Hindawi Publishing Corporation
International Journal of Antennas and Propagation
Volume 2012, Article ID 196925, 10 pages
doi:10.1155/2012/196925

Research Article

How FPGAs Can Help Create Self-Recoverable Antenna Arrays

Miroslav Joler

Faculty of Engineering, University of Rijeka, 51000 Rijeka, Croatia

Correspondence should be addressed to Miroslav Joler, mjoler@riteh.hr

Received 12 March 2012; Revised 20 September 2012; Accepted 27 September 2012

Academic Editor: Krishnasamy Selvan

An approach to utilize Field Programmable Gate Array (FPGA) technology to control antenna arrays is presented based on the scenario of sensing a failure of any array element, analyzing degradation of the radiation pattern due to that failure, and finding a new set of excitations to the array elements in order to recover the radiation pattern as close to the original state as possible, thus creating a *self-recoverable antenna array* (SRA). The challenges of the SRA concept and embodiment of the recovery algorithm(s) are discussed. The results of the radiation recovery are presented on a few array cases, followed by a discussion on the advantages and possible limitations of the FPGA-based array control.

1. Introduction

Nowadays, we live in a world of ubiquitous and never-stopping communication services such as terrestrial and satellite broadcasting, surveillance, remote sensing, or rescue operations, to mention a few. Due to a constantly increasing number of wireless standards, services, and subscribers who want to enjoy it without disruption of services, preferably at any location, there has been a constant need to offer more robust techniques and technologies that will cope with this demand. We have witnessed various techniques being proposed and implemented concerning the topics of modulation and multiplexing, direction of arrival estimation, diversity and interference suppression, antenna reconfiguration or antenna array radiation control, optimization problems, and so forth, while on the technology side we witness an increased processor power and storage space at reduced dimensions, increased speed of circuitry, or algorithm parallelization, enabling previously proposed approaches or novel approaches to be implemented in more advanced ways than it was possible in the past.

Antenna arrays fall in one of the three broad categories. Terrestrial broadcasting systems are one category, typically characterized by radiation invariant antenna arrays. Once mounted, their radiation pattern does not change. Unlike them, *smart antennas*, which are referred to as either *switched beam* or *adaptive beam*, are intended for deployment in (terrestrial) mobile phone communications or satellite-based broadcasting. While they should bring improvements to the capacity of the system and reduction of interference levels, their increased complexity and (initial) cost seem to have not been adopted by wireless providers at a more significant degree [1, 2]. The third category is referred to as *reconfigurable antennas*, which are meant to change some of their parameters during the operation, such as the resonant frequency, radiation pattern, or polarization [3–5], although recent papers were mostly presenting a single antenna, rather than an array.

Regardless of the category, a particular antenna array can be classified with, various approaches of array control have been proposed, but all of them on the premise that *all elements* of an antenna array remain fully functional during the array operation. However, if any element of the antenna array fails to maintain its normal operation due to any possible cause, the original radiation pattern may, and likely will, degrade, possibly severely and the subscribers are then affected by a substantial loss of signal quality, which is very inconvenient at present times that we all expect the service to have extremely high uptime.

Various approaches have been taken to help managing of the array performance. In [6], a conjugate gradient-based algorithm was proposed to minimize the sidelobe level by reconfiguring the amplitude and phase distribution of the properly working array elements. In [7], a method of replacing the signals from the failed array elements was proposed,

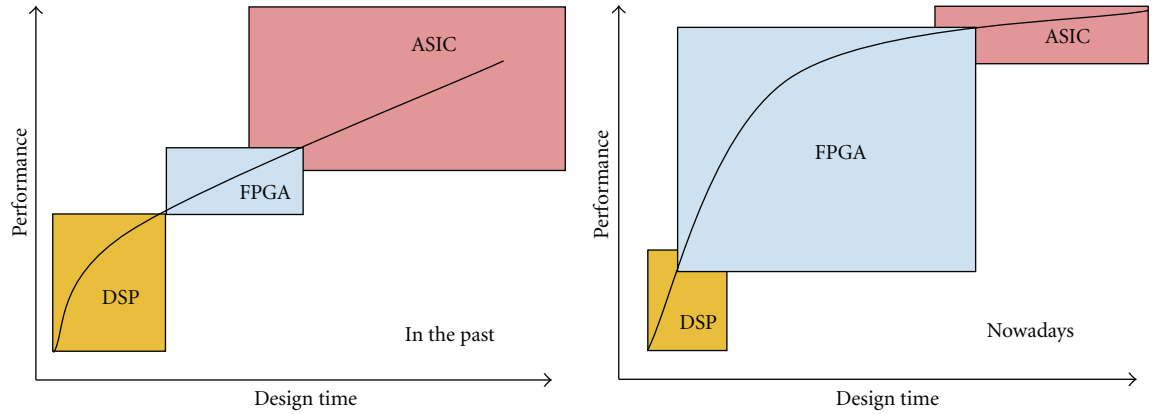


Figure 1: A comparison of DSP, FPGA, and ASIC embedded processors in terms of their characteristics in the past and nowadays.

yet applicable only to a receiving array. In [8], three mating schemes were examined within a genetic algorithm-based method proposed for an array failure correction. In [9], a Genetic Algorithm (GA) was used to synthesize antenna arrays. In [10], a genetic algorithm was used to *find* defective elements in a planar array. In [11], a degradation of the radiation pattern caused by the excitation coefficient error was analyzed. In [12], an artificial Neural Network (NN) was used to *find* faults in antenna array, whereas, in [13], a support vector machine was used to detect the location of failed elements within a planar array and the level of the failure. Recently, FPGA was utilized to control switches in a *single* reconfigurable antenna [14, 15] in a way that a control signal is generated to change the switch state.

In this paper, we discuss how *self-recoverable antenna arrays* could be created utilizing an FPGA device with embedded optimization algorithm(s). In the case of array element failure, it is desirable that the system has an ability to *heal itself* as much and fast as possible, before the service crew arrives, which is especially of interest for spaced-based systems or time-critical operations. It is known from antenna array theory [16] that a radiation pattern depends on the excitation magnitude and phase and the locations of the antenna array elements. Due to arbitrariness of the array layout (especially in the case of a random element failure), it is a challenging problem to tackle, even when numerical approaches are utilized.

The novelty of the approach proposed here is that although FPGAs have been utilized to help some tasks, as reviewed in Sections 1 and 2.1, no work, to the author's knowledge at this point, has been presented in the way to propose a complete solution: (a) monitoring of the array condition and detecting a flawed element; (b) finding a solution to recover the radiation pattern of a handicapped antenna array using FPGA (transmit mode primarily, but could be adapted for receive mode likewise) and send it to the array via signal conditioning circuitry; (c) proposing it as a general solution applicable to any array, if the proposed scheme is entirely implemented.

2. Proposed Concept

2.1. Advancements in FPGA Technology. FPGA technology has gained interest due to reprogrammable nature of its circuitry and possibility of algorithm parallelization that enables faster execution with respect to algorithms that are executed in a sequential order in typical processors. Because of such attractive features, researchers have tried it in a variety of studies, such as characterizing performance of FPGA in comparison with Application Specific Integrated Circuit (ASIC) [17] or Digital Signal Processor (DSP) [18], or use of FPGA for measurement calibration and digital beamforming processing [19]. Implementation of FPGA-based direction-of-arrival estimator using MUltiple SIgnal Classifier (MUSIC) algorithms was discussed in [20], a hardware-based particle swarm optimization method was related to an ASIC- or FPGA-implementation in the context of adaptive beam forming [21], whereas FPGA-based implementation of Minimum Mean Square Error (MMSE) algorithm was discussed for the benefit of adaptive arrays in [22].

Comparing where FPGAs stand with respect to DSPs and ASICs, the following characterization can be made. DSPs, as software (SW) programmable processors, are characterized by shortest design time, relatively lowest performance, and large flexibility, whereas ASICs, as hardware dedicated processors, are known for a relatively longest design time, least flexibility, yet highest performance. In comparison with DSPs and ASICs ten to fifteen years ago, FPGAs were practically viable for a small area of applications when plotted on a *performance versus design time* utility graph that is shown in Figure 1(a). Nowadays, however, FPGAs are recognized to occupy a much larger area on that utility graph (Figure 1(b)) thanks to technological advancements FPGAs that have undergone [23]. Besides the advancements in terms of their processor power, increased number of gates, storage space, power consumption, and cost, the reason also lies in the fact that their programming is today supported by high-level software tools that ease programmer's communication with the FPGA hardware (HW). In the past, FPGA HW could be

programmed only using some hardware description languages such as VHSIC (Very High Speed Integrated Circuits) Hardware Description Language (VHDL) or Verilog, whereas nowadays it is also possible to write a high-level model-based code in some of the popular tools such as Simulink, translate the code into a VHDL or Verilog using Simulink HDL Coder, and implement the code onto the FPGA using an FPGA vendor-specific tool, such as Altera's Quartus II, which provides synthesis, mapping, placing, and routing of the translated code onto the chosen FPGA. Besides that advancement, another limiting factor about FPGAs, up till recently, was to have the code written in fixed-point arithmetic, that is inherent to FPGAs, that could complicate the design and implementation of the code quite a bit. To work around it, one would typically take one of the following approaches: (a) use some look-up tables for complex mathematical expressions; (b) use COordinate Rotation DIgital Computer (CORDIC) algorithm [24]; (c) embed, that is, build, a soft core processor [25] into an FPGA chip, which would then use floating-point arithmetic and take advantage of some numeric library that would help with implementation of floating-point operations in HW [26–28].

Since recently, however, some FPGAs can also operate with floating-point arithmetic (see [25]), and even a mixed-arithmetic computations within same code, while some software tools (e.g., Simulink [29], HDL Coder [29], and DSP Builder [25]) enable automatic adaptation of floating-point-based algorithm into its fixed-point equivalent, on a bit-true, cycle-accurate basis [30], even including an automatic advisement on the necessary number of bits that have to be used for a certain parameter, which now greatly simplifies the programmer's effort in developing the code and substantially reduces the algorithm design time. That achievement will consequently make the use of CORDIC algorithm an obsolete idea when it comes to FPGAs. The use of CORDIC algorithm was a viable choice for more complex arithmetic operations in the past due to an ASIC-based notion that logic is cheap and multipliers are relatively expensive. Today, high-precision multiplications have become cheap in FPGA, also having benefits of predictable performance, low power, and low latency, which now makes the use of CORDIC approach quite an inefficient method to realize the arithmetic functions comparing to floating-point-enabled FPGAs [30] as complex equations can now be effectively built in single as well as double precision using a vendor's add-on software such as DSP Builder [25].

2.2. Principle of Operation of a Self-Recoverable Array. The principle of the *self-recoverable array* (SRA) system is shown in Figure 2. The central part of it is an FPGA device that receives an input signal i from the array *failure detector*, which can be realized in the following way: a small portion of the signal can be taken by a directional coupler (e.g., 20-dB directional coupler) from the lines feeding the antennas, in order to get scaled versions of the signals, V_{sc}^{i}. Each of these signals is then fed into a comparator circuit whose reference level V_{ref}^{i} is set to a desired fraction of the sampled signals to represent a scaled level of the failure threshold, that is,

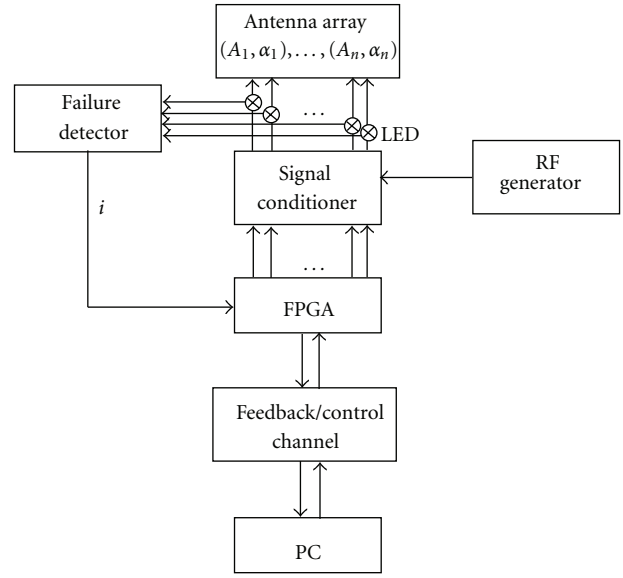


Figure 2: A concept of *a self-recoverable antenna array system*.

$V_{\text{ref}}^{i} = sV_{\text{sc}}^{i}$, $s \in [0, 1]$. Depending on the chosen digital logic of the comparators, the output of each comparator will give, for example a low level, that is, binary "0" if $V_{\text{sc}}^{i} > V_{\text{ref}}^{i}$, or a high level, that is, binary "1" if $V_{\text{sc}}^{i} \leq V_{\text{ref}}^{i}$. The sampled signals of the satisfactorily working elements will thus produce "0," while the signal from the flawed element will produce "1" at the output of their respective comparators and the outputs of all comparators will consequently form a binary word which will reveal the number (i.e., location) i of the failed array element when plugged into another digital-logic circuit. To prevent an ambiguity or higher complexity of the digital-logic circuits during the detection process, it is assumed here that failures occur one at a time, no matter how close in time subsequent failures may occur. To enable that, the location i of the failed element is to be stored in FPGA, to have the complete information in case of multiple failures, and V_{ref}^{i} is set to zero such that the output of this comparator always stays "0" and that way exclude this element in the next detection process when the comparator outputs of other array elements will be enabled to change their state and indicate new failure, still obeying the one-at-a-time principle. In this paper, the failure detector has not yet been implemented, but the element failure is simulated by manually changing the state of the switch on the FPGA development board, as it will be indicated in Section 4.

Upon receiving the information on the failed array element from the failure detector, the self-recovery algorithm in FPGA then analyzes the radiation pattern of the handicapped array, computes discrepancy between the original- and degraded-radiation pattern, decides whether to start the self-recovery algorithm based on the value of cumulative error with respect to a user-defined discrepancy tolerance level, and in the case of such a decision, computes a recovery solution to the failure, that is, a new set of excitation magnitudes and phases that need to feed the antenna array in order to recover the radiation pattern as close as possible to

the original condition and forwards their equivalent electrical signals to the *signal conditioner*.

The *signal conditioner* comprises power splitters, voltage controlled phase shifters, and voltage controlled attenuators. The power splitters divide the power received from the RF generator on N portions which then enter the respective voltage controlled attenuators and phase shifters (which are controlled by the voltage levels that come from the FPGA outputs). They altogether condition the excitation signal towards each array element. The concept is currently proposed for the transmitting system, but it could be adapted for the receiving system as well.

It is envisioned that with further development, the FPGA chip could be loaded with a few recovery algorithms, if desired, which could respond to a given failure in different ways—from the fastest, but the least general one (such as the response of a Look-Up Table (LUT) that would contain solutions to only a limited number of failure patterns), to the most general one (e.g., a GA or some other general optimization algorithm), yet a relatively slower one. Figure 3 illustrates this basic idea of having a few "recovery engines" loaded onto an FPGA board. A few algorithms can also be observed as just one algorithm that has three cases (e.g., GA, NN, and LUT) that are programmed within it. So, if one algorithm can be successfully embedded in FPGA, as it will be shown here in the case of GA, three algorithms, if coordinated within the same code, would practically mean a single algorithm, with just more lines of the code which is why it is viable to project that as doable, especially on more powerful FPGAs.

Whereas multiple recovery algorithms are not required, we consider it as beneficial to have for the following reason: in some larger N-element array, there are many failure patterns possible. To get the fastest possible recovery response, solutions to a small number of premeditated failure cases could be stored in an LUT and just read from it when failure occurs, which gives an instant response. However, it is not convenient to prepare solutions to all possible failure patterns of practical interest in advance, due to a large number of them and it would also have to be done again for every new array where SRA would be implemented, which does not constitute a generally applicable approach that was sought here. Also, such an extensive LUT would be quite large and it is questionable if an FPGA could store it within its available memory.

On the other hand, GA is capable of finding a solution to *any* failure pattern and does not require storing in advance anything besides the original set of excitations, yet GA is a relatively slow response due to its thorough search of the solution space.

Somewhere in between the LUT-based and GA-based recovery would be an NN-based approach upon this idea. An NN could be trained by some failure- and solution-patterns and should then provide recovery solutions to considerably more failure patterns than an LUT could provide, yet be faster than the GA.

In case all three algorithms were part of the SRA, the logic of the execution would be check if the failure falls within the cases that are stored in advance in the LUT. If *Yes*, read the solution from the LUT. If *No*, run the NN. If the NN produces a satisfactory solution based on the cumulative error measure that is defined in the algorithm, we got a quick solution. If not, run GA as it is the most general algorithm to find a preferable solution.

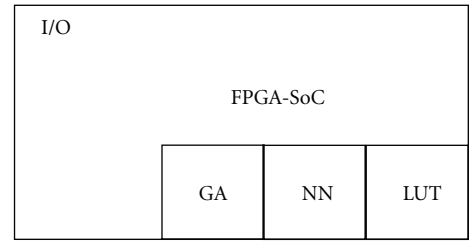


Figure 3: A vision of an FPGA chip with its input/output (I/O) circuitry and possibly loaded with a few recovery engines such as GA, NN, and an LUT.

While this is a possibility to consider in the future, at this stage of development, the GA-based approach, as the most demanding of the above three, was developed and embedded into the FPGA and the FPGA successfully ran it and found the recovery solutions. In the rest of the paper, the GA-based SRA algorithm will be explained in more detail and the results of running the SRA by an FPGA will be presented for some test cases.

2.3. Antenna Array Theory for the Self-Recovery Concept. An antenna array is a challenging case for a self-recovery scenario due to the complexity of the radiation pattern synthesis in case of an arbitrarily configured array. The radiation pattern of an arbitrarily laid out array of N antenna elements can be calculated by knowing the (x_i, y_i, z_i), $i = 1, \ldots, N$ coordinates of the array elements and the excitation of each array element, that is, the magnitude A_i and the electrical phase α_i. For the arrays comprising equal antennas, the total radiation pattern E in 3D space is computed as the product of the radiation pattern of a single array element E_0 and the so-called *Array Factor*, AF, given by [16]

$$E(\theta, \phi) = E_0(\theta, \phi) \cdot \mathrm{AF}(\theta, \phi). \tag{1}$$

The AF contains the vector sum of the individual contributions to the overall radiation

$$\mathrm{AF}(\theta, \phi) = \sum_{i=1}^{N} A_i e^{j(\psi_i(\theta,\phi)+\alpha_i)}, \tag{2}$$

where α_i is is the electrical phase that feeds the ith array element and ψ is the phase shift due to the position of an ith array element with respect to the given direction of observation and the phase center of the array, given as

$$\psi_i(\theta, \phi) = k(x_i \sin\theta \cos\phi + y_i \sin\theta \sin\phi + z_i \cos\theta), \tag{3}$$

where k is the wavenumber given by $k = 2\pi/\lambda$.

3. Embedded Algorithm

The flowchart of the major steps of the SRA code is shown in Figure 4. From (1)-(2), it is evident that if any array element

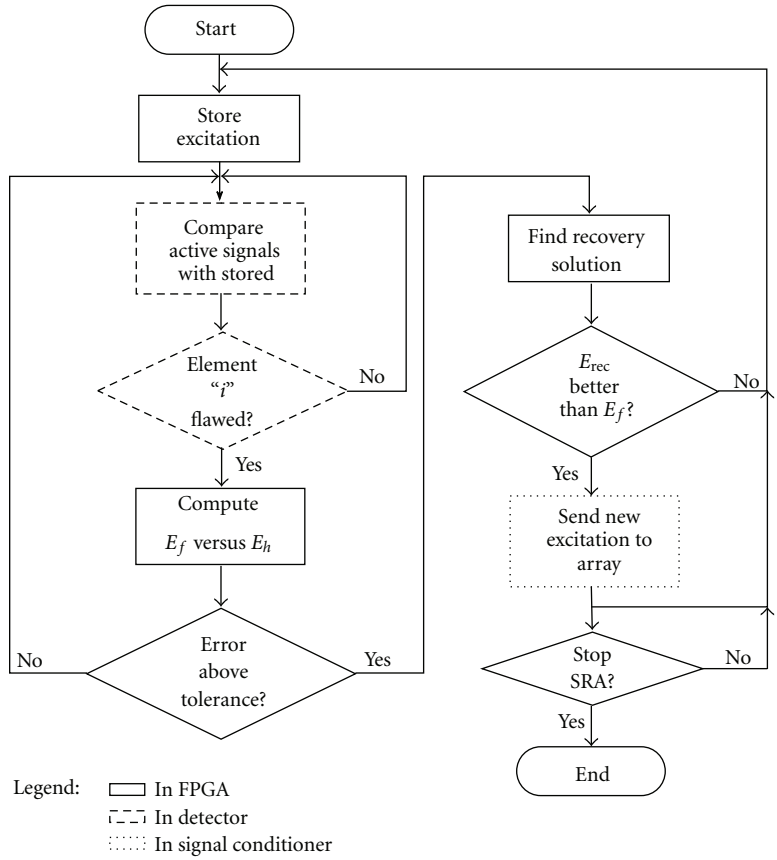


Figure 4: The flowchart of the principal steps in the SRA code.

stops delivering power due to some malfunctioning, the radiation pattern of the array will change, possibly severely. For the purpose of this computation, the array element is considered entirely failed (i.e., its magnitude coefficient equal to zero) even if it actually works at some reduced power, as was discussed in Section 2.2. During the testing on the FPGA, a particular array element failure was simulated by changing the states of the switches on the development board. To be able to analyze the damage due to a given failure, the original set of excitation magnitudes and phases is stored in the memory available on the particular FPGA. The positions of the array elements are considered fixed, upon this scenario, which is the case with most arrays in practical use. Thus, when the information about the flawed element is received from the *failure detector* (see Figure 2), the embedded SRA first computes the radiation pattern of the flawed array by (1)–(3) and compares it to the original radiation pattern. If the average cumulative error between the two patterns, defined by

$$e = \sum_{j=\phi_{\text{start}}}^{\phi_{\text{end}}} w_j \cdot \left| \dot{E}_{jo} - \dot{E}_{jf} \right|, \tag{4}$$

is greater than some tolerance level tol (here set to 1.5 dB), the SRA starts searching for a new set of magnitudes and phases that will feed the remaining healthy elements of the array and generate the radiation pattern that will be as close as possible to the original radiation pattern. In (4), j takes the values of the scanning angle, for example, ϕ, from its start value ϕ_{start} to the end value ϕ_{end} at every 1°, w_j is the weight factor whose purpose is to discriminate between the more- and less-important angles of the radiation pattern, while $\dot{E}_{jo}$ and $\dot{E}_{jf}$ are the complex values of the electric fields of the *original*- and the *flawed*-pattern at angle j, respectively. If $e <$ tol, the remaining original excitations are maintained because the flaw in the array did not corrupt the radiation more than acceptable.

As a basis for an SRA, one can choose from different optimization algorithms that have been proven useful for the array synthesis. In this work, genetic algorithm [31] was chosen to provide the self-recovery solutions. Without having available some of the more advanced tools that can automate the workflow, as discussed in Section 2, the algorithm was chosen to be done entirely in floating point, due to a need for complex functions containing square roots and complex numbers, such as in (1). To realize that, a NIOS II embedded processor [25] was created within the Altera Cyclone II FPGA using Embedded Design Suite environment. NIOS II now acts as a CPU, manages processing, input, and output streams, and communicates with an 8 MB Synchronous Dynamic Random Access Memory (SDRAM) and 512 KB Static Random Access Memory (SRAM) where the program code and general-purpose data are stored, respectively, as illustrated in Figure 5. The SRA code here was written in

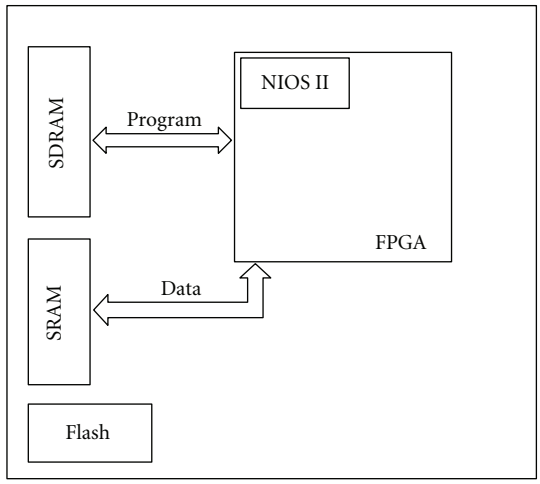


Figure 5: A NIOS II processor embedded into an FPGA chip.

C programming language and its libraries, whereas to speed up the computation of trigonometric functions in (1), a special *trigonometry* LUT was defined.

An N-element antenna array in this work is mapped into a GA such that each array element is presented by two genes—one for the *excitation magnitude* and another for the *excitation phase*. A single chromosome, that in the GA represents one possible recovery solution, thus contains $2N$ genes. The magnitude and phase are encoded by 4 bits and 8 bits, respectively, which provides 16 discrete levels of magnitude and the phase resolution of 1.4°, which is satisfactory for most practical purposes. To improve the GA efficiency, additional methods were included in this algorithm such as *rejuvenation* and *Gray coding*. Rejuvenation improves convergence of the GA by replacing the most of the population with new chromosomes, thus injecting a fresh genetic material in the solution space. It is applied if slow convergence is observed between a few consecutive iterations. Gray coding of the genes is used to help better explore the solution space. It is done by encoding the chromosomes from real values into Binary Coded Decimals (BCD) and then into Gray-coded binaries. That procedure better relates binary representatives in the search space with their decimal equivalents. A reverse process is performed before evaluating the fitness function. Additionally, to ensure monotonic convergence of the fitness function, the *elitist* strategy is applied between the consecutive generations by carrying over the fittest individual from the preceding generation, thus providing that the next generation will at least preserve the fitness of the preceding generation.

Whereas the genes are encoded from the real numbers, the *crossover* and *mutation* operations are performed directly on the binary words as these operations are perfectly fit for binary arithmetic. The selection of the parents for the operation of GA mating is based on the *tournament scheme*, as illustrated in Figure 6. A Random Number Generator (RNG) generates 2 integer indices in the GA population, for 2 possible parents. Their respective fitness values are then compared and the parent with the higher value is selected for mating. The procedure is then repeated to select the second parent that will undergo the mating process.

The mating is realized as a random two-point crossover in the way that the RNG selects the start point and the end point within two randomly selected parent chromosomes, as indicated in Figure 7. The respective memory addresses of those chromosomes are calculated and memory copy operations then swap the genetic content between the selected points of the two selected parents, thus creating two new offsprings.

The crossover in the SRA is always done with 100% probability as it was found to be beneficial for the quality of the solution (similar experience was reported in [8]).

The mutation, as a means of better exploring the solution space by constantly introducing a small amount of new genetic material is realized by two common parameters—the *mutation probability* and the *mutation rate*, complementing the bit values at random bit positions within randomly selected chromosomes with the exception that no mutation is performed on the fittest individual of the actual generation. Mutation always has to be well balanced between too high and too low a rate. Too high a rate leads to a random search and slows down the algorithm convergence, whereas too low a rate leads to a premature convergence towards a suboptimal solution.

In the spirit of GA, the list of the fittest chromosomes is obtained by decoding the genes of each chromosome, evaluating the radiation pattern of each chromosome by (1), computing the cumulative error with respect to the original radiation pattern using (4) and sorting the list of all the chromosomes by their cumulative error value, taking that the fittest chromosome is the one with the least error. The sorted list is the base from which the next GA generation is formed. The SRA stops after some predefined number of GA generations have been passed or when there is a chromosome which satisfies $e < \text{tol}$ and the difference between the smallest e's in the two consecutive GA generations is smaller than some user-defined margin (the latter ensures that SRA will not stop when the very first solution satisfying $e < \text{tol}$ comes up, but keep searching for the best possible solution and stop only when the difference in the solution quality in two consecutive generations is negligible to be worth running the SRA any longer or the maximum number of GA generations is reached).

4. Examples of Recovery Solutions

The SRA was tested on various test cases, a few of which are presented next. First, vertically polarized half-wave dipoles are placed in the xy-plane and equidistantly separated by $\lambda/4$. The radiation pattern in each case is computed in the xy-plane, where ϕ is the scanning angle. The three radiation patterns— of the *original*, *flawed*, and *recovered* array—are compared such that the flawed and recovered pattern are each normalized with respect to the *original* radiation pattern in order to track the effects that element failure and the SRA solution take on the radiation pattern with respect to the original condition.

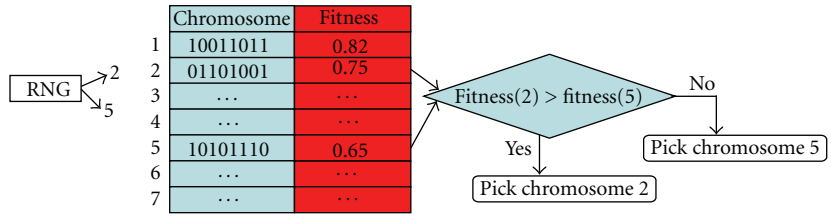


Figure 6: A tournament scheme for the parent selection in the GA.

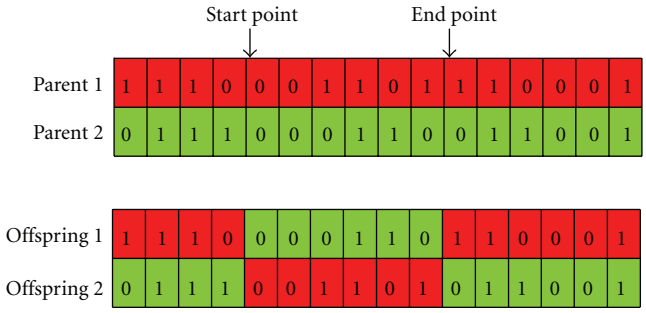


Figure 7: The two-point crossover scheme used in the SRA.

Table 1: A 4-element array parameters before and after failure.

Element	1	2	3	4
x	0	$\lambda/4$	$\lambda/2$	$3\lambda/4$
y	0	0	0	0
z	0	0	0	0
A_{orig}	1	3	3	1
α_{orig} (°)	0	0	0	0
A_{recov}	3	3	0	2.03
α_{recov} (°)	0	0	0	328.47

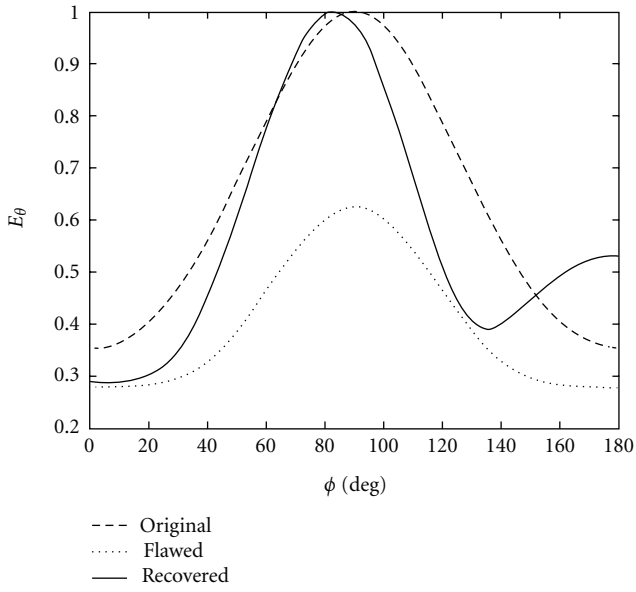


Figure 8: A PC-based recovery solution to a 4-element binomial linear array when run over 60 generations with the population size of 60 chromosomes.

Figure 8 shows a recovery solution the SRA found when run on a desktop PC over 60 generations and the population size of 60, 30% population replacement rate, 70% crossover probability, 2% mutation probability and equal weight factors, for a 4-element binomial linear array with element number 3 simulated as flawed. The array elements positions x, y, z, excitation magnitudes A and electrical phases α before and after the recovery are tabulated in Table 1.

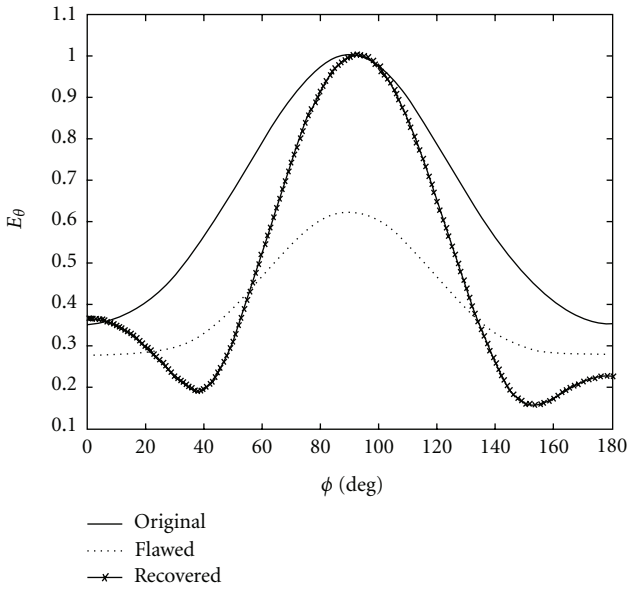
As our FPGA does not have as much computational power as a desktop PC, new GA parameters were being sought that would achieve a recovery solution of similar quality, but in fewer iterations of the SRA. The solution shown in Figure 9(a) was found in 12 min when SRA was executed within NIOS processor on FPGA over 8 generations and the population size of only 20 chromosomes, with 50% population replacement rate, 11.5% mutation probability, and 15% mutation rate. Figure 9(b) shows the development board during the execution of SRA. It can be noticed in Figure 9(b) that a blue Light Emitting Diode (LED) number 3 is *off*, corresponding to array element #3 being emulated as flawed by manually changing the state of the corresponding switch on the development board.

The tests were also performed on a 4×4 uniform- and binomial-planar array of dipoles placed in the xy-plane, as indicated in Figure 10, where element number 6 was taken as flawed (appearing as a hollow circle).

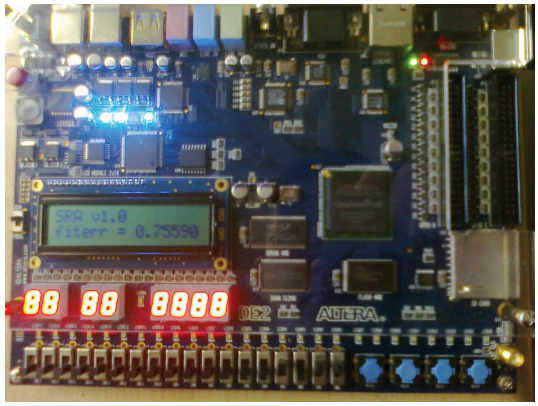
The scanning angle here is still ϕ in the xy-plane. The original uniform array radiation pattern, shown in Figure 11, has maximum at $\phi = 45°$. In 60 GA generations, it was possible to find a satisfactory recovery solution.

In a similar fashion, a superior recovery solution can be found for a 4×4 binomial array with 2 flawed elements (#6, #7) in only 15 generations and the population size of 40 chromosomes.

When simulations were done with linear (Figure 12) arrays of *Rectangular MicroStrip Antennas* (RMSA) placed along x-axis or y-axis, it was found that a single-element failure did not degrade the radiation pattern for more than the preset tolerable amount in all cases of uniform excitation and most cases of binomial excitation. As expected, failure of the central element of the binomial array causes the pattern degradation above the tolerance and initiates the recovery



(a) An FPGA-based recovery solution to a 4-element binomial linear array in only 8 generations and the population size of 20 chromosomes



(b) Execution of SRA on the development board with FPGA

Figure 9: An illustration of SRA recovery solution when executed on an FPGA.

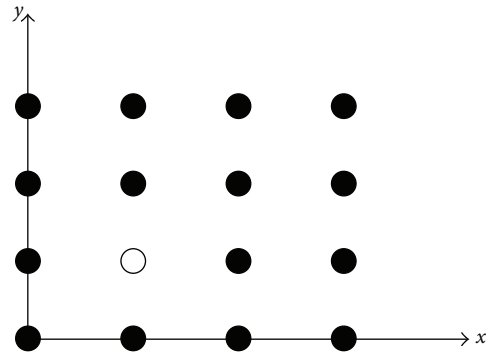


Figure 10: A 4×4 array layout in the xy-plane.

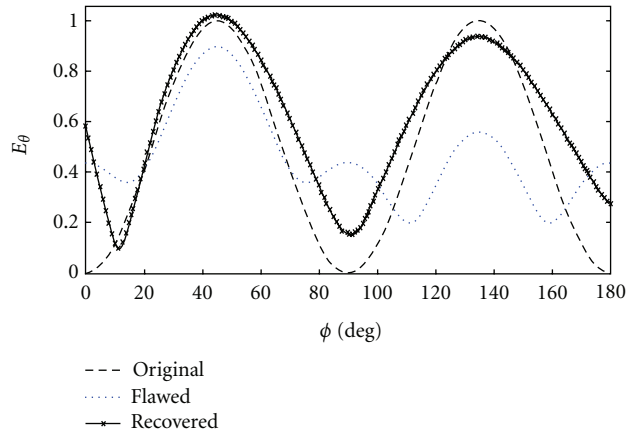


Figure 11: A recovery solution for a 4×4 uniform array with element #6 being flawed.

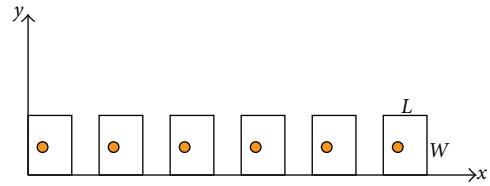


Figure 12: A schematic of a linear array of probe-fed rectangular microstrip antennas placed along x-axis.

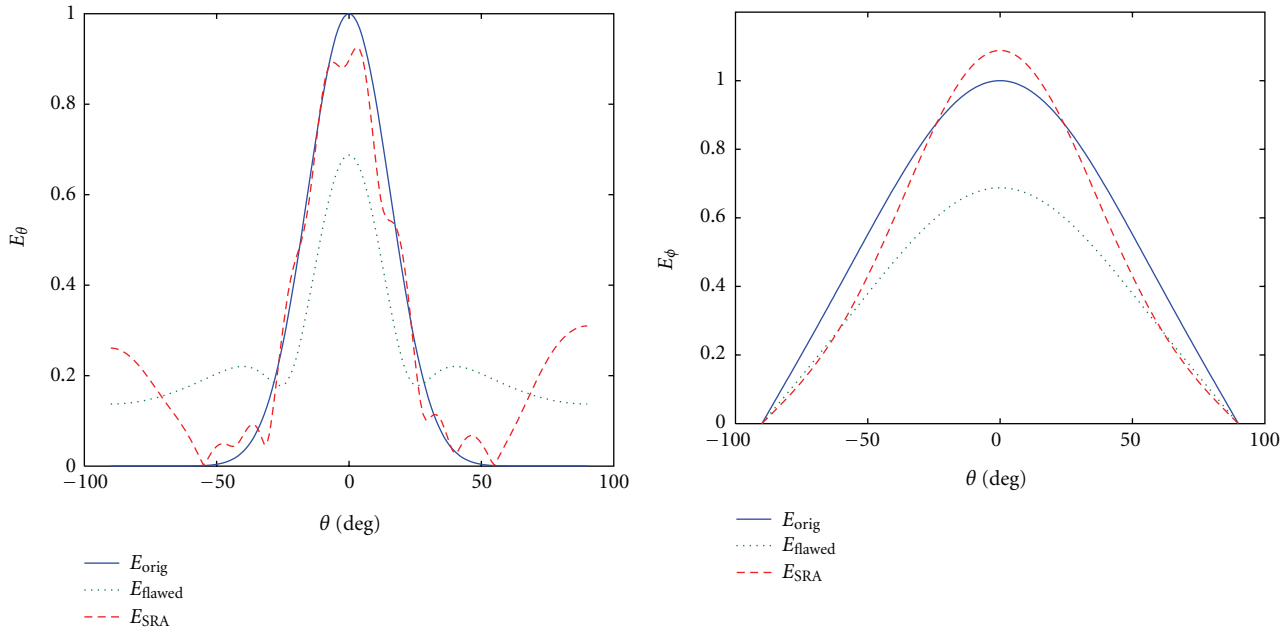
process, which can then find a preferable recovery solution, as illustrated in Figure 13 for the E-plane and H-plane of a 6-element linear binomial broadside array that was analyzed using the cavity model [16] expressions for the far field. The parameters of the RMSA used here were design frequency $f = 10\,\text{GHz}$, substrate relative permittivity $\epsilon_r = 2.2$, substrate height $h = 1.6\,\text{mm}$, patch length $L = 9.06\,\text{mm}$, patch width $W = 11.86\,\text{mm}$, and the element separation (between the same points on the patch) $d = 0.5\lambda$. Tests on a planar 4×4 array of RMSAs showed good robustness to failure of any single element.

From all the tests so far, it was noticed that besides the the antenna type, the number of flawed elements and their locations in the array, the fitness of self-recovery solutions also depends on the values of the GA parameters and the type of the array excitation. For example, the radiation patterns of binomial arrays of dipoles were found to be easier to recover than the radiation patterns of uniform arrays, whereas the radiation pattern of RMSA arrays were more robust to degradation than the dipole arrays, likely due to the wider major lobes of the former in both major cut planes.

While there is a great variety of array types that SRA can be tested on and optimized for, the bottom line here was essentially to test and show the prospect of using FPGAs for radiation pattern mitigation before further development takes place.

5. Conclusion

In this paper, advancements in the FPGA technology and accompanying software tools for FPGA programming were



(a) E-plane ($\phi = 0°$) (b) H-plane ($\phi = 90°$)

FIGURE 13: Pattern comparison in E-plane and H-plane of a 1×6 linear binomial broadside array of rectangular microstrip antennas placed along x-axis with element #3 flawed.

discussed and a concept for an FPGA-based creation of self-recoverable antenna arrays was proposed and successfully tested on a few scenarios of self-recovery in linear and planar arrays of moderate size.

At this stage of the development, the goal was to show that a complex SRA code can be embedded and run in an FPGA and recovery solutions to various failure patterns found. The future development is to include the surrounding circuitry (the failure detector and the signal conditioners) to enable testing on real physical arrays.

When it comes to large arrays, the drawback may presently lie in an insufficient number of I/O ports that FPGA board may have for a large number of external devices that are to be controlled by the excitation signals coming out of the FPGA board. There will also be an increased number of phase shifters and attenuators, but that factor is not related to FPGA boards, but rather to a large number of antennas and a need for an adjustable excitation for every one of them. Some alternative feeding schemes of the array elements may help circumvent this burden. One of them may be to form antenna groups with equal excitation settings.

Further optimization of the SRA and use of more powerful FPGA chips will achieve shorter computation times in the future. The SRA code used in this work was built using *C* language and floating-point arithmetic, whereas in the future development, the SRA code is planned to be realized in fixed-point or mixed arithmetic using the workflow discussed in Section 2, including more parallelization of the tasks. Moreover, embedding a few self-recovery algorithms (e.g., GA, NN, LUT) into an FPGA board, as discussed earlier, would equip the FPGA with a few possible levels of response to a particular failure situation.

Using the aforementioned opportunities of the FPGA technology and related SW tools, autonomous, smart, and reconfigurable FPGA-based controllers of antenna arrays could be built for multiple array-control scenarios, such as direction of arrival estimation, interference suppression, or radiation pattern synthesis, that may be needed due to changed conditions within or around the array system that are affecting the array performance and require the array feeding adjustment.

References

[1] F. Rayal, "Why have smart antennas not yet gained traction with wireless network operators?" *IEEE Antennas and Propagation Magazine*, vol. 47, no. 6, pp. 124–126, 2005.

[2] I. Stevanovic, A. Skrivervik, and J. Mosig, "Smart antenna systems for mobile communications," Tech. Rep., Ecole Polytechnique Federale De Lausanne, 2003.

[3] C. won Jung, M. J. Lee, G. P. Li, and F. De Flaviis, "Reconfigurable scan-beam single-arm spiral antenna integrated with RF-MEMS switches," *IEEE Transactions on Antennas and Propagation*, vol. 54, no. 2, pp. 455–463, 2006.

[4] S. Zhang, G. H. Huff, J. Feng, and J. T. Bernhard, "A pattern reconfigurable microstrip parasitic array," *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 10, pp. 2773–2776, 2004.

[5] H. Aïssat, L. Cirio, M. Grzeskowiak, J. M. Laheurte, and O. Picon, "Reconfigurable circularly polarized antenna for short-range communication systems," *IEEE Transactions on Microwave Theory and Techniques*, vol. 54, no. 6, pp. 2856–2863, 2006.

[6] T. J. Peters, "A conjugate gradient-based algorithm to minimize the sidelobe level of planar arrays with element failures,"

IEEE Transactions on Antennas and Propagation, vol. 39, no. 10, pp. 1497–1504, 1991.

[7] R. J. Mailloux, "Array failure correction with a digitally beamformed array," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 12, pp. 1543–1550, 1996.

[8] B. K. Yeo and Y. Lu, "Array failure correction with a genetic algorithm," *IEEE Transactions on Antennas and Propagation*, vol. 47, no. 5, pp. 823–828, 1999.

[9] D. Marcano and F. Durán, "Synthesis of antenna arrays using genetic algorithms," *IEEE Antennas and Propagation Magazine*, vol. 42, no. 3, pp. 12–20, 2000.

[10] J. A. Rodriguez, F. Ares, H. Palacios, and J. Vassal'lo, "Finding defective elements in planar arrays using genetic algorithms," *Progress in Electromagnetics Research*, vol. 29, pp. 25–37, 2000.

[11] S. Nakazawa, S. Tanaka, and T. Murata, "Evaluation of degradation of shaped radiation pattern caused by excitation coefficient error for onboard array-fed reflector antenna," in *Proceedings of the IEEE Antennas and Propagation Society International Symposium*, vol. 3, pp. 3047–3050, June 2004.

[12] A. Patnaik, B. Choudhury, P. Pradhan, R. K. Mishra, and C. Christodoulou, "An ANN application for fault finding in antenna arrays," *IEEE Transactions on Antennas and Propagation*, vol. 55, no. 3, pp. 775–777, 2007.

[13] N. Xu, C. G. Christodoulou, S. E. Barbin, and M. Martínez-Ramón, "Detecting failure of antenna array elements using machine learning optimization," in *Proceedings of the IEEE Antennas and Propagation Society International Symposium*, pp. 5753–5756, Honolulu, Hawaii, USA, June 2007.

[14] E. Al Zuraiqi, M. Joler, and C. G. Christodoulou, "Neural networks FPGA controller for reconfigurable antennas," in *Proceedings of the IEEE International Symposium on Antennas and Propagation and CNC-USNC/URSI Radio Science Meeting (AP-S/URSI '10)*, pp. 1–4, Toronto, Canada, July 2010.

[15] S. Shelley, J. Costantine, C. G. Christodoulou, D. E. Anagnostou, and J. C. Lyke, "FPGA-controlled switch-reconfigured antenna," *IEEE Antennas and Wireless Propagation Letters*, vol. 9, pp. 355–358, 2010.

[16] C. A. Balanis, *Antenna Theory: Analysis and Design*, Wiley-Interscience, 3rd edition, 2005.

[17] R. L. Walke, R. W. M. Smith, and G. Lightbody, "Architectures for adaptive weight calculation on ASIC and FPGA," in *Proceedings of the 33rd Conference Record of the Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1375–1380, 1999.

[18] C. Siriteanu, S. D. Blostein, and J. Millar, "FPGA-based communications receivers for smart antenna array embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2006, Article ID 81309, 2006.

[19] T. W. Nuteson, J. E. Stocker, J. S. Clark, D. S. Haque, and G. S. Mitchell, "Performance characterization of FPGA techniques for calibration and beamforming in smart antenna applications," *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, no. 12, pp. 3043–3051, 2002.

[20] M. Kim, K. Ichige, and H. Arai, "Implementation of FPGA based fast DOA estimator using unitary MUSIC algorithm," in *Proceedings of the IEEE 58th Vehicular Technology Conference, VTC '03)*, vol. 1, pp. 213–217, October 2003.

[21] G. Kókai, T. Christ, and H. H. Frhauf, "Using hardware-based particle swarm method for dynamic optimization of adaptive array antennas," in *Proceedings of the 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS '06)*, pp. 51–58, June 2006.

[22] A. Nakajima, M. Kim, and H. Arai, "FPGA implementation of MMSE adaptive array antenna using RLS algorithm," in *Proceedings of the IEEE Antennas and Propagation Society International Symposium and USNC/URSI Meeting*, vol. 3, pp. 303–306, July 2005.

[23] S. Yadati, Discovering FPGA advantages with MATLAB and Simulink. Online. MathWorks, 2012 http://www.mathworks.com/.

[24] R. Andraka, "Survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 191–200, February 1998.

[25] Altera Corporation, http://www.altera.com/.

[26] B. Lee and N. Burgess, "Parameterisable floating-point operations on FPGA," in *Proceedings of the 36th Asilomar Conference on Signals Systems and Computers*, pp. 1064–1068, November 2002.

[27] E. Roesler and B. Nelson, "Novel optimizations for hardware floating-point units in a modern FPGA architecture," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, vol. 2438 of *Lecture Notes in Computer Science*, pp. 637–646, Montpellier, France, 2002.

[28] G. Lienhart, A. Kugel, and R. Manner, "Using floating-point arithmetic on FPGAs to accelerate scientific N-body simulations," in *Proceedings of the 10th Annual IEEE Symposium Field-Programmable Custom Computing Machines*, pp. 182–191, Washington, DC, USA, April 2002.

[29] The MathWorks, http://www.mathworks.com/.

[30] E. Cigan and J. Inkeles, Introduction to FPGA design using MATLAB and Simulink. Online. MathWorks, 2012, http://www.mathworks.com/.

[31] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*, Wiley-Interscience, 2nd edition, 2004.