Self-Scaling Evolution of Analog Computation Circuits with Digital Accuracy Refinement

Steven D. Pyle, Vignesh Thangavel, Stephen M. Williams, and Ronald F. DeMara
Computer Architecture Lab
Department of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL, 32816-2362
demara@mail.ucf.edu

***Abstract*—We introduce *SCALER,* a two-pronged strategy utilizing digital resources for refining intrinsic evolution of analog computational circuits. A *Self-Scaling* Genetic Algorithm is proposed to adapt solutions to computationally-tractable ranges in hardware-constrained analog reconfigurable fabrics. *Differential Digital Correction* is developed utilizing an error metric computed from the evolved analog circuit to reconfigure the digital fabric intrinsically thereby enhancing precision. We demonstrate our methods 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 an error metric on average *7.18*-fold, up to *12.92*-fold for computational circuits that produce outputs beyond device range. Overall, *Differential Digital Correction* can reduce computational error by 23.1% compared to the performance of the evolved analog circuit.**

I. Introduction

As we continue to advance towards CMOS technology-scaling limits, new and innovative strategies to enhance computational performance are sought. One fundamental inefficiency in today's computational models comes from utilizing digital computation to solve continuous real-world phenomena [1]. An intriguing way of alleviating this inefficiency is to utilize analog devices to perform continuous time computations where applicable [1]. Analog computers are not an unprecedented concept in computation, and their speed and energy performance can be better than their digital counterparts for certain computations [1,2,3]. According to Gene's Law, utilizing analog computation where applicable could provide a 20-year leap in performance versus their digital counterparts [2]. Approaches presented in [1,3] demonstrated that analog computation reduced energy consumption 8-fold compared to the corresponding digital implementation. However, complex analog circuits can be challenging to design and lack precision. We show in this paper how both of these issues can be addressed using Evolutionary Algorithms (EAs).

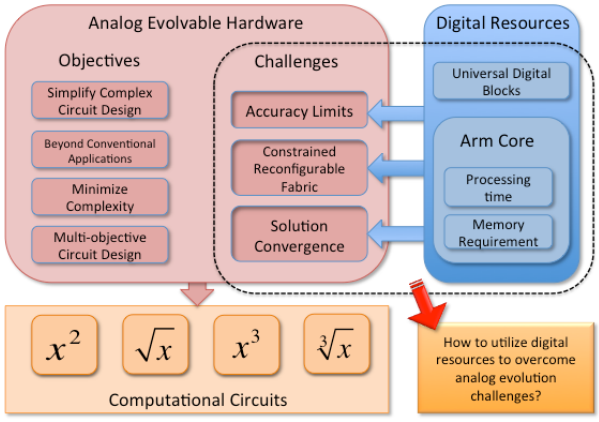

Figure 1: Objectives and challenges of evolving analog circuits for computation.

Precise and efficient complex analog circuits typically require design expertise and experience [4]. Nonetheless, [4,5,6,7] have demonstrated that EAs are a viable candidate to address the problem of automated analog design, having successfully evolved analog computational circuits and even evolved 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 is possible to evolve robust nonlinear analog circuits with EAs. However, due to the stochastic nature of EAs, 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.

A. Evolutionary Optimization Algorithms

Genetic Algorithms (GAs) are a well-known class of EAs that emulate natural forms of survival-of-the-fittest Darwinian evolution [9]. 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 [10]. The *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.

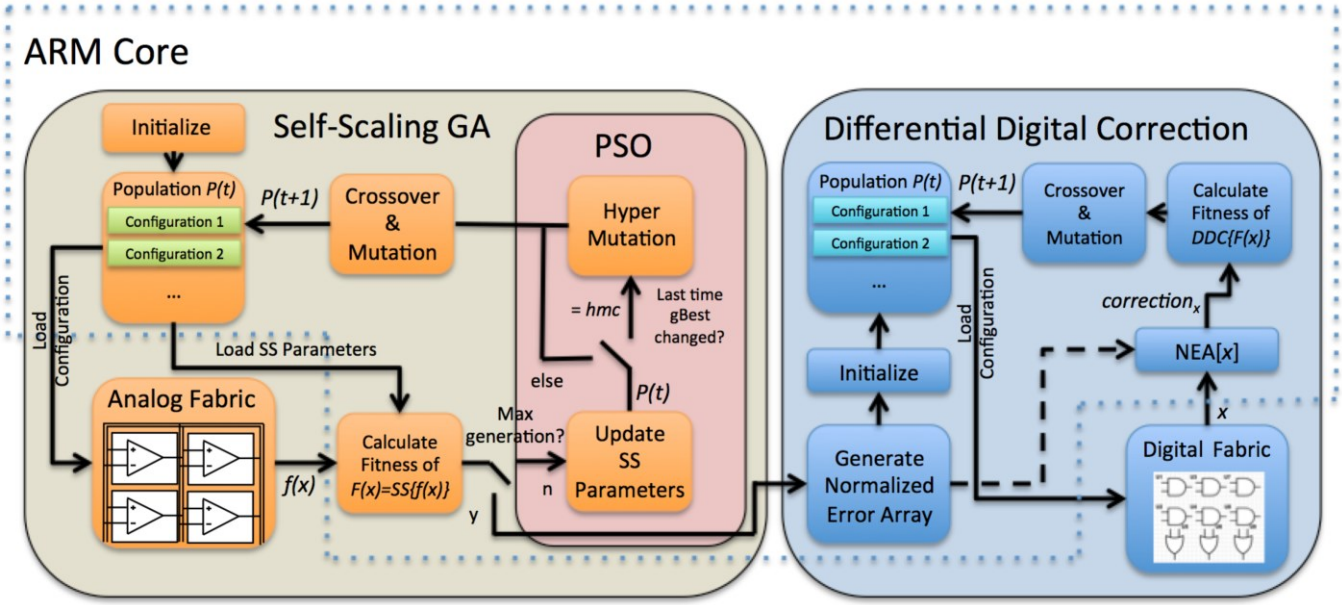

Figure 2: Scaling Evolutionary Refinement (SCALER).

Particle Swarm Optimization (PSO) is a parameter optimization algorithm inspired by bird or fish flocking and swarming theory [11]. 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, as determined by the desired functionality, 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 [11].

This paper delineates two new cooperating techniques to utilize digital resources address the challenges facing evolved analog computational circuits as depicted in Figure 1. In order to demonstrate our methods, we have developed the *Scaling Evolutionary Refinement (SCALER)* technique. SCALER is delineated in Figure 2 and consists of an analog *Self-Scaling GA (SSGA)* on an intrinsic commercial prototyping platform. Once an evolved solution is chosen, the proposed technique of *Differential Digital Correction (DDC)* is applied to refine the most-fit analog solution. 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 techniques operate and demonstrate their capabilities to intrinsically evolve, adapt, and refine the *Computational Circuits (CCs)* in [7], specifically the square, square-root, cube, and cube-root functions, and then utilize our contributions to refine the accuracy of the evolved circuits.

B. *Research Contributions*

The following research contributions are provided:

1) an extension to analog domain evolution called *Self-Scaling GA (SSGA)* which utilizes particle swarm optimization to allow genetic algorithms to self-scale the range of outputs to best fit an intrinsic computational domain's available resources,
2) use of an Island-like GA to explore multiple SSGA parameter-sets and exchange best-parameter-set information periodically, and
3) a novel refinement technique implemented with a small amount of digital logic and memory to improve the accuracy of computations performed by analog circuits significantly.

II. Related Work

A variety of EAs have been used to realize novel electronic circuit designs intrinsically on reconfigurable fabrics. 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. For example, Koza et al. demonstrated an approach for the automatic synthesis of analog circuits using Genetic Programming (GP) to synthesize crossover and lowpass filters at various frequencies, an amplifier, a source identification circuit, a CC (cube-root), a time-optimal controller circuit, a voltage reference circuit, and a temperature-sensing circuit, all extrinsically using DC sweeps for fitness evaluation [5]. Mydlowec and others followed the path of Koza, evolving other CCs extrinsically [7,12], some using multiple time domain simulations to improve robustness.

Table 1: Selected Related Research in Analog Evolvable Hardware.

Research Work	Analog Circuits Evolved	EA Type	Platform	Contribution
Cornforth 2014, ref [4]	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.
Koza 1997, ref [5]	Square root	Extrinsic	SPICE	Used GP to design analog circuits using DC sweeps.
Mydlowec 2000, ref [12]	Square, square root, multiplier, and lag circuit	Extrinsic	SPICE	Synthesis of computational circuits using multiple time-domain simulations for robustness fitness evaluation.
Keymeulen 2000, ref [14]	Multiplier	Intrinsic	Custom FPTA with 48 switchable transistor terminals in two 0.5um chips	Population-based and Fitness-based fault tolerance.
Streeter 2002, ref [17]	Cube	Extrinsic	Weakly-constrained virtual fabric under progressive voltage conditions	Average error 7-fold less than human design.
SCALER (work herein)	**Square root, cube root, square, cube**	**Intrinsic**	**Cypress PSoC-5LP**	**Digital resources enhance accuracy of self-scaling GA with PSO.**

McConaghy et al. showed that automated analog circuit synthesis using GP could build construction trustworthy circuits by using expert designed building blocks [13]. Keymeulen et al. demonstrated intrinsic EHW on Field Programmable Analog Arrays (FPAAs) for population-based and fitness-based evolution of fault-tolerant analog circuits [14]. Aggarwal et al. showed that PSO can be used to optimize an FPAA for PID control [15]. A novel FPAA architecture was developed in [16] to realize GA synthesizable and intrinsically adaptable analog filters. 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 [7] EAs were used to evolve four analog CCs as well as two digital circuits using analog components. In [18] 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]. Their work showed that a variety of stimuli are able to extrinsically evolve nonlinear analog circuits, which conform to randomly generated black-box circuits, demonstrating the strength of the method.

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 evolved digital circuits. The ARM core on the PSoC allows on-chip execution of EAs such as the GAs and PSO as developed herein.

III. Self-Scaling Evolution of Analog Circuits with Differential Digital Correction

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. Figure 3 shows a typical rudimentary GA-evolved cube circuit output measured on the PSoC-5LP intrinsic platform compared to the ideal curve. Since the PSoC

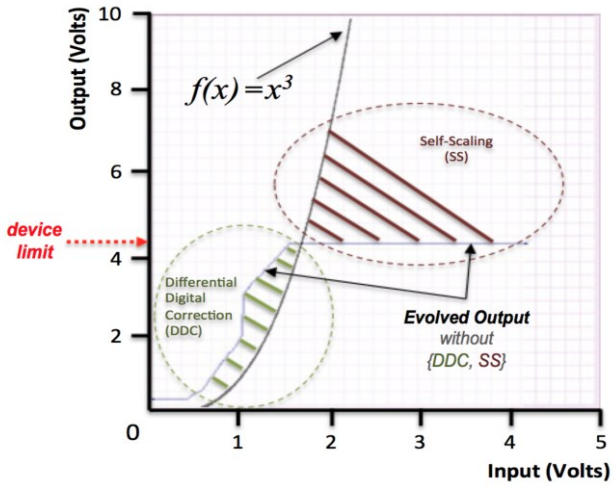

Figure 3: Analog cube CC output evolved with unrefined GA.

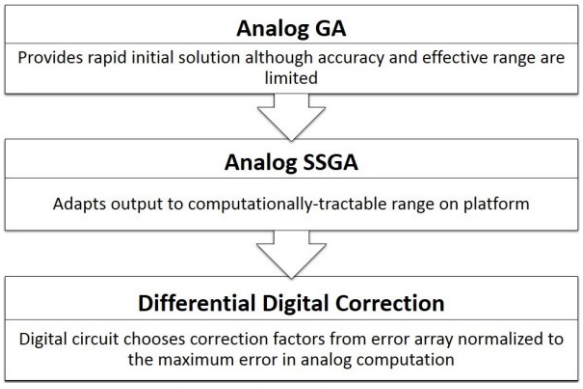

Figure 4: Phases of SCALER.

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. It would be possible to scale the inputs in such a way that the maximum output voltage would be within our device limit, but we seek an intelligent adaptive method. We also consider that there may be particular output voltage ranges that more effectively map the available resources to our desired CC, and we would like to adapt and search for such exploitations, which we show in Section IV.

SCALER provides interconnected GAs which operate in phases as shown in Figure 4. The first phase is the unrefined analog GA which evolves a coarse solution in a small voltage range. The second stage is the SSGA, which performs output scaling by working with the unrefined GA to scale output to a more computationally-tractable range. Finally, the DDC Genetic Algorithm (DDCGA) evolves digital fabric to select appropriate correction factors to offset errors in the SSGA output, thereby improving accuracy and to some extent precision where possible. Thus, solutions are accurate in some ranges, but have diminished accuracy in others. We show in Section V that SCALER is able to use DDC to refine our solutions to overcome such losses in accuracy without distorting the accurate ranges.

IV. Self-Scaling Genetic Algorithm

In order to evolve analog CCs, it can be beneficial to consider circuits to be composed of multiple *Computational Analog Elements* (*CAEs*) that the GA can operate on. CAEs could be higher-level analog blocks such as integrators, filters, adders, multipliers, etc., or they could be lower-level analog elements such as transistors and capacitors.

Each gene for our analog GA constitutes all of the information necessary to fully develop a CAE. Figure 5 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*. Routing is encoded by using bits to enable or disable CAE terminal connections to available routing lines. 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 *r* bits to determine routing.

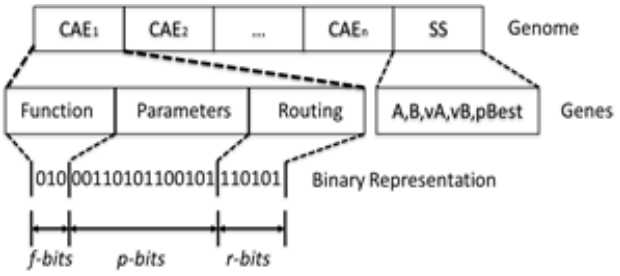

Figure 5: Breakdown of individual genome, gene expression, and binary representation for SCALER's analog GA.

For our study, an Island-like GA is utilized where the only information exchanged between the island populations consists of SS parameters. The initial *P* populations for our GA each consists of *N* individuals with genomes consisting of *n* randomly generated *CAEs*. Once the initial population is generated, the fitness for each individual is calculated by evaluating the circuit with *M* test inputs for which we know the desired output that we denote as the *oracle* output values [19], which are pre-computed by the ARM core. The fitness is then the sum of differences of all the test cases. If the difference between a particular test input and the oracle is greater than *penalty_case*, we add *penalty* to the fitness.

Once the fitness of each individual is established, both the best fit and the second best fit individuals from each population are transferred to the next generation implementing *elitism* of degree 2. 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 but the best fit individual. 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 the difference between the best fit individual's fitness and the average fitness. $P_{mutation}$ is doubled when $Fitness_{average} - Fitness_{best} < Fitness_{best}$ and tripled when $Fitness_{average} - Fitness_{best} < 0.5 \times Fitness_{best}$.

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

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

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 SSGA flow is shown in Algorithm 1.

$$fitness = \sum_{x=0}^{M} |SS\{f(x)\} - oracle_x| \qquad (2)$$

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.

The SS parameters *A* and *B* are initialized for each individual when the populations are initialized by randomly assigning values between *pmin* and *pmax*. Because PSO is now being performed on a dynamic population whose functional ranges are changing, we implement a *hypermutation* function, similar to [20], 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 the range [*pmin, pmax*], and reinitializes half of the population to random individuals so that there is enough genetic diversity to

make use of the new SS parameters. Exchange of SS parameters will be demonstrated in Figure 8. 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.

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);
```

Due to the stochastic nature of such algorithms, it is 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.

V. Differential Digital Correction

The solutions obtained from analog evolution can rapidly approximate the desired solutions, although their accuracy is limited and susceptible to imprecision. The DDC technique selects a correction factor for each test input from a *Normalized Error Array (NEA),* which contains fractions of the maximum analog error. DDC utilizes correction factors quantized to 256 levels and hence evolves PLDs to produce an 8-bit mapping for the 256 test inputs. The DDC fitness is evaluated as:

$$for\ i := 0\ to\ 255$$

$$fitness\ +=\ oracle[i] - out_{analog}[i] - NEA\ [D]$$

The NEA containing correction factor elements called normalized differences is indexed by D to realize the 8-bit output mapped to one of the 256 values which provides a correction factor of proportional magnitude. The fitness function represents an error correction factor which is minimized by a GA that intrinsically evolves the digital fabric.

A. Digital Genome

In order to evolve a single PLD, its configuration bits are encoded into a chromosome which genetic operators can act upon. The configuration bits consist of the following: active lines, `AND` array parameters and `OR` array parameters. Input lines that are active, i.e. not in high impedance state, in the actual implementation are determined from the register configuration immediately after booting and are marked as *active* lines; other lines cannot be written. There are 12 input lines for the AND array and four output lines for the OR array.

a) AND array parameters

For each input line that is active, the corresponding product term can be asserted as true or complement input. In order to encode the configuration, the configurations of the active input lines alone are to be recorded and hence up to eight such 32-bit integers store the configuration from the corresponding registers for two PLDs constituting a *Universal Digital Block (UDB)* in the fabric.

b) OR array parameters

Like the AND array, for the OR array, each of the product terms arriving from the AND array may be asserted or not asserted and are encoded likewise in the chromosome. We thus have four such 16-bit integers that store the configuration from the corresponding registers. The chromosome for each PLD is encoded as a structure with three arrays: AL (active lines), AND array parameters and OR array parameters representing configuration bits as described above and effectively describing contents of one UDB.

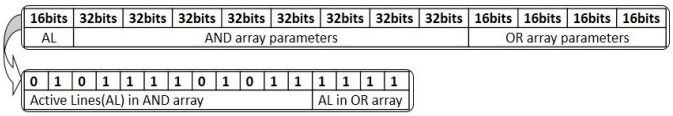

Figure 6: Chromosome for each UDB being evolved.

For evolution of four PLDs, two sets of such chromosomes, one for each UDB are evolved simultaneously.

B. Differential Digital Correction Genetic Algorithm

A separate *Differential Digital Correction GA (DDCGA)* reconfigures the digital fabric. The DDCGA evolves four PLDs of two chromosome sets per individual: one each for the AND and OR arrays, and one set of individuals for each pair of four PLDs, given constraints of the digital fabric.

Individuals are randomly initialized and their fitness is evaluated intrinsically to identify the top two elite individuals. The crossover operation is performed separately in each

iteration for the AND arrays and OR arrays owing to their inherent functional differences, at the boundaries indicated in Figure 6. For the remaining individuals, tournament selection is done with a tournament size of two and constitute 40 of the total 80 individuals per generation. Single-point crossover is then performed between one of these individuals and another individual randomly chosen from the whole population. New offspring then replace the lower fraction of the total population while the fitter individuals selected via tournament selection fill the upper fraction. All but the two most elite individuals undergo simple bit flip mutation with a default mutation rate of 0.1% per bit in the chromosome.

Mutation rate was observed to be a crucial factor in the performance of the DDCGA with adaptive mutation being very useful in overcoming stasis. Stasis is detected and reported if the best fitness achieved hasn't changed in 50 iterations. The difference between average fitness and best fitness achieved is compared and combined with stasis information to decide whether mutation should be enabled at the default rate or at an incremented rate in steps of 0.01, which improved performance.

The maximum deviation of the analog output from the oracle and the distribution of errors determine the extent to which the DDCGA can perform. With appropriate conditioning of analog outputs, DDC helps improve the accuracy of the solutions obtained and adds a few more bits of precision to evolved analog solutions.

VI. Experimental Configuration

A. Computational Analog Blocks

The primary reconfigurable analog elements in the PSoC-5LP consist of Switched Capacitor op-amp Blocks (SC Blocks). SC Blocks have a variety of topologies and parameters, which can be readily configured via memory-mapped registers accessible from the onboard ARM core. There are eight SC Block topologies available: naked op-amp, trans-impedance amplifier, continuous-time mixer, discrete-time mixer, unity gain buffer, first order modulator, programmable gain amplifier, and track and hold amplifier.

For our experimentation, we consider the SC blocks to be our CAEs, of which there are four in the PSoC-5LP. This leads to each individual having a genome of $n = 4$ CAEs along with the SS parameters.

B. Universal Digital Blocks (UDBs)

PSoC's digital fabric consists of PLDs, organized in pairs within the UDBs, which can all be interconnected under GA control. There are 24 UDBs or 48 PLDs in total. This PLD architecture is referred to as 12C4, where 12 stands for the number of input terms, C indicates that the PTs are constant and are accessible to the OR array and 4 indicates the number of output terms emerging from the OR array. Given this capability of the digital fabric, the routing is not easily reconfigured, unfortunately. These are determined at boot-time rather than during evolution.

Placement directives, such as force directives may be used to force placement of LUT implementation to any specific PLD, which is analogous to the **`PROHIBIT`** command in the Xilinx User Constraint File [21]. Owing to restrictions on routing, the initial configuration for the LUTs instantiated has to be carefully chosen to make available the maximum possible resources in the route whose logic configuration is then reconfigured by the DDCGA. Also, decoupling PLD outputs from the registers to ensure pure combinational outputs is essential and is done through additional register writes for every individual when their fitness is evaluated. Detection of active lines in Figure 6 is first performed on the initial configuration of LUTs to demarcate register writes that are legal.

C. Test Cases

The input/output ranges in Table 2 show the difference between attempting to evolve CCs with SSGA versus a rudimentary GA running on PSoC-5LP. Due to the native range of our platform, evolving without any form of scaling would severely reduce computational range for square and cube circuits.

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

Each computational circuit in Table 3 was evolved five times each with different random number generator seeds for both evolution with a simple GA and evolution with an SSGA. Globally, *max_generation* = 500, *pmin* = 0, $P = 4$, $N = 30$, *penalty* = 10, *hypermutation_condition* = 100, and *gen_share* = 200. Parameters *pmax* and *penalty_case* are delineated in Table 3. All parameters chosen are roughly optimized values based on trial experimentation. Average fitness values are computed along with best-case average error.

In order to compare our circuit complexity to [5,7,12], 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.

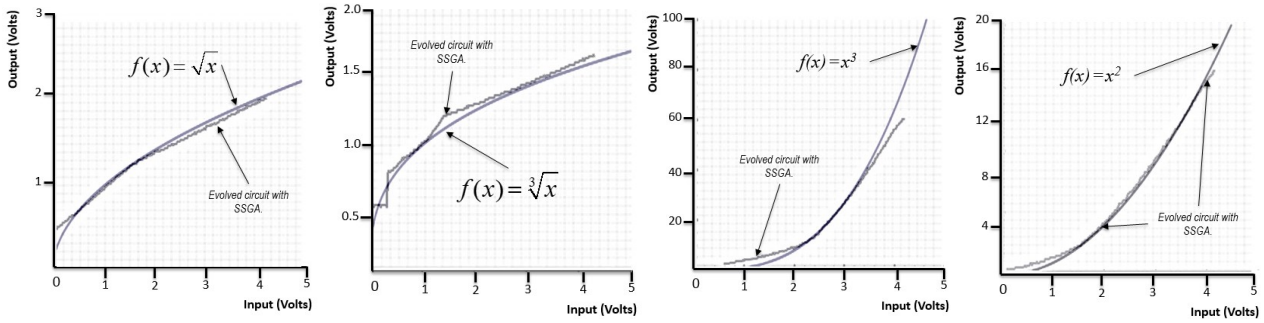

Figure 7: Evolved CCs with SSGA compared to ideal curves

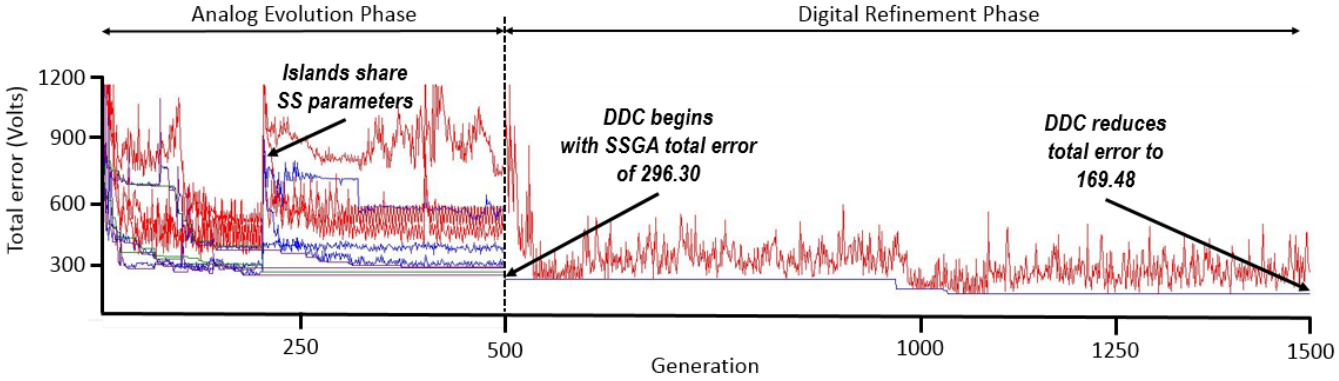

Figure 8: Total error over generation for SSGA and DDC when evolving cube CC. The top red line is the population average total error and the bottom blue line is the best fit individual's total error. During SSGA, four populations are evolved in parallel. Each population's average error and best fit error is shown.

VII. Results

Table 3 shows the fitness scores of the four CCs evolved. Each of the evolved CCs produced solutions, which closely matched their ideal outputs as shown in Figure 7. The worst performing circuit, with regards to fitness, 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. Even though the cube circuit had the worst fitness relative to the circuits tested, it was the best demonstration of the SSGA as it was able to increase its effective range seventeen-fold.

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 requires less scaling than the cube case. Furthermore, the square circuit only had an average error of 140 mV, so it is rarely penalized. Square-root and cube-root both were able to evolve good solutions with the unrefined GA. Interestingly, the best observed fitness amongst all of our tests was an unrefined 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 for using the SSGA.

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 30mV and 23mV, 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. It is interesting to note that DDC improved accuracy of all circuits to various degrees. The greatest reduction in average error was seen for

Table 3: Improvements of SSGA compared to unrefined GA

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

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	-

the cube circuit where a reduction from 1160mV to 732mV yielded a 36.89 percent reduction in error on average. Likewise, DDC improved accuracy by reducing average error in square, square root and cube-root CCs by 28.57, 10.67 and 16.3 percent respectively, on average. Some general trends were observed with regards to error reduction by DDC. For all CCs, the error reduction was larger when SSGA performance was worse than average, thus providing for a stabilizing effect to maintain accuracy within reasonable bounds. Performance for individual cases depended on error distribution of SSGA output. As shown in Figure 8, SSGA evolves four islands of populations in parallel to produce a best fit individual with a total error of 296.3. The DDC then evolves the digital fabric to correct errors and reduce it to 169.48. 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 rather than an unlimited number of resistors and BJTs. With an addition of only four PLDs, significant accuracy improvements were also achieved.

VIII. Conclusion

SCALER is able to scale, translate, and refine evolved analog computational circuits using evolved digital resources. PSO with an Island-like GA realizes a 12.9-fold fitness improvement of the best-fitness analog circuit. The novel hybrid analog-digital design that is evolved, leverages the relative advantages of both circuit domains. SCALER could benefit from exploration in the search space of PSO parameters and seeds for the unrefined GA. Also, the precision of DDC can further be improved by using values that can satisfy a 16-bit mapping instead of the 8-bit mapping used here. We would be interested to see our techniques applied to larger FPAA platforms with additional computational analog blocks and determine how large of a range of accurate computation is possible. Finally, the SSGA could be applied to frequency domain analysis via adjustment of FFT coefficients. Energy-conserving hybrid analog-digital computational circuits for scientific or low energy applications are being investigated.

References

[1] S. Sethumadhavan, R. Roberts, Y. Tsividis, "A Case for Hybrid Discrete-Continuous Architectures," *Computer Architecture Letters*, vol. 11, no.1, pp. 1-4, Jan.-June 2012.

[2] P. Hasler and D.V. Anderson, "Cooperative analog-digital signal processing," *IEEE Int'l Conference on Acoustics, Speech, and Signal Processing (ICASSP),* vol.4, pp.IV-3972 - IV-3975, 13-17 May 2002.

[3] S. Suh, A. Basu, C. Schlottmann, P. E. Hasler, J. R. Barry, "Low-Power Discrete Fourier Transform for OFDM: A Programmable Analog Approach," , *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 290-298, Feb. 2011.

[4] T. W. Cornforth and H. Lipson, "Reverse-Engineering Nonlinear Analog Circuits with Evolutionary Computation," in *Unconventional Computation and Natural Computation*. 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," *IEEE Transactions on Evolutionary Computation*, vol.1, no.2, pp.109-128, Jul 1997.

[6] Y. Jiang, J. Ju, X. Zhang, B. Yang, "Automated analog circuit design using Genetic Algorithms," *ASID 2009. 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication,* pp. 223 - 228, 20-22 Aug. 2009.

[7] Y. A. Sapargaliyev and T. G. Kalganova. "Open-ended evolution to discover analogue circuits for beyond conventional applications," *Genetic Prog. and Evolvable Machines,* 13.4 (2012): pp. 411-443.

[8] F. H. Bennett III, et al. "Evolution by Means of Genetic Programming of Analog Circuits that Perform Digital Functions," *GECCO*. 1999.

[9] G. Cowan, R. C. Melville, and Y. Tsividis. "A VLSI analog computer/digital computer accelerator," *Solid-State Circuits, IEEE Journal of* 41.1 (2006): 42-53.

[10] R. F. DeMara, K. Zhang, and C. A. Sharma, "Autonomic Fault-Handling and Refurbishment Using Throughput-Driven Assessment," Applied Soft Computing, Volume 11, Issue 2, Pages 1588-1599, March 2011.

[11] Russ C. Eberhart and James Kennedy, "A new optimizer using particle swarm theory." *Proceedings of the sixth international symposium on micro machine and human science*. Vol. 1. 1995.

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

[13] McConaghy, T.; Palmers, P.; Steyaert, M.; Gielen, G.G.E., "Trustworthy Genetic Programming-Based Synthesis of Analog Circuit Topologies Using Hierarchical Domain-Specific Building Blocks," *Evolutionary Computation, IEEE Transactions on* , vol.15, no.4, pp.557,570, Aug. 2011.

[14] D. Keymeulen, R. Zebulum, Y. Jin, A. Stoica,, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," IEEE Transactions on Reliability, vol. 49, no. 3, pp. 305-316, Sept. 2000.

[15] Aggarwal, V.; Mao, M.; O'Reilly, U.-M., "A Self-Tuning Analog Proportional-Integral-Derivative (PID) Controller," *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on* , vol., no., pp.12,19, 15-18 June 2006.

[16] Becker, J.; Trendelenburg, S.; Henrici, F.; Manoli, Y., "A field programmable Gm-C filter array (FPAA) for online adaptation to environmental changes.,"*Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on* , vol., no., pp.547,553, 5-8 Aug. 2007.

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

[18] S. L. Sabat, K. S. Kumar, and S. K. Udgata. "Differential evolution and swarm intelligence techniques for analog circuit synthesis." *NaBIC 2009 World Cong. Nature & Biologically Inspired Computing,* IEEE, 2009.

[19] K. Zhang, R. F. DeMara, C. A. Sharma, "Consensus-based Evaluation for Fault Isolation and On-line Evolutionary Regeneration," in *Proceedings of the International Conference in Evolvable Systems (ICES'05)*, pp. 12 - 24, Barcelona, Spain, September 12 - 14, 2005.

[20] Monica Sam, Sanjay K Boddhu, Kayleigh E. Duncan, John C.Gallagher, "Evolutionary strategy approach for improved in-flight control learning in a simulated Insect-Scale Flapping-Wing Micro Air Vehicle," *IEEE Conf. on Evolvable Systems (ICES), 2014*, pp.211,218, 9-12 Dec. 2014.

[21] R. S. Oreifej, C. A. Sharma, R. F. DeMara, "Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware," in *Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGAs* (Reconfig'06), San Luis Potosi, Mexico, September 20-22, 2006, pp 106-113.

This document is an author-formatted work. The definitive version for citation appears as: S. D. Pyle, V. Thangavel, S. M. Williams, and R. F. DeMara, "Self-Scaling Evolution of Analog Computation Circuits with Digital Accuracy Refinement," in Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2015), Montreal, QC, Canada, June 15 - 18, 2015.