Electronic Theses and Dissertations, 2004-2019

2015

Self-Scaling Evolution of Analog Computation Circuits

Steven Pyle
University of Central Florida

Showcase of Text, Archives, Research & Scholarship

SELF-SCALING EVOLUTION OF ANALOG COMPUTATION CIRCUITS

by

STEVEN D. PYLE
B.S. University of Central Florida 2013

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical Engineering & Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2015

Major Professor: Ronald F. DeMara

ABSTRACT

Energy and performance improvements of continuous-time analog-based computation for selected applications offer an avenue to continue improving the computational ability of tomorrow's electronic devices at current technology scaling limits. However, analog computation is plagued by the difficulty of designing complex computational circuits, programmability, as well as the inherent lack of accuracy and precision when compared to digital implementations. In this thesis, evolutionary algorithm-based techniques are utilized within a reconfigurable analog fabric to realize an automated method of designing analog-based computational circuits while adapting the functional range to improve performance.

A *Self-Scaling* Genetic Algorithm is proposed to adapt solutions to computationally-tractable ranges in hardware-constrained analog reconfigurable fabrics. It operates by utilizing a *Particle Swarm Optimization (PSO)* algorithm that operates synergistically with a *Genetic Algorithm (GA)* to adaptively scale and translate the functional range of computational circuits composed of high-level or low-level *Computational Analog Elements* to improve performance and realize functionality otherwise unobtainable on the intrinsic platform. The technique is demonstrated by evolving square, square-root, cube, and cube-root analog computational circuits on the Cypress PSoC-5LP System-on-Chip. Results indicate that the *Self-Scaling* Genetic Algorithm improves our error metric on average *7.18*-fold, up to *12.92*-fold for computational circuits that produce outputs beyond device range. Results were also favorable compared to previous works, which utilized extrinsic evolution of circuits with much greater complexity than was possible on the PSoC-5LP.

TABLE OF CONTENTS

LIST OF FIGURES

x

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

The exponential improvement in the ability of computers that we've observed over the past decades has led to a booming technology market, improvements in scientific understanding, and greater globalization as individuals are more able to connect with one another regardless of distance. With upcoming challenges facing the current status quo of Moore's Law, new and innovative strategies to continue improving computational performance are sought. This chapter elucidates the significance of the problem, overviews current techniques for applying analog computation, and then delineates the proposed contributions in the Contribution of Thesis section.

Need for Evolutionary Analog Computation

As we continue to advance towards CMOS technology-scaling limits, new and innovative strategies to enhance computational performance at our current technology scaling limits are sought. One possible approach for such enhancements lies in addressing the fundamental inefficiency in today's computational models that utilize discretized digital computation to solve continuous real-world phenomena [1], such as signal processing and differential equation computation. An intriguing way of alleviating this inefficiency is to utilize a "let the physics do the computing" approach by employing analog devices to perform continuous time computations where applicable [1]. According to Gene's Law as shown in Figure 1, utilizing analog computation for applicable applications could provide a 20-year leap in performance versus their digital counterparts, which translates into a theoretical 1,000 to 10,000 fold improvement [2]. Approaches

1

presented in [2, 3] demonstrated that analog computation reduced energy consumption by 8-fold compared to the corresponding digital implementation. However, complex analog circuits can be both challenging to design and lack precision.

Precise and efficient complex analog circuits typically requires an expert with many years of design expertise and experience [4]. Nonetheless, [4-7] have demonstrated that evolutionary approaches such as *Genetic Algorithms* (GAs) are a viable candidate to address the problem of automated analog design, having successfully evolved analog computational as well as analog circuits to perform digital functions, such as a NAND gate and a 2-input ALU [8]. In [4] it has been shown that it's possible to evolve robust nonlinear analog circuits with GAs, demonstrating the strength of the technique. However, due to the stochastic nature of GAs, it can be challenging to determine how accurately the evolved analog circuits map to the desired function, especially on realistic commercial devices with constrained hardware.
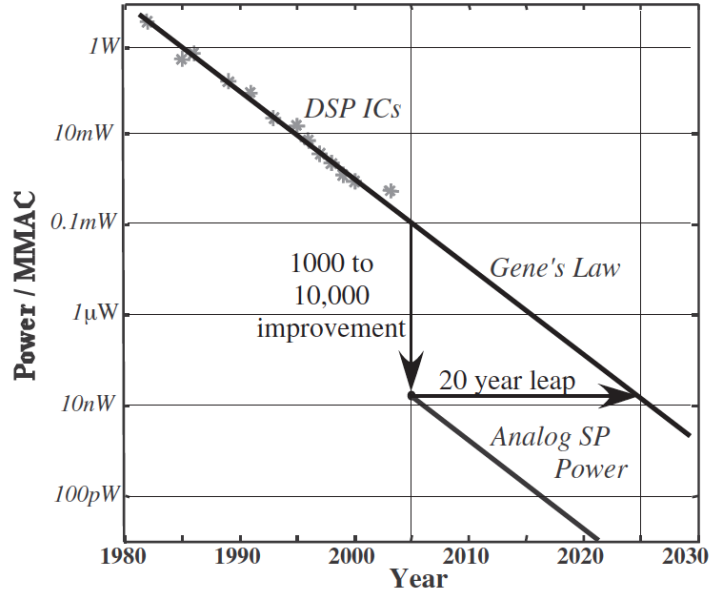
Figure 1: Gene's Law Curve Showing 20-year Leap in Performance of Analog SP Compared to Digital [9].

Characteristics of Analog Computational Circuits

Continuous-time analog computational circuits (CC) as well as discretized digital computers co-existed during the early stages of electronic computer development, as each domain offered benefits over the other for different computational needs [10-14]. However, digital-based computational models eventually won out over just about all their analog counterparts due to the benefits of noise-resilience, easy and sustainability of memory operation, and ease of programmability and reproducibility [1]. This does not imply that the benefits of analog computational models cannot be utilized in an intelligent fashion to improve current digital-only models in some hybrid fashion [2, 3]. The primary characteristics of analog-based computation to be considered when developing hybrid computing methods is shown in Figure 2, and is delineated

3

by 1) low-power, 2) speed of solution convergence, 3) low-precision, 4) noise-intolerance, and 5) difficulty of programmability or design [1].
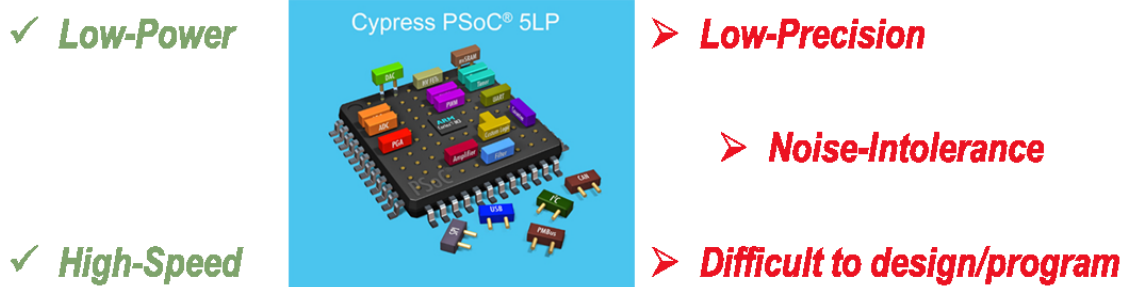
Figure 2: Characteristics of Analog Benefits and Challenges Explored Herein.

Characteristics of Evolvable Analog Hardware

Evolvable Hardware (EHW) can generally be broadly classified into two different categories shown in Figure 3 whether the application is in either the digital or analog domain: 1) intrinsic evolution, which is the evolution of circuits evaluated on a physical platform, or 2) extrinsic evolution, which is where the evaluation is conducted in a simulation environment and then can be implemented onto a physical device if so desired and the evolved circuit is compatible [15]. The majority of analog EHW studies are implemented extrinsically as there are few reconfigurable analog platforms available [4-8, 16-18]. However, some groups have developed their own *Field Programmable Transistor Array (FPTA)* to implement EHW techniques in the analog domain intrinsically [19].

4
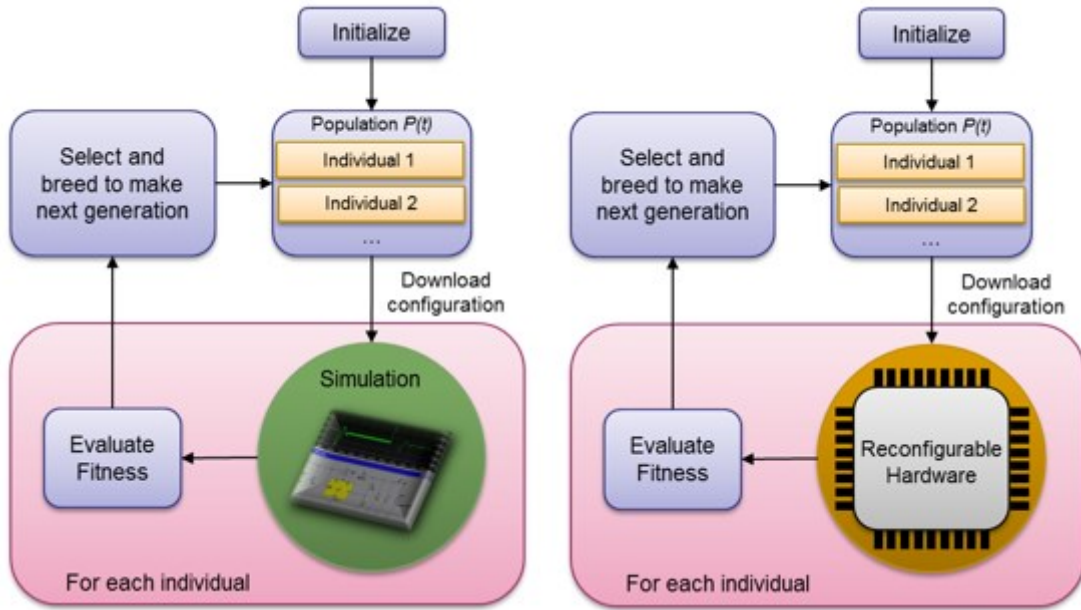
Figure 3: Extrinsic (left) and Intrinsic (right) Evolvable Hardware Techniques.

EHW techniques are applicable to more than just evolutionary design, as they can be utilized for intrinsic repair due to system faults. Faults can come in the form of soft-errors, which are caused when a bit in a register or along a datapath is flipped, or hard-errors, which are caused by the shorting or opening of wires in the hardware [20-23]. EHW techniques are typically utilized for hard faults by allowing the system to search for configurations which still provide the desired functionality even with the hardware faults in place [20, 22, 24-26].

Genetic Algorithms

GAs are a well-known class of metaheuristic EAs that emulate natural forms of survival-of-the-fittest Darwinian evolution [27]. GAs utilize a population of *configurations*, denoted as *individuals*, the relative quality of their solutions, called *fitness*, and various bio-inspired genetic

operators, such as *crossover* and *mutation*, to find solutions in large search spaces [28]. The individuals "compete" via the selection method and their relative fitness levels in order to combine their genetic material to produce new individuals for the next generation. This cycle of testing, selecting, and breeding gives rise to individuals that have a very high fitness, and based on the fitness function used for evaluation these individuals should be very adept at their application. The most important things to consider when developing a GA is the genetic representation of the circuit configuration, the choice of fitness function for the particular application, the selection mechanism for choosing which individuals undergo genetic operators to produce new individuals, and the mutation rate as too low of a mutation rate can lead to early solution convergence to local minima, and too high of a mutation rate will devolve the GA into random search [29]. An extension to GAs called an *Island GA* evolves multiple populations in parallel and periodically exchanges individuals between them; this helps to preserve genetic diversity while each island is allowed to follow different trajectories in the search space [15].

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a metaheuristic optimization algorithm inspired by bird or fish flocking and swarming theory [30, 31]. In operation, a population of particles is initialized with randomly-distributed optimization parameters within a specified range [*pmin, pmax*]. Each particle is then evaluated based on the quality of output given by the particles coordinates, or parameters, substituted into the function to be optimized. Each particle's previous best parameter configuration is saved (*pBest*) along with the global best parameter configuration

(*gBest*), and the particles are moved towards *pBest* and *gBest* parameters with a particular velocity. Using this method, particles are "flown" across the search space to realize optimizations within the problem space [30].

Platform Overview
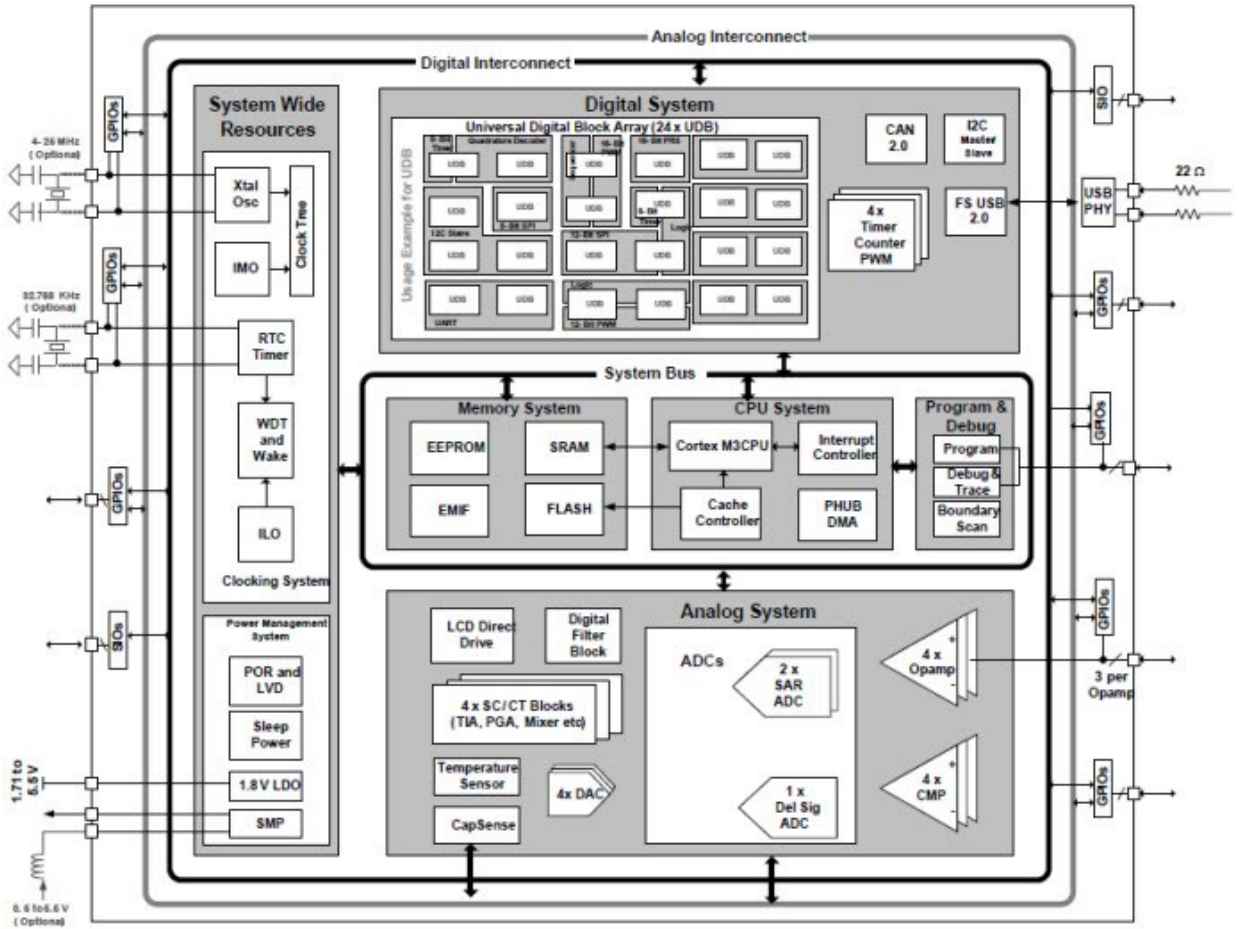
Figure 4: Cypress Semiconductor PSoC-5LP System Block Diagram.

The platform utilized for the research and development of the contributions herein is the PSoC-5LP programmable system on chip made by Cypress Semiconductor Inc. The PSoC-5LP is

a fantastic platform for the development of analog EHW techniques because it combines a Cortex-M3 ARM microprocessor with reconfigurable analog fabric in the form of switched capacitor *operational amplifier (op amp)* blocks along with necessary peripheral and interface circuitry such as analog to digital converters, digital to analog converters, and a variety of routing circuitry. The overall system block diagram for the PSoC-5LP is depicted in figure 4. The PSoC-5LP also has a reconfigurable digital fabric in the form of Universal Digital Blocks (UDBs) for custom digital circuitry.

Summary of the Thesis

This thesis delineates a new technique that utilizes PSO to optimize and refine evolved analog computational circuits to the intrinsic device voltage range. In order to demonstrate the method, an analog *Self-Scaling GA (SSGA)* is developed on an intrinsic commercial prototyping platform. In particular, case studies are examined on the Cypress PSOC-5LP commercially-available System on a Chip (SoC), which combines reconfigurable analog fabric in the form of four switched capacitor operational amplifier blocks, a PLD-based reconfigurable digital fabric, an ARM core, and other modules such as ADCs and DACs. We describe how the proposed technique operates and demonstrates its capability to intrinsically evolve, adapt, and refine the same *Computational Circuits (CCs)* that [7] did, specifically the square, square-root, cube, and cube-root functions.

8

Contributions of the Thesis

As far as the author is aware, this is the first demonstration of evolving analog computational circuits on a commercially-available platform. The techniques developed herein pave new ground for allowing research into exploring how analog circuits can improve the computational models that we use today. Previously, on-chip analog circuit design required analog circuit expertise to design un-reconfigurable ASICs, or to design using models which are then programmed onto a Field Programmable Analog Array (FPAA) [32]. Additionally, these solutions are inherently prone to process variations and device mismatch. In order to best utilize analog circuits to improve our computational models, a method for adapting such circuits to the desired functionality given all of the intrinsic device characteristics must be developed, and that is what was done herein.
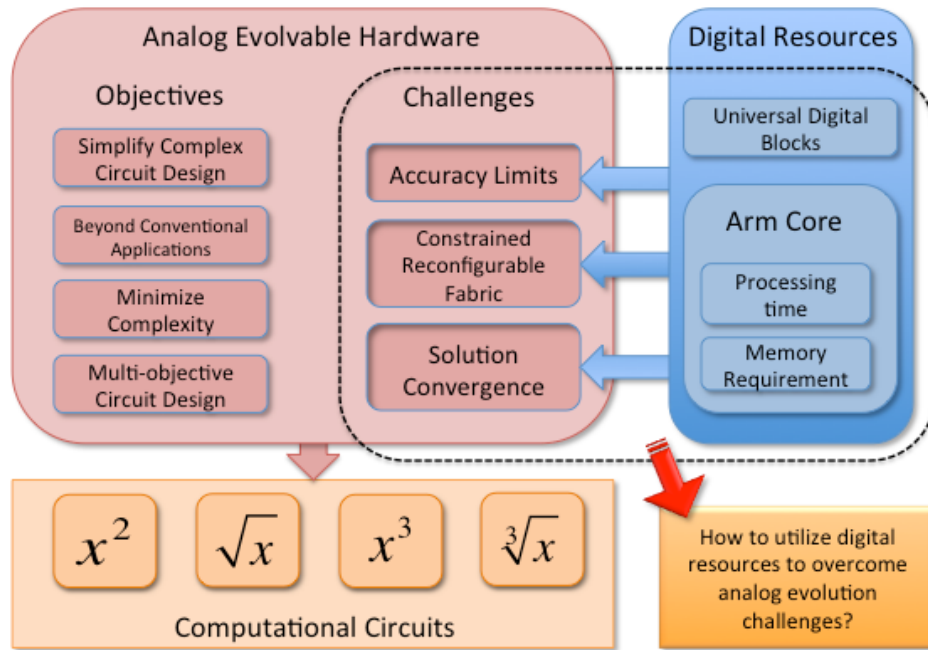
Figure 5: Contributions of the thesis.

The following research contributions are provided:

1) pioneered an extension to analog domain evolution called *Self-Scaling GA (SSGA)* which utilizes both particle swarm optimization in tandem with analog genetic algorithms. This results in the first demonstration of self-scaling functional voltage, whereby the range of outputs is adapted to the device's intrinsically tractable computational domain,

2) extended a new Island-like GA methodology to explore multiple SSGA parameters sets in order to locate and exchange best-parameter information periodically, and

3) providing a bitwise testing scheme along with normalized error metrics in preparation for digital refinement.

CHAPTER TWO: RELATED WORK

Utilizing analog circuits to perform various continuous-time computations currently represent a niche application domain with potential for expansion. Accordingly, analog computers were the first computational devices invented, with some of the earliest demonstrations of analog computational systems dating back to 1929, where AC network analyzers were used to solve electrical power system calculations that were too large to solve using the numerical methods of the time [33]. Analog computers performance and scope continued to grow along with modern computers after their introduction by Alan Turing [14]. Analog computers were even used into the 1960's to solve nonlinear ordinary differential equations [12]. However, several hurdles existed such as the error-prone nature of analog computation, the difficulty in storing analog data, and the limited dynamic range available to analog computers. By circumventing these challenges, digital computers, which lacked these drawbacks, eventually became the predominant computational method [1]. Nonetheless, with transistor scaling limits fast approaching, analog computation is being revisited in conjunction with digital computation for its low power and high speed computational abilities [1].

Hybrid Analog-Digital Computation

Recently, the benefits of analog computation, such as low-power and high-speed operation, are becoming increasingly interesting due to the upcoming limitations of transistor size scaling on digital logic as well as the increasing need for low-power computation in mobile devices [1, 34,

35]. Instead of keeping the analog and digital computational domains separate, researchers are interested in researching how we may combine the benefits of both domains together to perform hybrid analog-digital computation [1, 36].

For instance, an interesting platform developed for a hybrid approach is the *Reconfigurable Analog Signal Processor (RASP)* developed at the Georgia Institute of Technology over the last decade [2, 3, 34, 37, 38]. A hybrid approach using the RASP was demonstrated to improve particularly the energy required for a variety of computations, such as using analog components to compute the Discrete Fourier Transform block of an orthogonal frequency-division multiplexing system, achieving an 8.9 dB reduction in overall power consumption compared to the purely digital approach for just a 2-dB performance degradation [3]. It was also on the RASP that using analog components for Vector Matrix Multiplication could provide up to 1,000 times more computationally efficient in MMAC/µW (Million Multiply-Accumulate Computations per µW) compared to the digital approach [9].

Proceeding the development of the RASP, Cowan et al. developed and demonstrated an analog co-processor that worked in tandem with a digital processor to compute ordinary differential equations, partial differential equations, and stochastic differential equations to accelerate computations by more than 10-fold while dissipating less than 1% of the energy of a general purpose digital microprocessor [39]. They were able to achieve this by developing a custom Integrated Circuit (IC) of 80 analog integrators, 336 other linear and nonlinear analog functional blocks, routing interconnects, and additional circuitry allowing the IC to be programmed, read, and controlled by a PC via a data acquisition card. These custom analog

development platforms provide a fascinating infrastructure for soft-computing techniques for adaptive analog computation, and while they are a representation of experimental programmable analog signal processor, in this thesis we investigate a commercial version for prototyping the methods developed herein.

Recently, Huang et al. developed a chip architecture along with its corresponding Instruction Set Architecture (ISA) to implement Hybrid discrete-continuous computational systems to improve the performance of floating-point math, non-linear math, and differential equation computation [36]. The Hybrid Continous-Discrete Computer (HCDC) designed is a highly parallel tiled-based hardware design combining analog and digital functional units into an HCDC fabric. The HCDC has the capability to operate as a standalone processor for processing sensory input or directly controlling other physical devices. It can also be used in conjunction with a microcontroller for calibration, configuration, and adaptation. As the HCDC has not been implemented as of this writing, no results can be expounded on. However, the HCDC provides a novel and intriguing platform for future hybrid analog-digital computational methods.

Evolvable Hardware

Evolvable Hardware (EH) is a novel technique for the design of electronic systems by utilizing evolutionary algorithms in lieu of procedural hand-designed techniques. The purpose of EH is to utilize simulated evolution for composing systems in such a way that takes into account the possibility of configurations which might lie beyond human design capabilities [40]. Intrinsic EH takes this a couple steps further by providing a method of design which accounts for all of the

innate physical characteristics of each of the available resources [40-42]. Traditional hardware such as ASICs can be completely inflexibly, impossible to change its structure and functionality once the chip has been manufactured. This leads to issues when implemented in dynamic harsh environments or when the functionality needs to be changed or adapted.

EH can be used for the initial design of the system as well as allowing the system to self-adapt in the presence of environmental changes or hardware faults [21, 24, 43]. The simulated evolution can be carried out by a variety of stochastic metaheuristic search algorithms including GAs, GPs, or other Evolutionary Strategies (ES). EH is typically implemented on reconfigurable platforms such as FPGAs, Field Programmable Analog Arrays (FPAAs), or FPTAs. EH uses a binary bit-stream to encode the architecture that is to be implemented on the reconfigurable platform [44]. Regardless of the evolutionary strategy chosen, the strategy is used to find the best performing bit-stream, and therefore architecture, for the desired application.

Evolvable Hardware for Design

EH was first demonstrated by Thompson in his seminal paper "Silicon Evolution" where he demonstrated that artificial evolution via a genetic algorithm implemented on a Field Programmable Gate Array (FPGA) could design digital logic circuits that would oscillate at a given frequency [40]. Koza et al. took EH further into the analog domain by using *Genetic Programming (GP)* to evolve analog circuit topologies, which were evaluated extrinsically using a Simulation Program with Integrated Circuit Emphasis (SPICE) [5]. Koza et al. showed that the use of EH techniques could realize analog circuits to perform a suite of different prototypical

circuits such as a low-pass filter, a crossover filter, a source identification circuit, a computational circuit, a time-optimal controller circuit, a temperature-sensing circuit, and a voltage reference circuit.

A variety of *Evolutionary Algorithms (EAs)* have been used in tandem with reconfigurable fabrics to intrinsically realize novel electronic circuit designs. Numerous innovative works have contributed to the literature of which only a few are highlighted in Table 1 relating to analog and hybrid analog-digital domains. Mydlowec and others followed the path of Koza, evolving other CCs extrinsically [7, 16], in some cases using multiple time domain simulations to improve robustness. Later, Streeter et al. [17] also showed that GP was able to iteratively evolve circuits that could be attached to computational circuits to refine their performance. In [6] EAs were used to evolve four analog CCs as well as two digital circuits using analog components. In [12] swarming algorithms such as PSO were used to evolve analog circuit sizing. Recently, Cornforth et al. evolved non-linear circuits by utilizing a strategic fitness evaluation scheme without necessarily optimizing them for area [4]. They were able to show that a variety of stimuli can extrinsically evolve nonlinear analog circuits, which conform to randomly generated black-box circuits, demonstrating the strength of the method.

Deviating from the normal utilization of GAs and GP to run the evolutionary algorithms, a different ES called Univariate Marginal Distribution Algorithm (UMDA) was recently shown to be able to design the topology of desired analog circuits [45]. Slezak et al. showed that UMDA was an effective algorithm for designing the topology of circuits in conjunction with a local search

15

algorithm to determine device parameters [45]. By using a hybrid UMDA-local search algorithm, they were able to evolve fractional capacitor circuits with a given input impedance.

Table 1: Selected Previous Works.

Research Work	Analog Circuits Evolved	EA Type	Platform	Contribution
[Koza 1997]	Square root	Extrinsic	SPICE	Used GP to design analog circuits using DC sweeps.
[Mydlowec 2000]	Square, square root, multiplier, and lag circuit	Extrinsic	SPICE	Synthesis of computational circuits using multiple time-domain simulations for robustness fitness evaluation.
[Keymeulen 2000]	Multiplier	Intrinsic	Custom FPTA with 48 switchable transistor terminals in two 0.5um chips	Population-based and Fitness-based fault tolerance.
[Streeter 2002]	Cube	Extrinsic	Weakly-constrained virtual fabric under progressive voltage conditions	Average error 7-fold less than human design.
[Cornforth 2014]	Random Black Box Non-linear circuits	Extrinsic	NG-SPICE	Demonstrated that an age-fitness incremental algorithm is better for non-linear analog circuit fitness evaluation.
This work	**Square root, cube root, square, cube**	**Intrinsic**	**Cypress PSoC-5LP**	**Digital resources enhance accuracy of self-scaling GA with PSO.**

Evolvable Hardware for Reliability

Using EH techniques to self-adapt systems for faults and environmental changes can be very useful for systems that need high survivability such as space missions and defense applications [46, 47]. These applications typically require a certain level of functionality for long durations in harsh environments [46]. Kim et al. showed that using ES with robustness evaluations could be used to automate the synthesis of robust analog circuits that maintain their functionality in the presence of faults [48]. Even though Kim el al. demonstrated their methods using extrinsic evolution, their evolved designs still showed similar characteristics to the simulations when implemented using physical components [48]. Keymeulen et al. demonstrated intrinsic EHW on FPAAs for population-based and fitness-based evolution of fault-tolerant analog circuits [18].

Analog EH techniques have also been shown in [49] to be able to automatically generate multiple analog circuits with similar functionality and then combine the solutions to generate robust outputs. They utilized the inherent populations generated by a GA to obtain multiple designs in order to implement a modularly redundant analog low-pass filter. Results show that the modularly redundant designs performed better than the best single module design.

Taxonomy

The current state of research in analog computational systems is represented by the taxonomy shown in Figure 6. In the area of analog computation, there's two major research scopes. One is aimed at addressing the potential benefits of using analog computational systems to improve the performance of our current computational systems. The other is aimed at the challenges of implementing analog computational systems. Utilizing the benefits of high-speed, low-power, and high-efficiency computation, the RASP processor developed in [37] and the analog accelerator co-processor developed in [39] showed great merit. Both implementations were able to significantly improve the performance and efficiency of computations compared to the purely digital approaches. However, many of the challenges associated with analog computation have been largely left unanswered; having mostly been explored with extrinsic EH techniques in [4-7] and [16-17]. These extrinsic techniques showed us that EH techniques are a valid candidate in the automated design of analog computational systems. The research explored also showed that EH techniques can be utilized to improve the robustness and reliability of computational systems,

including analog. In this work, all of the challenges listed associated with analog computation is addressed in some fashion by using intelligent self-scaling intrinsic evolutionary techniques.
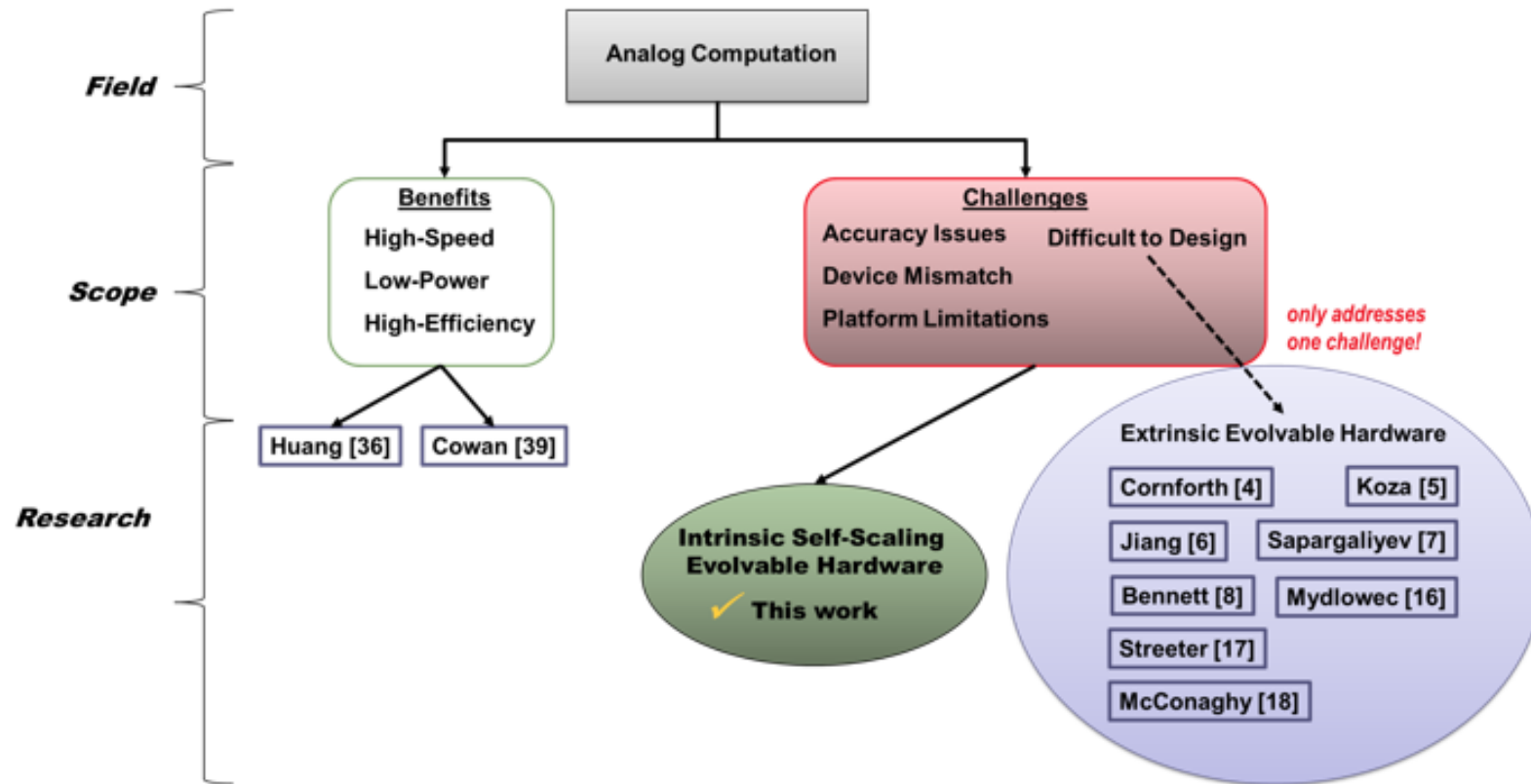
Figure 6: Analog Computation Taxonomy

Summary

In this chapter motivational applications of analog computation is explored, as well as current techniques and works in EH. The energy and performance benefits of analog computation was shown by the results obtained from analog implementations on the custom RASP platform developed in [2, 3, 34, 37, 38] as well as the analog accelerator co-processor developed in [39]. Current research topics involving HCDC architectures were explored, providing the necessary framework for future flexible HCDC computational models to be developed. Then, the field of EH was broken down. Starting from using EH for design, we explored Thompson's et al. seminal work using an FPGA to evolve circuits in the analog domain along with other EH techniques for evolving analog circuits on intrinsic, but primarily extrinsic, platforms. EH techniques for fault tolerance, redundancy, and self-repair was also explored. This chapter finished with a taxonomy relating the current research in the domain of analog computation.

While several previous works in analog CC design using EAs have involved simulation, recent *Programmable System on Chip (PSoC)* devices providing reconfigurable analog fabric, digital logic, and ARM cores enable new capabilities. Analog fabrics allow rapid evolution, but are limited by precision and/or accuracy, which may be refined with evolutionary techniques. The ARM core on the PSoC allows on-chip execution of EAs such as the GAs and PSO as developed herein. While this chapter introduced the prototypical infrastructures in EH approaches needed for this research, we now discuss how such capabilities can be utilized specifically for addressing the challenges of analog circuit evolution.

CHAPTER THREE: EVOLUTION OF ANALOG CIRCUITS

The evolutionary design of analog circuits typically consists of a GA used to evolve the circuit to a fitness function by either extrinsic or intrinsic means. This Chapter goes into greater detail of GA operation, implementation on the PSoC-5LP platform, and concludes with the issues facing intrinsic analog circuit evolution.

Delineation of Genetic Algorithms

The general flow of a GA is delineated in figure 7, and will be described herein as applicable to analog circuit evolution. A GA typically begins by initializing a group of *N* circuit configurations called *individuals* to make up a *population* of such individuals. Initialization is generally carried out by generating random circuit configurations for each individual. All individuals adhere to a specific circuit coding template, called a *genome,* which describes the components of the circuit as well as the routing between component terminals and input/output.

Once the initial population has been generated, each individual is evaluated against a *fitness function* to determine each individual's quality of solution, or *fitness*. After the fitness of each individual has been evaluated, a *selection* process occurs, which chooses individuals for breeding the next generation. Once individuals are selected for breeding, the *genetic operators* are used to produce offspring consisting of a combination of the genetic makeup of their parent's genomes. These offspring move on to populate the next *generation* of the population. In many GAs, designers choose to save the best individual(s) from being altered by the GA operators; this is

called *Elitism*. Elitism does not remove the best individuals from being potential parents as their genes are very valuable. After this stage, the new population is reevaluated based on the same fitness function that was used for the first, and then the subsequent population undergoes selection and GA operators as well as elitism and repeats until the maximum fitness is reached or the maximum generations allowed by the designer has been reached.
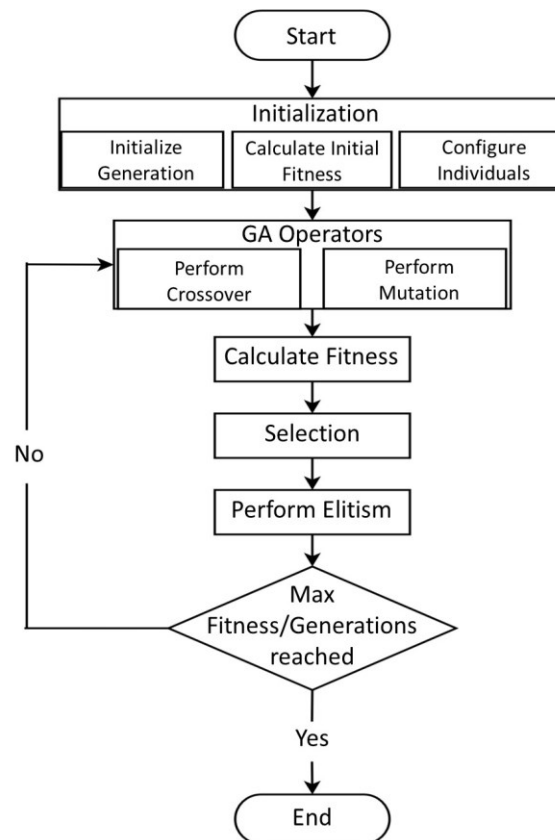
Figure 7: Genetic Algorithm Flow Chart.

Genes

Each individual's genome is broken down further into *Genes*, which describe the fundamental building block of the circuit the GA is operating on. Genes can describe high level components such as various op amp topologies like integrators, adders, filters, etc, or they can describe lower level components such as individual transistors, resistors, and capacitors as shown in Figure 8. Typically, the genes constitute the level of granularity the GA is able to operate on, by either device limitations or design choice.
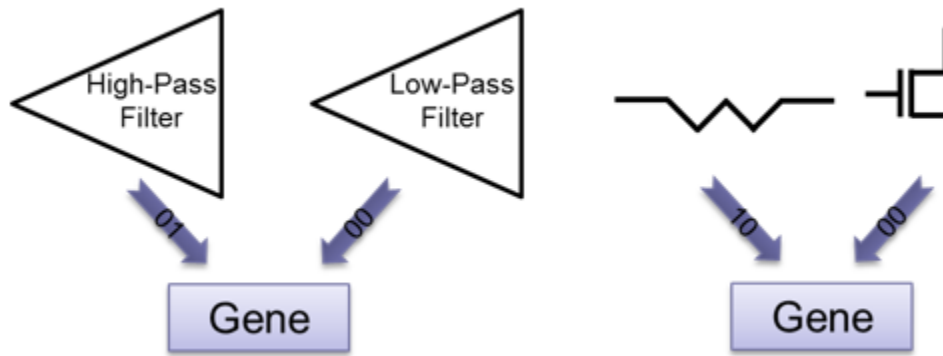
Figure 8: Genes used to represent high-level and/or low-level circuit components. Their configuration is determined by a binary string as shown.

For this study, each gene is corresponds to a *Computational Analog Element (CAE)*. Each CAE contains the functionality, parameters, and routing necessary to fully configure a single Switched Capacitor (SC) op amp block *(SC block)* found in the PSoC-5LP. The representation for these parameters are encoded in binary form as shown in Figure 10 to allow easy integration with the genetic operations. There are eight topologies available to choose from for each of the SC blocks that are shown in Figure 9. The selection of each resistor and capacitor value is

programmable within a palette of discrete values set by the manufacturer. Meanwhile, the routing between inputs, outputs, and other blocks is further explained later this Chapter.

Naked Opamp   **Unity Gain Buffer**   **Programmable Gain Amplifier**

Transimpedance Amplifier   **Continuous Time Mixer**   **Track and Hold Amplifier**

Delta Sigma Modulator   **Sampled Mixer**
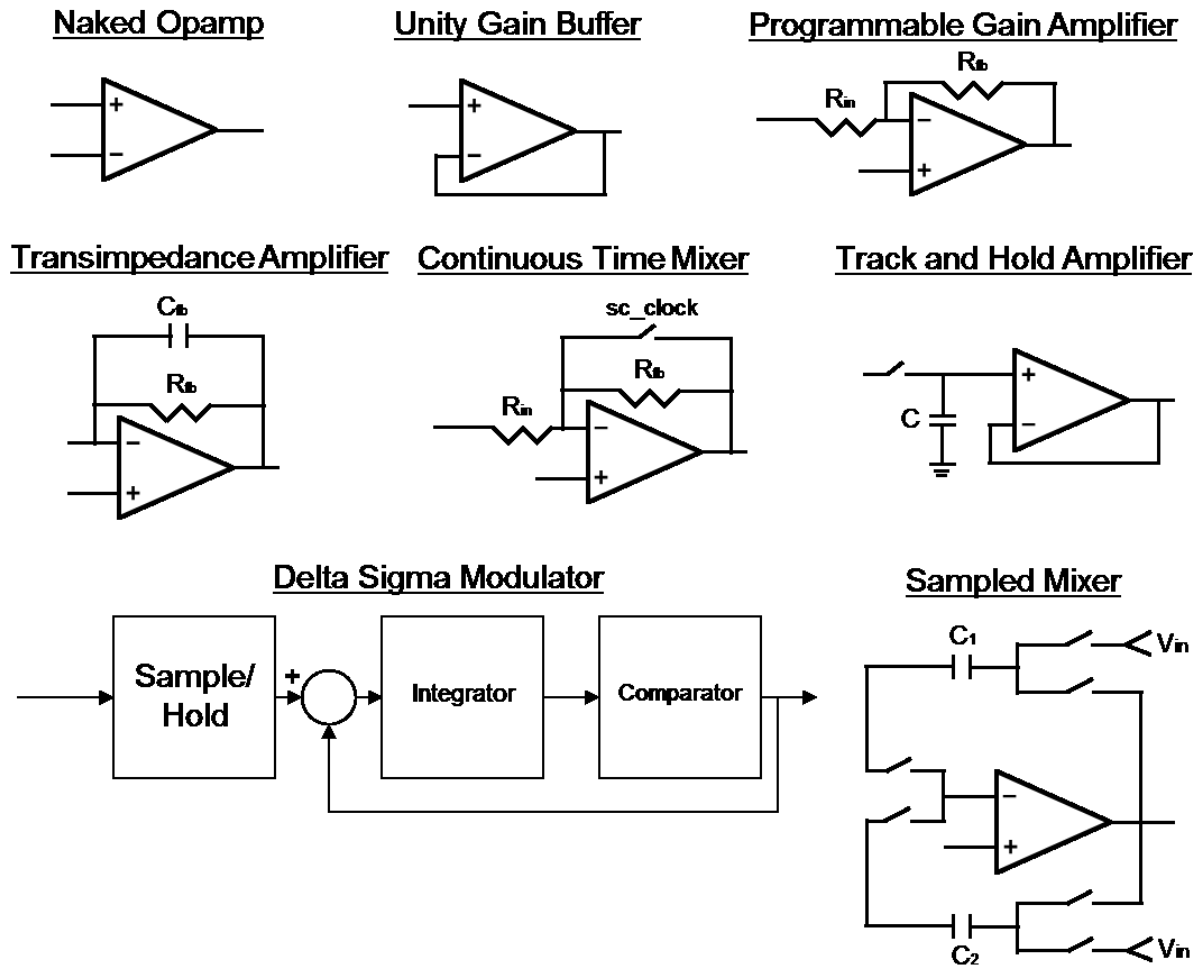
Figure 9: Eight different topologies possible for PSoC-5LP SC Opamp blocks.

Genome

Each individual's genome consists of all the genes necessary to configure the circuit's building blocks as well as the routing between the building blocks. The routing can be either encoded into each gene by choosing which routing lines the gene's terminals connect to, or

24

designated by a separate encoding within the genome but outside of the genes. Genomes are usually encoded into a binary representation as that is straighforward for computer software driven GAs to operate with as shown in Figure 10.
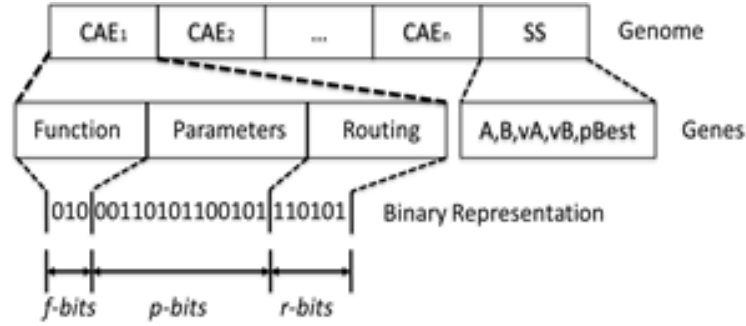
Figure 10: Breakdown of individual genome, gene expression, and binary representation for SSGA.

Fitness Evaluation

As the fitness function is what determines the quality of a candidate circuit, the definition of the fitness function is one of the most important aspects of the GA. The fitness evaluation stage of a GA can take on a variety of forms of which need to be chosen and tailored to the desired application. Often, and in this research, the process of fitness evaluation takes the form of applying a number of inputs to the circuit under evaluation and comparing the outputs with the pre-defined correct outputs for the particular input called the *oracle*. For the above case, fitness can be defined as $fitness = \sum_{t=0}^{T-1} output\big(input(t)\big) - oracle(t)$, where *t* is the particular test case, *input(t)* is the input value for test case *t*, *output()* is the circuit output for a given input, and *oracle(t)* is the predefined output that we desire our circuit to produce given our test case.

Selection

Individuals are selected for undergoing genetic operators to produce offspring based on their relative fitness levels. *Tournament Selection* is a popular method which randomly chooses `tournament_size` individuals from the population, and then chooses the best-fit individual as shown in Figure 11. This process is repeated again to select a second individual. Those two individuals are then chosen as parents to undergo genetic operators to produce two new individuals with similar genetic material to the parents. It can be advantageous to choose a `tournament_size` of just 2 in order to improve diversity, which generally provides greater GA performance.
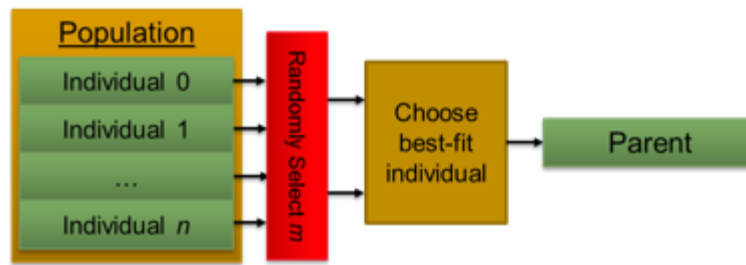
Figure 11: Pictorial representation of Tournament Selection.

Genetic Operators

The two primary genetic operators used in GAs are crossover and mutation. Crossover is performed on two individuals that have undergone the selection process by splitting the parent's genomes into one or more partitions as shown in Figure 12 and then recombining the genome components to produce two new offspring that contain the mixed genetic material of the two

parents. Partitions generally occur on the boundary between genes so that the functionality of the genetic makeup remains intact.

The mutation operation operates by scanning through each bit in each offspring's genome and with a low probability of `mutation_rate`, flips the bit as shown in Figure 13. `Mutation_rate` is generally in the range of around 1%, i.e. a probability of 0.001 to 0.05 on a scale from 0.0 to 1.0. The mutation operation provides an avenue for the GA to overcome limitations in the initial population's gene pool. For instance, the initial population rarely contains all possible genes available and may lack some genes which can provide better performance than the initial pool. If only crossover were used in the GA, these better-performing genes would never be realized as crossover would just be mixing genes around from the initial population. Mutation allows the opportunity for all genes to possibly be realized and utilized by the GA without devolving into random search.
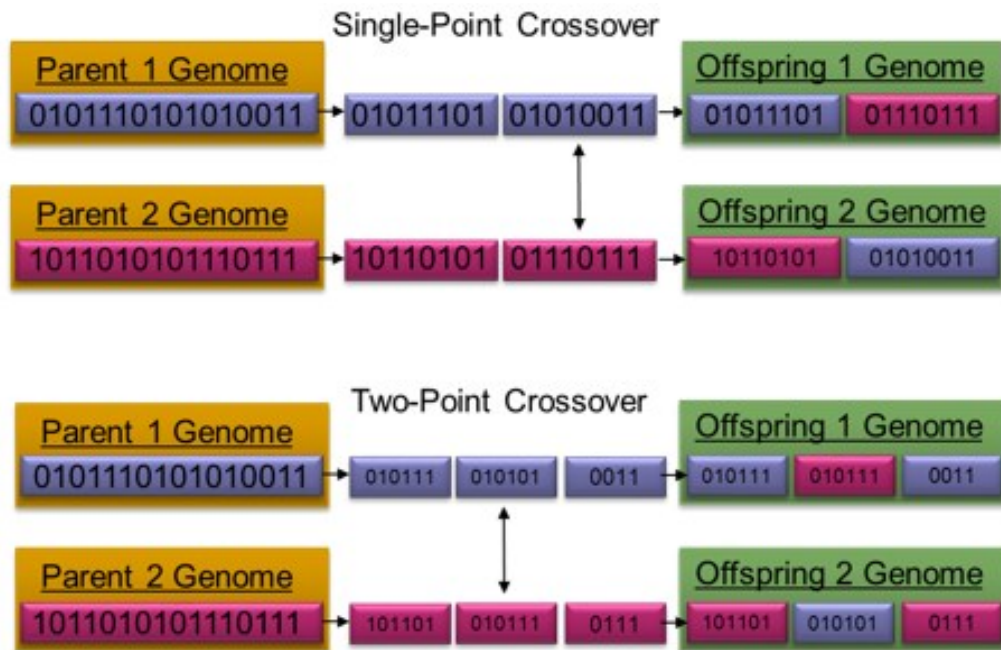
Figure 12: Pictorial Representation of the Crossover Genetic Operator. Both Single-Point and Two-Point variations are shown.
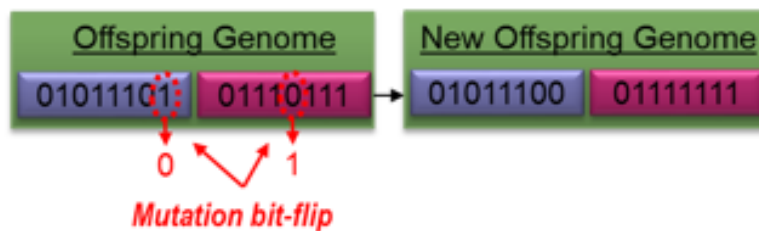
Figure 13: Mutation operation showing 2 different bit flips.

Elitism

Due to the stochastic nature of GAs, it becomes possible for the best fitness observed in each generation to degrade as the best-fit individual has the potential to produce offspring which have a lower fitness due to the crossover operation generating poor gene combinations or the mutation operator altering the genome to a less-fit configuration. For this reason, it's common for

GAs to implement *elitism*. Elitism transfers the best fit individual(s) to the next generation without altering their genomes. This will not exclude the best-fit individuals from the selection process, as their genes are valuable for the gene pool. When the GA is designed to transfer only the best-fit individual to the next generation, the GA is considered to have an elitism of one. If the GA retains the top two best-fit individuals, then it has an elitism of two, and so on. Although elitism is shown to improve GA-based hardware evolution, too much elitism decreases genetic diversity and can actually have negative effects on the evolution by causing early convergence. For this study the GA uses an elitism of two.

<center>Routing Encoding</center>

The routing scheme used for any EHW project needs to be tailored to the specific platform as different platforms have different resources and capabilities. Generally though, routing is encoded into the gene by choosing whether a particular terminal connects to an available routing wire or to another terminal on the same gene or a different one. For this study, the PSoC-5LP's analog routing capabilities can be summarized in Figure 14.

There are 4 primary analog busses, each of which consist of multiple wires that each available terminal in an SC block can connect to. The left and right analog buses each contain 4 wires, and the left and right global analog buses contain 8. Additionally, the left and right local buses can be connected together such that $analog_{local\_left}[0] = analog_{local\_right}[0]$ up to $analog_{local\_left}[3] = analog_{local\_right}[3]$; the left and right global buses can also be connected in such a manner. These buses are connected in this fashion for the experiments in this study

<center>29</center>

leading to a total of 12 common wires available for routing signals amongst the circuit. The routing in our genome is encoded as binary connection/no-connection values for each available terminal to each of the available analog wires in the local bus as well as the global bus. The routing bits correlate to configuration registers that handle the connections.
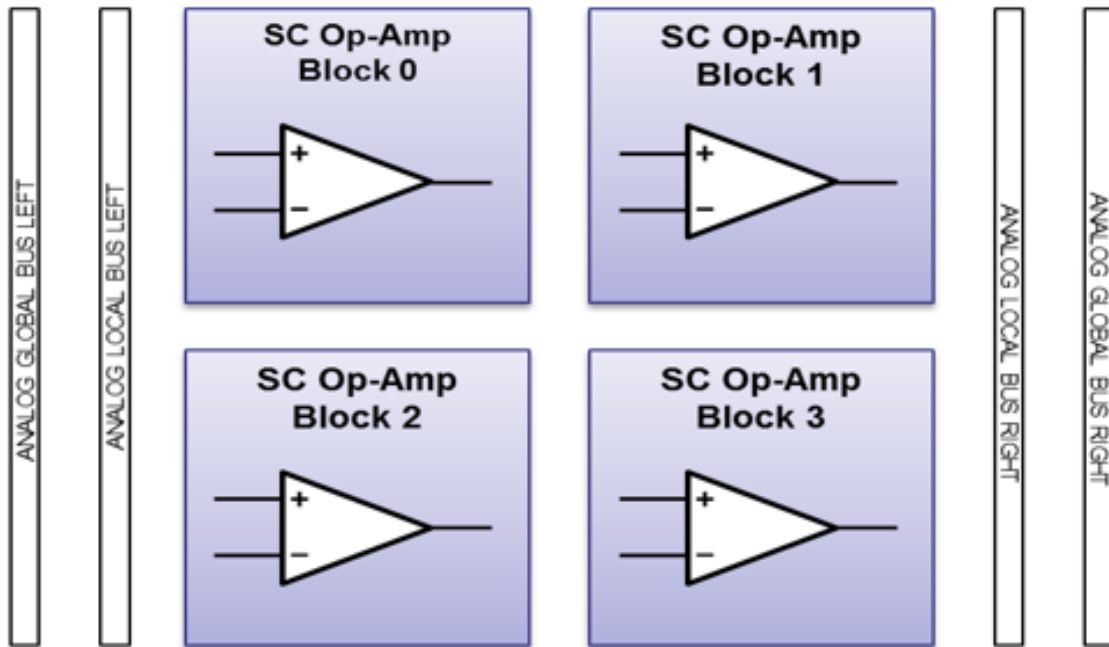
Figure 14: Routing architecture for PSoC-5LP Switched-Capacitor Op-Amp Blocks showing left/right local and global analog buses.

Analog Evolution Issues

When initially evolving analog CCs with a rudimentary GA, the evolution was observed to converge to solutions, which showed characteristics of the desired CC, but was limited by the available voltage range of the platform. Figure 15 shows a typical rudimentary GA-evolved cube circuit output measured on the PSOC-5LP intrinsic platform compared to the ideal curve. Since

the PSoC device on our prototype platform is limited to a 4.08V peak signal level, any input over

1.6V would exceed the platform's range for a cube CC. The evolved circuits were also observed

to have fluctuations in their accuracy as indicated by the deviations from the ideal curve. The

limited device range is mitigated using a SSGA, which is developed herein in Chapter 4, and the

accuracy issues are addressed using Differential Digital Correction, which is a technique

developed in conjunction with the SSGA techniques in this thesis; details are provided in a recent

Master's thesis [50].
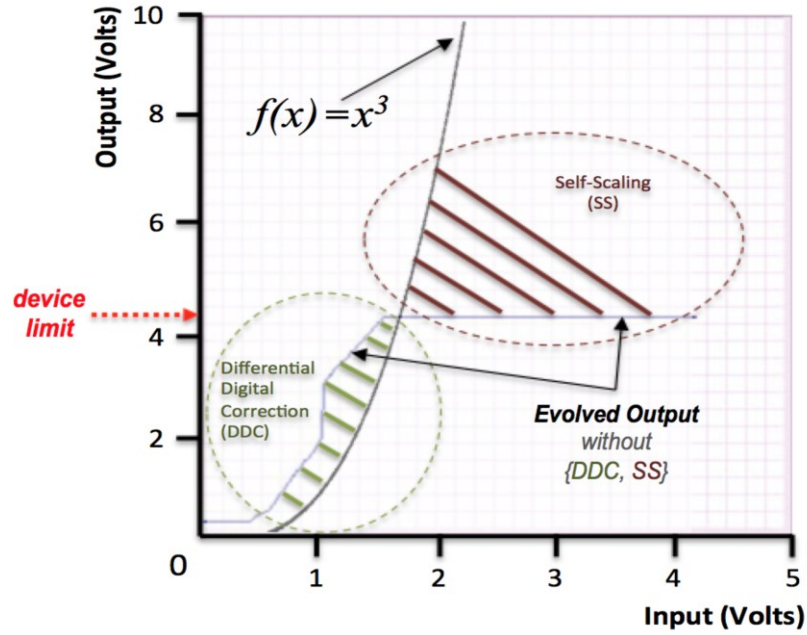
Figure 15: Analog Cube CC Evolved with Unrefined GA Compared to Ideal Curve.

Summary

This chapter went into depth concerning current techniques for EH that will be used in the

research herein, specifically these aspects of the GA:

- genes: genetic representation of the constituent computational blocks utilized by the GA,

31

- genome: combination of genetic material to fully realize a circuit,

- fitness function: the function which is to be optimized by the GA,

- selection: the method of choosing which individuals are to breed individuals for the next generation,

- genetic operators: the operators which mix and randomly alter the genomes of selected individuals to generate two new individuals made up of the genetic material of parents,

- and elitism: the aspect of saving top performing individuals to ensure there's never a reduction in best fitness.

This chapter also went over how routing can be encoded in GAs as well as how it's specifically implemented on the PSoC-5LP platform used in this research. The chapter was then wrapped up by delineating the issues encountered when evolving analog CCs and how this research plans to address these challenges.

CHAPTER FOUR: SELF-SCALING GENETIC ALGORITHM SUITABLE FOR ANALOG CIRCUIT EVOLUTION

As discussed in Chapter 3, one of the primary issues involved with utilizing an analog platform for CCs is that the dynamic range of computation available is limited to the maximum device voltage level. It is challenging to scale the inputs in such a way that the maximum output voltage would be within our device range limit, thus we seek an intelligent adaptive approach. We also consider that there may be particular atypical output voltage ranges that more effectively map our available resources to our desired CC, and we would like to allow the evolutionary algorithms to search for such exploitations.

To realize these objectives, the *Self-Scaling Genetic Algorithm (SSGA)* is proposed as an extension to GAs that interweaves PSO amongst the GA operation to search, adapt, and refine scaling and translation factors, which alter the CC output to improve dynamic range as well as search for possibly more computationally-tractable ranges. In order to develop the SSGA extension, a PSO algorithm must be tailored for our application, the GA fitness function needs to be adjusted to include the scaling and translation factors, and the individual's genome can be adapted to keep track of and utilize the scaling and translation factors. Two other alterations to the standard GA are developed to improve SSGA-based circuit evolution. Those include the development of an *island-like GA* and tailoring a *hypermutation* genetic operator to the SSGA. The SSGA is one of two techniques developed to work in relay to realize *Scaling Evolutionary Refinement (SCALER)* as proposed in [51] and shown in Figure 16.
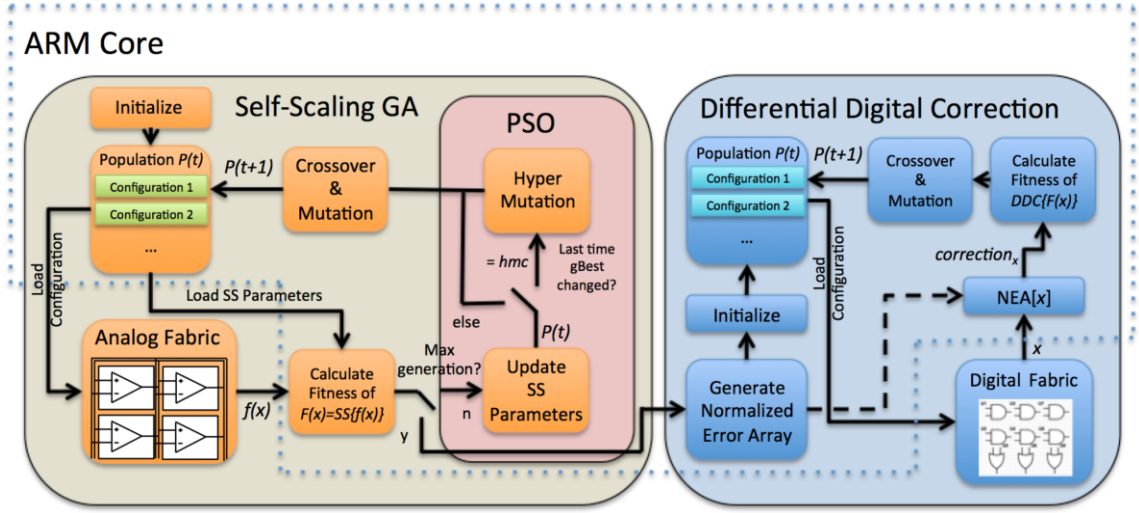
Figure 16: Scaling Evolutionary Refinement.

While SSGA is the primary focus of this thesis, the interested reader is referred to [50] for details on the design and operation of *Differential Digital Correction (DDC)*.

Self-Scaling Parameters

As shown in Figure 17, the flow to develop the SSGA consists of three major research objectives highlighted in blue. The leftmost step denotes the standard GA developed for analog CC evolution on the PSoC-5LP platform. The middle step denotes the development of a self-scaling functionality optimization scheme utilizing PSO. The rightmost step denotes the extensions developed to circumvent premature convergence and improve SSGA evolution.

The *Self-Scaling Parameters*, or *SS parameters* are the *scaling (A)* and *translation (B)* parameters. The scaling parameter is a real, positive, non-zero number that is multiplied to the output of the analog circuit under evaluation or operation to scale the circuit functionality to a particular range determined by the SSGA. The translation parameter is a real number that's added

34

to the analog circuit in order to translate the device range as determined by the SSGA. The output

of a CC evolved by the SSGA becomes $output = Af(x) + B$, where $f(x)$ is the original voltage

level output from the analog CC.
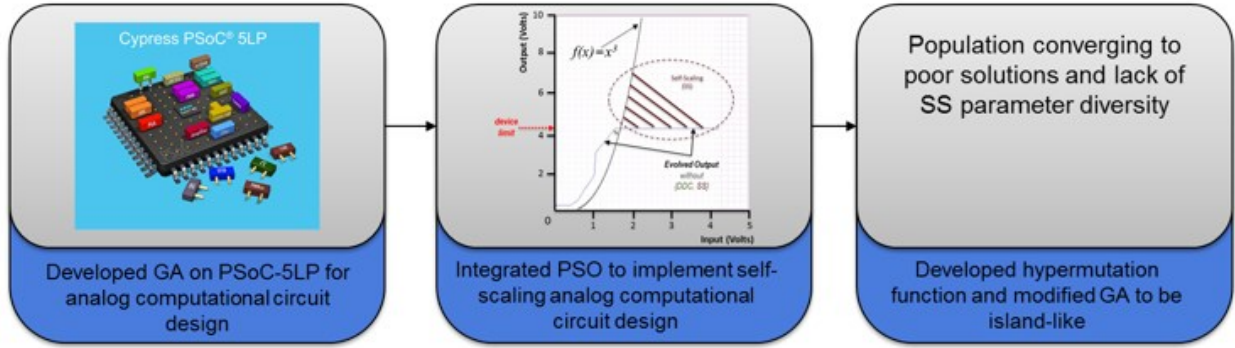
Figure 17: SSGA design flow.
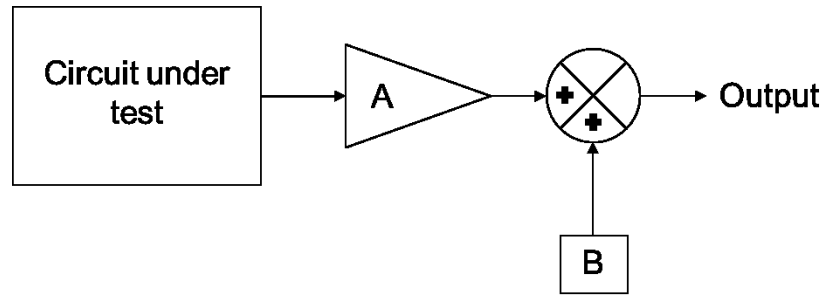
Figure 18: Implementation of scaling (A) and translation (B) parameters into circuit output.

<u>Delineation of Particle Swarm Optimization</u>

PSO is utilized in the SSGA to optimize the SS parameters expounded on in the previous

section. The PSO algorithm operates by iteratively maneuvering candidate solutions within the

search-space, testing each candidate each iteration, and then updating the candidates' positions

based on their previous best as well as the global best solution found.

For this study, a two-dimensional PSO algorithm is implemented within a GA to optimize the two SS parameters while the population is under evolution. It was tested whether the PSO algorithm was best implemented separately from the GA, as in the GA would evolve the best candidate circuit that it could, and then the PSO algorithm would commence to optimize that particular circuit, which would improve the fitness of the candidate circuit, but it was observed that operating the PSO algorithm while the populations were evolving produced the best-fit and most interesting solutions. The SS parameters *A* and *B* are initialized for each individual when the populations are initialized by randomly assigning values between `pmin` and `pmax`.
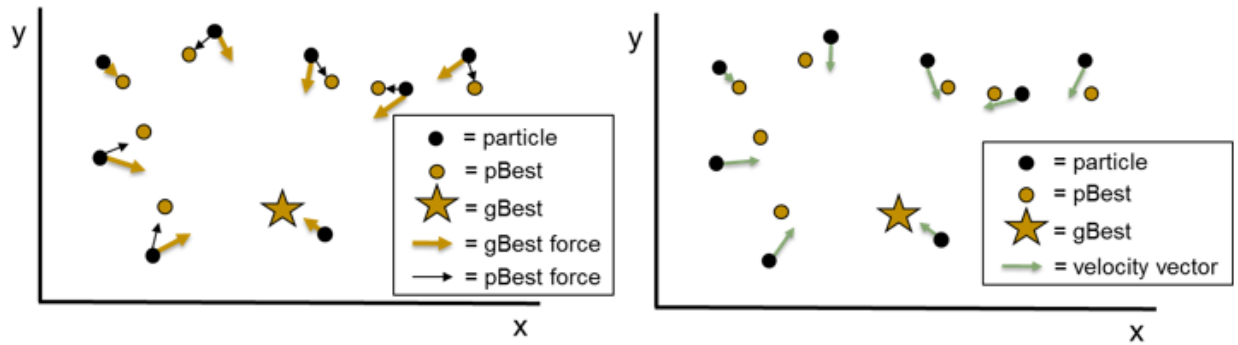
Figure 19: PSO particle forces (shown left) and cumulative velocity vectors (shown right).

Alterations of the Fitness Function

The fitness function utilized in the SSGA for our study is very similar to the general fitness scheme, that is, a number of test inputs are applied to the circuit under test, then the outputs are measured and compared against a pre-defined oracle. The fitness is then calculated as the sum of errors for all of the test points. This means that for this study a lower fitness value corresponds to a better fit candidate circuit as it has fewer errors.

36

One particular adaptation developed for the fitness function is the use of a penalty operation. The penalty operation is implemented by adding `penalty` points to the fitness of a candidate circuit under test for each error larger than `penalty_case`. This was shown to greatly improve the evolution of the population because circuits that deviated heavily from the desired functionality were penalized, and therefore had smaller chances to be selected for breeding.

To implement our SSGA, a two-dimensional PSO algorithm is used to optimize two *Self-Scaling* parameters, a scaling factor, *A*, and a translation factor, *B*, such that

$$SS\{f(x)\} = Af(x) + B \tag{1}$$

is more accurately mapped to our desired function, i.e. has a better fitness, where *f(x)* is our raw output from the evolved analog circuit and *SS* is our *Self-Scaling* transformation. The GA is extended to a SSGA by altering the fitness function, as shown in Equation 2, and updating the *SS* parameters with PSO as the GA is running. The flow of SSGA is shown in Algorithm 1.

$$fitness = \sum_{t=0}^{T-1} |SS(output(input(t))) - oracle(t)| \tag{2}$$

Initial tests attempted to adjust the fitness function to include a weighted combination, fixed or adaptive, of both the raw analog circuit fitness and the SSGA adapted fitness, which showed improvements in fitness. However, the best fitness improvement was obtained when we only considered the SSGA adapted fitness to evaluate our individuals.

Genetic Operators Details

The next generation for each population is developed by selecting 2 parents via tournament selection and performing crossover with their genomes to make two offspring, which contain the

mixed genes of their parents. Crossover has a 50/50 chance to perform single-point or 2-point crossover. Once selection and crossover has been completed *N/2-1* times to generate a total of *N* new individuals combined with the best fit and second best fit, mutation is performed on all except the best fit individual.

Adaptive Mutation Rate

Mutation is designed as a low probability chance, $P_{mutation}$, of flipping a single bit in each of the parameters in the genome, excluding the SS parameters. In order to help lift the GA out of local minimums, $P_{mutation}$ is varied dynamically based on convergence, which we define as as the fitness difference between the best fit individual and the average fitness. $P_{mutation}$ is doubled when $|Fitness_{average} - Fitness_{best}| < Fitness_{best}$ and tripled when $|Fitness_{average} - Fitness_{best}| < 0.5 \times Fitness_{best}$.

Alterations to the Genome

Figure 20 shows the complete individual genome consisting of *n* genes, which specifies configurable characteristics of all CAEs for the target application, as well as the *SS parameters* for scaling and translation, *A* and *B*, their velocity parameters, *vA* and *vB*, as well and their previous best value *pBest*. Each CAE requires *f* bits to describe its function, *p* bits to describe the various parameters of the components contained within, such as resistance values, and the *r* bits to determine routing.
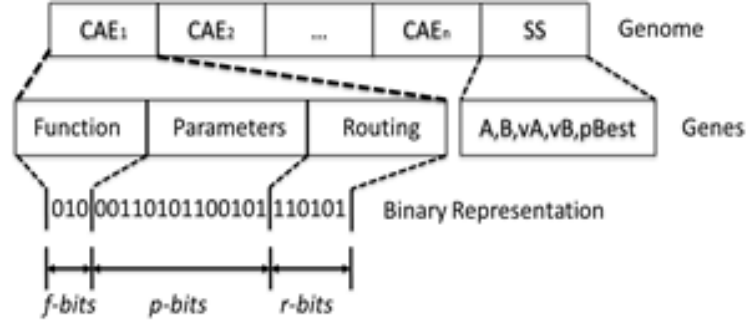
Figure 20: Breakdown of individual genome, gene expression, and binary representation for SSGA

Island-Like Genetic Algorithm

Due to the stochastic nature of such algorithms, it's observed that the SSGA sometimes converges to poor solutions. To alleviate this issue, an island-like GA is used with SSGA. We consider it island-like because neither individual's genomes nor genes are shared, but only SS parameters. At every `gen_share` generations, all of the island populations are checked, and the best-fit island's SS parameters are shared. This greatly increases the chance of finding high quality solutions and SS parameters, and allows populations consisting of different genes and genomes to evolve with known good SS parameters, possibly giving rise to further improvements.

Hypermutation Genetic Operator

Because PSO is now being performed on a dynamic population whose functional ranges are changing, we implement a *hypermutation* function, similar to [16], to help circumvent local minima and find new optimization parameters for new generations. The hypermutation simply reinitializes the scaling and translation factors (*A* and *B*) of all the particles to random values within

39

the range [`pmin, pmax`], and reinitializes half of the population to random individuals so that there is enough genetic diversity to possibly make use of the new SS parameters. The hypermutation function does not modify the *gBest* parameters, so the SSGA has the chance to explore new parameters without sacrificing current best parameters. Hypermutation is performed when *gBest* hasn't improved for `hypermutation_condition` generations.

<u>Self-Scaling Genetic Algorithm</u>

All of the techniques developed in this chapter are merged into the complete SSGA shown below in Algorithm 1. Starting with line one, both of the timer variables are initialized, which keep track of the current generation (`t`), as well as the number of generations without any improvement in the globally best fit individual (`gBest_time`). Then, the first generation of the population (`P(0)`) is initialized using the `Initialize_population()` function, which simply generates all of the individuals in the population randomly using the built in `rand()` functionality in C. One the first population is generated, each individual in the population is evaluated using the `Evaluate_Fitness()` function. Once all of the fitness' are calculated, we use lines 6-14 to update our previous best values for each of the particles in the PSO as well as the globally best value. Line 16 updates the SS parameters for each particle according to the PSO algorithm delineated earlier. Lines 17-20 check for an update to the globally best fit value, and if found true, it increments `gBest_time.` If not found true, that means the globally best fit value has improve, and therefore it resets `gBest_time` back to 0. Lines 21-22 hypermutate the population according to the hupermutation genetic operator delineated earlier if the globally best fit value hasn't changed for

a certain condition, that is `gBest_time == hypermuation_condition`. Line 23 performs the core

GA operators of crossover and mutation on the population to generate the next generation. Line

24 increments our generation counter as well as updates our `prev_gBest` to our current `gBest`. The

final line simply checks for our termination condition, which is when we reach `max_generation`

generations.

```
Algorithm 1: Self-Scaling Genetic Algorithm
1: t := 0; gBest_time = 0;
2: P(0) = Initialize_Population;
3: Repeat{
4:        Evaluate_Fitness(P(t));
5:        for each particle i in P(t)
6:                if fitness(i) < pBest(i){
7:                        pBest(i) = fitness(i)
8:                        pBest_A(i) = A(i);
9:                        pBest_B(i) = B(i);
10:               }
11:               if fitness(i) < gBest{
12:                       gBest = fitness(i);
13:                       gBest_A = A(i);
14:                       gBest_B = B(i);
15:               }
16:               PSO_Update (vA(i),vB(i),A(i),B(i));
17:       if (gBest == prev_gBest)
18:               gBest_time++;
19:       else
20:               gBest_time = 0;
21:       if (gBest_time == hypermutation_condition)
22:               Hypermutate(P(t));
23:       P(t+1) = GA_Operators(P(t));
24:       t = t + 1; prev_gBest = gBest;
25: }Until (t == max_generation);
```

Summary

This Chapter introduced the techniques developed in this Thesis as they are integrated into the SSGA. The SS parameters are defined and expounded upon how they interact with the outputs generated from the analog circuits. The PSO algorithm is also further delineated as it is implemented within a GA framework to adapt the SS parameters during GA driven evolution of the reconfigurable analog fabric on the PSoC-5LP. The alterations to the fitness function, genome, and genetic operators necessary for successful implementation of the SSGA is also described. The island-like GA utilized in this study is also defined and its benefits are expressed. The hypermutation genetic operator is also defined. Finally, the chapter is wrapped up with the full SSGA shown in a step-wise fashion.

CHAPTER FIVE: SELF-SCALING GENETIC ALGORITHM

PERFORMANCE ANALYSIS

Experimental Setup

To determine the parameters to be used for the study, a variety of configurations are explored and the best performing parameters are then utilized. Based on these findings, the global parameters are determined as follows. The maximum number of generations permitted for the GA to run is 500, as more generations did not produce a significant improvement in fitness, if at all. The minimum value that the scaling SS parameter can take is 0, and the minimum value the translation SS parameter can take is the negative of the `pmax` value shown in Table 2. The number of populations in the island-like GA is 4. The number of individuals in each population is 30. The level of elitism is 2. A tournament selection based selection scheme is utilized with a tournament_size of 2. The amount of penalty points that's added to the fitness of individuals with test outputs greater than `penalty_case` is 10. The number of stagnant generations that the hypermutation function is to occur is 100, and the number of generations before SS parameters are shared amongst the islands is 200. Parameters `pmax` and `penalty_case` are altered for each test case and are delineated in Table 2. All parameters chosen are roughly optimized values based on trial experimentation as described later.

Standard Genetic Algorithm Setup

In order to compare the results of using SSGA to standard GA evolution, we develop a standard GA that is as similar to our SSGA as possible, but with no scaling or translating, no island-like sharing of parameters, and no hypermutation. Since there's no islands in this GA, we utilize a single population of 120 individuals, which is the same total number of individuals for the SSGA. The fitness function used for the standard GA is simply $fitness = \sum_{t=0}^{T-1} output\big(input(t)\big) - oracle(t)$.

Circuit Complexity

In order to compare our circuit complexity to [5, 7, 16] , all of which used an unconstrained quantity of resistors and Bipolar Junction Transistors (BJTs) to evolve their CCs, we've identified a complexity cost metric, which relates roughly to computational capability. The complexity of a component is assigned according to the range of operations it can perform. Since resistors can only perform 1 operation, they are assigned a complexity of 1. BJTs can be wired up in 4 different configurations, and therefore have a complexity of 4. The SC blocks of the PSoC 5LP can perform 8 functions, and therefore it has a complexity of 8. Since we have 4 SC blocks for our CCs, our circuits have a fixed complexity of 32.

<u>Test Cases</u>

The input/output ranges in Table 2 show the difference between attempting to evolve CCs with SSGA versus a standard GA running on the PSoC-5LP. Due to the native range of the device, evolving without any form of scaling would severely reduce computational range for square and cube circuits. Each test case is evolved a total of 5 times each with both a standard GA and the SSGA and the average error and average fitness is computed and compared.

Table 2: Computational Circuit test cases used in literature and herein.

	Square	Square-root	Cube	Cube-root
Native input range w/ simple GA	0V-2.02V	0V-4.08V	0V-1.60V	0V-4.08V
Effective input Range with SSGA	0V-4.08V	0V-4.08V	0V-4.08V	0V-4.08V
Native Output Range	0V-4.08V	0V-2.02V	0V-4.08V	0V-1.60V
Effective Output range with SSGA	0V-16.65V	0V-2.02V *	0V-67.92V	0V-1.60V *
pmax	20	2	68	2
penalty_case	0.5V	0.1V	0.5V	0.1V

*Computational range falls within native device range

<u>Test Case Results</u>

In this section, each test case is evaluated along with a graph of the scaled and translated output from a typical SSGA evolutionary case is shown along with the fitness versus time graphs for a representative example.

45

Square-Root Computational Circuit
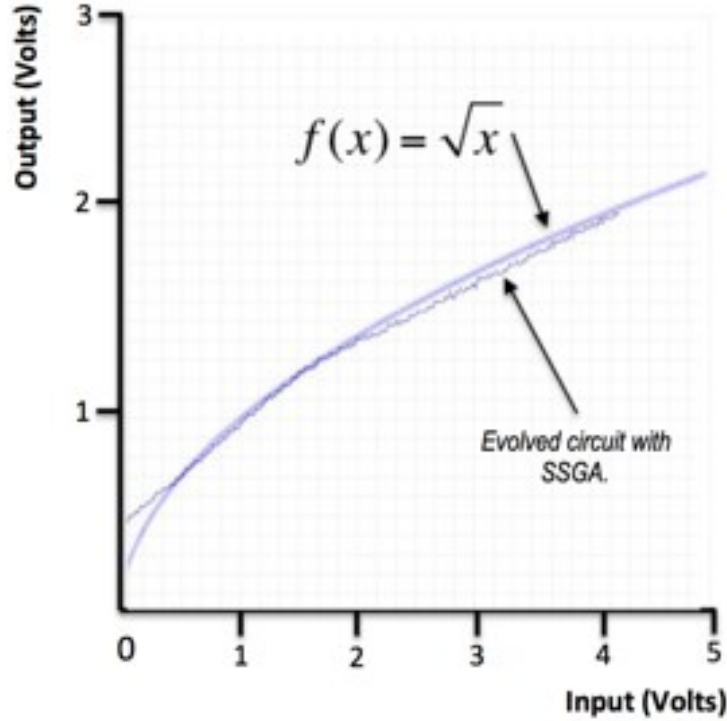
Figure 21: SSGA Evolved Square-Root Circuit Compared to Ideal Curve.

The first CC evolved with the SSGA was the square-root function, with a typical SSGA evolved output and ideal curve shown in Figure 21. The average error of the best solution found after 500 generations is only 30mV, which is much better compared to [5], that had an average error of 183.57mV, but worse when compared to [16] and [7], which had an average error of 20.0mV and 9.23mV, respectively. Even though the SSGA evolved square-root circuit on the PSoC-5LP performed comparably worse than [16] and [7], consider that those papers used extrinsic evolution in a simulation environment and developed circuits with much greater complexity (84 and 60, respectively) than the SSGA evolved CC. When evolving the square-root CC with a standard GA, the average fitness was 9.45. The average fitness when the square-root

46

CC was evolved with a SSGA was 1.35, leading to a 7x improvement in fitness when utilizing the SSGA versus the standard GA. Figure 22 shows a typical fitness versus generation graph for the SSGA evolution of the square-root circuit. Each island's average fitness value is plotted in red along with the best fitness in the island, which is plotted in blue.

Figure 22: Typical Fitness Over Time Graph for Square-root CC Evolution with SSGA.

Cube-Root Computational Circuit
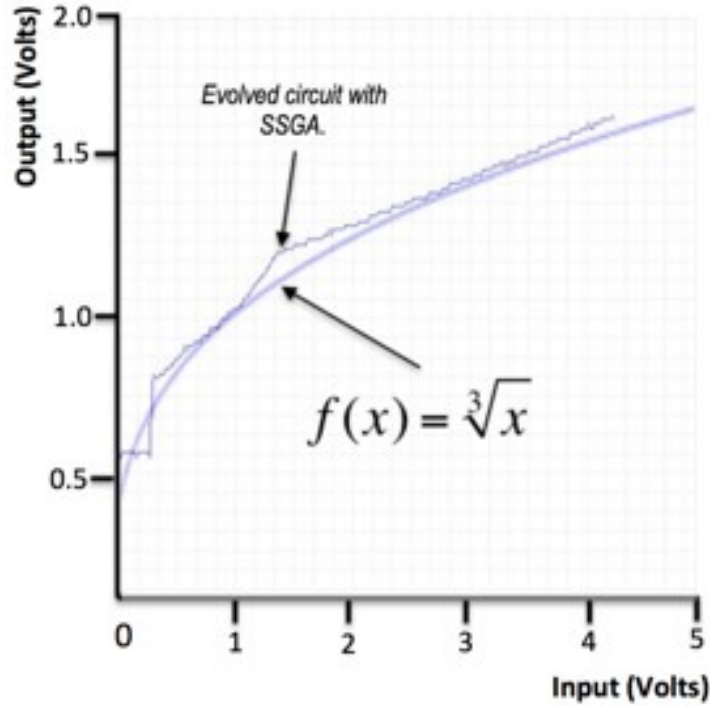
Figure 23: SSGA Evolved Cube-Root Circuit Compared to Ideal Curve.

The next CC studied is the cube-root function with a typical SSGA evolved output and ideal curve shown in Figure 23. The average error for the best solution evolved with the SSGA was found to be 23mV, which compares favorably to [5], which had an average error of 80mV, and is similar, albeit slightly poorer, to [7], which had an average error of 11.9mV. The complexity for both [5] and [7] were much greater than the complexity of the circuit evolved in this study, having a circuit complexity of 164 and 116, respectively. The average fitness of the cube-root CC evolved with a standard GA is 7.54. When evolved with the SSGA, average fitness was 1.49, giving a 5.07x improvement in circuit fitness. Figure 24 shows a typical fitness versus generation

graph for the SSGA evolution of the cube-root circuit. Each island's average fitness value is plotted

in red along with the best fitness in the island, which is plotted in blue.

Figure 24: Typical Fitness Over Time Graph for Cube-root CC Evolution with SSGA.

Square Computational Circuit
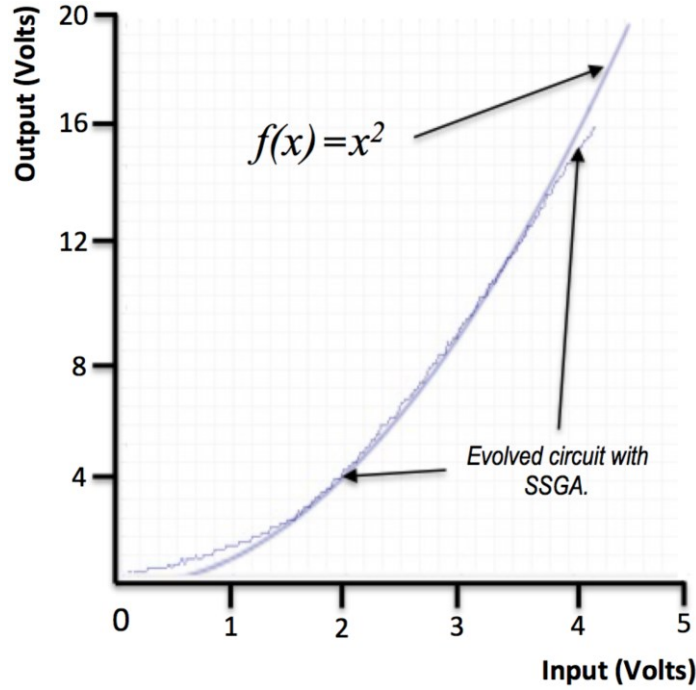
Figure 25: SSGA Evolved Square Circuit Compared to Ideal Curve.

The previous two CCs evolved were functions which could be evolved to the PSoC-5LP fabric without any self-scaling necessary to accommodate the device's voltage range. Now the square CC is evolved with a standard GA as well as the SSGA to demonstrate the SSGA's dynamic range scaling properties, as the square circuit does not natively map to the PSoC-5LP's voltage range. When evolved with a standard GA, the square CC had an average fitness of 506.95, but when evolved with the SSGA, the square CC had an average fitness of 39.23, leading to an improvement of 12.92-fold, the greatest improvement observed. The average error for the SSGA evolved square CC was determined to be 140mV, which is worse than in [5] and [7], which had an average error of 27mV and 1.44mV, respectively. However, consider that the circuit

complexities were 72 and 118, respectively, and that they were evolved extrinsically in a simulation environment. Figure 26 shows a typical fitness versus generation graph for the SSGA evolution of the square circuit. Each island's average fitness value is plotted in red along with the best fitness in the island, which is plotted in blue.
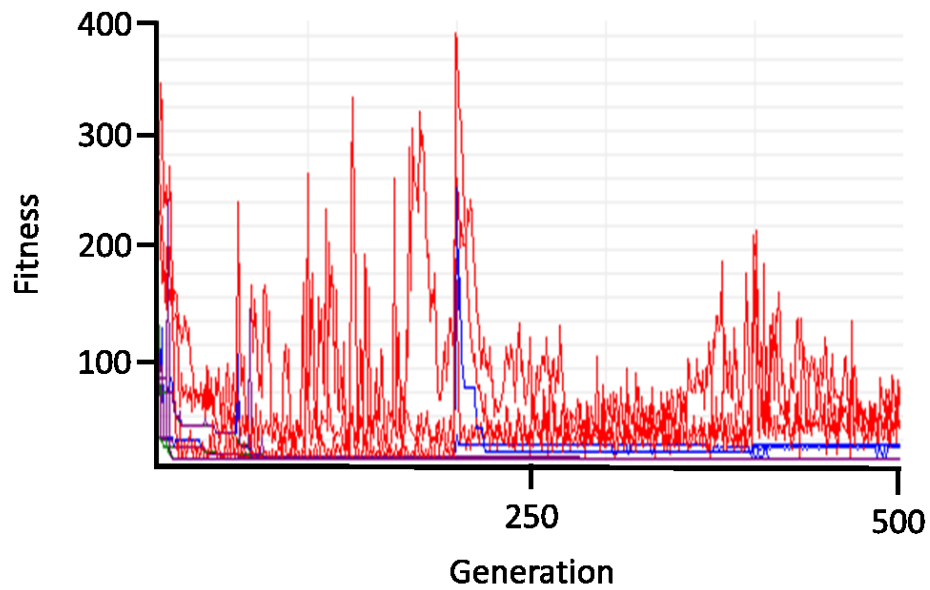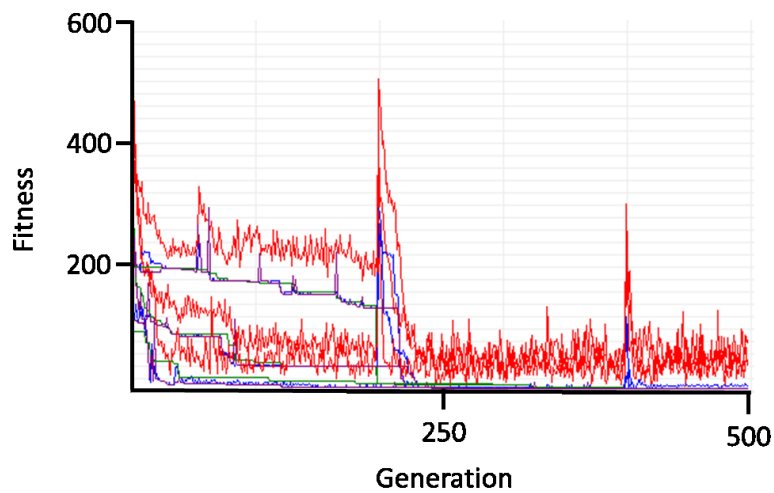

Figure 26: Typical Fitness Over Time Graph for Square CC Evolution with SSGA.

Cube Computational Circuit
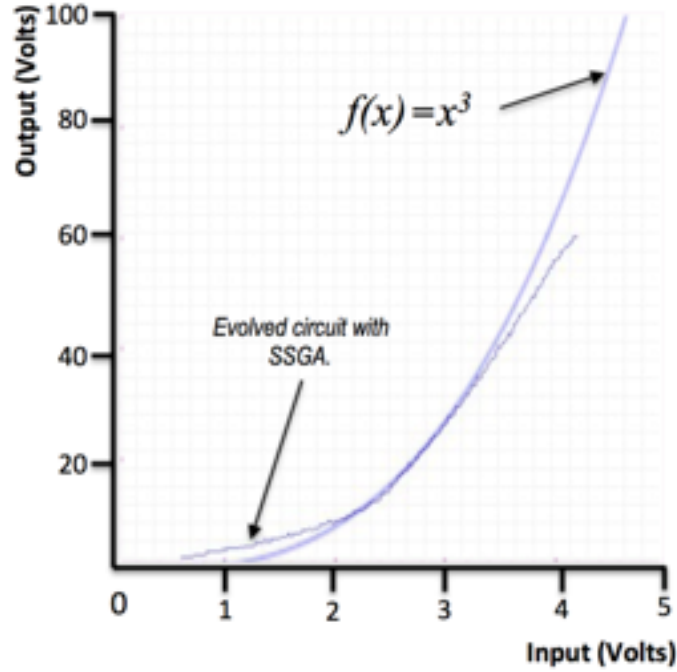
Figure 27: SSGA Evolved Cube Circuit Compared to Ideal Curve.

The final test case in this study is the cube CC, which has a functional range much larger than the intrinsic voltage range of the PSoC-5LP, so this test case is the best demonstration of the SSGA's scaling functionality. For the evolution of a cube CC with a standard GA, the average fitness was found to be 1084.45, while the average fitness when evolved with the SSGA was found to be 291.56, leading to an improvement of 3.72-fold. The high, and therefore relatively poor, fitness value of the SSGA evolved cube CC can be attributed to penalization. The average error was found to be 1.16V, and since the circuit is penalized whenever a test point is more than 0.5V different then the oracle, the cube CC is heavily penalized, even for the best evolved solution. This can be attributed to the fact that the cube CC required the greatest scaling, and therefore any errors

are scaled as well. Figure 28 shows a typical fitness versus generation graph for the SSGA evolution of the cube circuit. Each island's average fitness value is plotted in red along with the best fitness in the island, which is plotted in blue.
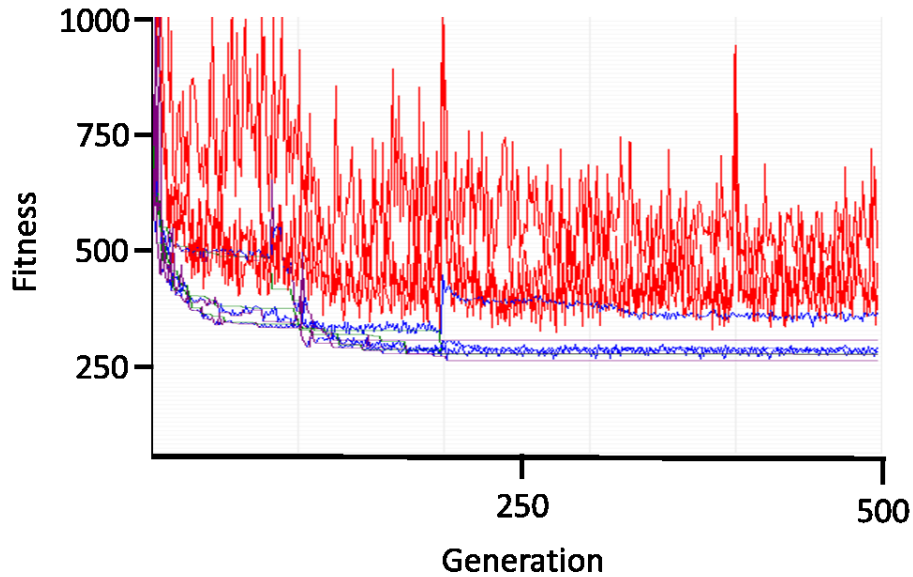
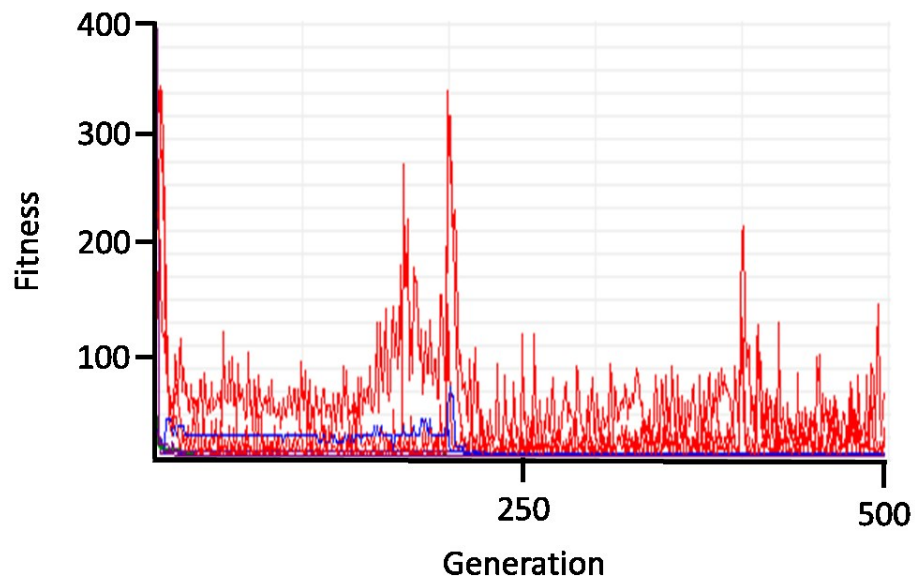Figure 28: Typical Fitness Over Time Graph for Cube CC Evolution with SSGA.

Overall Results Including Differential Digital Correction

As the companion technique to the SSGA, the results of DDC is delineated herein. DDC was able to improve the average error in all test cases as shown in Table 4. For the square-root CC, DDC was able to reduce the average error from 30mV to 26.8mV. For the cube-root CC, DDC reduced the average error from 23mV to 19.25mV. For the square CC test case, DDC reduced the average error from 140mV to 100mV. And for the cube CC, DDC reduced the average error from 1160mV to 732mV. A typical complete evolutionary run including both SSGA and DDC is shown in Figure 29. First, the analog evolution phase shows the generation-by-generation best-fit

53

individual's total error (shown in blue) and average total error (shown in red) for each of the island populations, and then after 500 generations, the digital refinement phase uses DDC to significantly reduce the total error.

These results indicate that digital refinement techniques are an intriguing method of using analog and digital resources in tandem to produce improved results in new and interesting computational fashions.
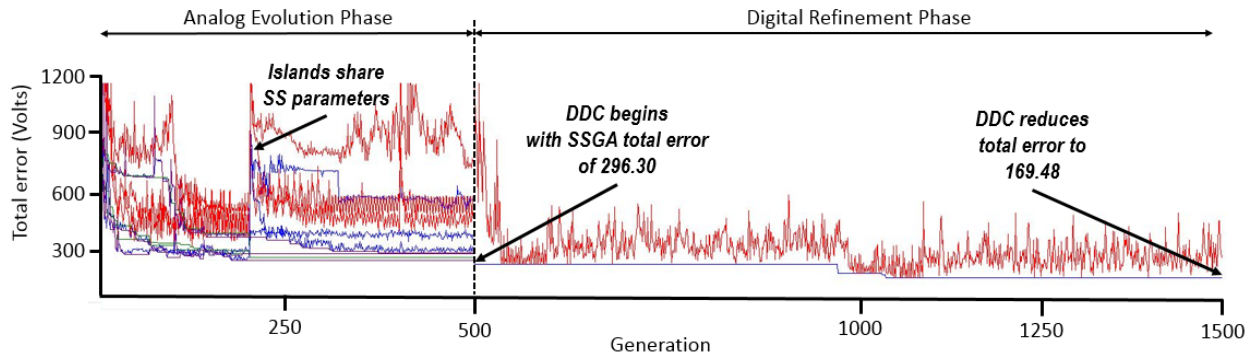
Figure 29: Average total error (red) and best total error (blue) evolution shown for both SSGA (analog evolution phase) and DDC (Digital Refinement Phase). During the analog evolution phase, the multiple average and best fitness lines are for each of the islands.

Results Overview

Table 3 shows the overall fitness scores of the four CCs evolved, indicating significant improvements of using the SSGA versus an unrefined GA. Each of the evolved CC produced solutions, which closely matched their ideal outputs as shown in the previous sections. The worst performing circuit, as far as fitness is concerned, is the cube circuit, which is understandable considering it required the greatest scaling beyond its native range. Considering that test points are penalized when they are more than 0.5V away from the oracle, and the cube circuit had an average

error of 1.19V, the penalization kept the cube circuit from obtaining better fitness values, even though it is relatively accurate as can be seen in Figure 27. Even though the cube circuit had the worst fitness, it was the best demonstration of the SSGA as it was able to increase its effective range by 17-fold.

Table 3: Standard GA and SSGA Evolved CC Fitness Results

Circuit	GA Average Fitness	SSGA Average Fitness	Improvement
Square	506.95	39.23	12.92
Square-root	9.45	1.35	7.00
Cube	1084.45	291.56	3.72
Cube-root	7.54	1.49	5.07
		Average Improvement:	7.18

The square circuit showed the most significant improvements, which is reasonable considering that the square circuits' effective range is unobtainable with an unrefined GA, but does not have to be scaled as greatly as the cube circuit required. Furthermore, the square circuit only had an average error of 187 mV, so it is rarely penalized. Square-root and cube-root both were able to evolve good solutions with the unrefined GA, but still showed significant improvements when evolved with the SSGA. Interestingly, the best observed fitness amongst all of our tests was a standard GA evolution of cube-root, which gave a fitness of 0.85. However, this was an atypical

case, as the average fitness scores show significant improvements from using the SSGA versus the unrefined GA.

Compared to the results of the previous works in Table 4, the square-root and cube-root CCs evolved with SSGA achieved an average error of 19.6mV and 20mV, respectively, and performed better than Koza et al. The square-root CC evolved in this paper performed marginally better than Mydlowee et al., with an average error of 20mV, but the square CC did not outperform. All test cases performed worse than Sapargaliyev et al., but considering their work evolved CCs extrinsically without device constraints, this is understandable. As far as the authors are aware, this is the first realization of intrinsic evolution of analog CCs on a commercial PSoC device utilizing a compact fabric of 4 SC op-amp Blocks.

Table 4: Results compared to previous works.

		[Koza 97]	[Mydlowec 00]	[Sapargaliyev 12]	SCALED SSGA	DDCGA
Square root	Average error, mV	183.57	20.00	9.23	30.00	26.8
	Average Fitness	3.86	70.40	0.19	8.14	6.786
	Complexity	-	84.00	60.00	32.00	-
Square	Average error, mV	-	27.00	1.44	140.00	100
	Average Fitness	-	4.81	0.03	35.23	25.11
	Complexity	-	72.00	118.00	32.00	-
Cube-root	Average error, mV	80.00	-	11.90	23.00	19.25
	Fitness	1.68	-	0.25	5.98	5.032
	Complexity	164.00	-	116.00	32.00	-
Cube	Average error, mV	-	-	11.90	1160.00	732.00
	Average Fitness	-	-	0.25	296.30	187.67
	Complexity	-	-	141.00	32.00	-

Summary

This chapter laid out the results generated from SSGA-driven intrinsic evolution of four computational circuits. All standard GA and SSGA parameters are described along with a definition of the complexity metric that is used to compare the constrained reconfigurable analog fabric on the PSoC-5LP to the circuits generated extrinsically by previous works. The results for evolving the cube, cube-root, square, and square-root CCs are then described. Each CC showed strong improvements when evolved with the SSGA versus the standard GA. Some of the results showed a reduced average error compared to previous works, but not for others. This can be

57

attributed to the small amount of components available on the PSoC-5LP compared to the simulation environments used in the compared previous works. A representative example of the digital refinement technique called DDC is then shown to reduce the average error of an evolved square circuit significantly.

CHAPTER SIX: CONCLUSION

This thesis developed the Self-Scaling Genetic Algorithm (SSGA), a method to scale, translate, and adapt evolved analog computational circuits. The SSGA supports the implementation of analog circuits on a resource-constrained and voltage-range restricted platform by utilizing PSO to optimize scaling and translation factors during the evolution of circuit topologies using a GA. As shown in Figure 30, several conclusions can be drawn from the results developed herein. First, extending GAs with PSO has proved to be an effective method of adapting analog solutions to the platform's computationally-tractable range. Next, it was demonstrated in this thesis that it is possible to use relatively limited reconfigurable resources to realize intrinsic analog CC evolution. Finally, results compared favorably to previous works using metrics of average error, fitness, and circuit complexity. A conclusion drawn from these results is that intrinsic evolution is clearly beneficial to minimizing these errors on a physical device, as will be discussed below.

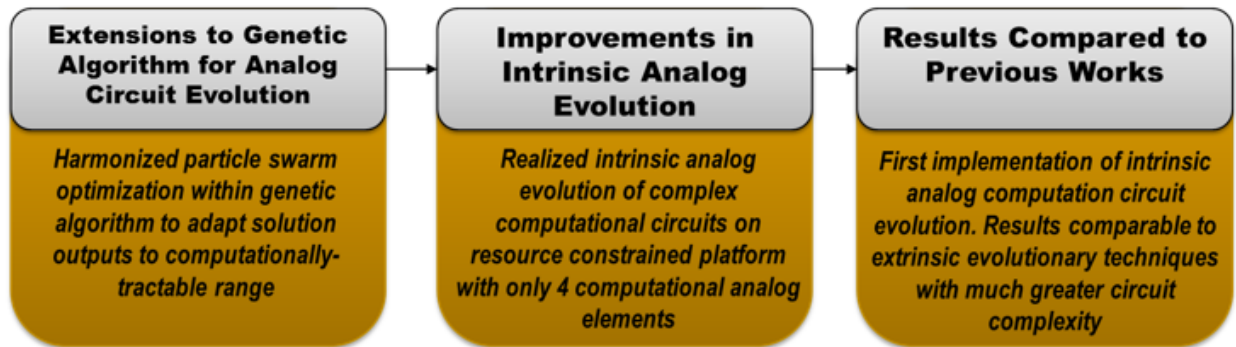| **Extensions to Genetic Algorithm for Analog Circuit Evolution**	**Improvements in Intrinsic Analog Evolution**	**Results Compared to Previous Works**
Harmonized particle swarm optimization within genetic algorithm to adapt solution outputs to computationally-tractable range	*Realized intrinsic analog evolution of complex computational circuits on resource constrained platform with only 4 computational analog elements*	*First implementation of intrinsic analog computation circuit evolution. Results comparable to extrinsic evolutionary techniques with much greater circuit complexity*

Figure 30: Conclusions drawn from study herein.

Technical Summary

The use of SSGA was shown to improve the fitness of four analog computational circuits by an average of 7.18-fold, up to 12.92-fold in the case of the square CC. Results were comparable to previous works, which used extrinsic evolution in simulation environments and a much greater number of resources as well as reconfiguration options. In all cases, the intrinsic evolution of analog CCs using SSGA reduced the average error compared to the 1997 results in [5]. However, all test cases showed poorer results compared to the 2012 results in [7].

In summary, we addressed three of the challenges related to analog computation as depicted in Figure 31. First, we utilized a GA to automate the analog circuit design to address the challenging task of designing analog CCs, especially on resource constrained devices. Next, the GA developed is able to intrinsically adapt for the device characteristics present on whatever platform is being utilized. Finally, we developed the SSGA which is able to improve the accuracy and precision of our analog CCs by adapting the outputs to the most computationally tractable range as shown in the results.
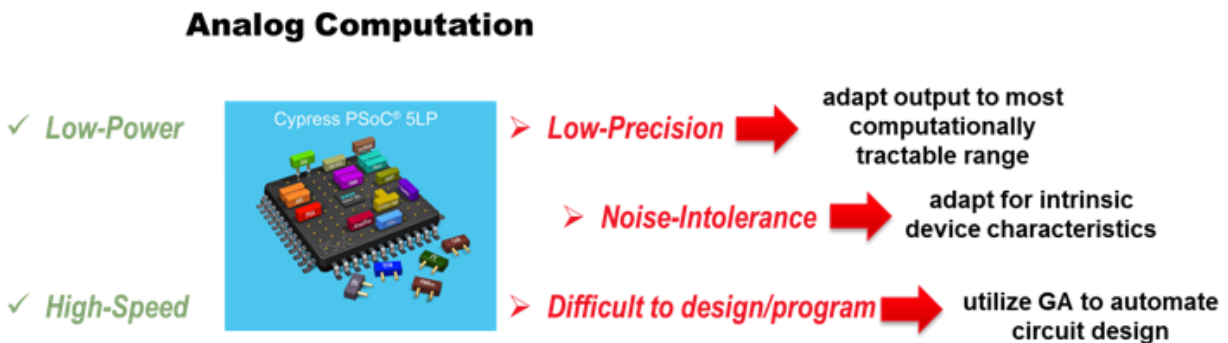

Figure 31: Summary of Challenges Addressed With Techniques Developed Herein

Technical Insights Gained

The intrinsic evolution of analog CCs on a reconfigurable fabric has shown great merit to realize functionality that would be incredibly difficult to hand-design given a resource-constrained platform. Extrinsic evolution has far too many luxuries that avoid many of the challenges of real implementation. Some of these luxuries include a lack of process variation, temperature variations, and device mismatch, as well as a possibly unconstrained amount of resources. Intrinsic evolution allows all of these characteristics to be considered during evolution as the fitness of the circuits are innately manipulated by all of them. Intrinsic evolution does allow the luxury of a reduced evaluation time as it's much faster to evaluate a real implementation rather than a simulation.

The PSoC-5LP provides an interesting platform to explore how analog and digital systems can cooperate in innovative ways. The onboard microcontroller allows adaptive design and refine algorithms to be implemented with the reconfigurable analog fabric in a single independent closed-loop package. The reconfigurable digital fabric allows the exploration of how reconfigurable analog and digital circuits can be synergistically utilized to improve old applications and explore the possibility of new applications.

When developing the techniques herein, the aspects which were the most straightforward to develop were:

- computational CCs which fell within the PSoC-5LP's native voltage range, as it was achievable with only the standard GA framework,
- implementing the PSO, as it only required a small modification to the fitness function and genome, as well as an additional small function to update the PSO parameters,

- the hypermutation function, as it only required a reinitialization of individuals genomes, which is just a simple function call in the framework developed,

- observing intra-evolutionary un-scaled functional tests as it only required a pin connected to the output routing line which was then connected to an oscilloscope for real-time observations,

- and the control of the EAs, as the onboard ARM microcontroller was able to store and run the algorithms with good speed and no external interfacing necessary.

Some of the most challenging aspects faced when developing the techniques herein include:

- developing the initial GA framework, as there were a large amount of registers and data to be stored and manipulated to configure each analog block, and bugs in the framework prevented good evolution until they were all found and addressed

- and observing intra-evolutionary details such as SS parameters and circuit topologies as it required manual halting of the evolutionary process in debug mode.

Scope and Limitations

The scope of this research centers around the automated synthesis of analog CCs on intrinsic reconfigurable fabrics for the potential benefit of greater efficiency in computation at current technology scaling limits as shown in Figure 32. Analog-based computation has interesting properties that could allow computational devices up to four orders of magnitude more efficient than purely-digital implementations, but there are still issues associated with their use. Although these efficiency improvements are a benefit, they lie outside the scope of this project.
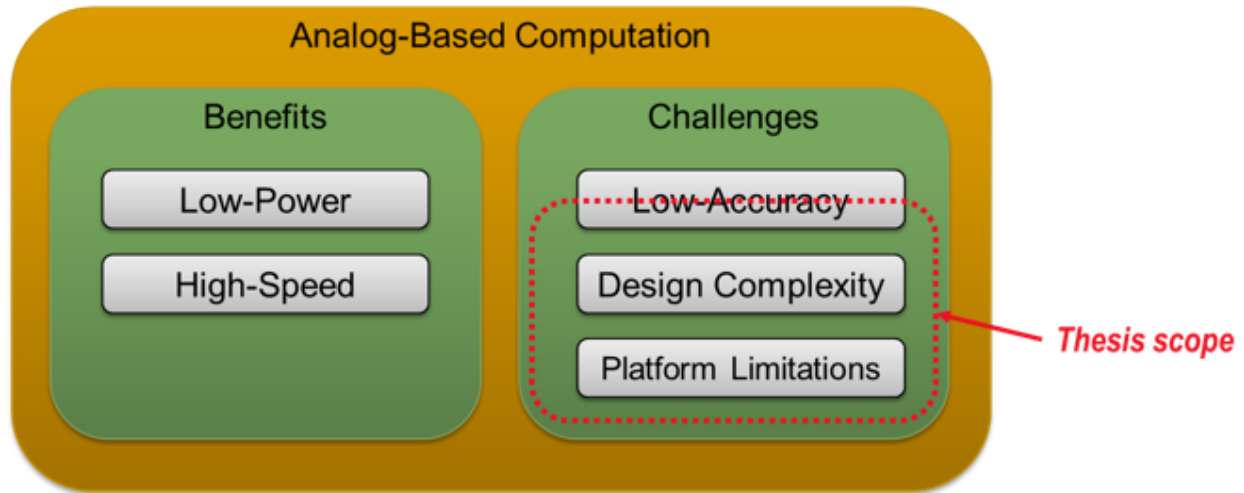
Figure 32: Analog-based computation benefits and challenges with the thesis scope outlined.

Future Directions

The performance of SSGA could benefit by more exploration by applying SSGA techniques to larger Field Programmable Analog Array platforms with additional computational analog blocks. Using such devices could perhaps show that a large of a range of accurate computation is possible, and that complex computational circuits such as differential equations could be accurately evolved.

Although the SSGA is used as a design technique, it could also function as a repair technique to sustain availability in long missions where a small number of resources are available by self-adapting faulty hardware to best map to the desired functionality. Additionally, an integrated reconfigurable analog platform and development environment with built in SSGA design functionality could allow analog-based computation to become more widely adopted.

Another interesting exploration would also be to conduct power measuring experiments on

SSGA evolved analog circuits compared to digital implementations; perhaps utilizing a multi-objective GA within the SSGA to account for power dissipation could lead to interesting findings. Finally, the SSGA could be applied to frequency domain analysis via adjustment of FFT coefficients.

REFERENCES

[1]   S. Sethumadhavan, R. Roberts, and Y. Tsividis, "A case for hybrid discrete-continuous architectures," *Computer Architecture Letters,* vol. 11, pp. 1-4, 2012.

[2]   P. Hasler and D. V. Anderson, "Cooperative analog-digital signal processing," in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, 2002, pp. IV-3972-IV-3975.

[3]   S. Suh, A. Basu, C. Schlottmann, P. E. Hasler, and J. R. Barry, "Low-power discrete Fourier transform for OFDM: A programmable analog approach," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 58, pp. 290-298, 2011.

[4]   T. W. Cornforth and H. Lipson, "Reverse-Engineering Nonlinear Analog Circuits with Evolutionary Computation," in *Unconventional Computation and Natural Computation*, ed: Springer, 2014, pp. 105-116.

[5]   J. R. Koza, F. H. Bennett III, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *Evolutionary Computation, IEEE Transactions on,* vol. 1, pp. 109-128, 1997.

[6]   Y. Jiang, J. Ju, X. Zhang, and B. Yang, "Automated analog circuit design using Genetic Algorithms," in *Anti-counterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference on*, 2009, pp. 223-228.

[7]   Y. A. Sapargaliyev and T. G. Kalganova, "Open-ended evolution to discover analogue circuits for beyond conventional applications," *Genetic Programming and Evolvable Machines,* vol. 13, pp. 411-443, 2012.

[8]   F. H. Bennett III, J. R. Koza, M. A. Keane, J. Yu, W. Mydlowec, and O. Stiffelman, "Evolution by Means of Genetic Programming of Analog Circuits that Perform Digital Functions," in *GECCO*, 1999, pp. 1477-1483.

[9]   P. Hasler, "Low-power programmable signal processing," in *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, 2005, pp. 413-418.

[10]   R. Rojas, "Konrad Zuse's legacy: the architecture of the Z1 and Z3," *Annals of the History of Computing, IEEE,* vol. 19, pp. 5-16, 1997.

[11]   D. Hartree, "The differential analyser," *Nature,* vol. 135, pp. 940-943, 1935.

[12]   M. White, "An Analog Computer Technique for Solving a Class of Nonlinear Ordinary Differential Equations," *Electronic Computers, IEEE Transactions on,* pp. 157-163, 1966.

[13]   A. S. Jackson, "Analog computation," 1960.

[14]   A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *J. of Math,* vol. 58, p. 5, 1936.

[15]   R. S. Zebulum, M. A. Pacheco, and M. Vellasco, "Analog circuits evolution in extrinsic and intrinsic modes," in *Evolvable Systems: From Biology to Hardware*, ed: Springer, 1998, pp. 154-165.

[16]   W. Mydlowec and J. Koza, "Use of time-domain simulations in automatic synthesis of computational circuits using genetic programming," in *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Las Vegas, Nevada*, 2000, pp. 187-197.

[17]   M. J. Streeter, M. A. Keane, and J. R. Koza, "Iterative Refinement Of Computational Circuits Using Genetic Programming," in *GECCO*, 2002, pp. 877-884.

[18]   T. McConaghy, P. Palmers, M. Steyaert, and G. G. Gielen, "Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks," *Evolutionary Computation, IEEE Transactions on,* vol. 15, pp. 557-570, 2011.

[19]   D. Keymeulen, R. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *Reliability, IEEE Transactions on,* vol. 49, pp. 305-316, 2000.

[20]   R. Al-Haddad, R. Oreifej, R. Ashraf, and R. F. DeMara, "Sustainable modular adaptive redundancy technique emphasizing partial reconfiguration for reduced power consumption," *International Journal of Reconfigurable Computing,* vol. 2011, 2011.

[21]   N. Imran, R. A. Ashraf, and R. F. DeMara, "Power and quality-aware image processing soft-resilience using online multi-objective GAs," *International Journal of Computational Vision and Robotics,* vol. 5, pp. 72-98, 2015.

[22]   M. G. Parris, C. A. Sharma, and R. F. Demara, "Progress in autonomous fault recovery of field programmable gate arrays," *ACM Computing Surveys (CSUR),* vol. 43, p. 31, 2011.

[23]   J. Lohn, G. Larchev, and R. DeMara, "A genetic representation for evolutionary fault recovery in Virtex FPGAs," in *ICES*, 2003, pp. 47-56.

[24]   R. Ashraf and R. F. DeMara, "Scalable FPGA refurbishment using netlist-driven evolutionary algorithms," *Computers, IEEE Transactions on,* vol. 62, pp. 1526-1541, 2013.

[25]   R. F. DeMara and K. Zhang, "Autonomous FPGA fault handling through competitive runtime reconfiguration," in *Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on*, 2005, pp. 109-116.

[26]   R. F. DeMara, K. Zhang, and C. A. Sharma, "Autonomic fault-handling and refurbishment using throughput-driven assessment," *Applied Soft Computing,* vol. 11, pp. 1588-1599, 2011.

[27]   M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer,* vol. 27, pp. 17-26, 1994.

[28]   D. Whitley, "A genetic algorithm tutorial," *Statistics and computing,* vol. 4, pp. 65-85, 1994.

[29]   M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," in *Proceedings of the first european conference on artificial life*, 1992, pp. 245-254.

[30]   R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, 1995, pp. 39-43.

[31] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence,* vol. 1, pp. 33-57, 2007.

[32] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, B. Degnan*, et al.*, "Hardware and software infrastructure for a family of floating-gate based FPAAs," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 2794-2797.

[33] T. P. Hughes, *Networks of power: electrification in Western society, 1880-1930*: JHU Press, 1993.

[34] C. Schlottmann and P. Hasler, "FPAA empowering cooperative analog-digital signal processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, 2012, pp. 5301-5304.

[35] E. Ozalevli, W. Huang, P. E. Hasler, and D. V. Anderson, "A reconfigurable mixed-signal VLSI implementation of distributed arithmetic used for finite-impulse response filtering," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 55, pp. 510-521, 2008.

[36] Y. Huang and S. Sethumadhavan, "Hybrid Continuous-Discrete Computer: from ISA to Microarchitecture," 2013.

[37] A. Basu, C. M. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan*, et al.*, "RASP 2.8: A new generation of floating-gate based field programmable analog array," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, 2008, pp. 213-216.

[38] C. Schlottmann, S. Nease, S. Shapero, and P. Hasler, "A mixed-mode FPAA SoC for analog-enhanced signal processing," in *Custom Integrated Circuits Conference (CICC), 2012 IEEE*, 2012, pp. 1-4.

[39]   G. E. Cowan, R. C. Melville, and Y. Tsividis, "A VLSI analog computer/digital computer accelerator," *Solid-State Circuits, IEEE Journal of,* vol. 41, pp. 42-53, 2006.

[40]   A. Thompson, "Silicon evolution," in *Proceedings of the 1st annual conference on genetic programming*, 1996, pp. 444-452.

[41]   A. A. Naseer, R. A. Ashraf, D. Dechev, and R. F. DeMara, "Designing Energy-Efficient Approximate Adders using Parallel Genetic Algorithms," presented at the SoutheastCon, 2015.

[42]   R. A. Ashraf, F. Luna, D. Dechev, and R. F. DeMara, "Designing digital circuits for FPGAs using parallel genetic algorithms (WIP)," in *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium*, 2012, p. 15.

[43]   N. Imran and R. F. DeMara, "A self-configuring TMR scheme utilizing discrepancy resolution," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, 2011, pp. 398-403.

[44]   R. S. Oreifej, R. N. Al-Haddad, H. Tan, and R. F. DeMara, "Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II Pro devices," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 299-304.

[45]   J. Slezak and T. Gotthans, "Design of Passive Analog Electronic Circuits using Hybrid Modified UMDA Algorithm," *Radioengineering,* vol. 24, 2015.

[46]   A. Stoica, A. Fukunaga, K. Hayworth, and C. Salazar-Lazaro, *Evolvable hardware for space applications*: Springer, 1998.

70

[47]   R. S. Oreifej, C. Sharma, and R. F. DeMara, "Expediting GA-based evolution using group testing techniques for reconfigurable hardware," in *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*, 2006, pp. 1-8.

[48]   A. W. Hyung-Joong Kim, Hod Lipson, "Automated synthesis of resilient and tamper-evident analog circuits without a single point of failure," *Genetic Programming and Evolvable Machines,* vol. 11, pp. 35-59, 2010.

[49]   S.-B. C. Kyung-Joong Kim, "Combining Multiple Evolved Analog Circuits for Robust Evolvable Hardware " in *Intelligent Data Engineering and Automated Learning - IDEAL 2009*, Burgos, Spain, 2009, pp. 359-367.

[50]   V. Thangavel, "Cascaded Digital Refinement for Intrinsic Evolvable Hardware," Master's in Electrical Engineering, Department of Electrical Engineering and Computer Science, University of Central Florida, 2015.

[51]   S. Pyle, Vignesh Thangavel, Stephen M. Williams, and Ronald F. DeMara, "Self-Scaling Evolution of Analog Computation Circuits with Digital Accuracy Refinement," in *NASA/ESA Conference on Adaptive Hardware and Systems*, Montreal, Quebec, CA, 2015.