Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2012, Article ID 762927, 14 pages
doi:10.1155/2012/762927

Research Article

An Adaptive Amplifier System for Wireless Sensor Network Applications

Mónica Lovay,[1] Gabriela Peretti,[1,2] Eduardo Romero,[1,2] and Carlos Marqués[2]

[1] *Mechatronics Research Group, Facultad Regional Villa María, Universidad Tecnológica Nacional, Avenida Universidad 450, 5900 Villa María, Argentina*
[2] *Electronics and Instrumentation Development Group, Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Medina Allende S/N, 5000 Córdoba, Argentina*

Correspondence should be addressed to Gabriela Peretti, gabi_peretti@yahoo.com.ar

Received 22 December 2011; Revised 13 February 2012; Accepted 27 February 2012

Academic Editor: Jose Silva-Martinez

This paper presents an adaptive amplifier that is part of a sensor node in a wireless sensor network. The system presents a target gain that has to be maintained without direct human intervention despite the presence of faults. In addition, its bandwidth must be as large as possible. The system is composed of a software-based built-in self-test scheme implemented in the node that checks all the available gains in the amplifiers, a reconfigurable amplifier, and a genetic algorithm (GA) for reconfiguring the node resources that runs on a host computer. We adopt a PSoC device from Cypress for the node implementation. The performance evaluation of the scheme presented is made by adopting four different types of fault models in the amplifier gains. The fault simulation results show that GA finds the target gain with low error, maintains the bandwidth above the minimum tolerable bandwidth, and presents a runtime lower than exhaustive search method.

1. Introduction

The advances in electronics have enabled the development of low cost, low power, and multifunctional wireless sensor nodes that consist of sensing, data processing, and communication components [1]. These small sensor nodes can be installed in a designated area to form a wireless network for performing specific functions. Usually, a host computer collects data from the sensors and carries out different actions depending on the particular purpose of the system. A broad range of applications has been proposed for this kind of systems such as industrial sensor networks, environmental monitoring, home automation, and medical care [2].

The processing and communication units typically found in wireless sensor nodes can be implemented with microcontrollers (μCs). These offer benefits like low cost and power consumption, ability to perform data processing tasks in the nodes, and usually powerful communication interfaces. In addition, some modern μCs offer a wide pool of configurable digital and analog sections that enhance the node adaptation to a broad range of applications.

In a number of applications, the nodes operate in harsh environments, under the action of a several agents that could potentially deteriorate their performances. If the application is critical, reliable operation of the node can require characteristics of safe operation, adaptation to a changing environment, or ability for compensating degradations in its own circuitry. For achieving this purpose, two related characteristics are necessary: fault detection and circuit self-adaptation.

The fault detection characteristic could be constrained by the low power operation of the node, which could make the use of dedicated test circuitry inconvenient for performing built-in self-test. Instead, a software-based self-test (SBST) strategy arises as an effective alternative that can provide in-field testing capabilities with very low area and performance overhead [3]. Particularly suitable for μCs, an SBST strategy utilizes the existing processing core to perform a self-test of the analog and digital components in a node [4, 5].

Providing adaptive characteristics to the node requires configurable hardware sections and a reconfiguration methodology. Evolvable hardware (EHW) is a methodology that

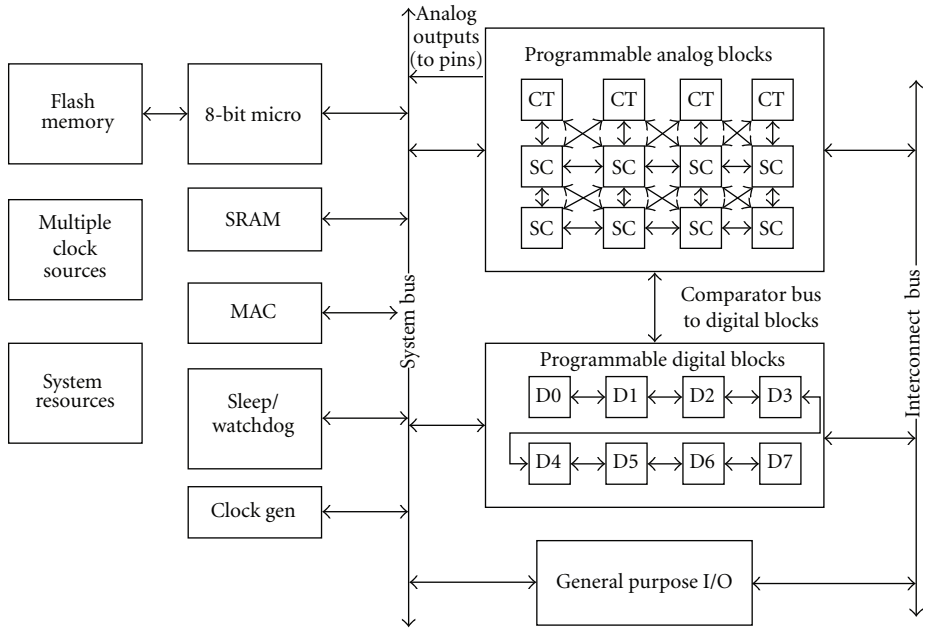

Figure 1: Architecture of PSoC1 device.

offers self-adaptation by combining reconfigurable hardware with evolutionary algorithms. In EHW, the designer establishes performance goals and usually a genetic algorithm [6] searches for the possible hardware configurations for reaching them [7, 8]. In this way, EHW offers an alternative to traditional fault tolerant schemes [7, 9]. Additionally, even if EHW does not always guarantee that a complete functionality can be restored, it allows maintaining the system operation with graceful degradation [10].

In this paper, we focus our efforts in the development of an adaptive amplifier embedded in the μC that is part of the node. We implement the amplifier on a platform that presents analog reconfigurable sections, a PSoC1 device from Cypress. One of the goals of our work is the development of a very low cost SBST strategy for the analog configurable sections. The other goal is the development of an EHW strategy that uses a GA-based reconfiguration strategy, which runs on the host computer. In this way, the amplifier maintains its targeted functional parameters between limits established by the user, without direct human intervention.

2. System Description

2.1. Architecture of the PSoC Device. As it was above mentioned, we adopt a PSoC1 device from Cypress for implementing the nodes. The PSoC1 device is a programmable system-on-chip platform with an 8-bit processor core [11]. The device presents an on-chip clocking solution and includes flash memory, SRAM, configurable blocks of analog and digital circuits, programmable interconnect, and configurable IO in a low-cost chip. Figure 1 shows its architecture.

Analog functions in the device are implemented by the combination (using programmable interconnection) of groups of general-purpose analog blocks that can be configured for user-determined functions with a design editing software tool. The control for these blocks is register-based and can be dynamically reconfigured during operation. There are two basic types of analog blocks: continuous-time (CT) and switched-capacitor (SC). The organization of the analog blocks is in columns. Each column contains one CT block and two types of SC blocks. The blocks are connected to direct port inputs, input multiplexers, column clock resources and output buffers for each column [12].

Some of the available configurations for the analog arrays are up to 14-bits analog-to-digital converters (ADC), up to 9 bits digital-to-analog converters (DAC), programmable gain amplifiers (PGA), programmable filters, and comparators. These functions could employ one analog PSoC block, a combination of more than one analog type (CT or SC), and even the inclusion of digital blocks.

2.2. The Programmable Gain Amplifier (PGA). The PGA user module implements an opamp-based noninverting amplifier with user-programmable gain (Figure 2). This amplifier, built in one CT block, has high input impedance, wide bandwidth, and selectable reference. The PGA gain is set by programming a selectable tap in a resistor string located in the opamp feedback path (R_a and R_b in Figure 2(a)). Figure 2(b) shows in detail the resistor ladder configuration and the signals; RtapMux and EXGAIN, which select the tap connection. For gains greater than or equal to one, the top of the resistor string is connected to the opamp output and the resistor tap is connected to the inverting input of the opamp. Under this condition, the output voltage (V_o) is

$$V_o = (V_{in} - V_{GND})\left(1 + \frac{R_b}{R_a}\right) + V_{GND}. \quad (1)$$

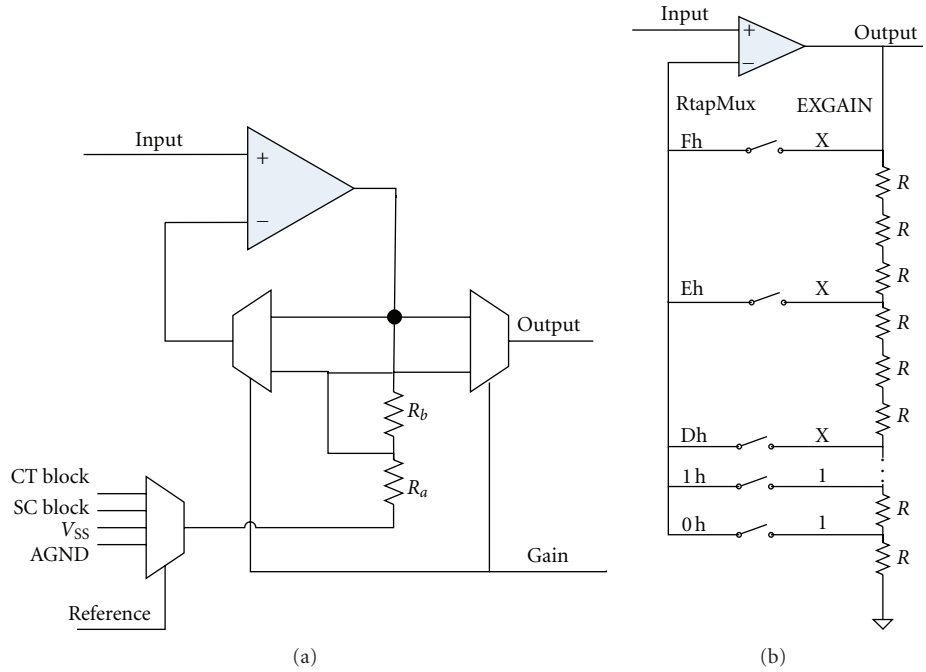

Figure 2: (a) PGA diagram. (b) Resistor ladder configuration.

For gains less than one, the opamp is set as voltage follower and the user module output is connected to the resistor tap. The amplifier output value is obtained by evaluating

$$V_o = (V_{in} - V_{GND})\left(\frac{R_a}{R_a + R_b}\right) + V_{GND}. \quad (2)$$

In (1) and (2), V_{in} is the voltage at the PGA input and V_{GND} is a reference voltage. The user can specify the reference as one of the following: a fixed value derived from the internal bandgap reference, a value ratiometric to the supply voltage, analog ground, or an external input. There are 36 programmable values for the PGA gain, ranging from 0.0208 to 48 [13].

The CT block allows a configuration (Figure 3) that connects V_{DD} at the top of the resistor string and V_{SS} at the bottom, while routes at the output the voltage value of the tap resistor. By the proper setting of the signals EXGAIN and RtapMux, it is possible to check all the values available at the tap resistor. This configuration is used in test mode to be explained in Section 3.

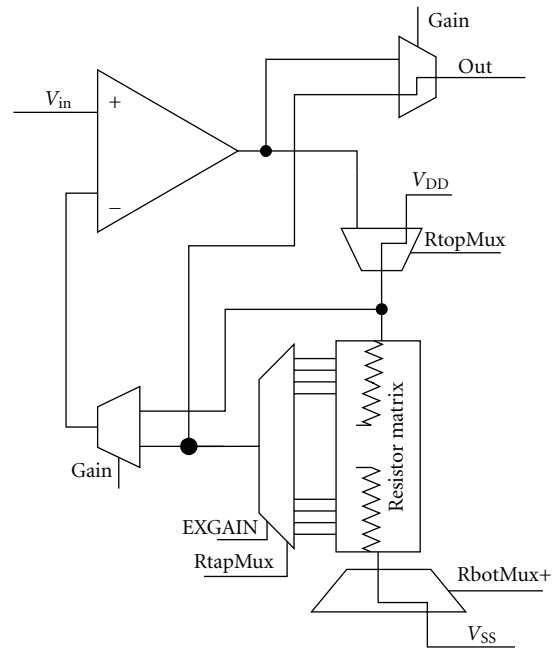

Figure 3: CT configuration for testing the resistor matrix (simplified diagram).

2.3. The Adaptive Amplifier. We assume that each node of the wireless sensor network requires an adaptive amplifier. For implementing the amplifier, we propose the use of programmable gain amplifiers (PGAs) in cascade connection. It should be noted that preliminary results about the amplifier were reported in [14]. However, the two key components of the system, the SBST and the GA used in the reconfiguration, are different in this new proposal.

The use of PGAs is frequent in systems that require an analog front-end for signal conditioning. Particularly, a wide range of applications employs PSoC1 PGAs for this end, [15–18]. This allows devising a wide range of application possibilities for the presented system, even if the analog blocks do not present the same high performance characteristics that could present specific-purpose analog devices.

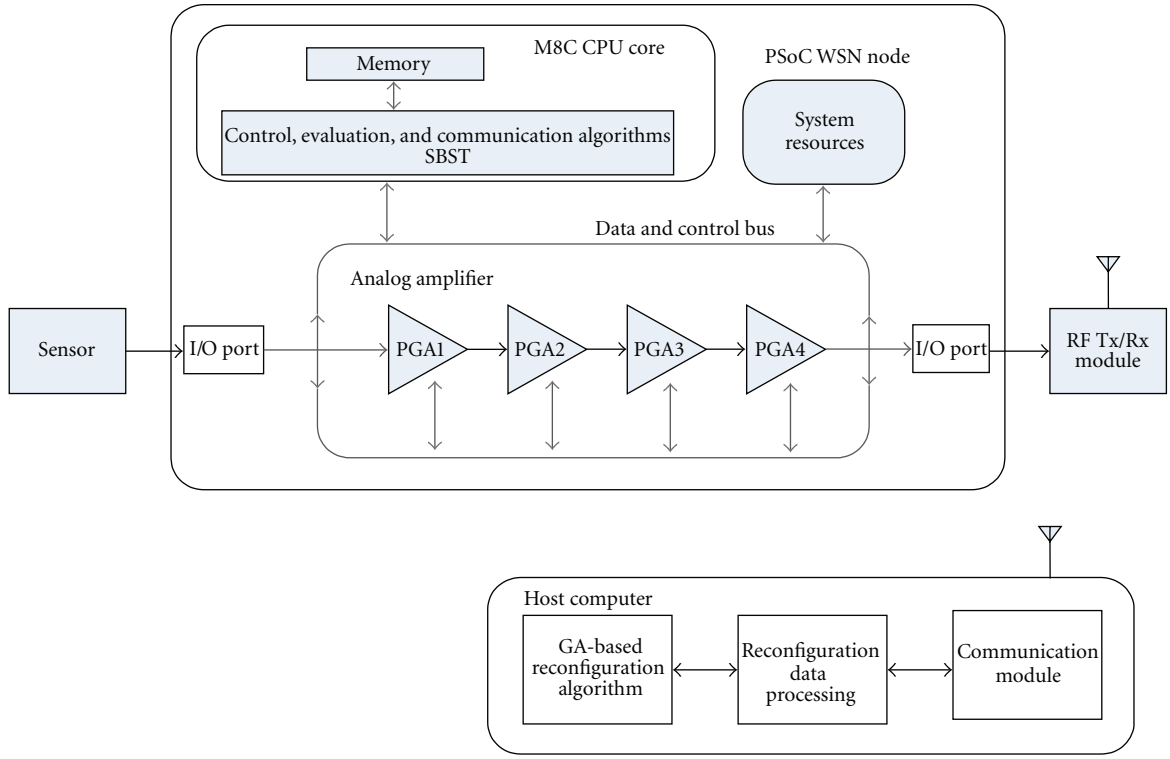

Figure 4: Amplifier system diagram.

In this paper, we use four amplifiers in the chain (PGA1, PGA2, PGA3, and PGA4 in Figure 4) one for each column of the PSoC analog array (shown in Figure 1). This configuration presents a broad range of alternative values for establishing the overall gain that contributes to achieve a better self-adaptation. However, the cascade connection could reduce the overall bandwidth below the user specifications. For this reason, we consider that the system has a target gain that has to be maintained despite the presence of faults, while its bandwidth must be larger than the one required by the application.

An SBST strategy checks during the idle times of the system the available gains of each amplifier and determines if it is necessary a system reconfiguration. Additionally, several routines (running in the μC) control the overall operation of the node. The host computer runs the genetic algorithm (GA) that finds the values for the system reconfiguration. The evolved values of gain are loaded back into the hardware for continuing the normal operation.

3. PGA Resistor Ladder Test

The test strategy proposed here uses the processor core, on-chip analog resources, and the dynamic reconfiguration characteristic of the PSoC device. Consequently, this SBST approach virtually eliminates the need for additional test-specific hardware. Another characteristic of the SBST proposed here is that it does not require the host computer intervention. Consequently, it could also be used without restrictions as a low-cost, no hardware overhead test for the resistor arrays of the CT blocks in any PSoC1-based system.

PSoC dynamic reconfiguration characteristic is a powerful feature that enables the designer to program multiple hardware configurations and dynamically change them while the device is running [19]. In this way, the amplifier system is placed in one hardware configuration, while the test hardware is configured in a different one. The processor loads from the program memory the setup of each configuration in runtime. The normal mode configuration presents only the PGAs connected in cascade, while the test mode configuration arranges the PGAs connections in order to obtain access to the resistor arrays (Figure 3). Additionally, the configuration employs other modules that are necessary for the test process, for example, ADCs.

In test mode, besides loading the test hardware configuration, the processor executes an embedded routine that acquires data and performs the required calculations. At the end of the test process, the test routine delivers to the host computer the parameters that characterize the resistor relations for the four PGAs.

The test configuration is shown in Figure 5. The CT block (where each PGA is placed) is configured for testing the resistor matrix (Figure 3). A 12-bit analog-to-digital converter (ADC12) (one by column) acquires all the available voltage values from the resistor matrix. The test routine programs these values using the settings of the RtapMux

Table 1: Test conditions and expected values for the resistor matrix.

Test condition	RtapMux (hexadecimal values)	EXGAIN	Nominal value for Vref (Vdd Units)	PGA gain value (gain ≥1)	PGA gain value (gain ≤1)
1	0	1	0.0208	48.000	0.0208
2	1	1	0.0417	24.000	0.0417
3	0	0	0.0625	16.000	0.0625
4	1	0	0.1250	8.000	0.1250
5	2	0	0.1875	5.333	0.1875
6	3	0	0.2500	4.000	0.2500
7	4	0	0.3125	3.200	0.3125
8	5	0	0.3750	2.667	0.3750
9	6	0	0.4375	2.286	0.4375
10	7	0	0.5000	2.000	0.5000
11	8	0	0.5625	1.778	0.5625
12	9	0	0.6250	1.600	0.6250
13	A	0	0.6875	1.455	0.6875
14	B	0	0.7500	1.333	0.7500
15	C	0	0.8125	1.231	0.8125
16	D	0	0.8750	1.143	0.8750
17	E	0	0.9375	1.067	0.9375
18	F	0	1.0000	1.000	1.0000

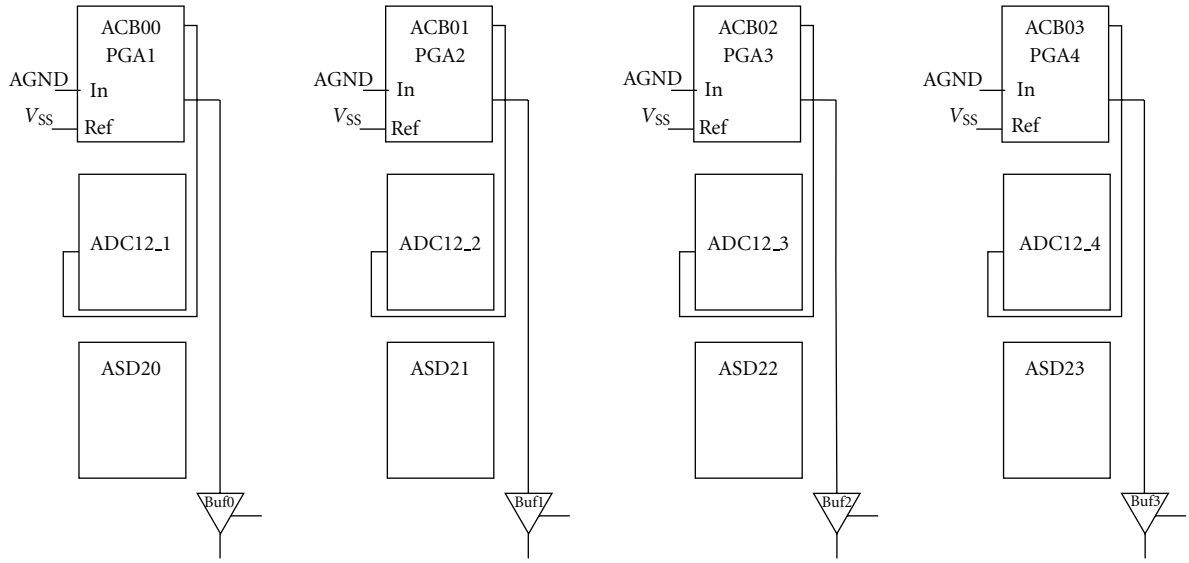

Figure 5: Test hardware configuration.

and EXGAIN registers. The combination of RtapMux and EXGAIN used during the test and the nominal voltage values of the resistor tap (Vref) are shown in Table 1. Also presented in the table are the gain values that the PGA adopts (that also depends on the gain signal value) for the corresponding resistor-matrix relations.

From the measurement of the voltage values, it is possible to establish all values of the programmed relations of R_a and R_b (Figure 2) and check if they are within the limits specified by the user. Because to the resistance ratios determine the gain values for the PGA module, the routine indirectly tests the correctness of the PGA gain values, avoiding the use of a test stimulus. If the gains are outside the limits allowed by the application needs (including the measurement errors), it is accepted that the system is faulty and a reconfiguration process starts in the host computer.

It should be noted that in order to consider the whole node reliable, we assume that the tests of the other parts of the node are solved for instance with techniques and strategies like those presented in [20, 21].

4. System Self-Adaptation

4.1. Overview of GA. As previously stated, GA evolves the gain values of the four amplifiers with the goal of maintaining the system overall gain within specifications and the bandwidth as large as possible. As we consider that gain is the most important objective, we employ the ε-constraint method for implementing the optimization. This method performs the optimization with respect to one objective and transforms the others into restrictions [22, 23]. The use of this method allows employing traditional GAs, as described in [6, 7, 24].

Figure 6 shows a flowchart of the GA used in this work. For the sake of clarity, in the following paragraph, numbers are related to the corresponding block in the flowchart. Appendix B shows Matlab code for the operations selection, crossover, and mutation.

(1) The algorithm randomly generates (with uniform probability) an initial population of individuals that are possible solutions to the problem. Each solution in the population is a string of bits, also called chromosomes or genotypes.

(2) For every evolutionary step, known as a generation, the individuals in the current population are evaluated according to some predefined quality criterion called the fitness function. To form a new population (the next generation), individuals are selected according to their fitness; high fitness individuals present better chances to appear (survive) in the next generation, while low fitness ones are more likely to disappear.

(3) The GA takes into account the restrictions by the use of a penalty technique (3.1 in the flowchart). By using this technique, the solutions violating the restrictions are modified in their fitness values (based on the violation degree) in order to decrease their chances of being selected.

(4) The selection of the individuals for the next generation is performed through the method of the rotating roulette. The probability of an individual to be selected for crossover is proportional to its fitness.

(5) The crossover operator selects two individuals, called parents, and exchanges parts of their information (the string of bits) to form two new individuals, called offspring. In its simplest form, it exchanges bits from one parent for bits from the other parent, creating two offspring. If the two parents do not undergo the crossover operation, they are copied unchanged to the new pool of individuals.

(6) The mutation operator is applied to the new pool of individuals produced after the application of crossover. This operator prevents premature convergence to local optima by flipping bits (of individuals) at random with some probability.

(7) If the individual generated does not represent a legal solution due to the nature of the encoding technique, then it is repaired (7.1 in the flowchart).

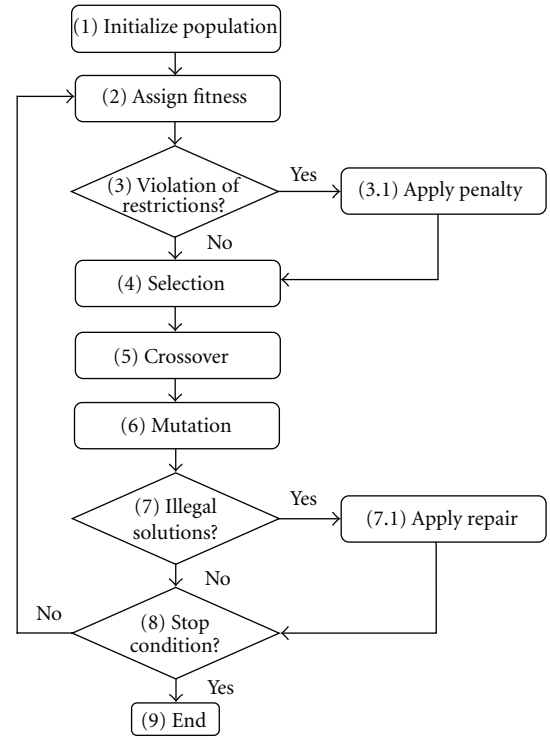

Figure 6: Flowchart of the genetic algorithm.

(8) This new generation goes through the process described above, from the fitness evaluation to the repair step. The cycle repeats until a stop criterion is met, such as a maximum number of generations is reached or a desired solution is found.

4.2. GA Parameters. The GA has to find the four PGA gain values (G_1 for PGA1, G_2 for PGA2, G_3 for PGA3, and G_4 for PGA4) that reach the condition:

$$\begin{aligned} \text{Min} \quad & (|A_{\text{tar}} - G_1 \cdot G_2 \cdot G_3 \cdot G_4|) \\ \text{subject to} \quad & \text{BW} \geq \varepsilon. \end{aligned} \tag{3}$$

In (3), A_{tar} is the target gain and ε is the minimum tolerable bandwidth (BW) of the system. The values available for each PGA are obtained by the SBST strategy presented in Section 3.

The BW value of the adaptive amplifier is found as the real positive solution to (4) [25]. For formulating this equation, each PGA is modeled as a first-order system, as reported by the vendor

$$\prod_{k=1}^{4}\left[1 + \left(\frac{BW}{p_k}\right)^2\right] = 2. \tag{4}$$

In (4), p_k is the pole of the kth PGA. Its value is calculated as

$$\begin{aligned} p_k &= \frac{\text{GBWP}}{G_k} \cdots \quad \text{if } G_k \geq 1, \\ p_k &= \text{GBWP} \cdots \quad \text{if } G_k \leq 1. \end{aligned} \tag{5}$$

In (5), GBWP is the gain bandwidth product reported by the vendor and G_k is the gain of the kth amplifier.

For simplifying the operation of the GA, the equation of the fitness function (f) is formulated for obtaining a maximum [6]:

$$f = B - |A_{\text{tar}} - G_1G_2G_3G_4|, \tag{6}$$

where B is a constant added for avoiding negative numbers.

For each PGA gain value, a simple binary codification of 6 bits is used. The length of the chromosome results consequently in 24 bits.

The creation of the population in the first cycle of the algorithm is made by using uniform initialization. The population size is 30, and the number of generations is 25. The fitness of each individual is calculated using (6). The size of the population is chosen according to the guidelines of [24] and ensures that the probability of finding a binary value 1 or a binary value 0 at each position in the chromosome exceeds 99.9%.

In order to apply the restrictions, the algorithm increases the f value to the individuals that present an overall gain within specification and BW greater than or equal to ε. The individuals with higher BW are assigned with a higher fitness value (Hf) as follows:

$$\text{Hf} = f + f \cdot \frac{(\text{BW} - \varepsilon)}{\varepsilon}. \tag{7}$$

On the other hand, the algorithm penalizes the individuals with BW below ε, even if they present a gain within specification. Consequently, these individuals will adopt a lower fitness (Lf), according to the following expression:

$$\text{Lf} = f - f \cdot \frac{(\varepsilon - \text{BW})}{\varepsilon}. \tag{8}$$

The selection of the individuals for the crossover is performed through the method of the rotating roulette. The probability of an individual to be selected for crossover is proportional to its fitness. The probability of crossover is 0.5 and the probability of mutation is 0.3. These values are chosen using previous experimental guidelines [10, 24].

In the new generation can exist illegal individuals, since there are 28 values (2^6–36) that cannot be adopted by the PGA. We propose for solving this problem a repair technique that replaces illegal individuals for legal ones. The technique assigns a legal value proportional to the value of the illegal solution (bigger illegal solutions are repaired as bigger legal solutions).

4.3. Fault Models and Fault Injection. The performance of the scheme presented here is evaluated by means of fault injection. To this end, it is necessary to define a fault model. As the system addressed in this paper is configurable, we adopt fault models similar to those used by the authors of [26–29]. In these papers, the authors propose fault models for the evaluation of test strategies for field-programmable analog circuits.

If the PGA is well designed, the operational amplifier can present wide deviations in its functional parameters without effects in its closed loop performance. Consequently, we consider that the main cause of PGA gain faults comes from degradations in the resistances that establish the gain. In each PGA, we consider four different fault models for the gain determined by the resistances R_a and R_b (Figure 2(a)).

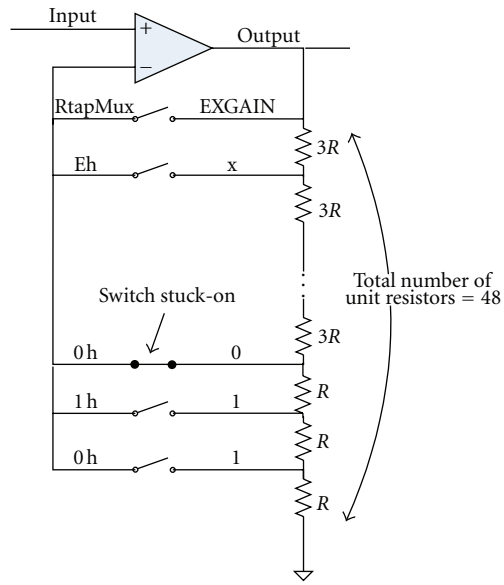

Figure 7: Stuck-on fault injection example.

The named Model 1, Model 2, and Model 3 are catastrophic, single fault models. Model 1 assumes that is not possible to establish a gain value in a PGA due to a stuck-open fault in one of the switches that configure the PGA. Model 2 considers that it is not possible to establish two gain values. This model takes into account that a stuck-open fault in one of the switches that connect the resistors (Figure 2(b)) can lead to two different values of gain not available for the PGAs. These two gain values present the same setup for EXGAIN and RtapMux signals but differ in the value of the signal Gain, which determines if the PGA gain is less than or greater than one (see Section 2.2). Model 3 considers a stuck-on fault in a switch, which is always in on state. Finally, Model 4 is a parametric one and assumes that there is a deviation in the gain values. This model takes into account parametric faults present in the resistor array.

Because PGAs are embedded in the PSoC devices, it is impossible to inject faults directly in the hardware. For this reason, we adopt a different approach. To inject a fault using Model 1, we eliminate from the search space used by the GA the gain value that is assumed as faulty. When Model 2 is used, we remove the two gain values that present the same combination of EXGAIN and RtapMux.

The injection of fault Model 3 requires considering the location of the switch that presents the stuck-on fault and establishing the gain value under this condition. For instance, Figure 7 shows a stuck-on in the switch that establishes a gain value of 16 (EXGAIN = 0, RtapMux = 0, and test condition 3 in Table 1). If under this fault it is programmed a gain value of 24 (EXGAIN = 1, RtapMux = 1, and test condition 2 in Table 1), then a gain value of 23.5 is obtained instead. This

Table 2: Resistor-matrix test results.

Test condition	PGA1 Vref (V_{DD} units)	PGA2 Vref (V_{DD} units)	PGA3 Vref (V_{DD} units)	PGA4 Vref (V_{DD} units)
1	0.0199	0.0215	0.0223	0.0211
2	0.0407	0.0423	0.0430	0.0411
3	0.0615	0.0634	0.0641	0.0604
4	0.1238	0.1258	0.1267	0.1255
5	0.1861	0.1882	0.1893	0.1867
6	0.2488	0.2508	0.2520	0.2499
7	0.3114	0.3132	0.3143	0.3125
8	0.3741	0.3756	0.3767	0.3742
9	0.4366	0.4380	0.4391	0.4373
10	0.4992	0.5006	0.5012	0.4993
11	0.5617	0.5633	0.5636	0.5634
12	0.6239	0.6254	0.6257	0.6253
13	0.6867	0.6880	0.6881	0.6860
14	0.7492	0.7504	0.7505	0.7504
15	0.8120	0.8131	0.8129	0.8120
16	0.8745	0.8752	0.8750	0.8745
17	0.9370	0.9376	0.9374	0.9376
18	1.0000	1.0000	1.0000	1.0000

process is repeated for all the test conditions described in Table 1.

For injecting faults using Model 4, we shift the gains values in the search space. Particularly, we consider that a PGA presents a deviation in its gain values in a percentage of its nominal values of ±10%, ±20%, ±30%, ±40%, and ±50%.

5. Experimental Results

5.1. Test Results. The test routine is written in assembly language. The experiments were performed in a CY3210-PSoCEval1 board that provides the necessary hardware to evaluate the PSoC1 device CY8C29466-24PXI. In order to simplify the experiments, we implement in the device a communication module to the host computer through an RS-232 serial interface.

Table 2 shows the mean values of the resistor array for the test conditions depicted in Table 2. In the table, the values are very close to the ideal reported in Table 1 and the highest errors are for test conditions 1 to 4. The relative error to the nominal value is always below 5%, with the exception of PGA3, test condition 1. In this case, the error rises to 7%.

5.2. Fault Free Operation. We propose three different values for the target gain (A_{tar}): 2, 8, and 15 with the aim of evaluating the ability of GA for finding an acceptable solution in different scenarios. We set the value of the minimum −3 dB BW (ε) in $4E+06$ rad/s (636 kHz), also for demonstration purposes.

As GA is a stochastic process, its results could change according to the statistical distribution of the initial population. In order to see how the results could be affected by the setting of the initial population, we change the seed of its random generation and consider that all gains are available for running the algorithm. The distribution of the obtained results can be observed in the dispersion diagram depicted in Figure 8. In the figure, each point is a solution to the optimization problem changing the seed and shows the relative error of the gain versus the bandwidth for the three target gains. From the figure, the three target gains present relative errors in the range [−4.982%, 4.831%]. The lowest BW obtained for all the evaluated gains is $4.06E+6$ rad/s, above the required.

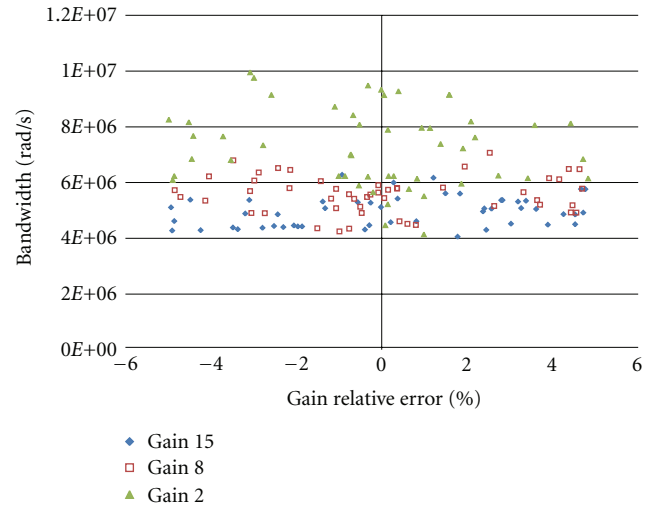

Figure 8: Gain relative error versus bandwidth. Fault-free operation.

Table 3 shows a characterization of the relative error for the three target gain values. We adopt the median as a measurement of central tendency because the data distribution is

Table 3: Gain relative error characterization under fault-free condition.

Target gain	Median (%)	Minimum error (%)	Maximum error (%)
15	0.244%	−4.932%	4.767%
8	−0.307%	−4.847%	4.686%
2	0.070%	−4.982%	4.831%

Table 4: Gain relative error characterization under fault model 1.

Target gain	Median (%)	Minimum error (%)	Maximum error (%)
15	−0.458%	−4.918%	4.746%
8	−0.825%	−4.975%	4.963%
2	−0.401%	−4.996%	4.841%

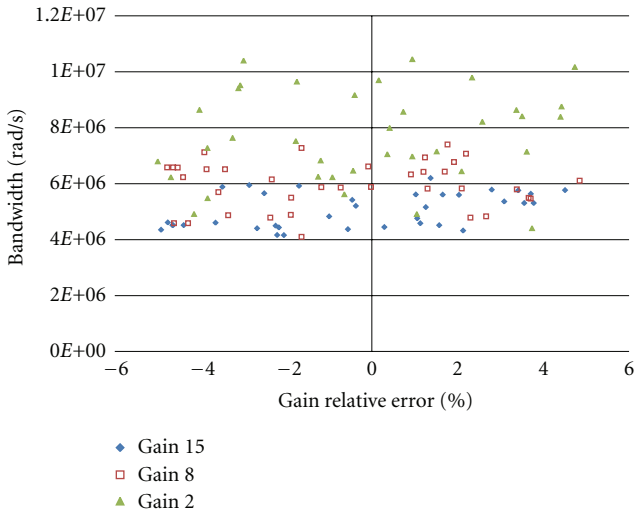

Figure 9: Gain relative error versus bandwidth. Operation under fault model 1 injected in PGA1.

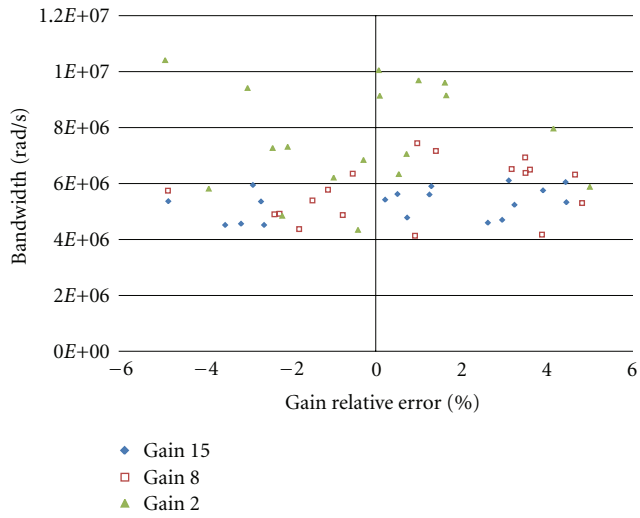

Figure 10: Gain relative error versus bandwidth. Operation under fault model 2 in PGA3.

not normal. We also present the maximum and minimum as a measurement of dispersion. The table shows that the relative error for all the cases is below the required (±5%). Additionally, the median of the relative error is close to zero for all target gains.

5.3. Operation under Fault Condition

5.3.1. Fault Model 1. For space reasons, we only report in Figure 9 the fault injection results corresponding to PGA1 which presents the worst results. The figure shows the relative error of the overall gain versus the bandwidth for the three target gains. Each point is a solution to the optimization problem under Model 1 fault. The figure shows that the error in the gain is near ±5% for all the target gains, while the bandwidth is above the required.

Table 4 summarizes the fault injection results for the four PGAs. In this table, we grouped the results by gain value. Comparing the gain error results obtained from fault-free operation (Table 3) and faulty operation (Table 4), the faulty system presents in the worst case an increase of 0.405% in the error range for gain 8. The median of the relative error decreases for the three gains, suggesting that the error distribution changes between the fault-free and faulty operation. In all the experiments, the GA is capable of reaching the target gain, with errors for all the gains in the range [−4.996%, 4.963%]. The lowest BW obtained for all the simulated conditions is 4.03*E*+6 rad/s above the required.

5.3.2. Fault Model 2. Figure 10 shows the results using Fault Model 2 in PGA3. We report only these results because this PGA presents the worst performance in the fault injection process. Each point in the figure is a solution to the problem when a Model 2 fault is injected in the PGA.

Table 5 summarizes the fault simulation results using Model 2. Comparing the gain error results obtained from fault-free operation (Table 3) and fault operation (Table 5), the faulty system presents in the worst case an increase of

Table 5: Gain relative error characterization under fault model 2.

Target gain	Median (%)	Minimum error (%)	Maximum error (%)
15	0.283%	−4.841%	4.988%
8	0.154%	−4.853%	4.966%
2	−0.762%	−4.964%	4.995%

Table 6: Gain relative error characterization under fault model 3.

Target gain	Median (%)	Minimum error (%)	Maximum error (%)
15	−0.052%	−4.996%	4.996%
8	−0.062%	−4.982%	4.803%
2	−0.508%	−4.897%	4.903%

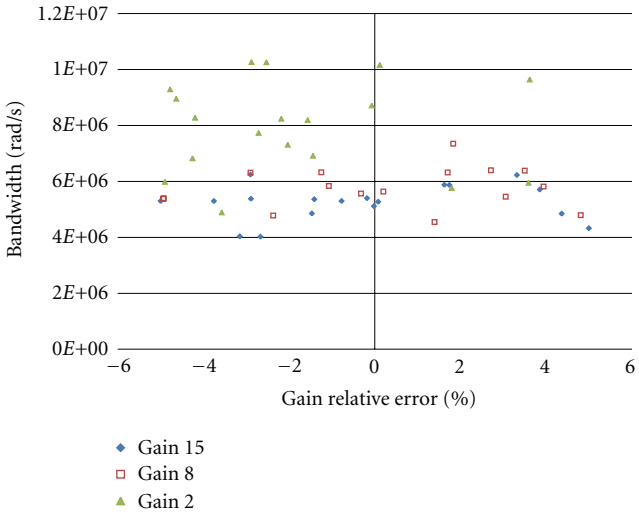

Figure 11: Gain relative error versus bandwidth. Operation under fault model 3 in PGA3.

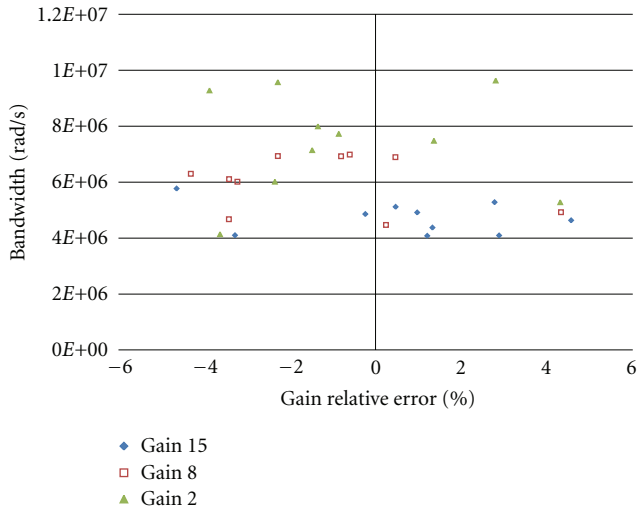

Figure 12: Gain relative error versus bandwidth. Operation under fault model 4 in PGA2.

0.286% in the error range for gain 8. The median of the relative error increases for the gains 8 and 15, while for gain 2 it decreases.

In all the experiments, the GA is capable of reaching the target gain, with errors for all the gains in the range [−4.964%, 4.995%]. The lowest BW obtained for all the simulated conditions is $4.13E+6$ rad/s above the required.

5.3.3. Fault Model 3. Figure 11 shows the results using Fault Model 3 in PGA3. We report only these results because this PGA presents the worst performance in the fault injection process. Each point in the figure is a solution to the problem when a Model 3 fault is injected in the PGA.

From the simulation results (Table 6), it is observed that the GA is able to reach the target gain with errors for all the gains in the range [−4.996%, 4.996%] and BW greater than $4.01E+06$ rad/s. The comparison of these results with the ones obtained under fault-free conditions (Table 3) shows that the faulty system presents in the worst case an increase of 0.293% in the error range for gain 8. The median of the relative error decreases for the gains 2 and 15, while for gain 8 it increases.

5.3.4. Fault Model 4. Figure 12 shows the deviation-fault simulation results for the PGA2, which presents the worst performance. The figure depicts the relative errors in the target gains versus the BW obtained for each deviation value in the gain. From the simulation results, it is observed that the GA is able to reach the target gain with errors for all the gains in the range [−4.943%, 4.571%] and BW greater than $4.08E+06$ rad/s. In the worst case, the obtained BW is above the required.

Table 7 summarizes the effects of Model 4. Comparing the relative error under fault-free (Table 3) and fault conditions, the faulty system presents a diminution for all the gains, despite the presence of relatively high deviation

Table 7: Gain relative error characterization under Fault Model 4.

Target Gain	Median (%)	Minimum error (%)	Maximum error (%)
15	0.718%	−4.809%	4.561%
8	−1.580%	−4.792%	4.571%
2	−0.946%	−4.943%	4.393%

Table 8: Performance comparison between ESM and GA.

Method	Max Runtime (sec)				
	Fault-free condition	Model 1	Model 2	Model 3	Model 4
GA	0.3164	0.404	0,395	0.388	0.371
ESM	202.94	195.20	192.57	202.94	202.94

faults. The median of the relative error increases for gain 15 while for the other two gains decreases, suggesting that the error distribution changes between the normal and faulty operation.

6. Comparison with Exhaustive Search Method

For a better characterization of the efficiency of the GA, we compare it with exhaustive search method (ESM). This method consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem statement [30].

We performed the comparison using two parameters: number of objective function evaluations (OFE) and runtime. GA and ESM are both implemented in Matlab.

GA performs at most 750 OFE (population size x number of generations), while ESM performs 1.679.616 OFE (6) to find the best solution for fault-free, Model 3, and Model 4 fault operation. For Model 1, this method performs 1.632.960 evaluations and for Model 2 it performs 1.586.304 evaluations.

Table 8 summarizes the performance comparison between ESM and GA. The runtime of GA shown is the maximum value of the runtime obtained in the worst case (gain 8 for Fault Model 3 and gain 15 for the other fault models). In the worst condition (Model 1), GA is 482.81 times faster than ESM. In the best condition (fault-free), GA is 641.40 times faster than ESM.

Regarding BW, we compare the one obtained by the GA against that obtained by using ESM, under normal and faulty operation. The bandwidths for the GA are close to the optimal ones obtained with the ESM, with the median between 17.19% and 31.23% lower. However, AG obtains these values in considerably less time and with fewer objective function evaluations.

7. Conclusions

We presented an adaptive amplifier implemented with programmable gain amplifiers in a PSoC device that is part of a sensor node in a wireless sensor network. The system is composed by an SBST scheme that checks all the available gains in the amplifier and by a GA for reconfiguring the chip resources that runs on a host computer. The GA presented is robust for the types of faults addressed in our evaluation. The fault simulation results show that the system maintains the overall gain and the bandwidth within specifications, despite the presence of catastrophic and deviation faults. In addition, its runtime is considerably lower than exhaustive search method.

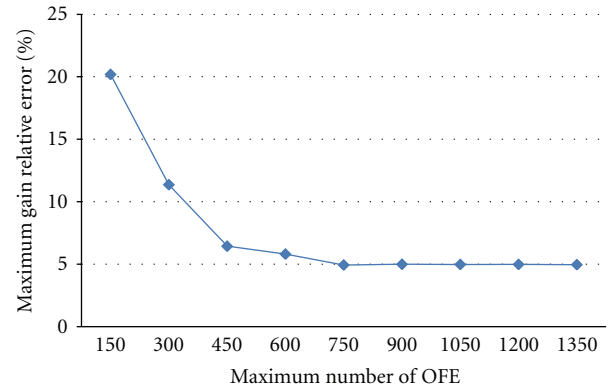

Figure 13: Gain relative error versus maximum number of objective function evaluations (fault-free operation).

Appendices

A. Selection of the Number of Objective Function Evaluations

In order to determine the maximum number of OFEs, we performed an experimental characterization of the GA. Particularly we observed the behavior of the gain relative error, the bandwidth, and the runtime versus the number of OFEs.

For illustrative purposes, we report in Figure 13 the error in the gain for the objective gain of 15 under fault-free conditions. As can be observed from the figure, the error diminishes up to the value of 750 OFE. Surpassing this number,

```
Roulette = rand(1,populationSize);% Matrix of random numbers for applying roulette.
% For each element of "Roulette" determines the appropriate selection range.
for i = 1: populationSize;
    p = 1;
    while FitnessRange(1,p)< Roulette(1,i)      % FitnessRange contains the range of
      p = p + 1;                                % selection of each individual in the
   end;                                         % population.
   PopRoulette(i,:)= Individual(p,:);           % The corresponding individual is
end;                                            % transferred to the matrix PopRoulette.
crossProbabIndividual = rand(1, populationSize);         % Random numbers for selection
% Individuals with crossover probability less than or equal to crossProbabGA
% are transferred to PopCross.
j = 1;
for i = 1: populationSize;
    if crossProbabIndividual (1,i)<= crossProbabGA       % crossProbabGA contains the
      PopCross (j,:)= PopRoulette (i,:);                 % defined crossover probability
      xpos(j) = i;                                       % for the algorithm
      j = j + 1;
   end;
end;
j = j - 1;
numberPartners = fix(j/2);                   % Calculation of the number of partners.
%
```

Figure 14: Matlab code for selection operation.

```
                                                % L is the length of the chromosome
cPoint = round(1+((L-1)*rand(1,numberPartners))); % Crossover point for each partner.
z = 0;
for i = 1:2:(numberPartners*2);
   z = z + 1;
% Mix from PopCross(i,:) and PopCross(i + 1,:)
   Offspring(i,1:L) = [PopCross(i,1:cPoint(1,z)),PopCross(i + 1,cPoint(1,z) + 1:L)];
   Offspring (i + 1,1:L) = [PopCross(i + 1,1:cPoint(1,z)),PopCross(i,cPoint(1,z) + 1:L)];
end;
% Insertion of new individuals in the population.
signal = 0;
for i=1: populationSize;
   for j = 1: numberPartners*2;
      if i == xpos(j)                           % Determines whether the position
         signal = 1;                            % "i" corresponds to the new
         p = j;                                 % individuals.
      end;
   end;
   if signal == 1
      PopNew(i,:) = Offspring(p,:);             % Transfer of a new individual.
      signal = 0;
   else
      PopNew(i,:) = PopRoulette(i,:);           % Transfer of an individual
   end;                                         % not involved in the crossing.
end;
%
```

Figure 15: Matlab code for crossover operation.

the error remains almost constant. It should be noted that for all the experiments the BW always remains above the minimum required.

Regarding the runtime, we observed that it also remains constant above 750 OFE because the GA finishes before reaching the maximum OFE. However, the algorithm could reach the maximum number of OFE due to its stochastic nature. For this reason, we decided to maintain the number of OFEs as low as possible (750).

It should be mentioned that similar behaviors have been observed for other gains and under fault conditions.

B. Matlab Code for GA Operations

B.1. Selection. The algorithm randomly generates a matrix called "crossProbabIndividual" containing the crossover probability of each individual. Then it transfers to PopCross the individuals with crossover probability less than or equal to the defined crossover probability for making the crossover operation. Also, GA calculates the number of partners (Figure 14).

B.2. Crossover. GA randomly determines the crossover point for each partner. The algorithm performs crossover and stores the new individuals in the matrix Offspring. Then, GA inserts these individuals and the ones that were not involved in the crossing in PopNew (Figure 15).

B.3. Mutation. GA calculates the total bits of the population. Then it randomly generates a matrix RandMut that is the mutation probability for each bit. The bits with probability less than or equal to the mutation probability are mutated

```
totalBits = populationSize*L;                          % Total bits of the population
RandMut = rand(1,totalBits);                           % Mutation probability of each bit
j = 0;
% posmut stores the position of those bits mutation probability less than to
% mutationProbabGA.
for i = 1:totalBits;
   if RandMut(1,i)< mutationProbabGA                   % mutationProbabGA contains the
      j = j + 1;                                       % defined mutation probability
      posmut(j) = i;                                   % for the algorithm
   end;
end;
if j > 0 % Mutation exists.
% For each bit to be mutated were determined by its location (row and column)
% in the mmatrix PopNew.
   for i = 1:j;
      c = posmut(i)/L;
      if c == fix(c)
         row = c;
         column = L;
      else
         column = posmut(i)-fix(c)*L;
         row = fix(c) + 1;
      end;
% Mutation is performed.
      if PopNew(row,column)== 1
         PopNew(row,column)= "0";.
      else
         PopNew(row,column)= "1";
      end;
   end;
end;
%
```

Figure 16: Matlab code for mutation operation.

after the calculation of its location on the matrix PopNew (row and column) (Figure 16).

References

[1] J. Chen, S. Kher, and A. Somani, "Distributed fault detection of wireless sensor networks," in *Proceedings of the 2006 Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks (DIWANS '06)*, pp. 65–72, September 2006.

[2] A. Flammini, P. Ferrari, D. Marioli, E. Sisinni, and A. Taroni, "Wired and wireless sensor networks for industrial applications," *Microelectronics Journal*, vol. 40, no. 9, pp. 1322–1336, 2009.

[3] K. Zhang, Z. Zilic, and K. Radecka, "Energy efficient software-based self-test for wireless sensor network nodes," in *Proceedings of the 24th IEEE VLSI Test Symposium*, pp. 186–191, May 2006.

[4] A. Krstic, L. Chen, W. C. Lai, K. T. Cheng, and S. Dey, "Embedded software-based self-test for programmable core-based designs," *IEEE Design and Test of Computers*, vol. 19, no. 4, pp. 18–27, 2002.

[5] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda, "Microprocessor software-based self-testing," *IEEE Design and Test of Computers*, vol. 27, no. 3, Article ID 5396292, pp. 4–19, 2010.

[6] D. Goldberg, *Genetic Algorithm. Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass, USA, 1989.

[7] R. S. Zebulum, M. Pacheco, and M. Vellasco, *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*, CRC Press, New York, NY, USA, 2002.

[8] Q. Ji, Y. Wang, M. Xie, and J. Cui, "Research on fault-tolerance of analog circuits based on evolvable hardware," in *Proceedings of the 7th International Conference on Evolvable Systems: From Biology to Hardware (ICES '07)*, pp. 100–108.

[9] P. C. Haddow, M. Hartmann, and A. Djupdal, "Addressing the metric challenge: evolved versus traditional fault tolerant circuits," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS '07)*, pp. 431–438, gbr, August 2007.

[10] J. M. Hereford, "Fault-tolerant sensor systems using evolvable hardware," *IEEE Transactions on Instrumentation and Measurement*, vol. 55, no. 3, pp. 846–853, 2006.

[11] Cypress Semiconductor Corporation, *PSoC Programmable System-on-Chip Technical Reference Manual*, Cypress Semiconductor Corporation, Boca Raton, Fla, USA, 2008.

[12] D. Seguine, "Just add sensor. Integrating analog and digital signal conditioning in a programmable system on chip," in *Proceedings of the 1st IEEE International Conference on Sensors, IEEE Sensors 2002*, pp. 665–668, June 2002.

[13] Cypress Semiconductor Corporation, *Programmable Gain Amplifier Data Sheet*, Cypress Semiconductor Corporation, Boca Raton, Fla, USA, 2009.

[14] M. Lovay, A. Arregui, J. Gonella, G. Peretti, E. Romero, and M. Lubaszewski, "Fault tolerant amplifier system using evolvable hardware," in *Proceedings of the 4th Argentine School of Micro-Nanoelectronics, Technology and Applications and 1st Uruguay School of Micro-Nanoelectronics, Technology and Applications (EAMTA '10)*, pp. 50–55, October 2010.

[15] L. Bissi, P. Placidi, and A. Scorzoni, "Offset voltage evaluation of analog blocks in a configurable mixed architecture for smart capacitive sensor applications," *Sensors and Actuators A*, vol. 140, no. 2, pp. 162–167, 2007.

[16] V. Mattoli, A. Mondini, K. Razeeb et al., "Development of a programmable sensor interface for wireless network nodes," in *Proceedings of the IEE International Workshop on Intelligent Environments*, pp. 1–6, 2005.

[17] Y. Chin, F. Chu, S. Huang, and H. Yang, "Based on PSoC electric angle meter," in *Proceedings of the 2011 1st International*

Conference on Robot, Vision and Signal Processing (RVSP '11), pp. 256–259, Kaohsiung, Taiwan, 2011.

[18] M. Hrgeti, I. Krois, and M. Cifrek, "Accuracy analysis of dissolved oxygen measurement system realized with cypress PSoC configurable mixed signal array," in *Proceedings of the IEEE International Symposium on Industrial Electronics 2005 (ISIE '05)*, pp. 1105–1110, June 2005.

[19] A. Doboli, P. Kane, and D. Van Ess, "Dynamic reconfiguration in a PSoC device," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '09)*, pp. 361–363, December 2009.

[20] B. Mihajlovic, Z. Zilic, and K. Radecka, "Infrastructure for testing nodes of a wireless sensor network," in *Handbook of Research on Developments and Trends in Wireless Sensor Networks: From Principle to Practice*, IGI Global, Hershey, Pa, USA, 2010.

[21] K. Virk and J. Madsen, "Functional testing of wireless sensor node designs," in *Proceedings of the 4th International Conference on Innovations in Information Technology (IIT '07)*, pp. 123–127, November 2007.

[22] Y. Collette and P. Siarry, *Multiobjective Optimization: Principles and Case Studies*, Springer, New York, NY, USA, 2003.

[23] J. Branke, K. Deb, and K. Miettinen, *Multiobjective Optimization: Interactive and Evolutionary Approaches*, Springer, New York, NY, USA, 2008.

[24] C. Reeves and J. Rowe, *Genetic Algorithms: Principles and Perspective. A Guide to GA Theory*, Kluwer Academic, Boston, Mass, USA, 2002.

[25] F. Centurelli, P. Monsurro, and A. Trifiletti, "Power-constrained bandwidth optimization in cascaded open-loop amplifiers," in *Proceedings of the 18th European Conference on Circuit Theory and Design (ECCTD '07)*, pp. 651–654, Seville, Spain, 2007.

[26] A. Laknaur and H. Wang, "A methodology to perform online self-testing for field-programmable analog array circuits," *IEEE Transactions on Instrumentation and Measurement*, vol. 54, no. 5, pp. 1751–1760, 2005.

[27] T. R. Balen, A. Q. Andrade, F. Azaïs, M. Lubaszewski, and M. Renovell, "Applying the oscillation test strategy to FPAA's configurable analog blocks," *Journal of Electronic Testing*, vol. 21, no. 2, pp. 135–146, 2005.

[28] T. R. Balen, J. V. Calvano, M. S. Lubaszewski, and M. Renovell, "Built-in self-test of field programmable analog arrays based on transient response analysis," *Journal of Electronic Testing*, vol. 23, no. 6, pp. 497–512, 2007.

[29] G. Pereira, J. Andrade, T. Balen, M. Lubaszewski, F. Azais, and M. Renovell, "Testing the interconnect networks and I/O resources of field programmable analog arrays," in *Proceedings of the 23rd IEEE VLSI Test Symposium*, pp. 389–394, 2005.

[30] K. Price, R. Storn, and L. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Springer, New York, NY, USA, 2005.