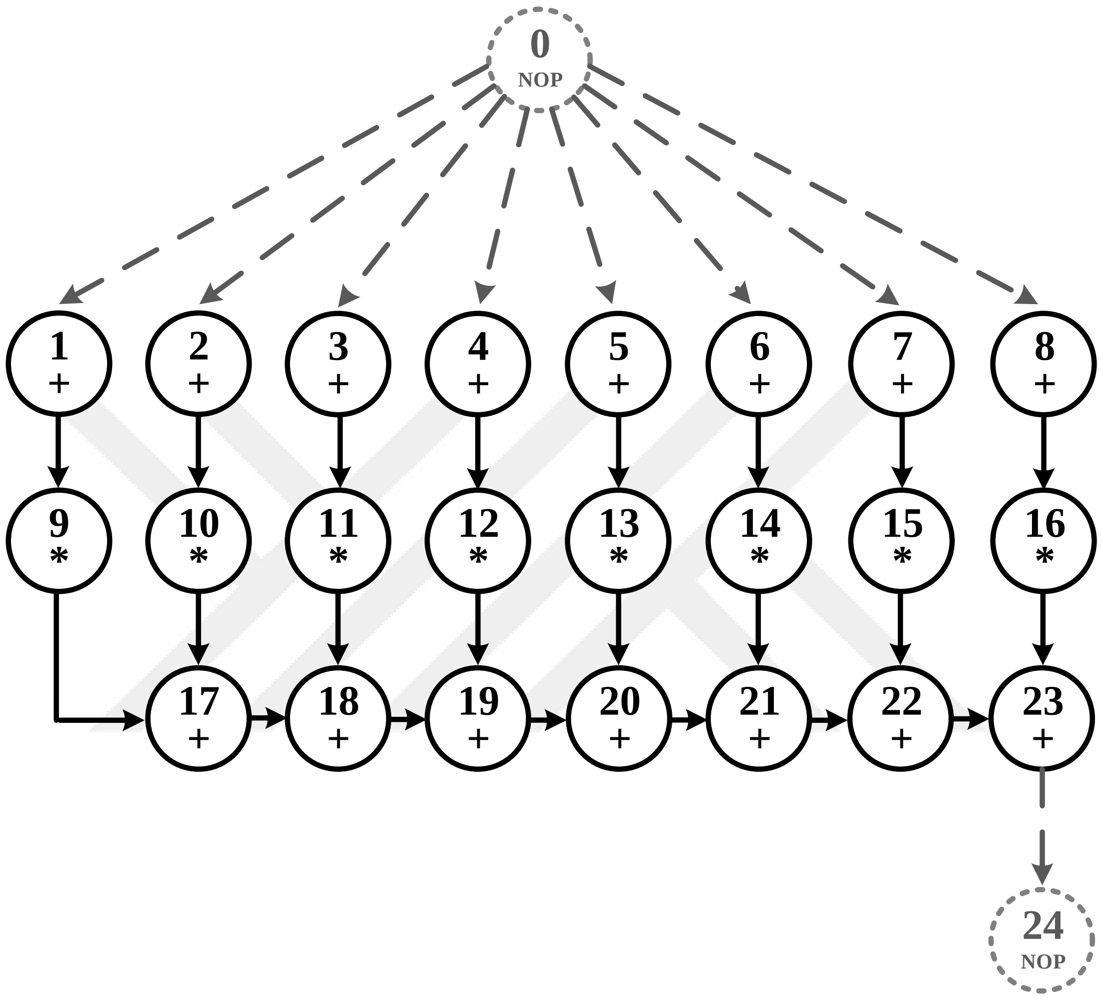


Figure B.2 DAG representation of the FIR design specification.

## B.3 DAG of the Elliptic Wave Filter Benchmark

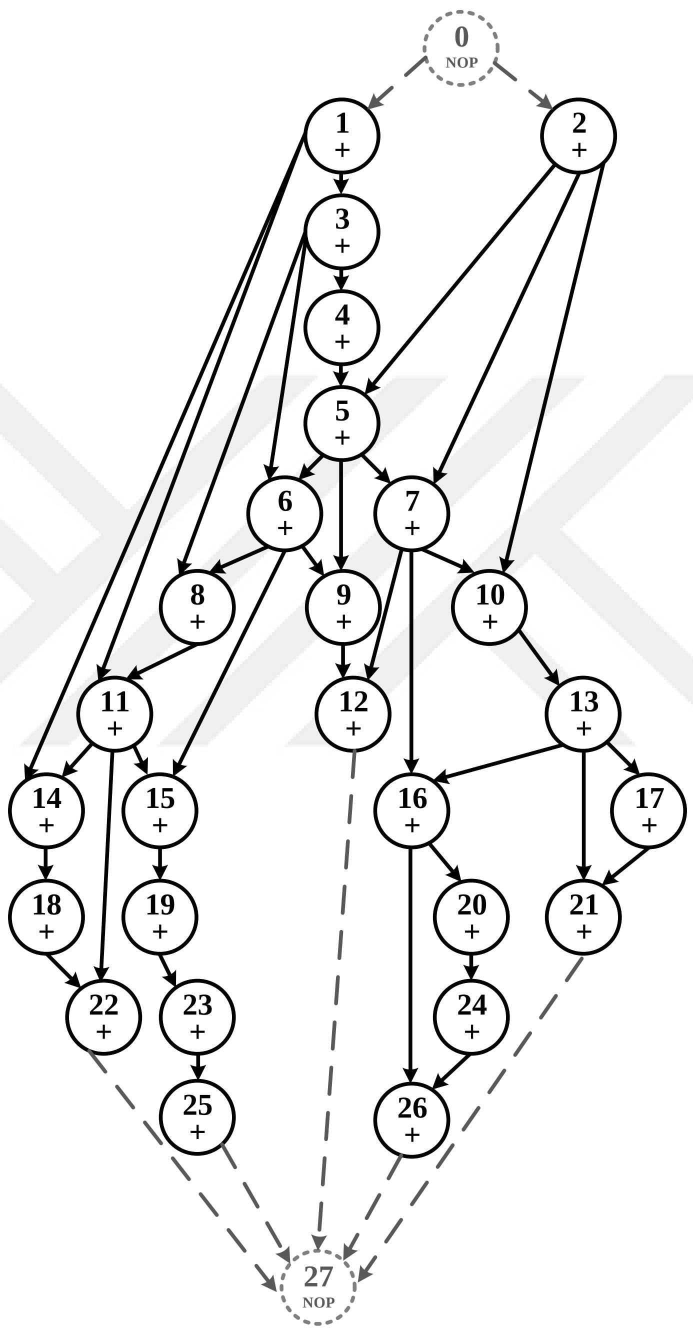


Figure B.3 DAG representation of the EWF design specification.

## B.4 DAG of the Auto-Regressive Filter Benchmark

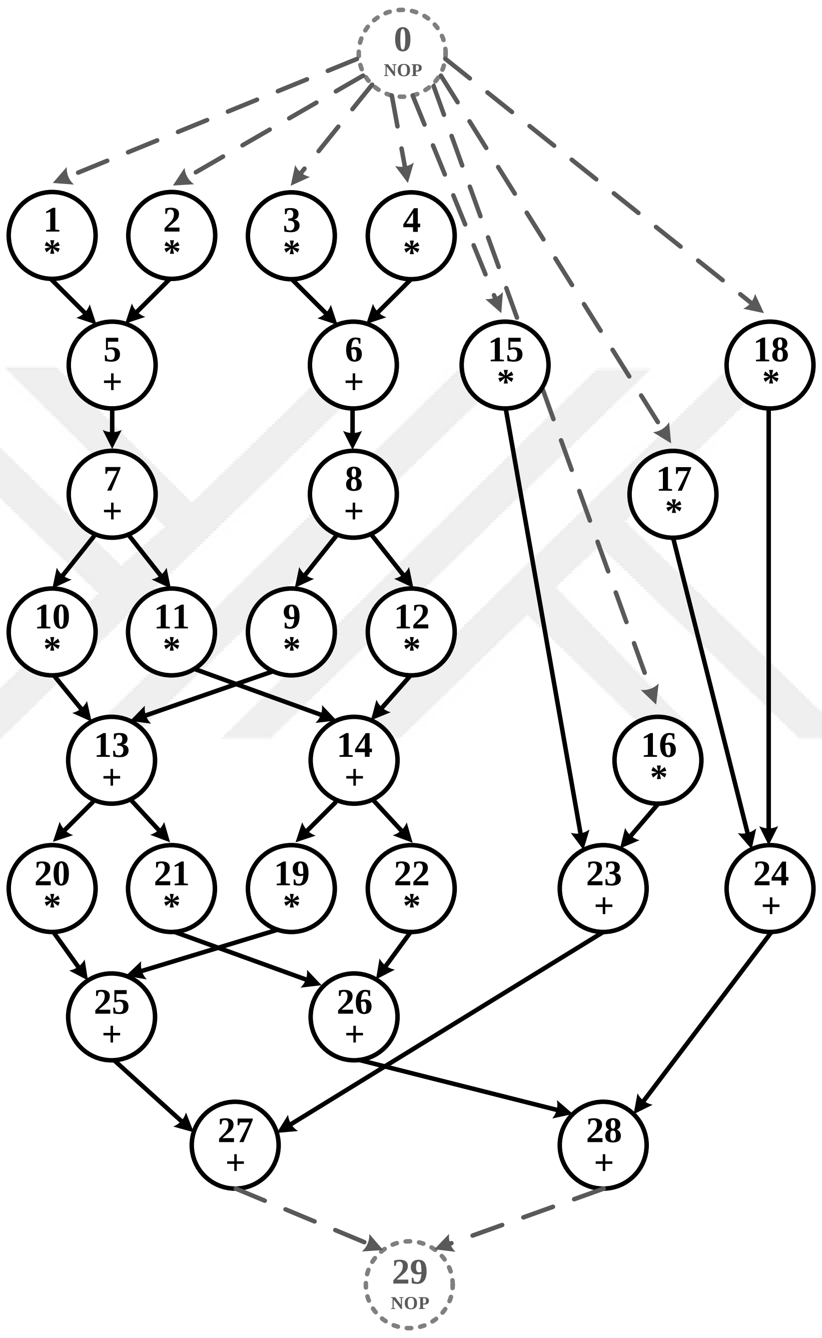


Figure B.4 DAG representation of the AR design specification.

# APPENDIX C - Additional Experimental Results

## C.1 Additional Experimental Results for Section 6.1.

Table C.1 Comparison of the energy results of ILP and GA duplication methods for all benchmarks when $\alpha = 1.0$.

| **(L, A)** | **ILP-PD** | **ILP-PD-C** | **ILP-FD** | **GA-SD** | **ILP-PD/GA-SD** | **ILP-PD-C/GA-SD** | **ILP-FD/GA-SD** |
|---|---|---|---|---|---|---|---|
| | | **DES Energy Results** | | | | $\Delta$ (%) | |
| (28,20) | 564.00 | 564.00 | 1067.80 | 629.00 | -10.33 | -10.33 | 69.76 |
| (28,30) | 1068.00 | 619.12 | 1068.00 | 629.00 | 69.79 | -1.57 | 69.79 |
| (28,40) | 1068.00 | 622.22 | 1068.00 | 629.00 | 69.79 | -1.08 | 69.79 |
| (25,20) | 558.00 | 558.00 | - | 623.00 | -10.43 | -10.43 | - |
| (25,30) | 1056.00 | 618.22 | 1056.00 | 623.00 | 69.50 | -0.77 | 69.50 |
| (25,40) | 1056.00 | 607.12 | 1056.00 | 623.00 | 69.50 | -2.55 | 69.50 |
| | | | | **Average $\Delta$ (%):** | **42.97** | **-4.46** | **69.67** |
| | | **EWF Energy Results** | | | | $\Delta$ (%) | |
| (30,10) | 312.00 | 229.23 | 312.00 | 256.00 | 21.88 | -10.46 | 21.88 |
| (30,20) | 456.00 | 255.27 | 456.00 | 256.00 | 78.13 | -0.29 | 78.13 |
| (30,30) | 456.00 | 255.27 | 456.00 | 256.00 | 78.13 | -0.29 | 78.13 |
| (40,10) | 312.00 | 312.00 | 312.00 | 316.00 | -1.27 | -1.27 | -1.27 |
| (40,20) | 552.00 | 308.84 | 552.00 | 310.00 | 78.06 | -0.37 | 78.06 |
| (40,30) | 552.00 | 315.33 | 552.00 | 316.00 | 74.68 | -0.21 | 74.68 |
| (50,10) | 575.00 | 339.96 | 582.00 | 340.00 | 69.12 | -0.01 | 71.18 |
| (50,20) | 600.00 | 339.96 | 600.00 | 340.00 | 76.47 | -0.01 | 76.47 |
| | | | | **Average $\Delta$ (%):** | **59.40** | **-1.61** | **59.66** |
| | | **FIR Energy Results** | | | | $\Delta$ (%) | |
| (30,20) | 868.00 | 868.00 | - | 1088.00 | -20.22 | -20.22 | - |
| (35,20) | 874.00 | 874.00 | - | 929.00 | -5.92 | -5.92 | - |
| (40,20) | 880.00 | 880.00 | - | 908.00 | -3.08 | -3.08 | - |
| (50,20) | 1456.00 | 892.00 | - | 998.00 | 45.89 | -10.62 | - |
| (35,30) | 1568.00 | 981.14 | 1556.00 | 989.00 | 58.54 | -0.79 | 57.33 |
| (40,30) | 1592.00 | 897.28 | 1556.00 | 908.00 | 75.33 | -1.18 | 71.37 |
| (45,30) | 1616.00 | 1010.56 | 1556.00 | 1011.00 | 59.84 | -0.04 | 53.91 |
| (50,30) | 1628.00 | 996.60 | 1556.00 | 998.00 | 63.13 | -0.14 | 55.91 |
| | | | | **Average $\Delta$ (%):** | **34.19** | **-5.25** | **59.63** |
| | | **AR Energy Results** | | | | $\Delta$ (%) | |
| (65,20) | 2848.00 | 1611.00 | 2848.00 | 1611.00 | 76.78 | 0.00 | 76.78 |
| (50,30) | 2764.00 | 1516.92 | 2764.00 | 1534.00 | 80.18 | -1.11 | 80.18 |
| (55,30) | 2800.00 | 1776.00 | 2800.00 | 1776.00 | 57.66 | 0.00 | 57.66 |
| (60,30) | 2836.00 | 1544.00 | 2836.00 | 1544.00 | 83.68 | 0.00 | 83.68 |
| (50,40) | 2764.00 | 1721.40 | 2764.00 | 1730.00 | 59.77 | -0.50 | 59.77 |
| (55,40) | 2800.00 | 1638.00 | 2800.00 | 1638.00 | 70.94 | 0.00 | 70.94 |
| | | | | **Average $\Delta$ (%):** | **71.50** | **-0.27** | **71.50** |

Table C.2 Comparison of the reliability results of ILP and GA duplication methods for all benchmarks when $\alpha = 0.0$.

| (L, A) | ILP-PD | ILP-PD-C | ILP-FD | GA-SD | ILP-PD/GA-SD | ILP-PD-C/GA-SD | ILP-FD/GA-SD |
|---|---|---|---|---|---|---|---|
| **DES Reliability Results** | | | | | | $\Delta$ (%) | |
| (28,20) | 0.881353 | 0.881353 | - | 0.878488 | 0.33 | 0.33 | - |
| (28,30) | 0.854871 | 0.907873 | 0.995197 | 0.902782 | -5.31 | 0.56 | 10.24 |
| (28,40) | 0.854871 | 0.907873 | 0.988170 | 0.902782 | -5.31 | 0.56 | 9.46 |
| (25,20) | 0.901332 | 0.957215 | - | 0.951513 | -5.27 | 0.60 | - |
| (25,30) | 0.900430 | 0.990286 | 0.997241 | 0.951513 | -5.37 | 4.07 | 4.81 |
| (25,40) | 0.890395 | 0.968549 | 0.988171 | 0.951513 | -6.42 | 1.79 | 3.85 |
| | | | | **Average $\Delta$ (%):** | **-4.56** | **1.32** | **7.09** |
| **EWF Reliability Results** | | | | | | $\Delta$ (%) | |
| (30,10) | 0.995615 | 0.885586 | 0.995615 | 0.826462 | 20.47 | 7.15 | 20.47 |
| (30,20) | 0.997625 | 0.989771 | 0.997625 | 0.826462 | 20.71 | 19.76 | 20.71 |
| (30,30) | 0.997625 | 0.989771 | 0.997625 | 0.826462 | 20.71 | 19.76 | 20.71 |
| (40,10) | 0.995615 | 0.995615 | 0.995615 | 0.973232 | 2.30 | 2.30 | 2.30 |
| (40,20) | 0.998966 | 0.995330 | 0.998966 | 0.952262 | 4.90 | 4.52 | 4.90 |
| (40,30) | 0.998966 | 0.995758 | 0.998966 | 0.973232 | 2.64 | 2.31 | 2.64 |
| (50,10) | 0.997067 | 0.996963 | 0.997097 | 0.988461 | 0.87 | 0.86 | 0.87 |
| (50,20) | 0.999638 | 0.996963 | 0.999638 | 0.988461 | 1.13 | 0.86 | 1.13 |
| | | | | **Average $\Delta$ (%):** | **9.22** | **7.19** | **9.22** |
| **FIR Reliability Results** | | | | | | $\Delta$ (%) | |
| (30,20) | 0.611809 | 0.826491 | - | 0.781564 | -21.72 | 5.75 | - |
| (35,20) | 0.638632 | 0.862726 | - | 0.798425 | -20.01 | 8.05 | - |
| (40,20) | 0.708066 | 0.827502 | - | 0.672103 | 5.35 | 23.12 | - |
| (50,20) | 0.498108 | 0.714613 | - | 0.524482 | -5.03 | 36.25 | - |
| (35,30) | 0.649775 | 0.846371 | 0.992996 | 0.759082 | -14.40 | 11.50 | 30.82 |
| (40,30) | 0.603160 | 0.800247 | 0.991731 | 0.585273 | 3.06 | 36.73 | 69.45 |
| (45,30) | 0.539284 | 0.791328 | 0.961597 | 0.579129 | -6.88 | 36.64 | 66.04 |
| (50,30) | 0.478715 | 0.686790 | 0.955639 | 0.495619 | -3.41 | 38.57 | 92.82 |
| | | | | **Average $\Delta$ (%):** | **-7.88** | **24.58** | **64.78** |
| **AR Reliability Results** | | | | | | $\Delta$ (%) | |
| (65,20) | 0.792693 | 0.801310 | 0.999972 | 0.793572 | -0.11 | 0.98 | 26.01 |
| (50,30) | 0.784443 | 0.835048 | 0.994783 | 0.831569 | -5.67 | 0.42 | 19.63 |
| (55,30) | 0.773373 | 0.804704 | 0.988506 | 0.802083 | -3.58 | 0.33 | 23.24 |
| (60,30) | 0.796366 | 0.806719 | 0.988503 | 0.802888 | -0.81 | 0.48 | 23.12 |
| (50,40) | 0.756717 | 0.868565 | 0.990628 | 0.859713 | -11.98 | 1.03 | 15.23 |
| (55,40) | 0.770776 | 0.822441 | 0.991069 | 0.820738 | -6.09 | 0.21 | 20.75 |
| | | | | **Average $\Delta$ (%):** | **-4.71** | **0.57** | **21.33** |

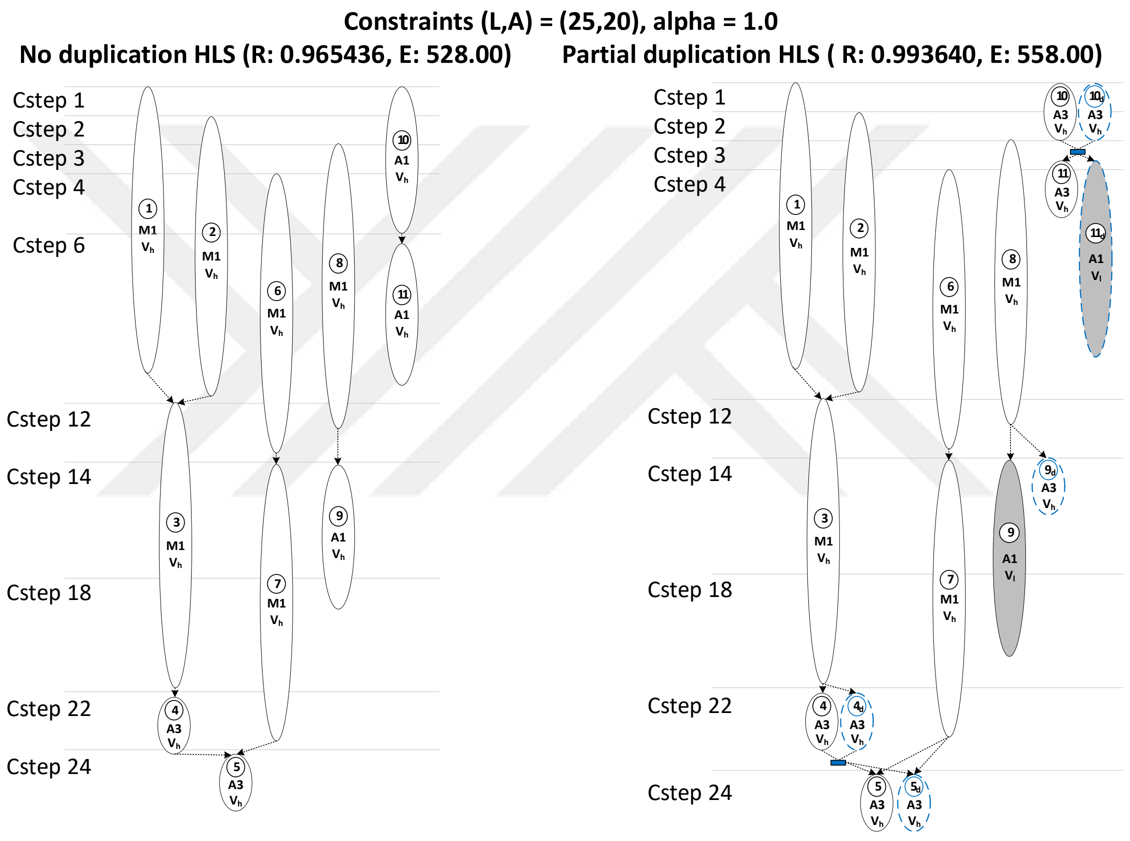

Constraints (L,A) = (25,20), alpha = 1.0
No duplication HLS (R: 0.965436, E: 528.00)
Partial duplication HLS ( R: 0.993640, E: 558.00)
Cstep 1
Cstep 2
Cstep 3
Cstep 4
Cstep 6
Cstep 12
Cstep 14
Cstep 18
Cstep 22
Cstep 24
M1
A1
A3
Vh
Vl

Figure C.1 Solutions without and with partial DMR obtained from the ILP models for the DES benchmark when $\alpha = 1.0$ under the constraints $A = 20, L = 25$.

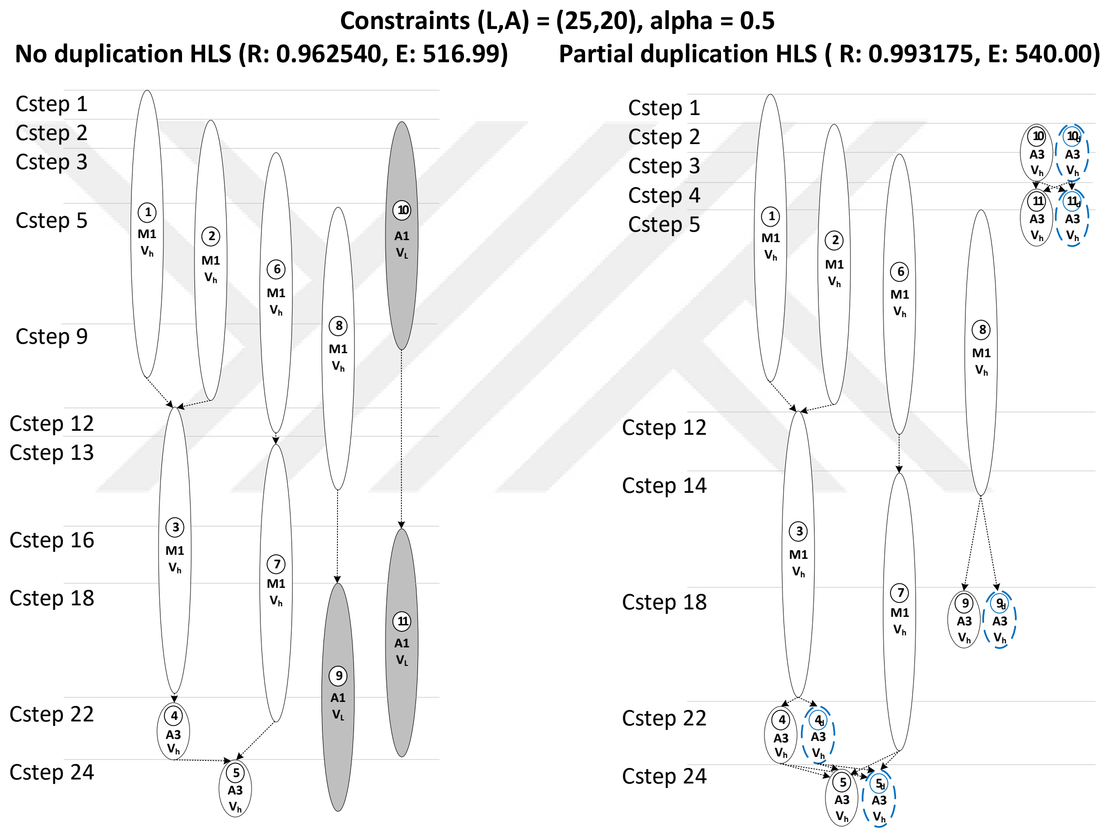

Constraints (L,A) = (25,20), alpha = 0.5
No duplication HLS (R: 0.962540, E: 516.99)
Partial duplication HLS ( R: 0.993175, E: 540.00)
Cstep 1
Cstep 2
Cstep 3
Cstep 4
Cstep 5
Cstep 9
Cstep 12
Cstep 13
Cstep 14
Cstep 16
Cstep 18
Cstep 22
Cstep 24
M1
A1
A3
$V_h$
$V_L$

Figure C.2 Solutions without and with partial DMR obtained from the ILP models for the DES benchmark when $\alpha = 0.5$ under the constraints $A = 20, L = 25$.

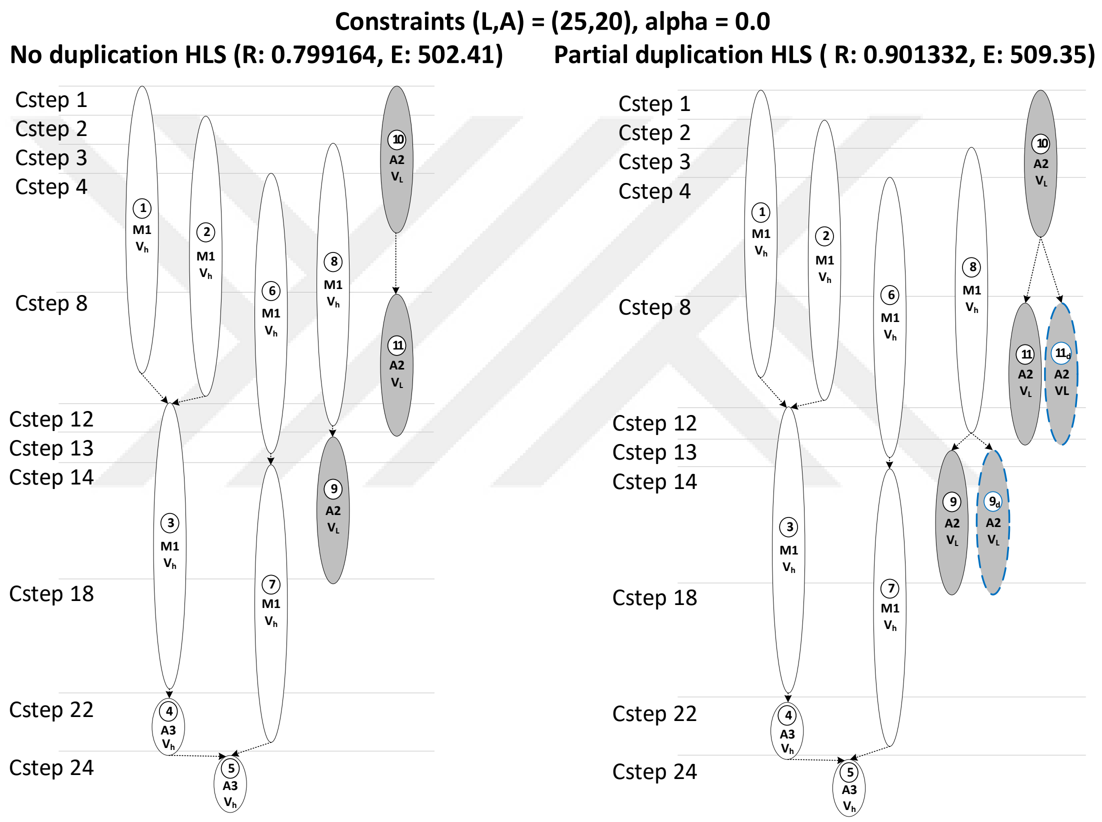


Figure C.3 Solutions without and with partial DMR obtained from the ILP models for the DES benchmark when $\alpha = 0.0$ under the constraints $A = 20, L = 25$.

## C.2 Additional Experimental Results for Section 6.2.

Table C.3 Comparison of the energy results of SA-based method with ILP and GA-based methods for all benchmarks when $\alpha = 1.0$.

| **(L, A)** | **SA-ND** | **ILP-ND** | **GA-ND** | **SA-ND/ILP-ND** | **SA-ND/GA-ND** |
|---|---|---|---|---|---|
| **DES Energy Results** | | | | $\Delta$ (%) | |
| (31,20) | 540.00 | 540.00 | 540.00 | 0.00 | 0.00 |
| (31,30) | 540.00 | 540.00 | 540.00 | 0.00 | 0.00 |
| (28,20) | 534.00 | 534.00 | 534.00 | 0.00 | 0.00 |
| (28,30) | 534.00 | 534.00 | 534.00 | 0.00 | 0.00 |
| (28,40) | 534.00 | 534.00 | 534.00 | 0.00 | 0.00 |
| (25,20) | 528.00 | 528.00 | 528.00 | 0.00 | 0.00 |
| (25,30) | 528.00 | 528.00 | 528.00 | 0.00 | 0.00 |
| (25,40) | 528.00 | 528.00 | 528.00 | 0.00 | 0.00 |
| | | | **Average $\Delta$ (%)** | **0.00** | **0.00** |
| **EWF Energy Results** | | | | $\Delta$ (%) | |
| (30,10) | 228.00 | 228.00 | 210.00 | 0.00 | 8.57 |
| (30,20) | 228.00 | 228.00 | 210.00 | 0.00 | 8.57 |
| (30,30) | 228.00 | 228.00 | 210.00 | 0.00 | 8.57 |
| (40,10) | 276.00 | 276.00 | 276.00 | 0.00 | 0.00 |
| (40,20) | 276.00 | 276.00 | 270.00 | 0.00 | 2.22 |
| (40,30) | 276.00 | 276.00 | 276.00 | 0.00 | 0.00 |
| (50,10) | 300.00 | 300.00 | 300.00 | 0.00 | 0.00 |
| (50,20) | 300.00 | 300.00 | 300.00 | 0.00 | 0.00 |
| (50,30) | 300.00 | 300.00 | 300.00 | 0.00 | 0.00 |
| | | | **Average $\Delta$ (%)** | **0.00** | **3.10** |
| **FIR Energy Results** | | | | $\Delta$ (%) | |
| (30,20) | 778.00 | 778.00 | 739.00 | 0.00 | 5.28 |
| (35,20) | 784.00 | 784.00 | 784.00 | 0.00 | 0.00 |
| (35,30) | 790.00 | 790.00 | 778.00 | 0.00 | 1.54 |
| (40,20) | 796.00 | 796.00 | 796.00 | 0.00 | 0.00 |
| (40,30) | 796.00 | 796.00 | 796.00 | 0.00 | 0.00 |
| (45,30) | 808.00 | 808.00 | 808.00 | 0.00 | 0.00 |
| (50,20) | 820.00 | 820.00 | 814.00 | 0.00 | 0.74 |
| (50,30) | 820.00 | 820.00 | 814.00 | 0.00 | 0.74 |
| | | | **Average $\Delta$ (%)** | **0.00** | **1.04** |
| **AR Energy Results** | | | | $\Delta$ (%) | |
| (55,20) | 1412.00 | 1400.00 | 1400.00 | 0.86 | 0.86 |
| (55,30) | 1412.00 | 1412.00 | 1406.00 | 0.00 | 0.43 |
| (55,40) | 1424.00 | 1424.00 | 1418.00 | 0.00 | 0.42 |
| (60,20) | 1412.00 | 1424.00 | 1412.00 | -0.84 | 0.00 |
| (60,30) | 1424.00 | 1424.00 | 1424.00 | 0.00 | 0.00 |
| (65,15) | 1424.00 | 1424.00 | 1412.00 | 0.00 | 0.85 |
| (65,20) | 1399.56 | 1424.00 | 1412.00 | -1.72 | -0.88 |
| | | | **Average $\Delta$ (%)** | **-0.24** | **0.24** |

Table C.4 Comparison of the reliability results of SA-based method with ILP and GA-based methods for all benchmarks when $\alpha = 0.0$.

| (L, A) | SA-ND | ILP-ND | GA-ND | SA-ND/ILP-ND | SA-ND/GA-ND |
|---|---|---|---|---|---|
| **DES Reliability Results** | | | | $\Delta$ (%) | |
| (31,20) | 0.720339 | 0.720339 | 0.758729 | 0.00 | -5.06 |
| (31,30) | 0.787886 | 0.795969 | 0.833106 | -1.02 | -5.43 |
| (28,20) | 0.848354 | 0.848354 | 0.864112 | 0.00 | -1.82 |
| (28,30) | 0.757969 | 0.757969 | 0.838299 | 0.00 | -9.58 |
| (28,40) | 0.757969 | 0.757969 | 0.838299 | 0.00 | -9.58 |
| (25,20) | 0.799164 | 0.799164 | 0.897412 | 0.00 | -10.95 |
| (25,30) | 0.798364 | 0.798364 | 0.897412 | 0.00 | -11.04 |
| (25,40) | 0.789466 | 0.789466 | 0.897412 | 0.00 | -12.03 |
| | | | **Average $\Delta$ (%)** | -0.13 | -8.19 |
| **EWF Reliability Results** | | | | $\Delta$ (%) | |
| (30,10) | 0.568721 | 0.568721 | 0.550670 | 0.00 | 3.28 |
| (30,20) | 0.424540 | 0.433327 | 0.520567 | -2.03 | -18.45 |
| (30,30) | 0.424540 | 0.433327 | 0.535491 | -2.03 | -20.72 |
| (40,10) | 0.282804 | 0.304956 | 0.377268 | -7.26 | -25.04 |
| (40,20) | 0.284847 | 0.304956 | 0.378641 | -6.59 | -24.77 |
| (40,30) | 0.284847 | 0.304956 | 0.392794 | -6.59 | -27.48 |
| (50,10) | 0.221953 | 0.213312 | 0.393347 | 4.05 | -43.57 |
| (50,20) | 0.221953 | 0.213312 | 0.307238 | 4.05 | -27.76 |
| (50,30) | 0.213312 | 0.213312 | 0.249176 | 0.00 | -14.39 |
| | | | **Average $\Delta$ (%)** | -1.82 | -22.10 |
| **FIR Reliability Results** | | | | $\Delta$ (%) | |
| (30,20) | 0.542459 | 0.542459 | 0.716684 | 0.00 | -24.31 |
| (35,20) | 0.555915 | 0.555915 | 0.769706 | 0.00 | -27.78 |
| (35,30) | 0.576121 | 0.576121 | 0.697999 | 0.00 | -17.46 |
| (40,15) | 0.808726 | 0.808726 | 0.845330 | 0.00 | -4.33 |
| (40,20) | 0.691288 | 0.691288 | 0.625021 | 0.00 | 10.60 |
| (40,30) | 0.505155 | 0.538654 | 0.577630 | -6.22 | -12.55 |
| (45,30) | 0.464718 | 0.478155 | 0.533088 | -2.81 | -12.83 |
| (50,15) | 0.424877 | 0.429112 | 0.790506 | -0.99 | -46.25 |
| (50,20) | 0.421407 | 0.441647 | 0.478155 | -4.58 | -11.87 |
| (50,30) | 0.424451 | 0.424451 | 0.459538 | 0.00 | -7.64 |
| | | | **Average $\Delta$ (%)** | -1.46 | -15.44 |
| **AR Reliability Results** | | | | $\Delta$ (%) | |
| (55,20) | 0.669042 | 0.669042 | 0.712711 | 0.00 | -6.13 |
| (55,30) | 0.789620 | 0.831064 | 0.753171 | -4.99 | 4.84 |
| (55,40) | 0.700950 | 0.725778 | 0.790472 | -3.42 | -11.33 |
| (60,20) | 0.617024 | 0.667035 | 0.635332 | -7.50 | -2.88 |
| (60,30) | 0.854603 | 0.747594 | 0.748425 | 14.31 | 14.19 |
| (65,15) | 0.827741 | 0.827741 | 0.785492 | 0.00 | 5.38 |
| (65,20) | 0.764145 | 0.773910 | 0.764145 | -1.26 | 0.00 |
| | | | **Average $\Delta$ (%)** | -0.41 | 0.58 |

## C.3 Additional Experimental Results for Section 6.4.

Table C.5 EWF benchmark reliability and energy results of full DMR solutions for a different number of supply voltages.

| | | **EWF Full DMR Reliability Results** | | | | | **EWF Full DMR Energy Results** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Supply Voltage Levels Used** | | | Δ (%) | | **Supply Voltage Levels Used** | | | Δ (%) | |
| Alpha | (L, A) | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI |
| 1.0 | (30,10) | 0.995615 | 0.995615 | 0.995615 | 0.00 | 0.00 | 312.00 | 312.00 | 312.00 | 0.00 | 0.00 |
| | (30,20) | 0.997625 | 0.997625 | 0.997625 | 0.00 | 0.00 | 456.00 | 456.00 | 456.00 | 0.00 | 0.00 |
| | (30,30) | 0.997625 | 0.997625 | 0.997625 | 0.00 | 0.00 | 456.00 | 456.00 | 456.00 | 0.00 | 0.00 |
| | (40,10) | 0.995615 | 0.995615 | 0.995615 | 0.00 | 0.00 | 312.00 | 312.00 | 312.00 | 0.00 | 0.00 |
| | (40,20) | 0.998966 | 0.998966 | 0.998966 | 0.00 | 0.00 | 552.00 | 552.00 | 552.00 | 0.00 | 0.00 |
| | (40,30) | 0.998966 | 0.998966 | 0.998966 | 0.00 | 0.00 | 552.00 | 552.00 | 552.00 | 0.00 | 0.00 |
| | (50,10) | 0.996140 | 0.997097 | 0.997097 | 0.10 | 0.10 | 568.00 | 582.00 | 582.00 | 2.46 | 2.46 |
| | (50,20) | 0.999638 | 0.999638 | 0.999638 | 0.00 | 0.00 | 600.00 | 600.00 | 600.00 | 0.00 | 0.00 |
| | (50,30) | 0.999638 | 0.999638 | 0.999638 | 0.00 | 0.00 | 600.00 | 600.00 | 600.00 | 0.00 | 0.00 |
| | | | | **Average Δ (%):** | **0.01** | **0.01** | | | **Average Δ (%):** | **0.27** | **0.27** |
| 0.5 | (30,10) | 0.995615 | 0.995615 | 0.995615 | 0.00 | 0.00 | 312.00 | 312.00 | 312.00 | 0.00 | 0.00 |
| | (30,20) | 0.995615 | 0.987541 | 0.987780 | -0.81 | -0.79 | 312.00 | 238.80 | 238.80 | -23.46 | -23.46 |
| | (30,30) | 0.995615 | 0.986335 | 0.986335 | -0.93 | -0.93 | 312.00 | 227.82 | 227.82 | -26.98 | -26.98 |
| | (40,10) | 0.995615 | 0.985131 | 0.985131 | -1.05 | -1.05 | 312.00 | 216.84 | 216.84 | -30.50 | -30.50 |
| | (40,20) | 0.995615 | 0.985131 | 0.985131 | -1.05 | -1.05 | 312.00 | 216.84 | 216.84 | -30.50 | -30.50 |
| | (40,30) | 0.995615 | 0.985131 | 0.985131 | -1.05 | -1.05 | 312.00 | 216.84 | 216.84 | -30.50 | -30.50 |
| | (50,10) | 0.995615 | 0.985131 | 0.985131 | -1.05 | -1.05 | 312.00 | 216.84 | 216.84 | -30.50 | -30.50 |
| | (50,20) | 0.995615 | 0.985131 | 0.985131 | -1.05 | -1.05 | 312.00 | 216.84 | 216.84 | -30.50 | -30.50 |
| | (50,30) | 0.995615 | 0.985131 | 0.985131 | -1.05 | -1.05 | 312.00 | 216.84 | 216.84 | -30.50 | -30.50 |
| | | | | **Average Δ (%):** | **-0.90** | **-0.89** | | | **Average Δ (%):** | **-25.94** | **-25.94** |
| 0.0 | (30,10) | 0.995615 | 0.995615 | 0.995615 | 0.00 | 0.00 | 312.00 | 312.00 | 312.00 | 0.00 | 0.00 |
| | (30,20) | 0.979502 | 0.987541 | 0.987660 | 0.82 | 0.83 | 270.00 | 238.80 | 238.80 | -11.56 | -11.56 |
| | (30,30) | 0.977633 | 0.964372 | 0.963874 | -1.36 | -1.41 | 266.00 | 221.68 | 221.64 | -16.66 | -16.68 |
| | (40,10) | 0.975312 | 0.985131 | 0.985131 | 1.01 | 1.01 | 260.00 | 216.84 | 216.84 | -16.60 | -16.60 |
| | (40,20) | 0.975312 | 0.940980 | 0.940008 | -3.52 | -3.62 | 260.00 | 197.24 | 197.16 | -24.14 | -24.17 |
| | (40,30) | 0.975312 | 0.940980 | 0.938168 | -3.52 | -3.81 | 260.00 | 197.24 | 197.08 | -24.14 | -24.20 |
| | (50,10) | 0.975312 | 0.985131 | 0.934091 | 1.01 | -4.23 | 260.00 | 216.84 | 202.22 | -16.60 | -22.22 |
| | (50,20) | 0.975312 | 0.913647 | 0.934091 | -6.32 | -4.23 | 260.00 | 184.64 | 202.22 | -28.98 | -22.22 |
| | (50,30) | 0.975312 | 0.913647 | 0.912281 | -6.32 | -6.46 | 260.00 | 184.64 | 184.56 | -28.98 | -29.02 |
| | | | | **Average Δ (%):** | **-2.02** | **-2.43** | | | **Average Δ (%):** | **-18.63** | **-18.52** |

Table C.6 AR benchmark reliability and energy results of full DMR solutions for a different number of supply voltages.

| | | **AR Full DMR Reliability Results** | | | | | **AR Full DMR Energy Results** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Supply Voltage Levels Used** | | | **Δ (%)** | | **Supply Voltage Levels Used** | | | **Δ (%)** | |
| Alpha | (L, A) | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI | 1 VI (h) | 2 VIs (h,l) | 3 VIs (h,m,l) | 2VIs/1VI | 3VIs/1VI |
| 1.0 | (65,20) | 0.999972 | 0.999972 | 0.999972 | 0.00 | 0.00 | 2848.00 | 2848.00 | 2848.00 | 0.00 | 0.00 |
| | (50,30) | 0.998797 | 0.998797 | 0.998797 | 0.00 | 0.00 | 2764.00 | 2764.00 | 2764.00 | 0.00 | 0.00 |
| | (55,30) | 0.999300 | 0.999300 | 0.999300 | 0.00 | 0.00 | 2800.00 | 2800.00 | 2800.00 | 0.00 | 0.00 |
| | (60,30) | 0.999804 | 0.999804 | 0.999804 | 0.00 | 0.00 | 2836.00 | 2836.00 | 2836.00 | 0.00 | 0.00 |
| | (50,40) | 0.998797 | 0.998797 | 0.998797 | 0.00 | 0.00 | 2764.00 | 2764.00 | 2764.00 | 0.00 | 0.00 |
| | (55,40) | 0.999300 | 0.999300 | 0.999300 | 0.00 | 0.00 | 2800.00 | 2800.00 | 2800.00 | 0.00 | 0.00 |
| | | | | **Average Δ (%):** | **0.00** | **0.00** | | | **Average Δ (%):** | **0.00** | **0.00** |
| 0.5 | (65,20) | 0.999972 | 0.999972 | 0.999972 | 0.00 | 0.00 | 2848.00 | 2848.00 | 2848.00 | 0.00 | 0.00 |
| | (50,30) | 0.998797 | 0.998797 | 0.998797 | 0.00 | 0.00 | 2764.00 | 2764.00 | 2764.00 | 0.00 | 0.00 |
| | (55,30) | 0.999300 | 0.999300 | 0.998625 | 0.00 | -0.07 | 2800.00 | 2800.00 | 2307.84 | 0.00 | -17.58 |
| | (60,30) | 0.999804 | 0.999804 | 0.998961 | 0.00 | -0.08 | 2836.00 | 2836.00 | 2331.84 | 0.00 | -17.78 |
| | (50,40) | 0.998797 | 0.998280 | 0.998458 | -0.05 | -0.03 | 2764.00 | 2542.24 | 2552.22 | -8.02 | -7.66 |
| | (55,40) | 0.999300 | 0.998277 | 0.998277 | -0.10 | -0.10 | 2800.00 | 2468.92 | 2187.18 | -11.82 | -21.89 |
| | | | | **Average Δ (%):** | **-0.03** | **-0.05** | | | **Average Δ (%):** | **-3.31** | **-10.82** |
| 0.0 | (65,20) | 0.999972 | 0.999972 | 0.999972 | 0.00 | 0.00 | 2848.00 | 2848.00 | 2848.00 | 0.00 | 0.00 |
| | (50,30) | 0.995392 | 0.994783 | 0.994783 | -0.06 | -0.06 | 2693.00 | 2691.35 | 2691.35 | -0.06 | -0.06 |
| | (55,30) | 0.988513 | 0.988506 | 0.995848 | 0.00 | 0.74 | 2680.00 | 2508.92 | 2250.49 | -6.38 | -16.03 |
| | (60,30) | 0.988513 | 0.988503 | 0.988502 | 0.00 | 0.00 | 2680.00 | 2435.60 | 2151.36 | -9.12 | -19.73 |
| | (50,40) | 0.988513 | 0.990628 | 0.996219 | 0.21 | 0.78 | 2680.00 | 2522.80 | 2626.34 | -5.87 | -2.00 |
| | (55,40) | 0.988513 | 0.991069 | 0.992660 | 0.26 | 0.42 | 2680.00 | 2279.00 | 2166.56 | -14.96 | -19.16 |
| | | | | **Average Δ (%):** | **0.07** | **0.31** | | | **Average Δ (%):** | **-6.07** | **-9.50** |

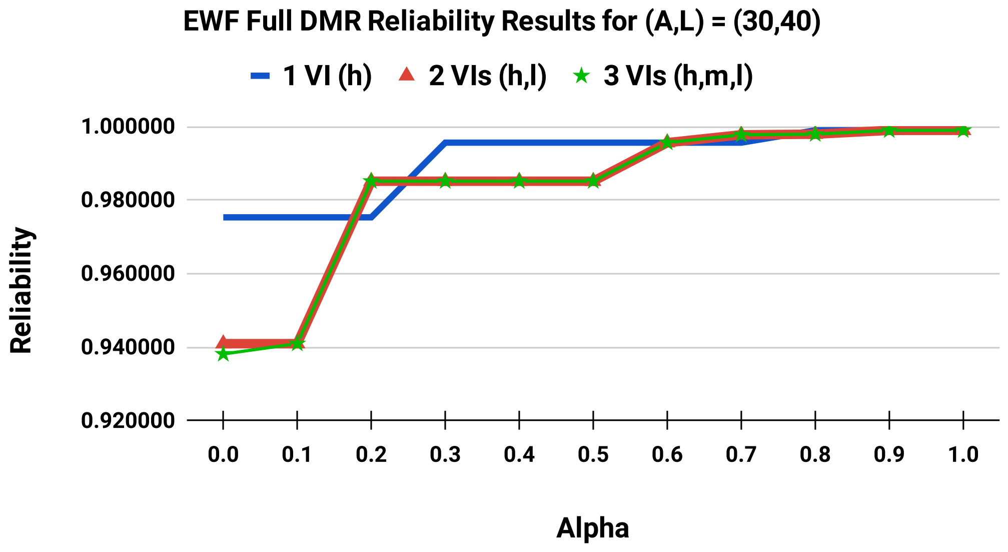


Figure C.1 Changes in reliability over different $\alpha$ values for EWF benchmark ($A = 30, L = 40$) under different numbers of supply voltages.

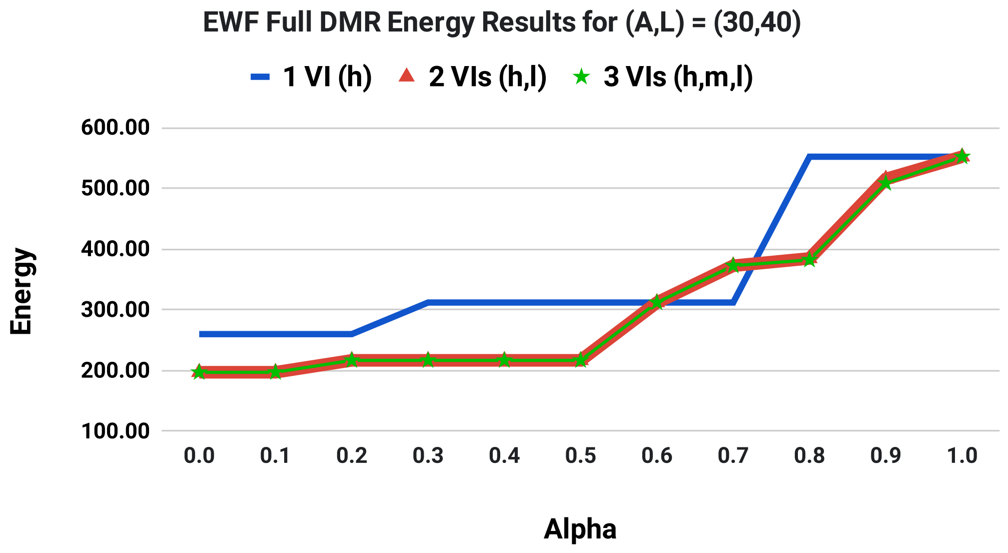


Figure C.2 Changes in energy over different $\alpha$ values for EWF benchmark ($A = 30, L = 40$) under different numbers of supply voltages.

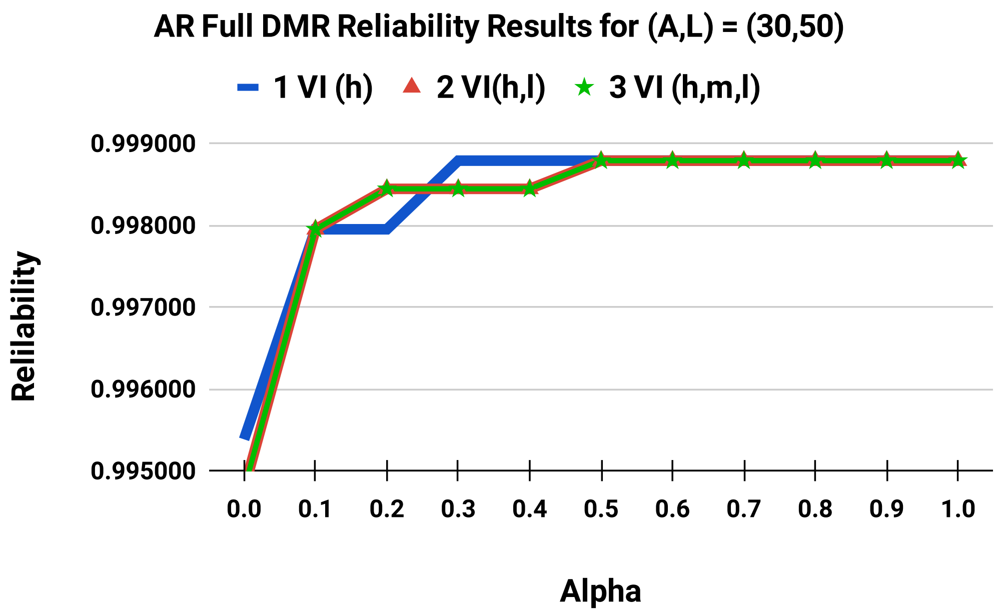


Figure C.3 Changes in reliability over different $\alpha$ values for AR benchmark $(A = 30, L = 50)$ under different numbers of supply voltages.

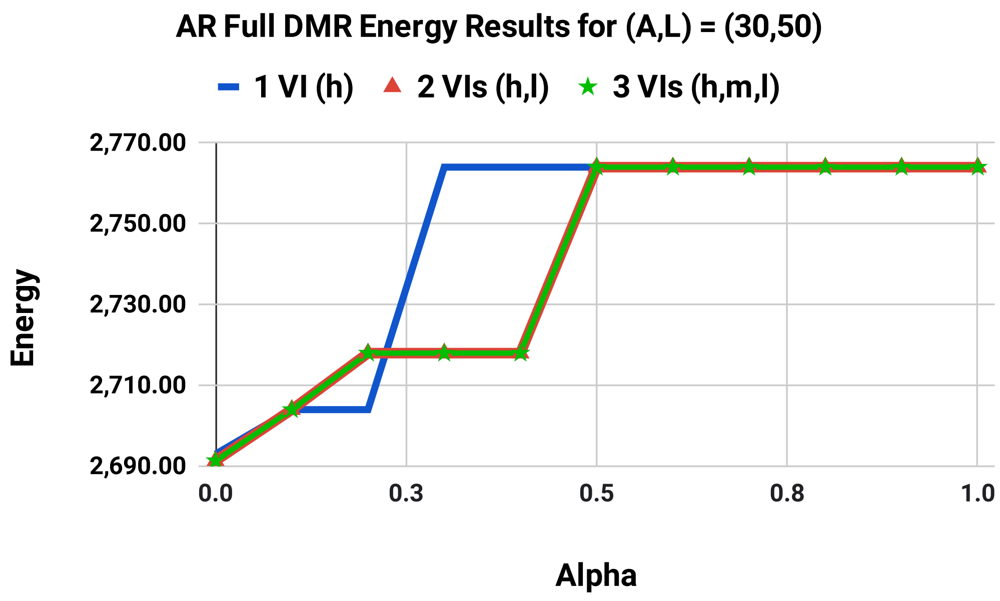


Figure C.4 Changes in energy over different $\alpha$ values for AR benchmark $(A = 30, L = 50)$ under different numbers of supply voltages.