Development of Automatic Design Optimization Method for Ultrawide Bandwidth (UWB) Multi-Layer Dielectric Rod Antenna

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Chia-Wei Liu

Graduate Program in Electrical and Computer Science

The Ohio State University

2011

Abstract

Dielectric rod antenna has been shown to provide wideband, dual-polarization, and symmetric pattern. A two layer dielectric rod antenna (DRA) was shown to achieve more than 4:1 bandwidth of stable gain and with more than 55 degree HPBW. Although, it was predicted that a 3-layered DRA design should be able to achieve 8:1 bandwidth and a 4-layer DRA design could reach 16:1 bandwidth, such design has never been realized due to its design and fabrication complexity.

The key challenge in designing a multi-layer DRA is to choose proper thickness and dielectric constant of each layer to meet desired VSWR, pattern, and phase center requirements. This becomes even more difficult when the when the number of layers increases for achieving a greater bandwidth.

This paper discusses about how we overcome these challenges via the utilization of genetic-algorithm (GA) for design optimization procedure, and polymer-ceramic fabrication method.

For my family

Acknowledgments

I would like to give my deep gratitude to Dr. Chi-Chih Chen, my advisor, for his patience and all of the technical advise throughout my graduate work and research at the ElectroScience Laboratory, the Ohio State University. All his technical suggestion and encouragement provide me invaluable experience in both theoretical and practical aspects and give me improvement in this work.

I'd also like to thank my colleagues. Their opinions give me valuable idea to solve some technical problems.

Finally, I would like to express my special appreciation to my mom Kuang-Hou Han for her endless support of all my decisions.

Vita

November 26, 1983	Born – Taoyuan, Taiwan
June, 2006	B.S. Communication Engineering, Yuan-Ze University, Taoyuan, Taiwan
July, 2006 – Aug 2007	Communication Sergeant, Taiwan Army, Kinmen, Taiwan
November 2007 – May 2008	Research associate, Communication Research Center, Yuan-Ze University, Taoyuan, Taiwan
January, 2009 - Present	Graduate Research Associate, The ElectroScience Laboratory, The Ohio State University

Publications

Refereed Journal Articles

1. T. Chou, **C. -W. Liu**, W. -J. Liao, "Optimum Horn Antenna Design based on an Integration of HFSS Commercial Code and Genetic Algorithms for the Feed Application of Reflector Antennas," ACES Journal, Volume 25**,** Number 2**,** 2010

Fields of Study

Major Field: Electrical Engineering

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Ultra wideband (UWB) technology has been widely studied and extensively applied in both commercial and military areas in recent decades [1] because UWB system can receive more information by using wide bandwidth signal than narrow bandwidth signal.

An important application of UWB antennas is feed antennas inside an anechoic chamber which is a facility for antenna and RCS measurement. For the measurement in an anechoic chamber, a parabolic reflector is responsible to convert the spherical wave fronts which are radiated from a feed antenna into plane wave. A good feed antenna should provide constant beamwidth over the operation bandwidth, a stationary phase center to be located at the focal point of the parabolic antenna, minima size to avoid blockage, and low sidelobe level to reduce leakage. It's very difficult to design a single antenna that provides a wide bandwidth, stationary phase center, constant gain, dual polarization, and desired beamwidth. Moreover, an antenna designed only with conducting material usually has undesired trade-off between above design goals.

For instance, horn antennas are commonly used as feed antennas in a chamber because of wide bandwidth. But horn antennas are well-known for asymmetric E- and H-plane patterns, frequency-dependent patterns, and single polarization [2]. A square coaxial feed structure method solves the pattern asymmetric problem of TEM horn antennas, but the feed technique constraint its bandwidth [3]. Dual-polarization is hard to achieve because of the interactions between two planes. Using coaxial to quadruple-ridged method [4] obtains wider bandwidth and dual polarization, but the patterns are still frequency-dependent. The phase center of most horn antenna designs varies with frequencies.

Plane spiral antennas are also well-known as broadband, symmetric patterns, and predictable phase center [5]. Dual-linear polarization can also be achieved by sinuous spiral antennas. But the disadvantages of these antennas are fixed radiation patterns and not flexible for all applications.

Using dielectric material as main radiator can be able to conquer the undesired trade-off discussed above. For example, a dielectric horn antenna (DHA), which can achieve wideband, fixed phase center, and stationary desired patterns, has been presented [6]. By properly designing DHA shapes, it can also achieve different beamwidth requirements. But, since the DHA is always formed by single low dielectric constant material to reduce the reflection at the radiation end, the size of a DHA can't be miniaturized easily.

Another example that uses dielectric material as main radiator is dielectric rod antenna (DRA). DRA was first invented in 40's [7] and DRA is also refereed as a polyrod antenna. A traditional DRA can be decomposed a solid dielectric material into excitation, waveguide and radiation sections. Normally, electromagnetic waves are excited by a metallic waveguide into dielectric waveguide section and guided to radiation section. By properly tapering radiation section, electromagnetic field can be gradually excited. Several different DRA designs have been discussed in [8][9]. However, the drawback of these designs is the bandwidth is limited since single layer dielectric can avoid the excitation of undesired high order modes which cause pattern variation and bandwidth reduction.

The design of concentric two-layer DRA with improved bandwidth, phase center, and pattern performances has been presented [10]. By properly choosing dielectric constant and thickness for each layer, it can achieve 4:1 bandwidth, stable phase center and stationary patterns. The exiting two-layer DRA has elevated cross-polarization performance above 8GHz because only two layer dielectric materials can not provide enough boundary to avoid the excitation of high order modes when the operation frequency excesses 4:1 bandwidth. Hence, in order to design a multi-layer DRA to achieve 8:1 bandwidth, it’s necessary to use at least 3 or more layers.

However, the key challenge in designing a multi-layer DRA is to choose proper thickness and dielectric constant of each layer to meet desired VSWR, pattern, and phase center requirements. This becomes even more difficult when the when the number of layers increases for achieving a greater bandwidth.

In order to solve this problem, a new approach that uses genetic algorithm to optimize the multi-layer DRA will be demonstrated in this thesis. Although most of the new commercial EM simulation codes such as FEKO and HFSS have built-in parameter optimization functions, this approach consumes too much computer resources and is not easy to implement complex design goal that involves many parameters and different frequencies. This thesis will demonstrate an alternative antenna design optimization method that employs an external GA program which interacts with commercial EM simulation software such as HFSS, to automatically choose optimized thicknesses and dielectric constants of a 3-layer DRA for operating from 2 to 18 GHz with constant gain, beamwidth, and phase center performance objectives.

In the chapter 2, the operation principles of UWB multi-layer dielectric rod antenna will be discussed. The chapter 3 will presents the automatic optimization method for UWB DRA design. Chapter 4 will present the optimized three-layer DRA design that achieves constant gain, pattern, impedance, and phase center. Chapter 5 will provide the brief conclusion and future development.

CHAPTER 2

Operation Principles of UWB Multi-layer Dielectric Rod Antenna (DRA)

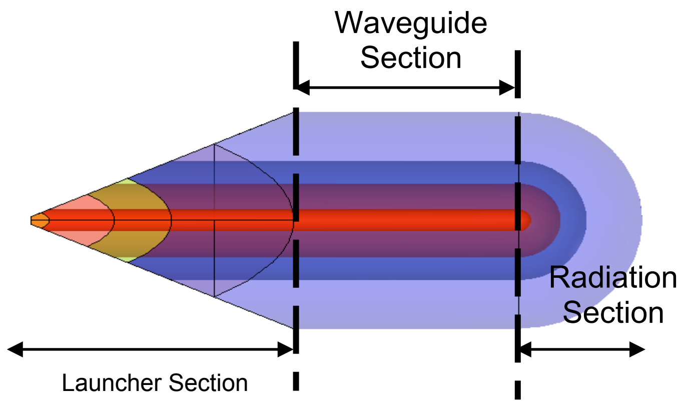
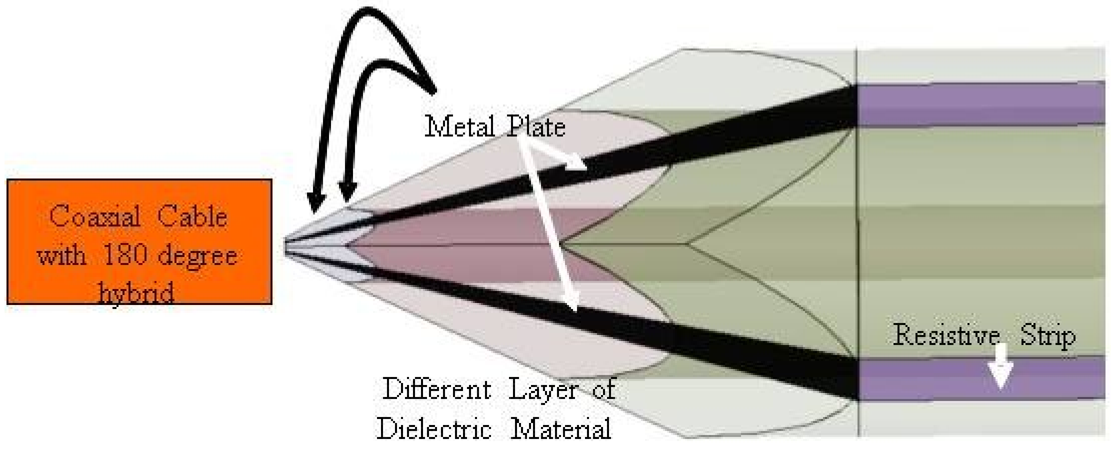


Figure 2.1 Geometry of multi-layer DRA

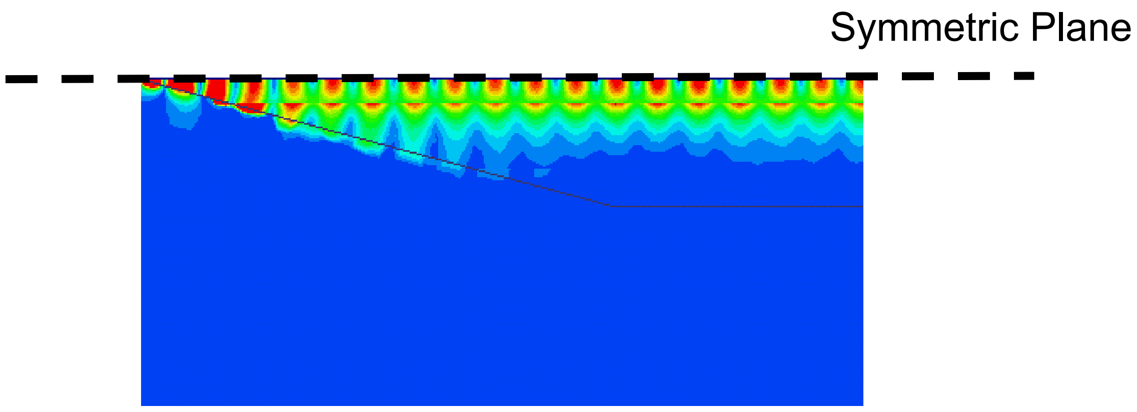
Figure 2.1 shows the basic concept of a wideband multi-layer DRA. Similar to previous single-layer and double-layer DRAs [10]-[12], it contains three sections: a wideband feed, a multi-layer UWB dielectric waveguide, and a radiation tip. The wideband feed launches wideband transverse electromagnetic (TEM) waves into the waveguide section. These TEM waves are then converted into hybrid HE_{11} modes [10] which are then guided along the multi-layer dielectric waveguide without producing high-order modes. The guided waves then reach the radiation tip where most guided energy is converted into radiation. The radiation tip needs to be designed properly to produce radiated fields with desired gain level, pattern, and polarization over the frequency range of interest

2-1 Feed Section

The feeding section is responsible for exciting wideband linearly-polarized TEM waves into the waveguide section. Chung and Chen [10] proposed a resistively terminated TEM horn as illustrated in Figure 2.2 for the feed section. The feed section is formed by machining the end of a multi-layer DRA into pyramid shape with isosceles triangular metal arms attached to the center of each face. By properly choosing the angles of the pyramid and triangular plates, the input impedance can achieve 100 ohms for matching to a balanced pair of 50 ohm cables whose outer ends are connected to the output ports of a 0-180 degree hybrid. It’s important to note that all the outer conductors of coaxial cables should be connected to each other to cancel any unbalance current. The far end each triangle plate is attached to a resistive strip for minimizing undesired end reflection and diffraction. Each pair of arms located at opposite sides of the pyramid is responsible for exciting linearly polarized TEM waves. Additional pair of launcher arms can be added to the other two sides to support dual-linear polarization operations as has been demonstrated in [10,13]. Figure 2.2(b) illustrates the TEM fields launched by a pair of the arms. The length of feed section is important to provide a smooth transition between feed and waveguide section without exciting high-order modes.


Metal Plate
Coaxial Cable with 180 degree hybrid
Resistive Strip
Different Layer of Dielectric Material

(a) Geometry of practical feed for multi-layer DRA



(b) Field distribution between launcher and waveguide section

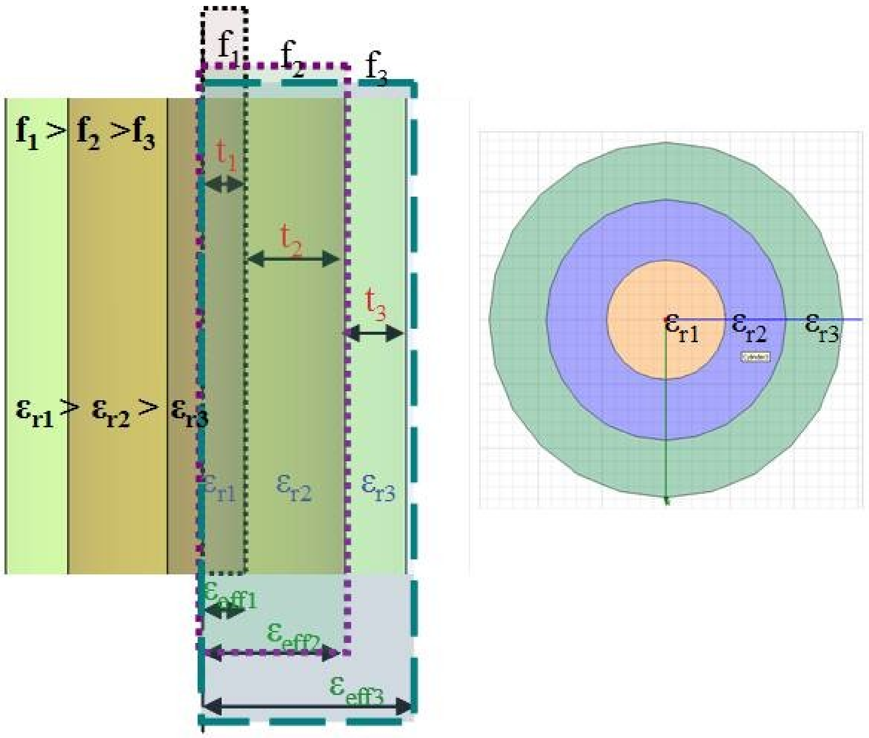
Figure 2.2 Geometry and field distribution of a practical feed for multi-layer DRA

2.2 Multi-Layer Dielectric Waveguide Section

The multi-layer dielectric waveguide section is responsible for guiding a HE_{11} mode [14] over a wide range of frequency without producing high-order modes. The TEM fields excited in the feed section naturally convert into desired HE_{11} mode in the waveguide section. The HE_{11} mode is chosen because it does have a cutoff and is linearly polarized within waveguide.

In order to achieve a wide bandwidth with single HE_{11} mode propagating inside the waveguide, it requires multiple layers of dielectric materials. The amount of energy guided within a dielectric waveguide is determined by the electrical dimension of its cross section [15]. The electrical size of the waveguide needs to be greater than half guided wavelength in order to effectively guide the EM fields. On the other hand, if the size is too large, undesired high-order mode could be excited inside the waveguide. Such high-order modes cause undesired pattern variation, elevated cross polarization compounds and bandwidth reduction.

To avoid exciting high-order modes, one can insert a dielectric core with a much higher dielectric constant to keep high frequency energy within a small dimension. This also helps miniaturizing the overall size as well. Chung and Chen [10] have proposed a relationship between the operation frequency of a multi-layer waveguide and effective dielectric constants and radii. Figure 2.3 shows the top and side view of an exemplary three-layer waveguide three layer waveguide. The dielectric constant of each layer increased as it gets closer to the center. Figure 2.4 shows the field distribution in the waveguide, and illustrates the phenomena that high frequency energy is guided inside the inner layer and vise verse. The optimal combination of layer thicknesses and dielectric constants will be optimized in the next chapter to meet all performance requirements.



(a)Side view (b) Cross section view

Figure 2.3 Example of a three-layer dielectric waveguide configuration

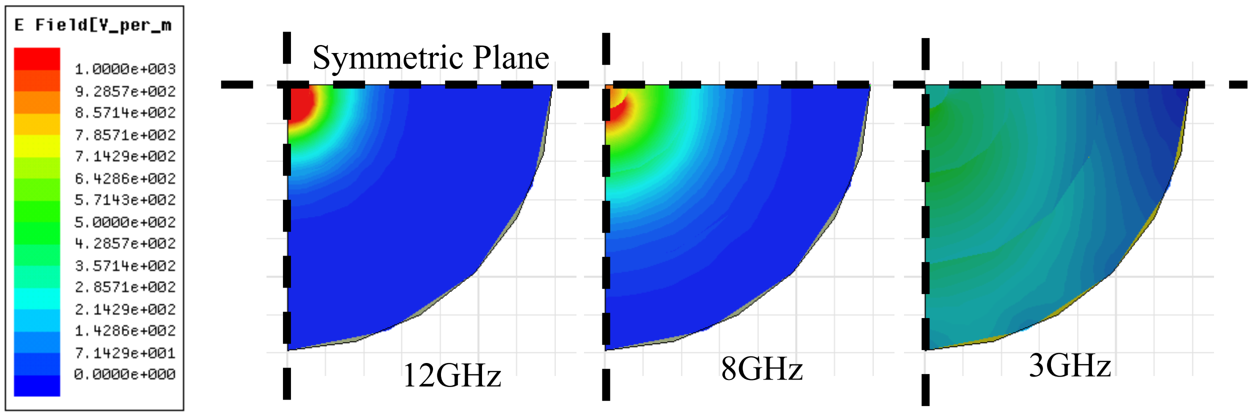
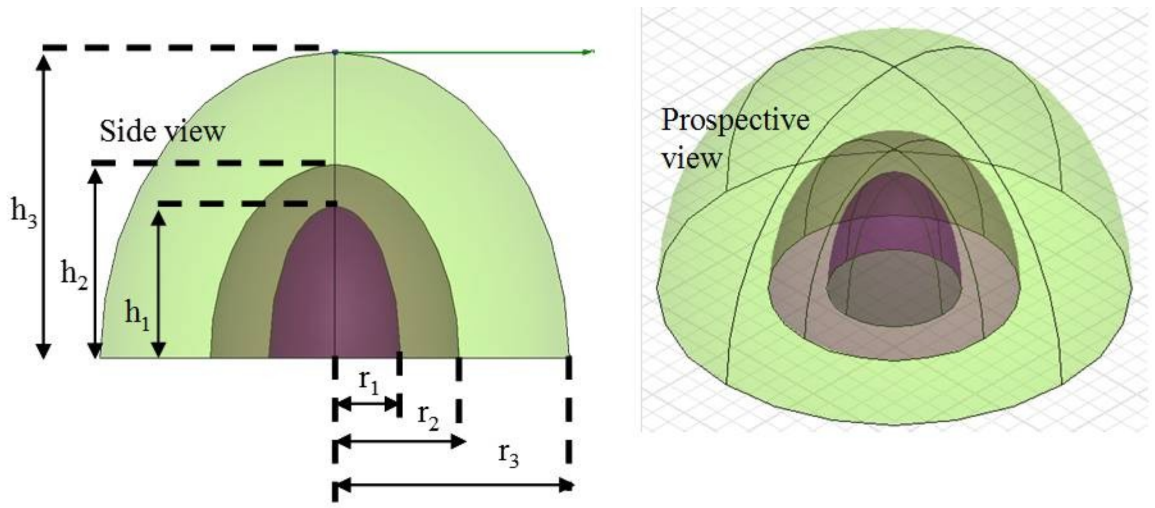


Figure 2.4 Field distributions in a three-layer waveguide section

2.3 Radiation Tip

The radiation tip section is responsible for producing stable patterns and stationary phase over the entire frequency range. Figure 2.5 illustrates the elliptical shape of radiation tip of a three layer DRA. The envelope of each layer takes the shape of a half ellipsoid. The axial ratio controls patterns, phase center, as well as any undesired reflection during transition from waveguide to radiation section.

The design of the radiation section is the most challenging part of a multi-layer DRA and need to rely on proper optimization procedure that involves full-wave simulations. This will be discussed in chapter 4.



(a)Side view (b)Prospective view

Figure 2.5 Example of a three layer radiation section

2.4 An Initial Design Example

Figure 2.6 and 2.7 present a preliminary design and HFSS simulation results of a three layer DRA design with hemispheric radiation tip. By using waveport that excites the waveguide with its eigenmode one can understand the pure radiation from the waveguide and radiation sections without influence of the feeding structure at this moment.

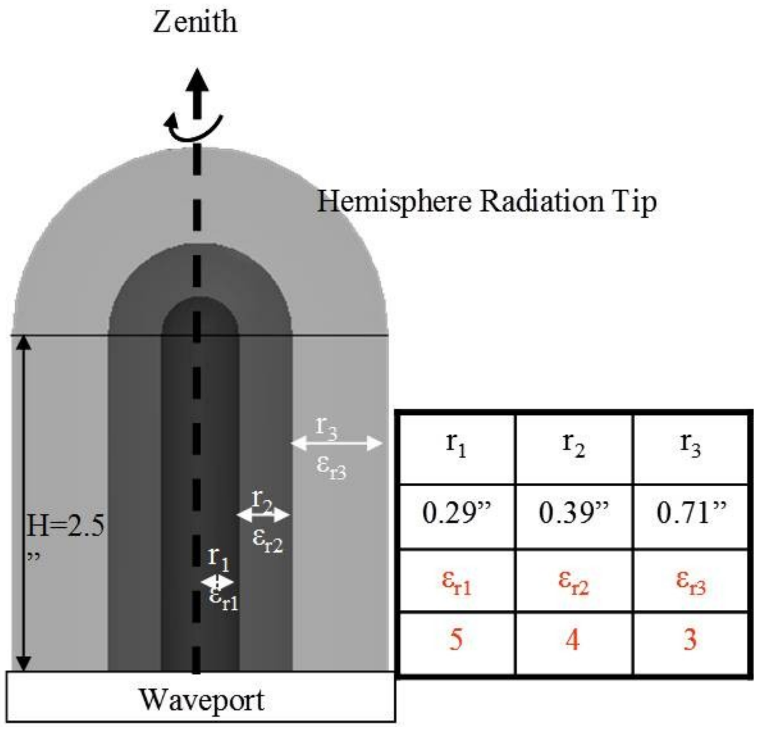
The S_{11} and realized gain results over an 8:1 bandwidth with fairly stable gain (Figure 2.7(b)) and S_{11} (Figure 2.7(a)). Furthermore, the radiation patterns in Figure 2.7(a)-(d) shows that E and H plane are symmetric from 2GHz to 14GHz. Figure 2.8(e) also shows desirable stable phase center with respect of the tip of the antenna. Ant the phase center is calculated based on the least square fit method by [16] as

$$PhaseCenter(f) = \min_{d}\left(\sum_{i=1}^{n}\Phi(\theta_i, d, f) - \Phi_{simulation}(\theta_i, f)\right) \qquad f_H \geq f > f_L \qquad (2.1)$$

$$\Phi(\theta, d, f) = -\frac{2\pi}{\lambda} d\cos\theta \qquad (2.2)$$

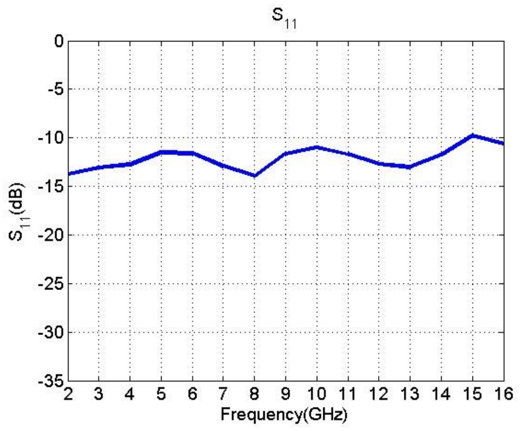
which compares the phase results from the simulation with phase distribution of ideal spherical wave and approximates the phase center position.

Because the current design is not optimized, the pattern beamwidth varies as frequency increases. The preliminary design has presented the potential to achieve the desired performance. In the next chapter, we will demonstrate that further design optimization using GA can lead to a multi-layer DRA with stable patterns, stationary phase center and more than 8:1 bandwidth.

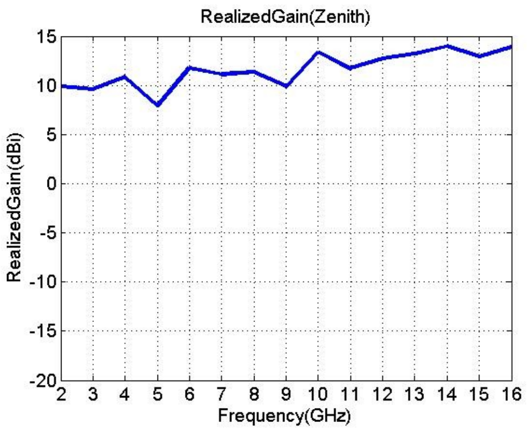


r_1	r_2	r_3
0.29"	0.39"	0.71"
ε_{r1}	ε_{r2}	ε_{r3}
5	4	3

Figure 2.6 Geometry of design example

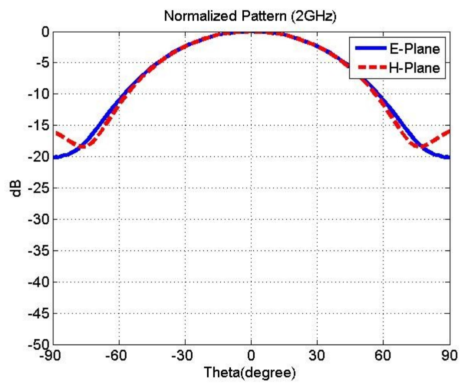


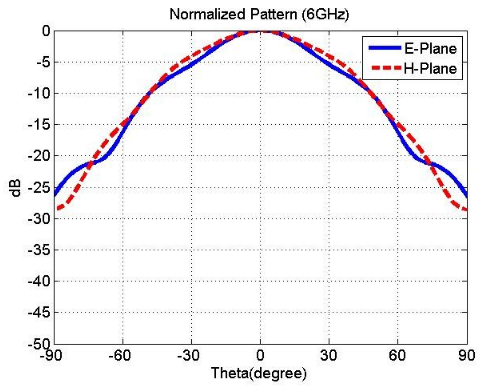
(a) S_{11}



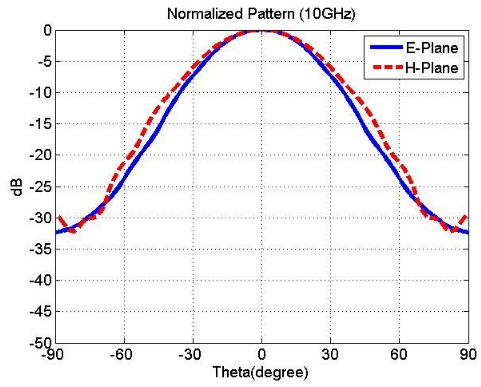
(b) Zenith Gain

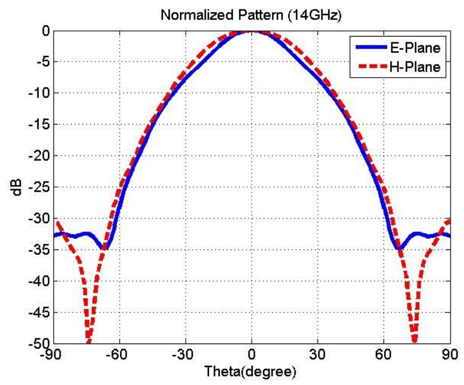
Figure 2.7 Simulation results



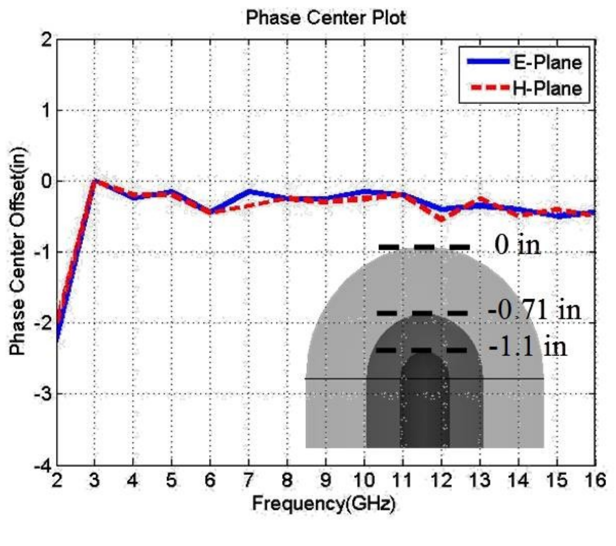


(a) f=2GHz (b) f=6GHz





(c) f=10GHz (d) f=14GHz



(e) Phase Center

Figure 2.8 Normalized Gain pattern and phase center plot

CHAPTER 3

Automatic Optimization Method for UWB DRA Design

In the design of a multi-layer DRA, the dielectric constant and thickness of each layer need to be chosen properly to meet the VSWR, pattern, and phase center requirement for wide bandwidth. It becomes increasingly difficult to achieve this goal as the number of layers increases. Moreover, these performance requirements are often dependent, and determining the optimized design that reaches the best trade-off among all the requirements is even harder. Therefore, in order to design a UWB DRA effectively and optimally, we proposed to use a global optimization algorithm to reach the optimal design.

Even though most of the current commercial EM software have build-in parametric optimization, the flexibility of those optimizers are limited. In other words, these optimizations can only deal with simple objective functions. For example, the phase center requires an additional computation from pattern data, most simulation software, such as FEKO and HFSS, can not perform optimization associated with phase center. In addition, in the case of multiple performance objectives such as VSWR, pattern, phase center, gain, etc. using built-in optimization become very inconvenient and very limited in terms of cost combination and weighting.

Moreover, though the superior goal of any optimization procedure is focused on the optimal results, it's also important to keep the performances of other designs during the optimization procedure. However, in order to save the data in the commercial software, it needs to store all the mesh date in the memory and wastes large computational resource especially for wideband optimization.

Therefore, our optimization approach was to use genetic algorithm optimization master program to control the simulation of commercial codes. This approach has greater flexibilities and better computation efficiency. The GA optimization approach has been shown to be effective in obtaining global optimum [17] without subject to the choice of initial cases. This chapter discusses the optimization procedure employed for obtaining optimum design of a three-layer DRA with 9:1 bandwidth.

3.1 Optimization Flowchart

Figure 3.1 shows the flowchart of the proposed automatic optimization procedure. It begins with a group (generation) of random generated DRA designs (population). Each design is defined by a set of design parameters such as a dielectric constant and radius of each layer. These antenna performances such as S_{11}, phase center, beamwidth and gain are then used to compute the fitness value according to fitness function which is defined according to target performance. The new generation with improved radiation performances is created from combining superior antenna designs with better fitness among the first generation. This process continues until the fitness value converges.

The whole optimization procedure can be subdivided into four different sections: (1) geometries controller, (2) GA operation, (3) HFSS simulation, and (4) fitness calculation. The geometries controller assigns values to the design parameters of HFSS model and creates new antenna geometries for performances analysis. The GA operation performs mating and mutation to generate new design parameters from designs with fitness values of current generation. The HFSS employs EM analysis core and exports radiation performances. The fitness calculation computes the fitness value of each design based on the radiation performance and pro-defined fitness function.

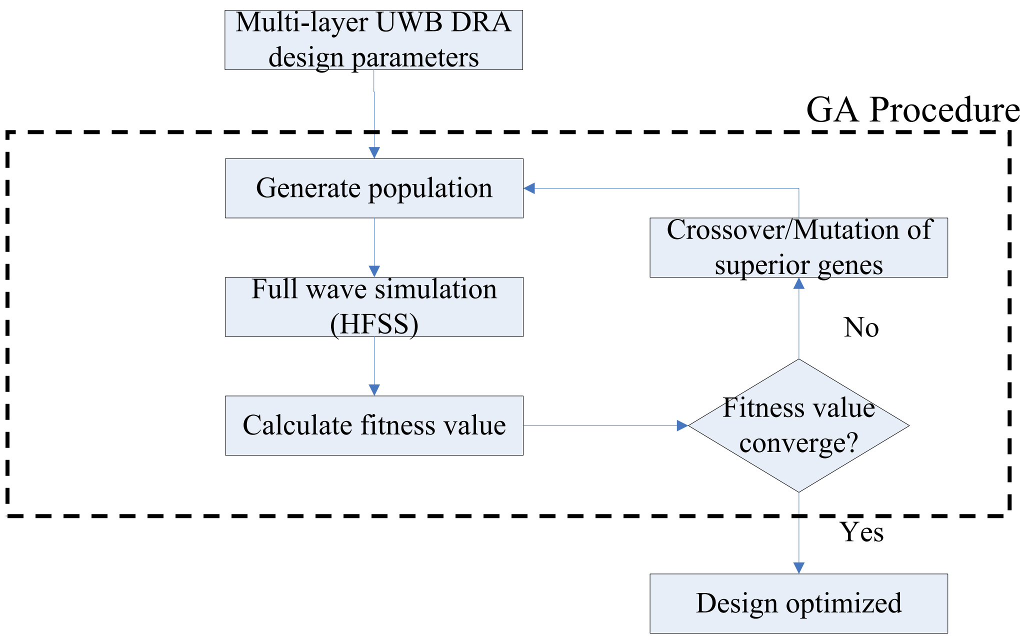

Multi-layer UWB DRA design parameters
GA Procedure
Generate population
Crossover/Mutation of superior genes
Full wave simulation (HFSS)
No
Calculate fitness value
Fitness value converge?
Yes
Design optimized

Figure 3.1 Flowchart of automatic GA optimization procedure for multi-layer UWB DRA design

3.2 Genetic Algorithm Description

Genetic algorithm [18] is employed in the master program to optimize the antenna geometry and material arrangement. GA can be specified into binary coding, selection, crossover and mutation four sections. The binary coding section coverts the decimal value into binary and vise verse. However, even though more bits stands for better resolution, the convergence efficiency decreases significantly if the number of bits become too large. In this program, each design parameter is presented by 2-4 bits. All the bits for antenna design are assembled together to form a chromosome with predetermined lower and upper bounds.

3.2.1 Selection Method

The selection method in the selection section determines the convergence rate of the optimization procedure. Many different selection methods, such as roulette, tournament, top percent, best, and random method, have been proposed [19]. Each method has each own advantages and disadvantages. Since the multi-layer DRA design is an UWB antenna and it requires significant computational time to analyze each design. Therefore, smaller population helps reducing computation time in each generation. Moreover, it's important to ensure that the members (genes) in the initial population span the solution to grantee convergence to global optimum. The tournament selection was adopted for out optimization. As shown in Figure 3.2, tournament selection method randomly selects N competitors form the population and the competitor with highest fitness value is chosen and put into mating pool to produce new population via bit crossover process. By changing the tournament size N, the convergence rate can also be adjusted. In the program, the tournament size N is set to 4.

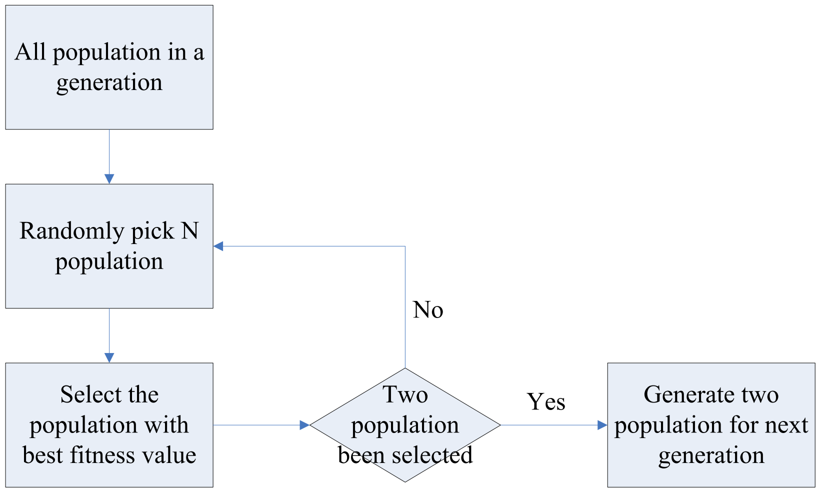

All population in a generation
Randomly pick N population
Select the population with best fitness value
Two population been selected
No
Yes
Generate two population for next generation

Figure 3.2 Tournament selection method

3.2.2 Crossover Method

Crossover and mutation procedures are genetic operators which are used to generate new offspring from selected parents in the mating pool. The main concept of crossover is that the offspring could be better than the parent if it combines the better characteristic from both parents. In our algorithm adopts the one-point crossover method [19] illustrated in the Figure 3.3. A crossover point is randomly selected for a pair of parents. The offspring are then generated by exchanging the genes after the crossover point.

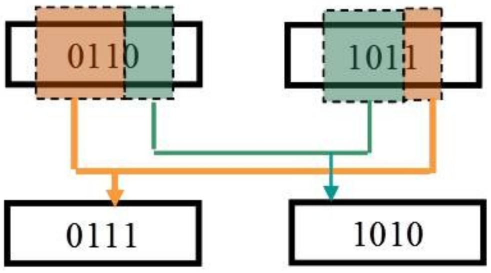


Figure 3.3 One point crossover method

3.2.3 Mutation Method

Mutation is also a genetic operator that randomly alters one or more gene in the chromosomes in the new generations from its original bit state. The mutation process can add new gene into the gene pool, which allows the genetic algorithm to achieve a better optimization results and avoid converging to a local optimum. The mutation method adopted in our GA procedure is the flip bit method [19] as illustrated in the Figure 3.4. This mutation method flips a randomly selected bit in the offspring with a probability P_m. The probability is predetermined to 0.1 in the program.

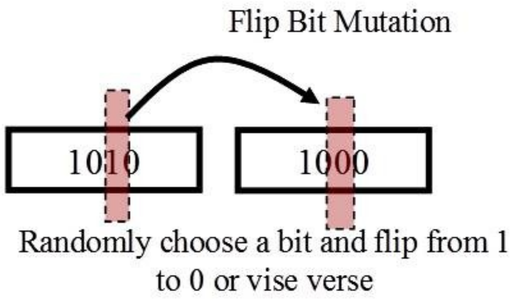


Figure 3.4 Flip bit mutation method

3.2.4 Fitness Function

Fitness function is the objective function for genetic algorithm to determine the optimization direction for maximizing the fitness value such that the user defined performance goal can be reached. Several different methods such as weight sum, weight product and utopia method which are commonly used in GA multi-objective optimization have been discussed in [20] and each method is suitable for different scenario. For our multi-layer DRA optimization, the fitness function is defined to equally emphasis on the return loss, pattern, gain, and phase center. The fitness function U is defined as the product of individual fitness function $F_i(x)$. That is,

$$U = \prod_{i=1}^{k} [F_i(x)] \qquad (3.1)$$

Each individual fitness function is defined as a function of specific performance parameters such as S_{11}, VSWR, gain, patter, etc. and can be defined differently according to the problem and use. In general, when the performance is close to performance goal, a fitness function produces a larger value. The optimization procedure will be determined as converge when variation of the maxima fitness values in constructive three generations is less than 5%.

3.3 External GA Optimization Program

Figure 3.5 shows the connection between the master GA and the slave HFSS. In the beginning, all variables are set up in a HFSS model and external GA program. Then, the external program can generate suitable VBScript file which can be executed by HFSS and send to HFSS to perform EM analysis. After simulation, HFSS automatically export all the desired reports to justify the fitness value.

Figure 3.5 also illustrates the external program that is composed of GA and simulation control parts. The GA part performs the procedures of chromosome generating, mutation and crossover. The simulation control part generates the VBScript files for the HFSS models, reads in the simulation results, and calculates fitness values of the performance. The flexibility for the design optimization can be applied to all kinds of antennas optimization problems by slightly changing the coding in HFSS control section.

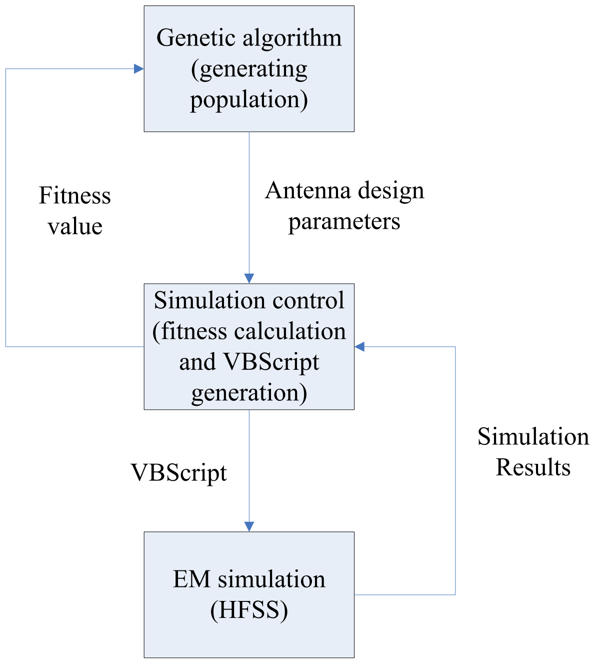

Genetic algorithm (generating population)
Fitness value
Antenna design parameters
Simulation control (fitness calculation and VBScript generation)
Simulation Results
VBScript
EM simulation (HFSS)

Figure 3.5 Flowchart of external GA program

3.4 Optimization Example

This section provides a design optimization example which optimizes the thickness and dielectric constant of a four-layer DRA with hemi-spherical radiation tip to achieve

(1) 8:1 impedance bandwidth and (2) constant gain performance. The cross section of this DRA geometry is shown in Figure 3.6. This antenna is fed by a waveport at the bottom of the waveguide section. There are 8 design parameters (4 dielectric constants and 4 thicknesses). The each dielectric constant is digitized into 3 bits and each thickness is digitized into 4 bits. The upper and lower bounds for each parameter are presented in the Table 3.1. The crossover probability is 0.9 and the mutation probability is 0.1. The fitness function for the optimization is described as

$$\overline{S_{11}} = \frac{1}{f_H - f_L} \sum_{f_L}^{f_H} |S_{11}(f)| \quad (3.2)$$

$$F_1(S_{11}) = \frac{\overline{S_{11}}}{\sqrt{\frac{1}{(f_H - f_L)} \sum_{f=f_L}^{f_H} \left[|S_{11}(f)| - \overline{S_{11}} \right]^2}} \quad (3.3)$$

$$\overline{D} = \frac{1}{f_H - f_L} \sum_{f_L}^{f_H} Dir_{Zenith}(f) \quad (3.4)$$

$$F_2(Dir) = \frac{\overline{D}}{\sqrt{\frac{1}{(f_H - f_L)} \sum_{f=f_L}^{f_H} \left[Dir_{Zenith}(f) - \overline{D} \right]^2}} \quad (3.5)$$

$$U = F_1(S_{11}) \times F_2(Dir) \quad (3.4)$$

, which targets at flat zenith directivity with good S_{11} performance.

	Bits	***Max(in)***	***Min(in)***		***Bits***	***Max(in)***	***Min(in)***
ε_1	3	9	2	**t_1**	4	0.5	0.05
ε_2	3	9	2	**t_2**	4	0.5	0.05
ε_3	3	9	2	**t_3**	4	0.5	0.05
ε_4	3	9	2	**t_4**	4	0.5	0.05

Table 3.1 Parameter bounds for the optimization design example

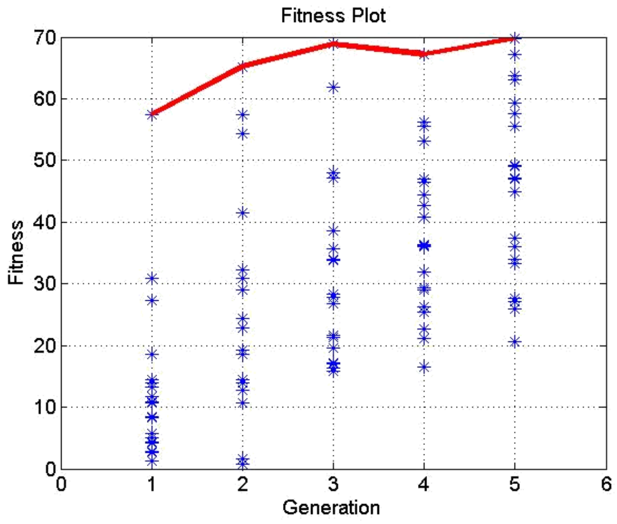
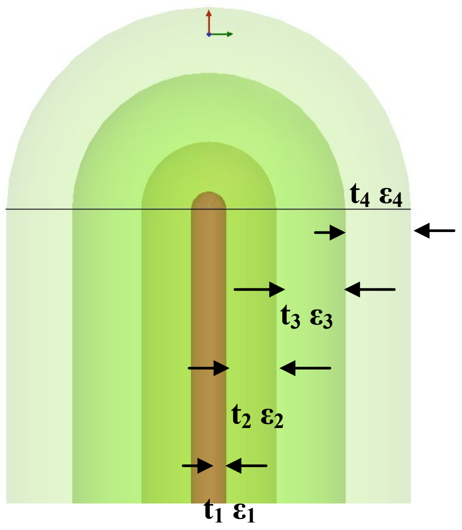


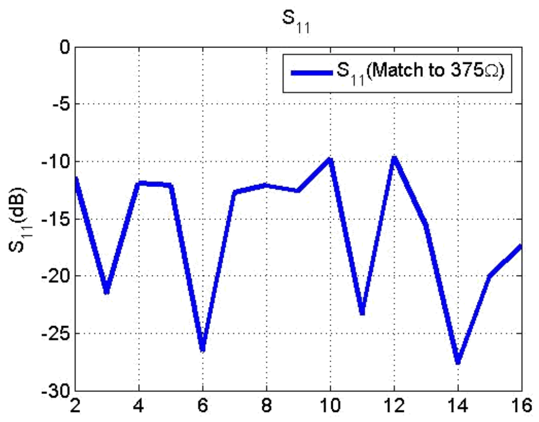
Figure 3.6 Convergence plot of optimization design example

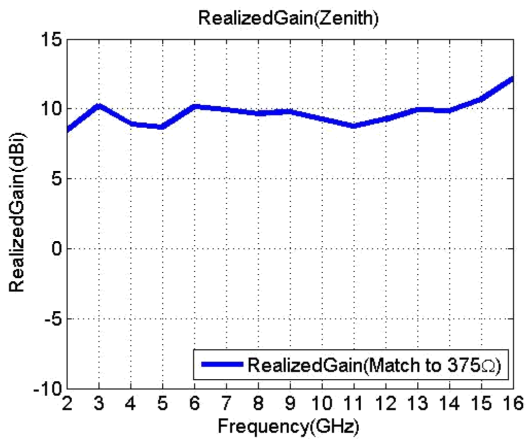
Figure 3.6 shows the convergence curve of the optimization procedure and presents a good convergence rate. Figure 3.7 demonstrates the side view of optimum design and the optima parameters. Figure 3.8 shows the optimized results of this optimization procedure. It is observed that the S_{11} remains -10dB and the zenith gain remains fairly stable from 2 to 16GHz. Figure 3.9 plots the Normalized Gain patterns which indicate similar E- and H-plane patterns at all frequencies. However, the current optimization design is not able control the pattern beamwidth because the fitness function doesn't include pattern shape parameter. This will be added in the next chapter.



t_1	t_2	t_3	t_4
0.127”	0.34”	0.47”	0.447”
ε_{r1}	ε_{r2}	ε_{r3}	ε_{r4}
8	4	3	2

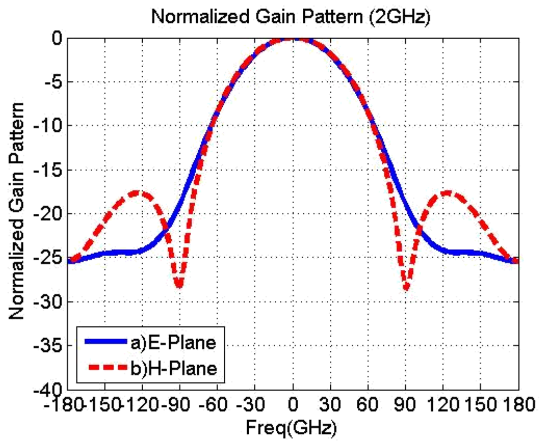
Figure 3.7 Geometry and design variables of optimization example



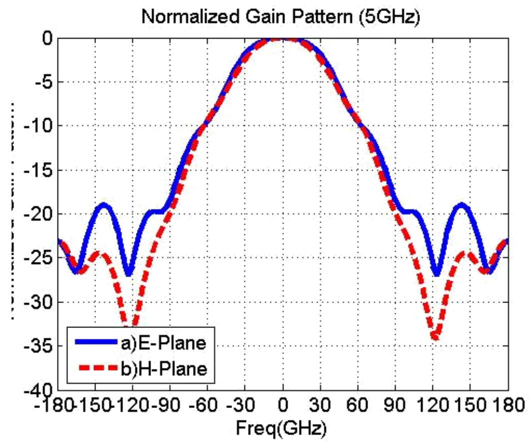


(a) S_{11} (b) Zenith Gain

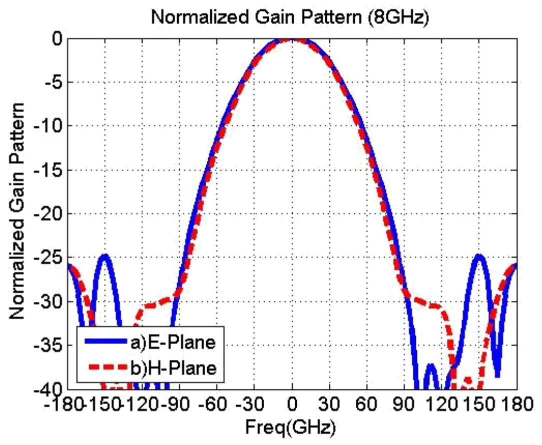
Figure 3.8 Simulation results of optimization example



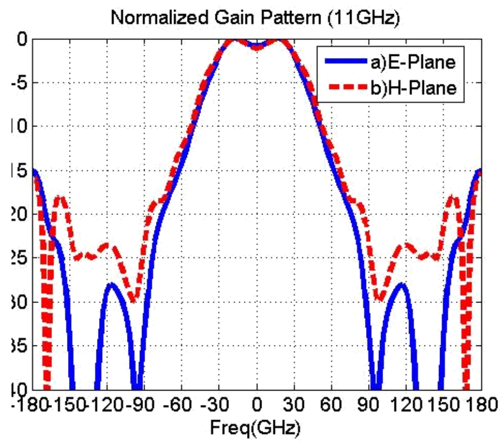
(a) f = 2GHz



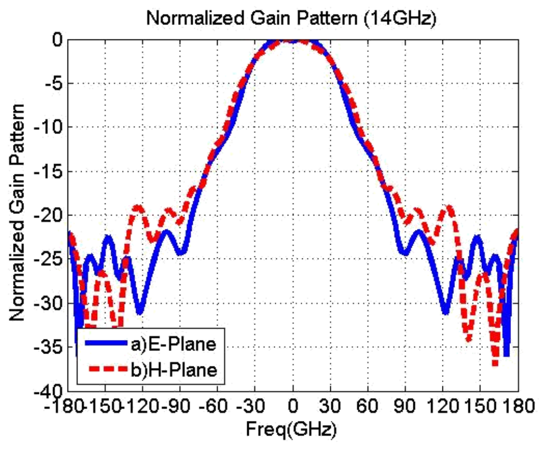
(b) f= 5GH



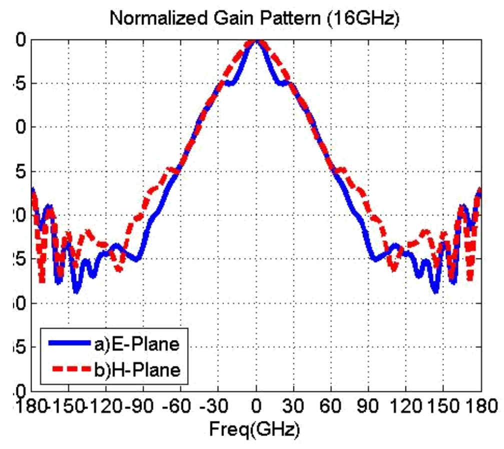
(c) f = 8GHz



(d) f= 11GHz



(e) f = 14GHz



(f) f= 16GHz

Figure 3.9 E and H-plane Normalized Gain patterns

CHAPTER 4

Application of Optimization Method for Designing UWB Multi-Layer DRA with Constant Gain, Pattern, Impedance, and Phase Center

In the previous chapters, we have discussed the design concept of a multi-layer DRA, and the design optimization procedure using GA and full-wave simulations. The chapter focuses on the two different design optimizations of multi-layer DRA which provide constant gain, stationary phase center, symmetric E- and H-plane pattern, more than 50 degrees HPBW and 100 degrees -10dB beamwidth with 2 to 18GHz and 1 to 18GHz bandwidth respectively.

Because the optimized results depend strongly on the fitness function, defining proper functions is the most important. We will present the pattern matching fitness function used in the optimization in the first section. An optimized three-layer DRA with 2 to 18GHz bandwidth and an optimized four-layer DRA with 1 to 18GHz bandwidth performance will also be presented respectively. Also, a practical feeding structure will be introduced in the launcher section to complete the antenna design.

4.1 Fitness Function with Pattern Matching Method

The fitness value of the optimization is defined by the performances of S_{11}, realized gain, and pattern matching.

$$F = F_S \times F_{PC} \times F_{TPM} \quad (4.1)$$

4.1.1 F_S

The fitness value for S_{11} performance is defined by

$$F_S = \frac{\bar{M}_S}{Std_S} \quad (4.2)$$

$$\bar{M}_S = \frac{1}{f_H - f_L}\left|\sum_{f_L}^{f_H} S(f)\right| \quad (4.3)$$

$$Std_S = \sqrt{\frac{\sum_{f_L}^{f_H}\left(S_{11}(f) - \bar{M}_S\right)^2}{f_H - f_L}} \quad (4.4)$$

$$\begin{cases} S(f) = -10 & if \quad S_{11}(f) < -10 \\ S(f) = S_{11}(f) & else \end{cases} \quad (4.5)$$

Figure 4.1 shows t between F_s and S_{11}(f) performance and demonstrates that F_s aims to reach $S_{11}(f) < -10dB$ stably.

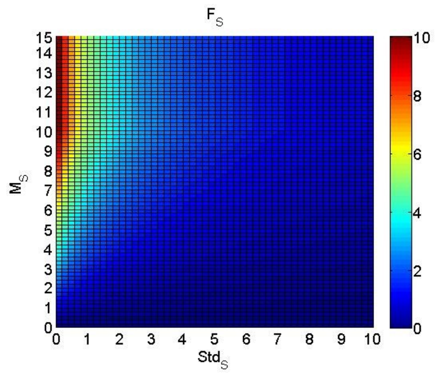


Figure 4.1 Relationship between F_s and S_{11}(f) performance

4.1.2 F_{PC}

F_{PC} aims to fix phase center location over operation frequency, and is defined as

$$F_{PC} = \mathrm{e}^{(-Std_{PC})} \qquad (4.6)$$

$$\bar{M}_{PC} = \frac{1}{f_H - f_L - 1}\left|\sum_{f_L+1}^{f_H} PhaseCenter(f)\right| \qquad (4.7)$$

$$Std_{PC} = \sqrt{\frac{\sum_{f_L+1}^{f_H}\left(PhaseCenter(f) - \bar{M}_{PC}\right)^2}{f_H - f_L - 1}} \qquad (4.8)$$

Figure 4.2 shows the relationship between F_{PC} and standard deviation of phase center Std_{PC}. F_{PC} value drops rapidly when Std_{PC} increases, and enforces GA to search for stable phase center results.

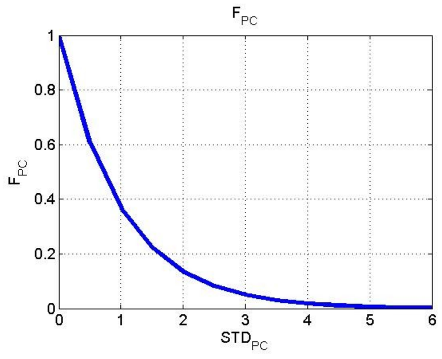


Figure 4.2 Relationship between F_{PC} with standard deviation of phase center Std_{PC}

4.1.3 F_{TPM}

F_{TPM} is design such that the normalized patterns conform to a user-defined target pattern as illustrated in Figure 4.3. This is achieved by adopting

$$F_{TPM} = \frac{1}{(1+\bar{M}_{\Delta P})\times(1+Std_{\Delta P})} \quad (4.9)$$

$$\Delta P(f) = \underset{\theta}{Max}\left|Pattern_{simulation}(f) - P_{target}\right| \quad (4.10)$$

$$\bar{M}_{\Delta P} = \frac{1}{f_H - f_L}\sum_{f_L}^{f_H}\Delta P(f) \quad (4.11)$$

$$Std_{\Delta P} = \sqrt{\frac{\sum_{f_L}^{f_H}\left(\Delta P(f) - \bar{M}_{\Delta P}\right)}{f_H - f_L}} \quad (4.12)$$

The choice of target pattern needs to be realistic and achievable since it controls the convergence of the optimization procedure. Hence, to ensure the realistic of the target pattern, the target pattern is obtained from the average of E- and H-plane pattern of 2GHz shown in the Figure3.9 (a). The pattern beamwidth reaches 60 degrees HPBW and 110 degrees -10dB beamwidth which fits to the design goals.

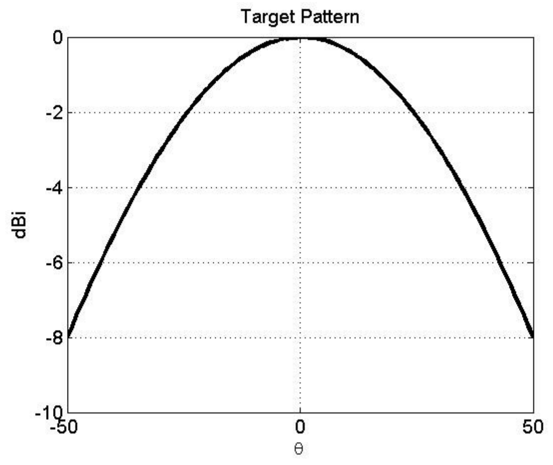


Figure 4.3 Target pattern for optimization

4.2 Three-Layer UWB DRA Optimization

The optimization starts with a three-layer DRA and figure 4.4 shows the initial geometry. Table 4.1 presents the upper bound, lower bound, and digitalization of all design parameter. The dielectric constant of the first layer (ε_{r3}) is set to 2 to reduce number of variables. The dielectric constant for other layers are determined by

$$\varepsilon_2 = \varepsilon_3 + \Delta_2 \quad \varepsilon_1 = \varepsilon_2 + \Delta_1 \qquad (4.13)$$

The elliptical radiation tip has included three axial ratios (P_1,P_2,P_3) for controlling patterns. The relation tip lengthy of each layer (h_1,h_2,h_3) is

$$h_n = \left(\sum_{i}^{n} t_i \right) P_n \qquad (4.14)$$

The mutation rate is 0.1, the crossover rate is 0.9 and each generation contains 20 populations. And the target pattern has shown in the Figure 4.3.

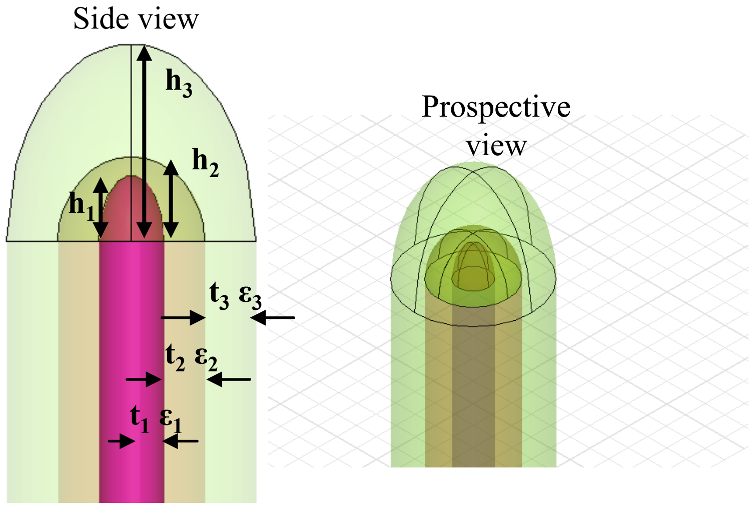


Figure 4.4 Initial geometry of three-layer DRA design

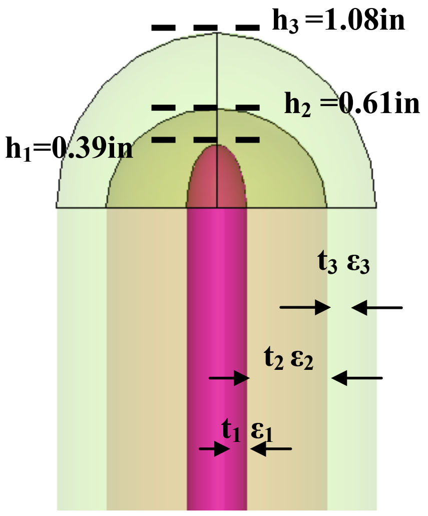
	Bits	***Max***	***Min***		***Bits***	***Max(in)***	***Min(in)***
$\mathbf{\Delta_1}$	2	4	1	$\mathbf{t_1}$	4	0.6	0.1
$\mathbf{\Delta_2}$	2	4	1	$\mathbf{t_2}$	4	0.6	0.1
P_1	3	2.5	0.5	$\mathbf{t_3}$	4	0.6	0.1
P_2	3	2	0.5				
P_3	3	2	0.5				

Table 4.1 Parameter bounds for the three-layer DRA design

Figure 4.5 shows the optimized three-layer DRA geometry and design parameters and it's worthy to note that the ratio between the dielectric constants of adjacent layers low. In fact, it shows that the ratio is always less than 2. This indicates that the ratio of dielectric constant between adjacent layer needs to be lower than 2 to maintain low internal reflections and field distortions. Figure 4.6 shows that the GA optimization procedure converges after 8 iterations.

The antenna performances of the optimized three-layer DRA design are shown in Figure 4.7. Figure 4.7 (b) shows the maxima directivity is located between 9.5 to 10.5dB within 2-18GHz. The phase center remains stable from 4GHz. Note from (4.9) that F_{PC} only considers frequency above 3GHz to provide more miniaturization since the EM energy is only loosely guided along the rod at 2-3GHz. This also explains the S_{11} results in 2-3GHz region. And the problem can simply be solved the exemption from the fitness function, and allow for larger maximum rod diameter. Figure 4.7 (d) shows the HPPW of E- and H-plane. This result is much improved compared to the design example in previous chapter due to the new fitness F_{TPM} based on pattern matching.

The normalized gain patterns shown in Figure 4.8-4.10 demonstrate all patterns conform to target pattern very well, all E- and H-plane patterns are similar. These patterns achieve more than 50 degrees HPBW and 100 degrees -10dB beamwidth. The E-field plots have been presented in Figure 4.11, and it' obviously that the fields are naturally expending from 2:18GHz. Moreover, it's clear that the field is well-guided inside the waveguide section and properly radiated at the radiation section.



Parameters of Optimized Result

ϵ_1	ϵ_2	ϵ_3
6	3	2
t_1(in)	t_2(in)	t_3(in)
0.19	0.49	0.3
P_1	P_2	P_3
2.1	0.9	1.1

Figure 4.5 Geometry and design parameters of optimized three-layer DRA

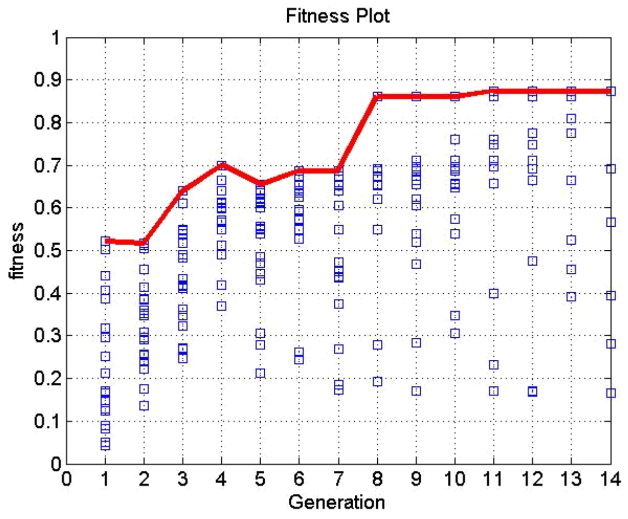
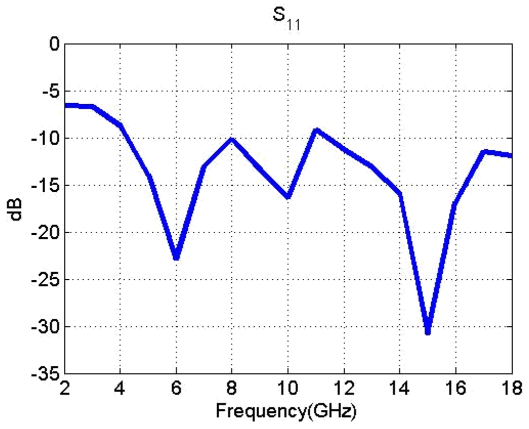
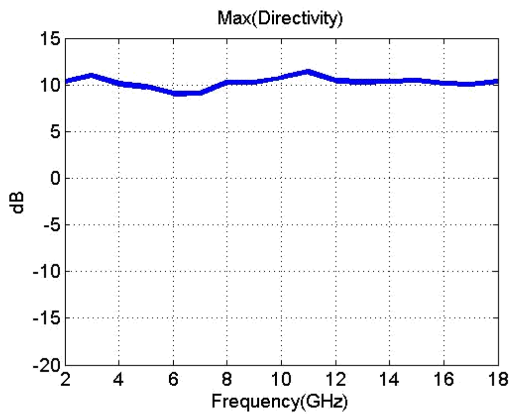


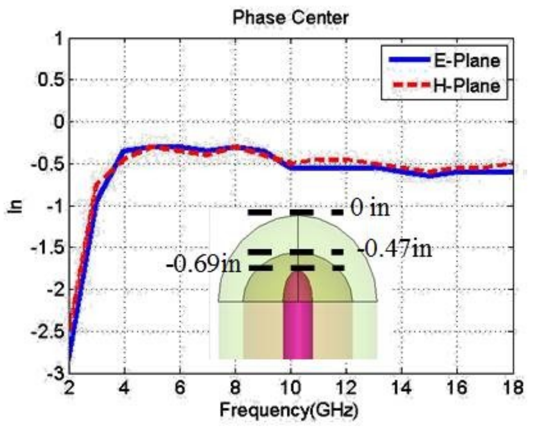
Figure 4.6 GA convergence plot



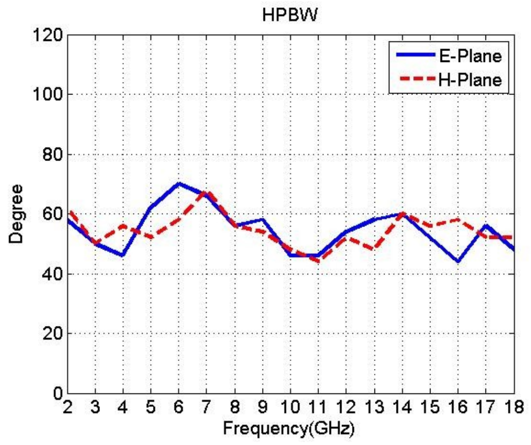
(a) S_{11}



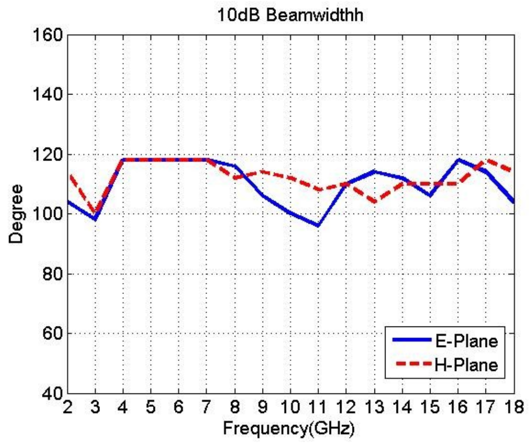
(b) Maxima Directivity



(c) Phase Center

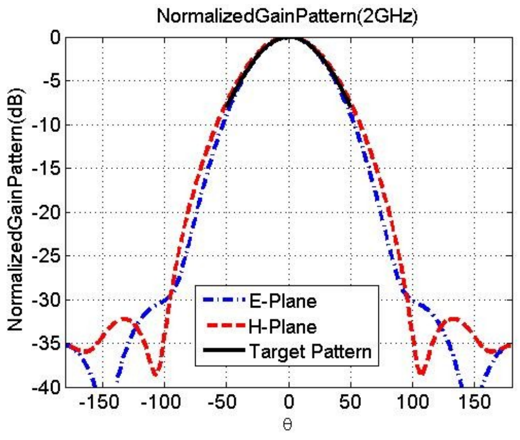


(d) HPBW

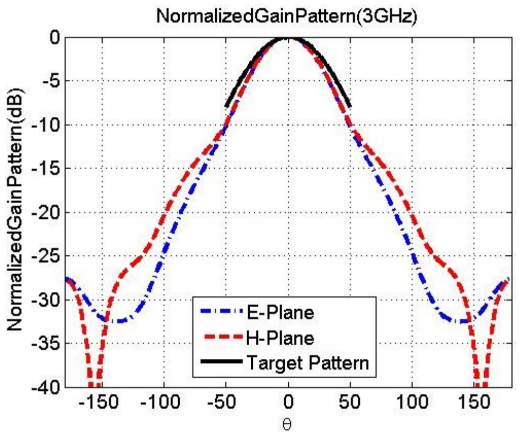


(e) -10dB Beamwidth

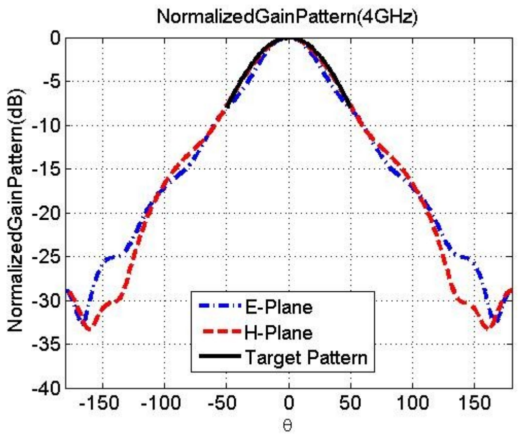
Figure 4.7 Simulation results of optimized three-layer DRA



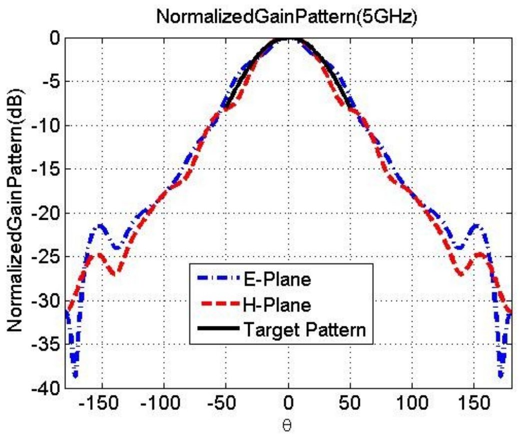
(a) f = 2GHz



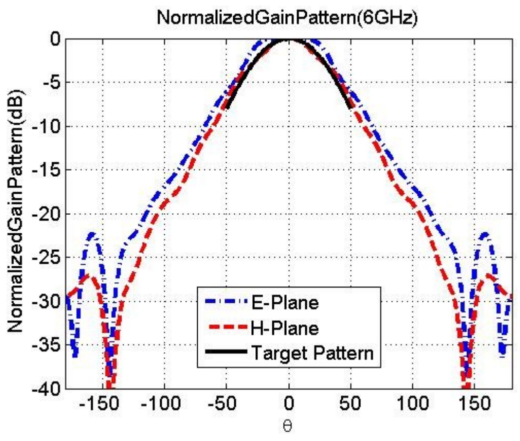
(b) f = 3GHz



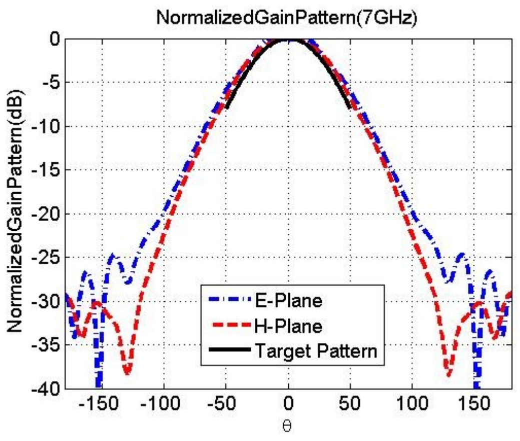
(c) f = 4GHz



(d) f = 5GHz

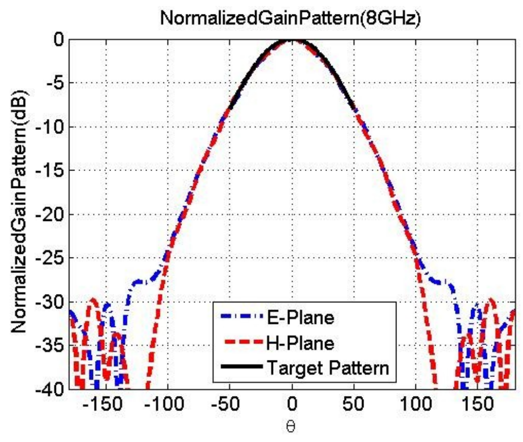


(e) f = 6GHz

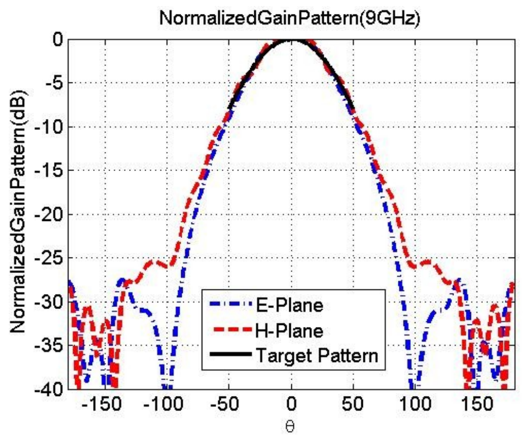


(f) f = 7GHz

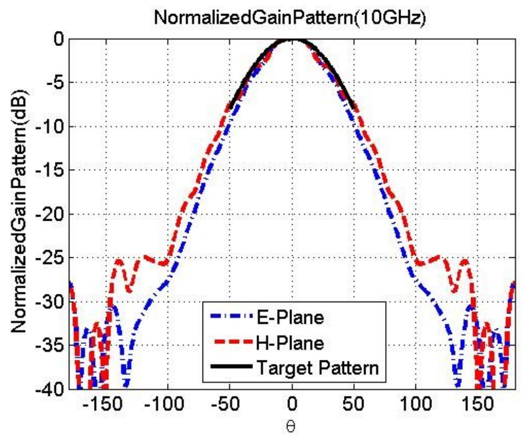
Figure 4.8 Normalized Gain patterns of optimized three-layer DRA(1)



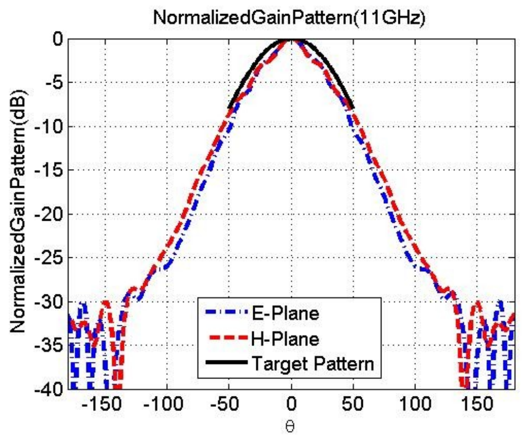
(g) f = 8GHz



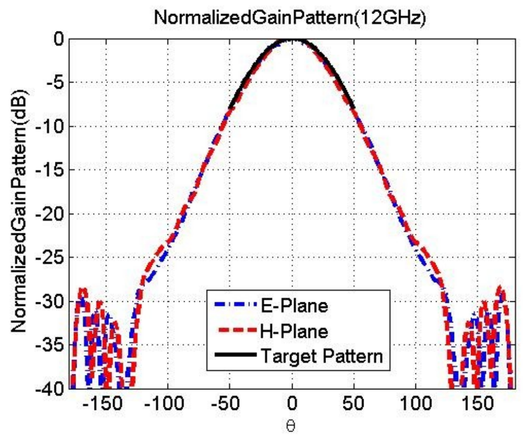
(h) f = 9GHz



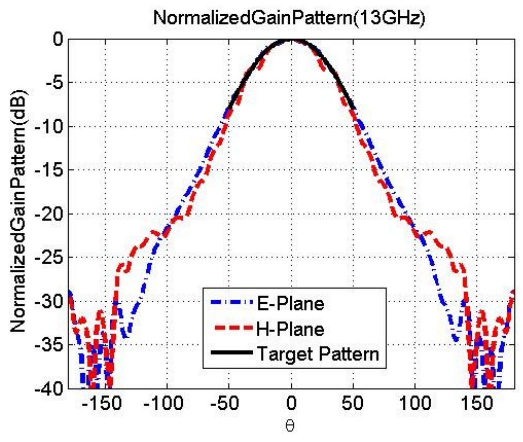
(i) f = 10GHz



(j) f = 11GHz

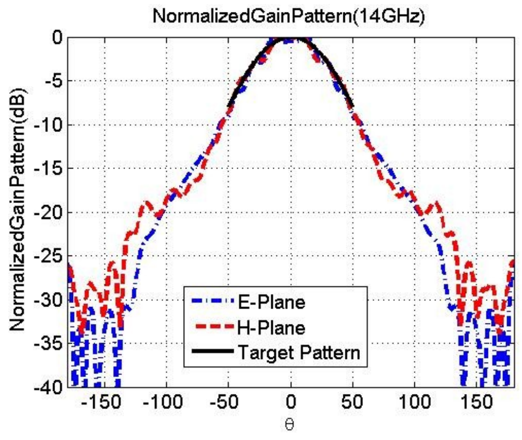


(k) f = 12GHz

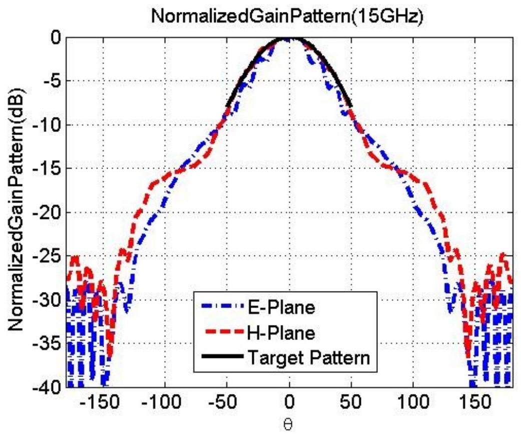


(l) f = 13GHz

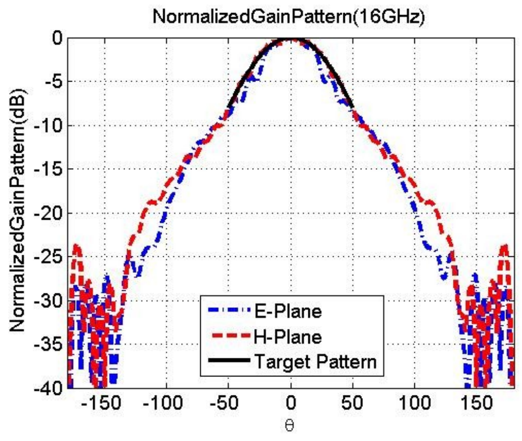
Figure 4.9 Normalized Gain patterns of optimized three-layer DRA (2)



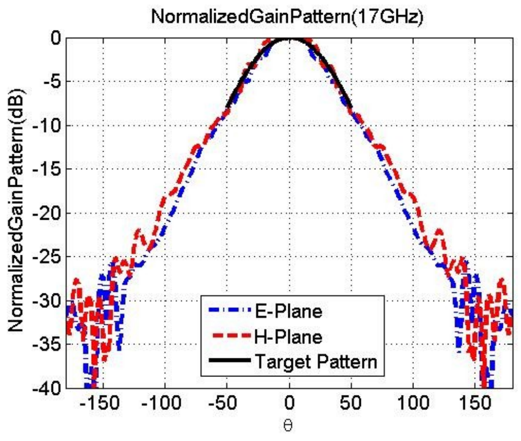
(m) f = 14GHz



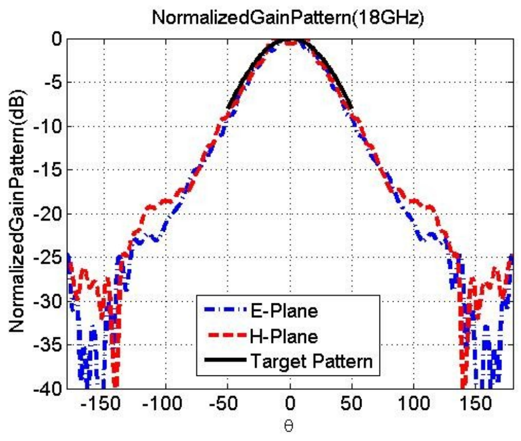
(n) f = 15GHz



(o) f = 16GHz

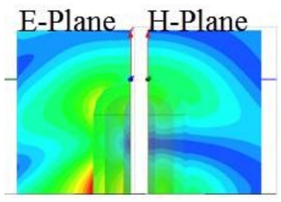


(p) f = 17GHz

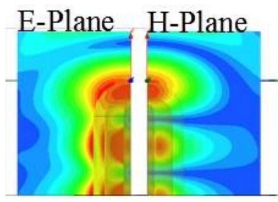


(q) f = 18GHz

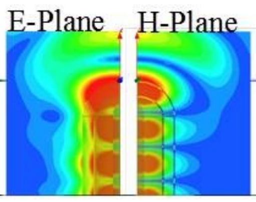
Figure 4.10 Normalized Gain patterns of optimized three-layer DRA (3)



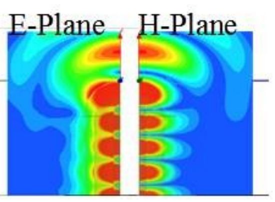
(a) f = 2GHz



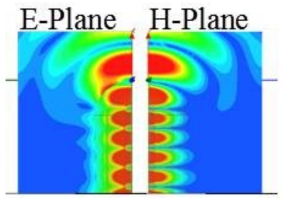
(b) f = 3GHz



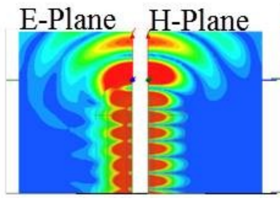
(c) f = 4GHz



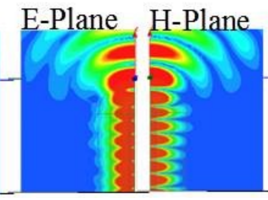
(d) f = 5GHz



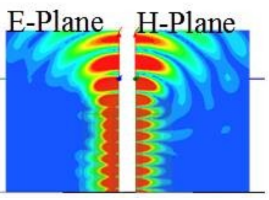
(e) f = 6GHz



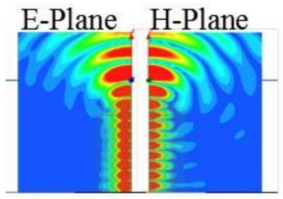
(f) f = 7GHz



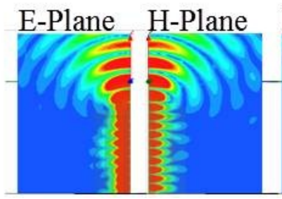
(g) f = 8GHz



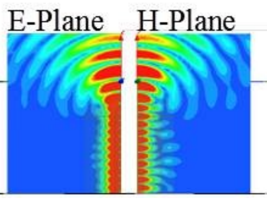
(h) f = 9GHz



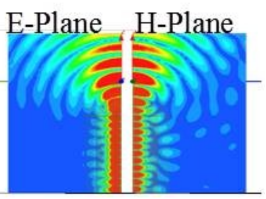
(i) f = 10GHz



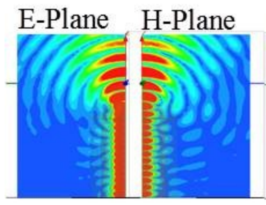
(j) f = 11GHz



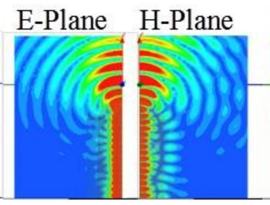
(k) f = 12GHz



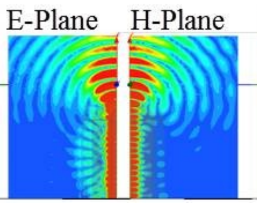
(l) f=13GHz



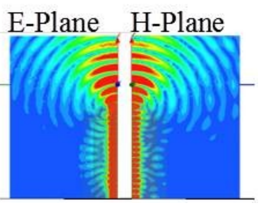
(m) f = 14GHz



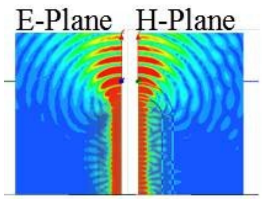
(n) f = 15GHz



(o) f = 16GHz



(p) f = 17GHz



(q) f = 18GHz

Figure 4.11 E-filed plot at E- and H-plane of the optimized three-layer DRA

4.3 Four-Layer UWB DRA Optimization

4.3.1 Optimizing with Existing Fitness Function

The GA optimization procedure has proven the compatibility of optimizing a three-layer UWB DRA which achieves 2 to 18GHz bandwidth, flat gain, stable pattern, and stationary phase center. In order to further verify the ability of the automatic optimization procedure, this section will employ the optimization program to optimize a four-layer DRA which reaches flat gian, stable pattern, stationary phase center and 1 to 18GHz bandwidth.

The initial geometry of four-layer DRA has been show in Figure 4.12. Table 4.2 presents the upper bound, lower bound, and digitalization of all design parameter. The dielectric constant of the first layer (ε_{r4}) is set to 2.2 here because it's easier to find the material with dielectric constant 2.2. The dielectric constant for other layers are determined by

$$\varepsilon_3 = \varepsilon_4 + \Delta_3 - 0.2 \quad \varepsilon_2 = \varepsilon_3 + \Delta_2 \quad \varepsilon_1 = \varepsilon_2 + \Delta_1 \qquad (4.17)$$

The elliptical radiation tip also includes four axial ratios (P_1,P_2,P_3,P_4) for controlling patterns. The mutation rate is 0.1, the crossover rate is 0.9 and each generation contains 20 populations. And the target pattern is same as Figure 4.3

	Bits	***Max***	***Min***		***Bits***	***Max(in)***	***Min(in)***
Δ_1	2	4	1	**t_1**	4	0.2	0.05
Δ_2	2	4	1	**t_2**	4	0.3	0.1
Δ_3	2	4	1	**t_3**		0.6	0.1
P_1	3	6	2	**t_4**	4	0.6	0.1
P_2	3	2.5	0.5				
P_3	3	2	0.5				
P_4	3	2	0.5				

Table 4.2 Parameter bounds for the four-layer DRA design

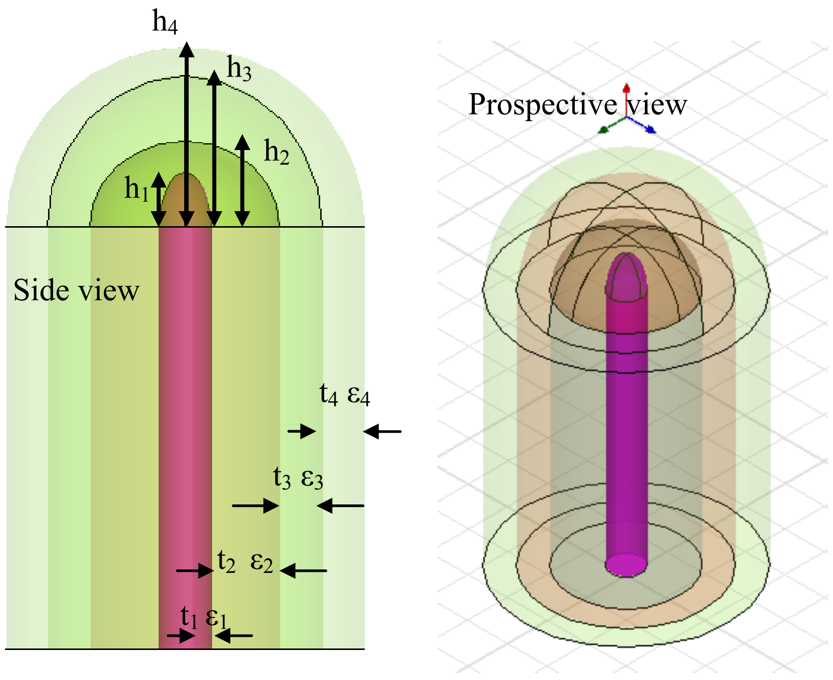
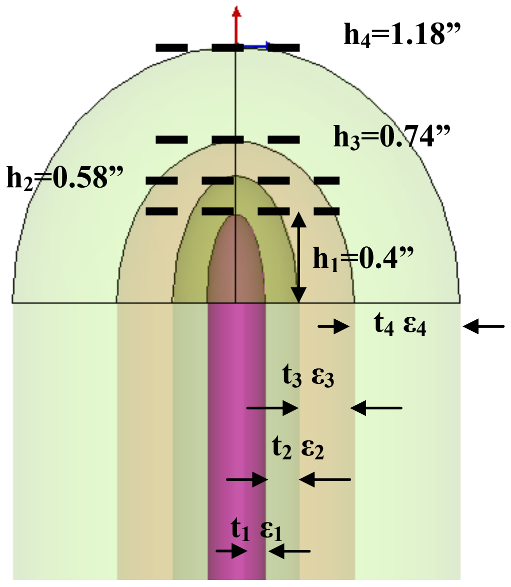


Figure 4.12 Initial geometry of four-layer DRA design



Parameters of Optimized Result

ε_1	ε_2	ε_3	ε_4
6	5	3	2.2
t$_1$	**t$_2$**	**t$_3$**	**t$_4$**
0.135”	0.157”	0.257”	0.485”
P$_1$	**P$_2$**	**P$_3$**	**P$_4$**
3	2	1.35	1.14

Figure 4.13 Geometry and design parameters of optimized four layer DRA

Figure 4.13 shows the optimized three-layer DRA geometry and design parameters and he ratio between the dielectric constants of adjacent layers is also lower than 2. Figure 4.14 shows the convergence plot of the GA optimization procedure. Though the design parameters are more than the three-layer design, the optimization program still performs a good convergence rate and converges in ten generations.

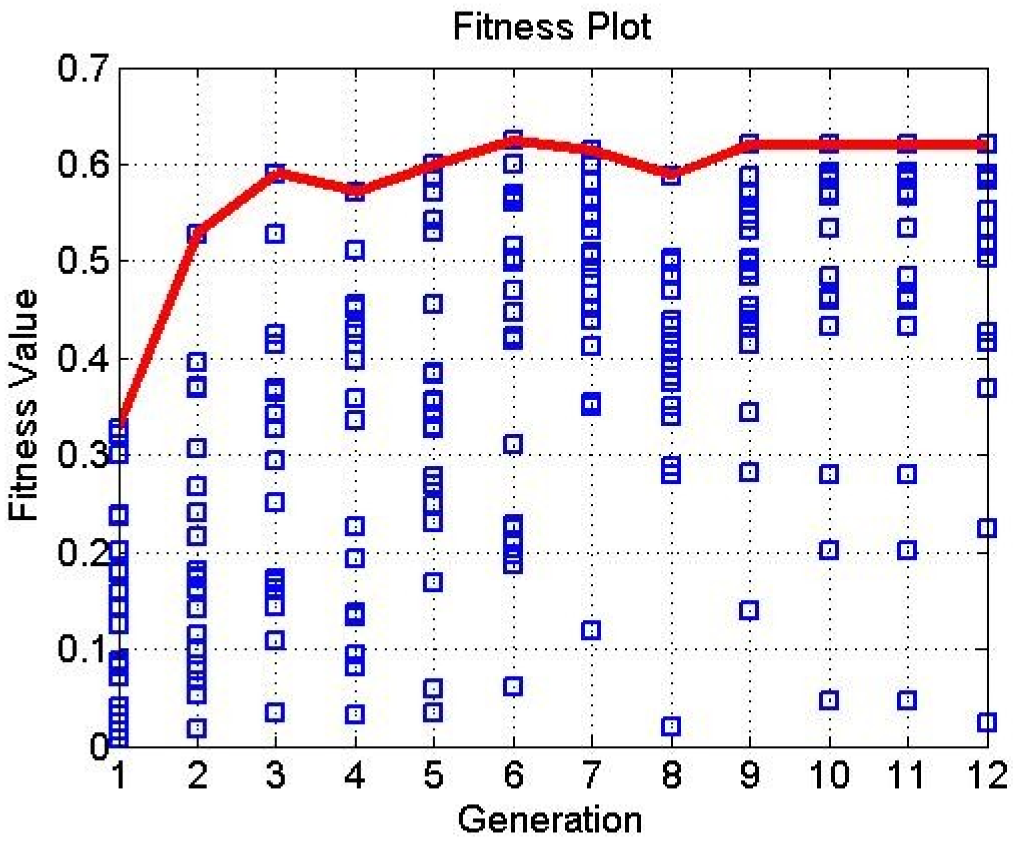
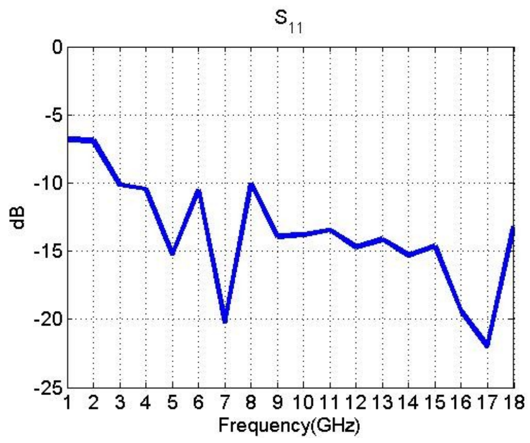


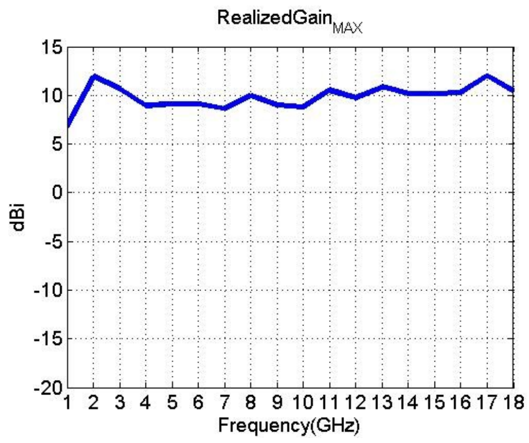
Figure 4.14 Convergence plot of four-layer DRA optimization

The antenna performances of the optimized four-layer DRA design are shown in Figure 4.15. Figure 4.15 (b) shows the maxima realized is lower 11dB within 1-18GHz. Since the fitness function (4.9) doesn't take the phase center at lowest frequency in to calculation for more miniaturization, it's expected that the phase center (Figure 4.7 (c)) becomes stable after 3GHz. It also results in the S_{11} performance from 1-2GHz region because the fields are loosely guided in this frequency range. And, as mentioned in previous section, this issue can be solved the exemption from the fitness function, and allow for larger maximum rod diameter. The HPBW result shown in the Figure 4.15(d) presents a stable HPBW pattern and reaches at least 50 degrees HPBW at most of the frequencies.

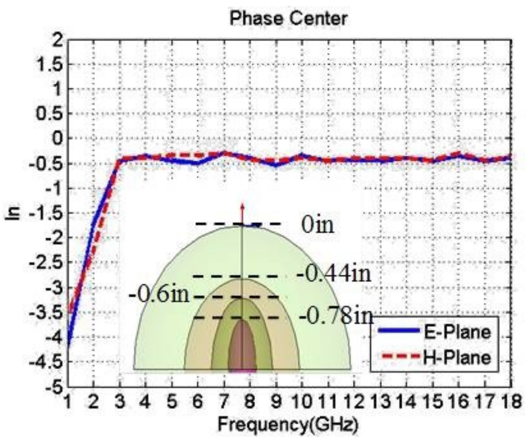
The Normalized Gain patterns shown in Figure 4.16-4.18 also demonstrate that all patterns conform to target pattern, and all E- and H-plane patterns are similar as well. Most patterns achieve more than 50 degrees HPBW and 100 degrees -10dB beamwidth. The E-field plots have been presented in Figure 4.19, and it clearly states that the fields are naturally expending from 1:18GHz. It also demonstrates the fields are loosely guided from 1to 2GHz.



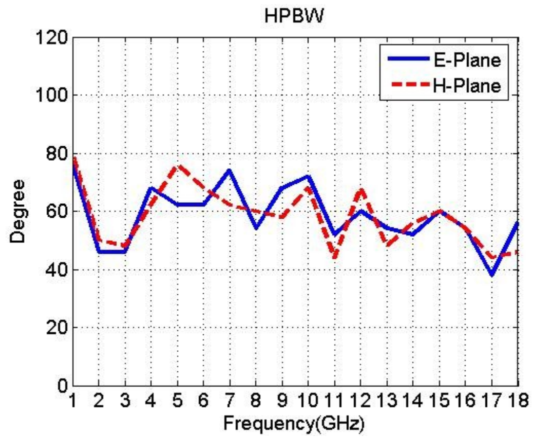
(a) S11



(b) RealizedGain

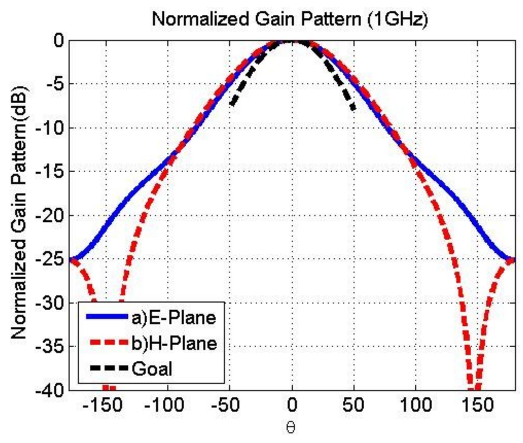


(c) Phase Center

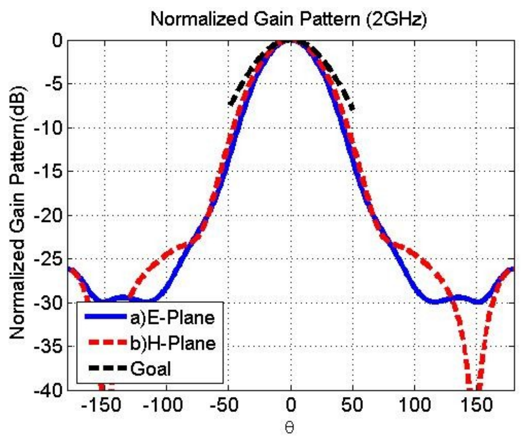


(d) HPBW

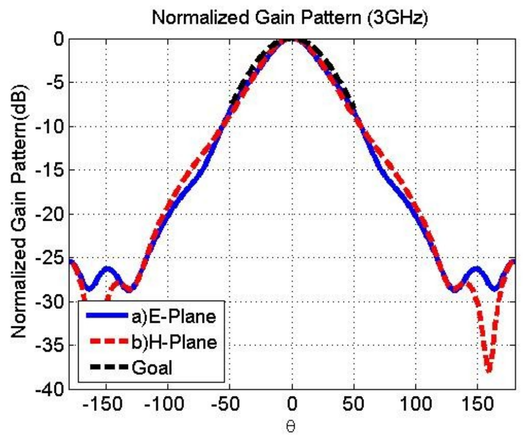
Figure 4.15 Simulation results of optimized four-layer DRA



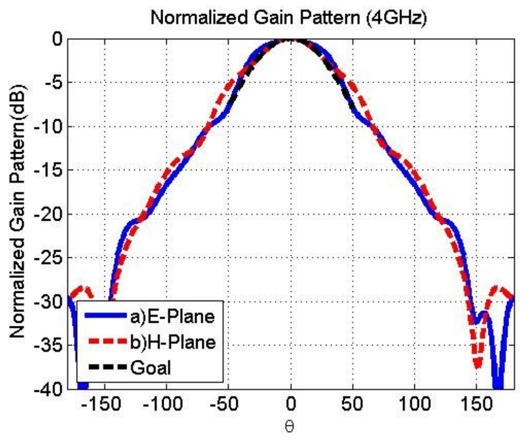
(a) f = 1GHz



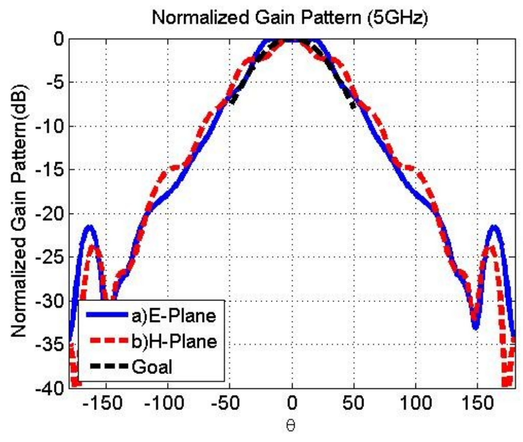
(b) f = 2GHz



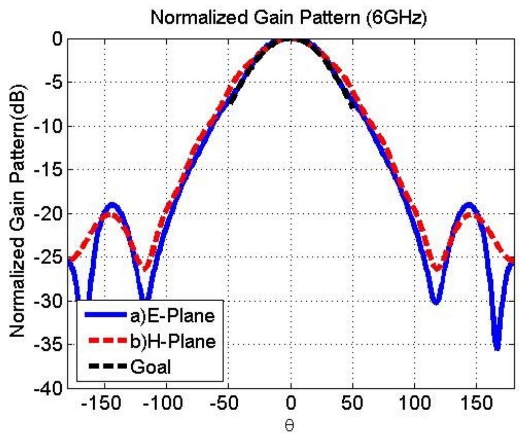
(c) f = 3GHz



(d) f = 4GHz

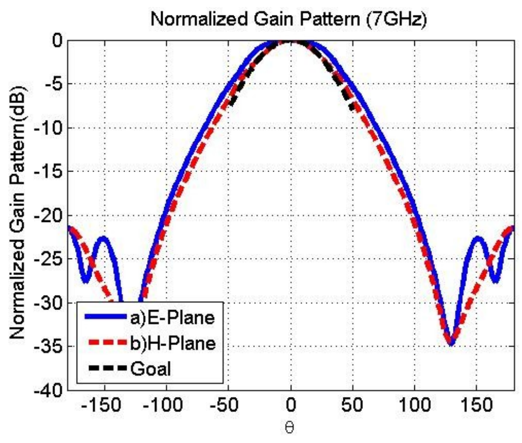


(e) f = 5GHz

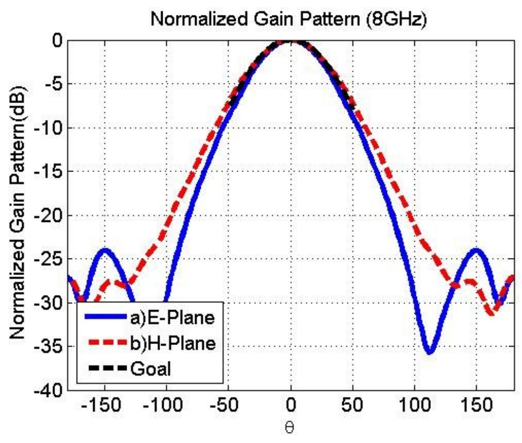


(f) f = 6GHz

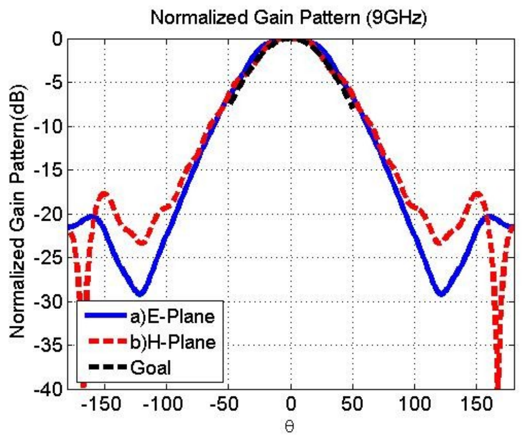
Figure 4.16 Normalized gain pattern of the optimized four-layer DRA (1)



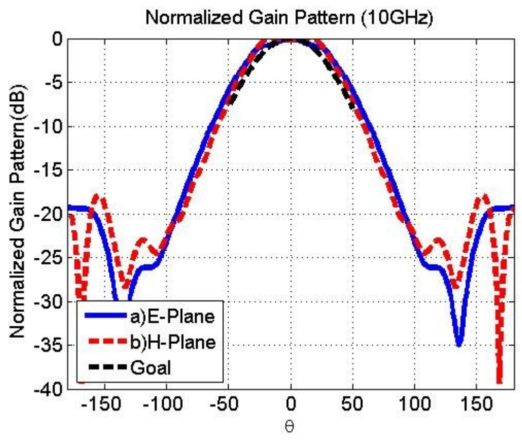
(g) f = 7GHz



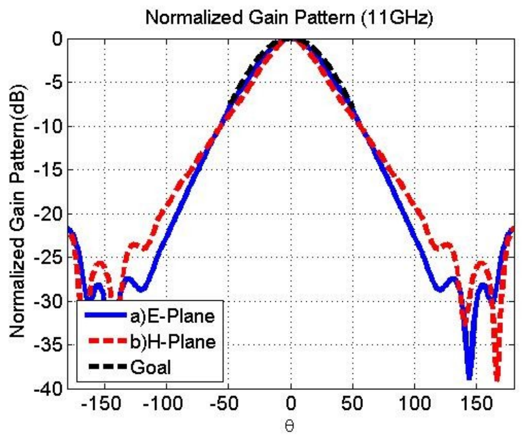
(h) f = 8GHz



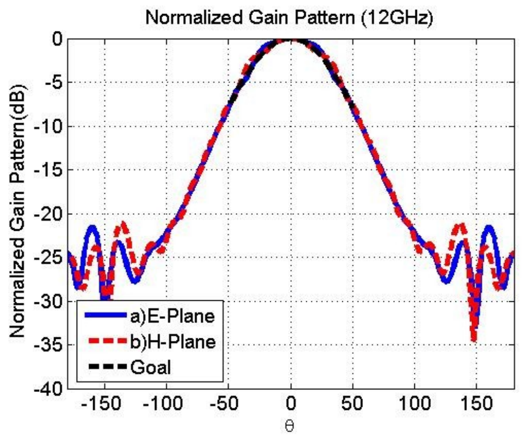
(i) f = 9GHz



(j) f = 10GHz

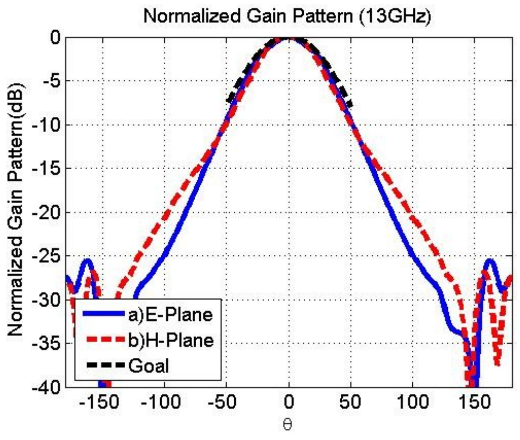


(k) f = 11GHz

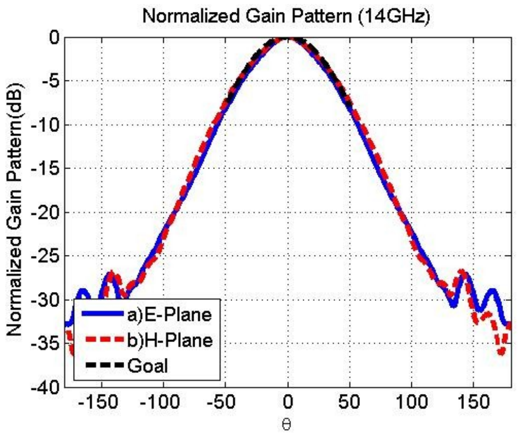


(l) f = 12GHz

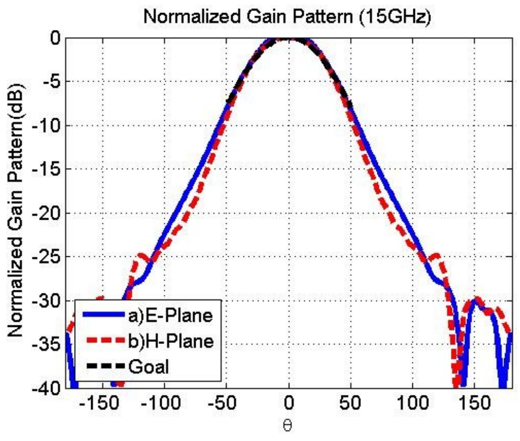
Figure 4.17 Normalized gain pattern of the optimized four-layer DRA (2)



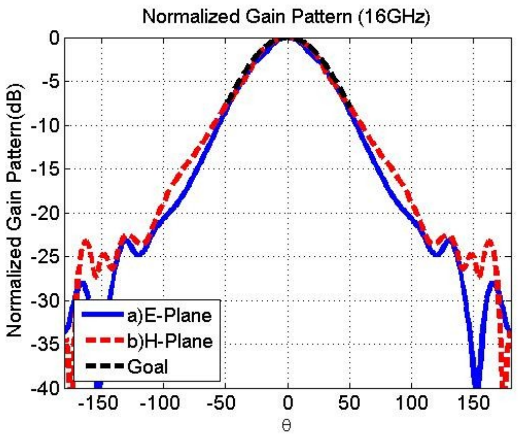
(m) f = 14GHz



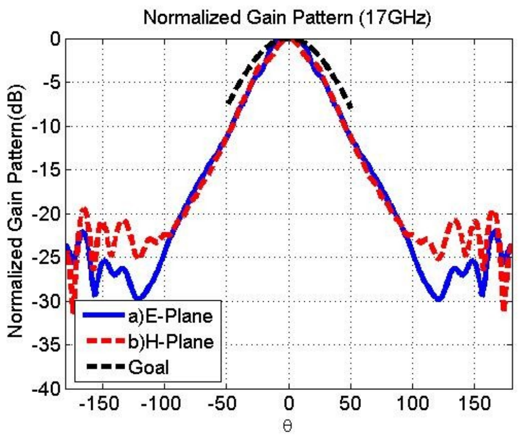
(n) f = 13GHz



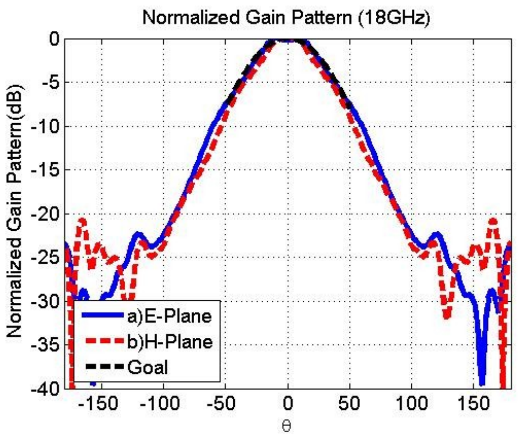
(o) f = 15GHz



(p) f = 16GHz

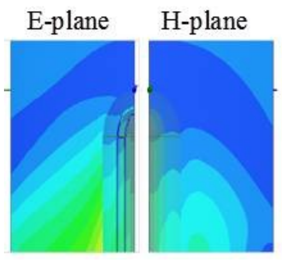


(q) f = 17GHz

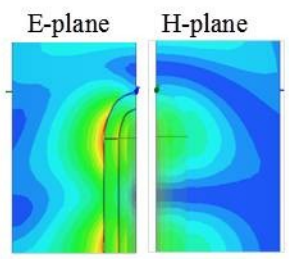


(r) f = 18GHz

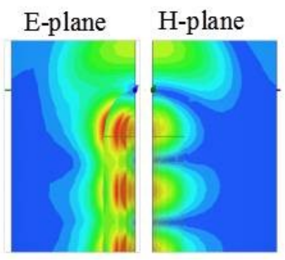
Figure 4.18 Normalized gain pattern of the optimized four-layer DRA (3)



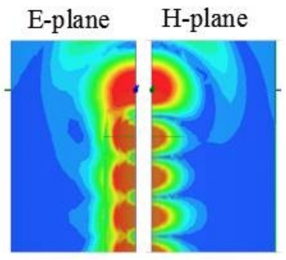
(a) 1GHz



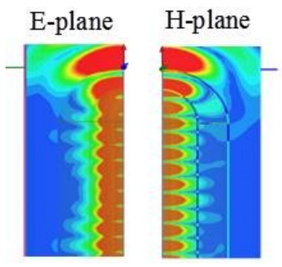
(b) 2GHz



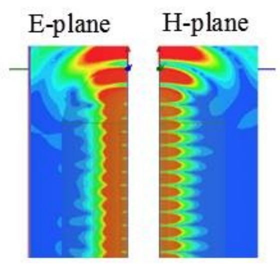
(c) 3GHz



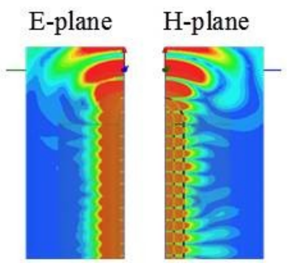
(d) 4GHz



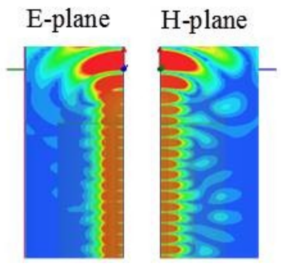
(e) 9GHz



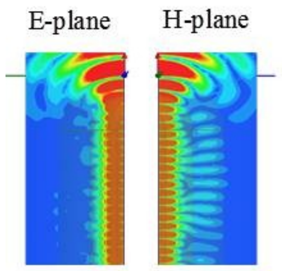
(f) 10GHz



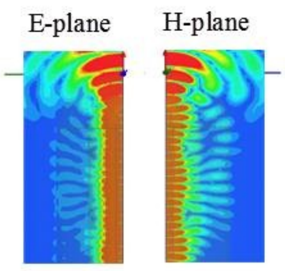
(g) 11GHz



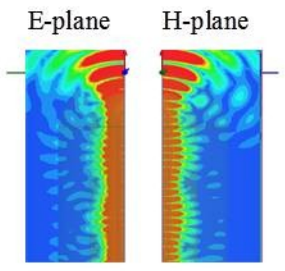
(h) 12GHz



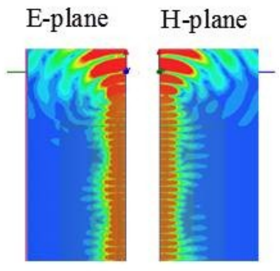
(i) 15GHz



(j) 16GHz



(k) 17GHz



(l) 18GHz

Figure 4.19 E-field at E- and H-plane of the optimized four-layer DRA

4.3.2 Optimizing without the Exemption in Phase Center Fitness Calculation

As mentioned above, both the optimized three- and four-layer designs have less stable phase center and slightly poor S_{11} performances at the very low frequencies due to the fact that the energies are loosely guided along the waveguide section. In order to mitigate this issue, we purpose to remove the exemption criteria in the fitness function F_{pc} (4.6)-(4.8) into

$$F_{PC} = \mathrm{e}^{(-Std_{PC})} \qquad (4.16)$$

$$\bar{M}_{PC} = \frac{1}{f_H - f_L}\left|\sum_{f_L}^{f_H} PhaseCenter(f)\right| \qquad (4.17)$$

$$Std_{PC} = \sqrt{\frac{\sum_{f_L}^{f_H}\left(PhaseCenter(f) - \bar{M}_{PC}\right)^2}{f_H - f_L - 1}} \qquad (4.18)$$

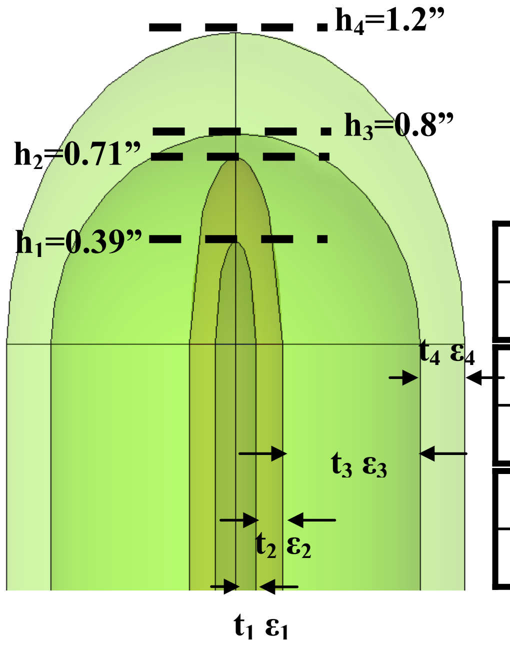
, and optimized another 4-layer DRA with the modified fitness function.

The design geometry of a 4-layer design has been shown in the Figure 4.10, and the parameter setting remains the same as discussed above. Table 4.3 shows the ranges of each parameter. Most of the upper and lower bounds remain the same as the previous 4-layer optimization to clearly compare the difference between two fitness function.

	Bits	***Max***	***Min***		***Bits***	***Max(in)***	***Min(in)***
Δ_1	2	4	1	**t_1**	4	0.25	0.05
Δ_2	2	4	1	**t_2**	4	0.4	0.1
Δ_3	2	4	1	**t_3**	4	0.6	0.1
P_1	3	6	1	**t_4**	4	0.6	0.1
P_2	3	4	0.75				
P_3	3	2	0.5				
P_4	3	2	0.5				

Table 4.3 Parameter bounds for the four-layer DRA design with modified F_{pc}

Figure 4.20 shows the optimized design parameters and geometry as well. Figure 4.21 presents the convergence rate of this optimization and it also shows a good convergence after 8 generations.



Parameters of Optimized Result

ε_1	ε_2	ε_3	ε_4
10	7	3	2.2
t_1	t_2	t_3	t_4
0.078”	0.1”	0.53”	0.174”
P_1	P_2	P_3	P_4
5	4	1.14	1.35

Figure 4.20 Geometry and parameters of optimized four-layer DRA with modified F_{pc}

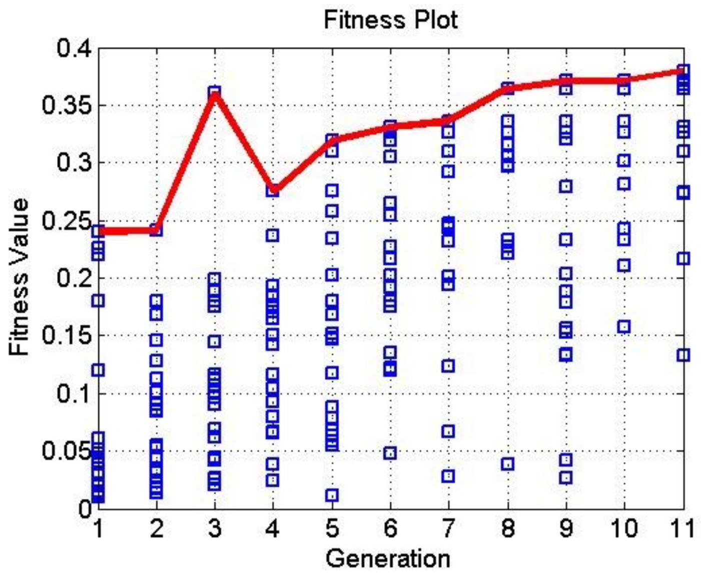
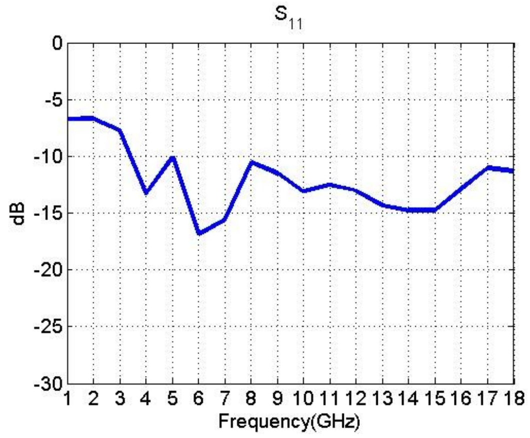


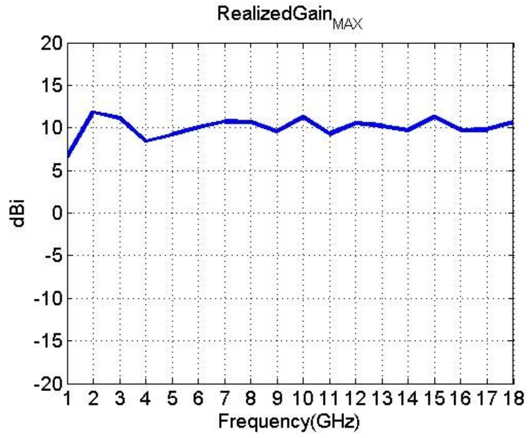
Figure 4.21 Convergence plot of 4-layer DRA optimization with modified F_{pc}

Figure 4.22 – 4.25 present the simulation results of the optimized 4-layer DRA design with modified F_{pc}. As one can tell, the S_{11} (Figure 4.22(a)), realized gain (Figure 4.22(b)), and HPBW (Figure 4.22(d)) performances are still similar to the previous optimized results and achieve the requirements. The normalized gain patterns (Figure 4.23-4.25) also conform to the target patter well. But the phase center performances (Figure 4.22(c)) only improve slightly compared to the previous results. The minor improvement demonstrates that the modified F_{pc} didn't strongly enforce the guidance at low frequency effectively because the fitness function is aimed for statically stable over the operation band. Moreover, because the phase center performances at higher frequencies are most likely to be stable, the fitness function is biased when it searches for statically stable over the whole band. In order to effectively improve the low frequency

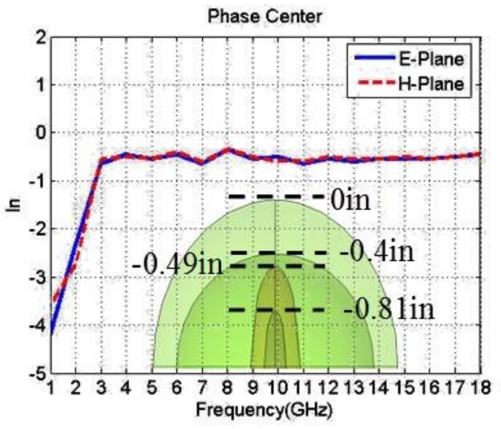
performance, it might need to separate the whole operation frequency range into several bands and enforce the performance at lower frequencies more.



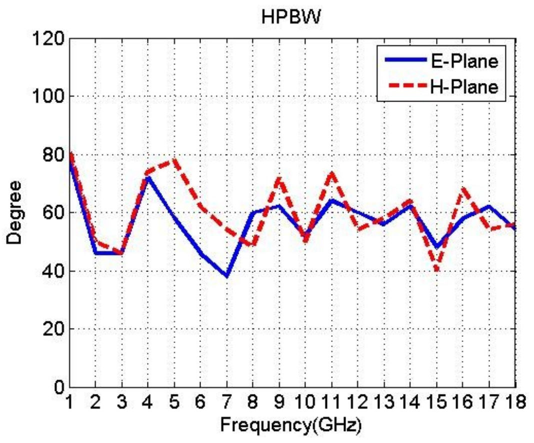
(a) S11



(b) RealizedGain

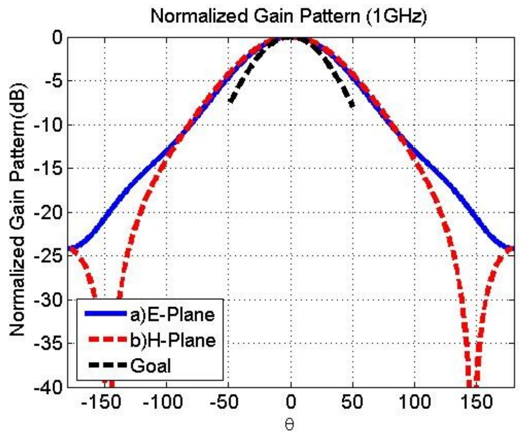


(c) Phase Center

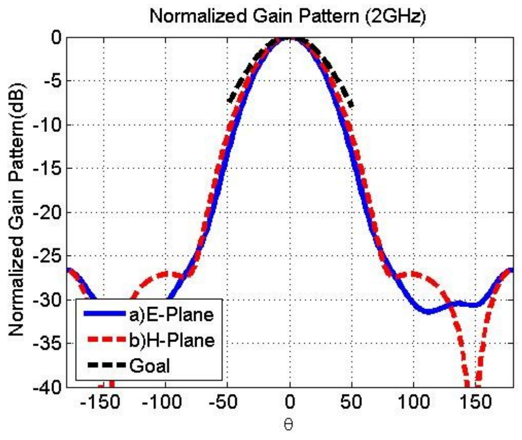


(d) HPBW

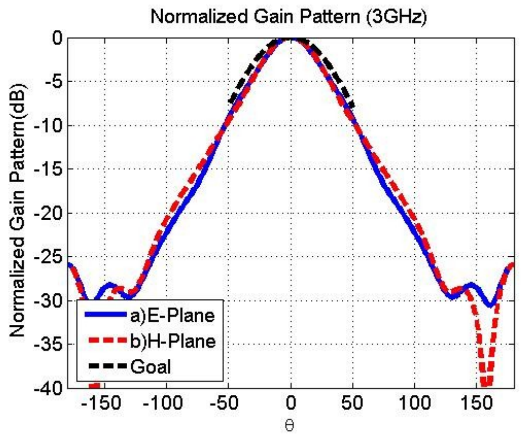
Figure 4.22 Simulation Results of optimized four-layer DRA with modified F_{pc}



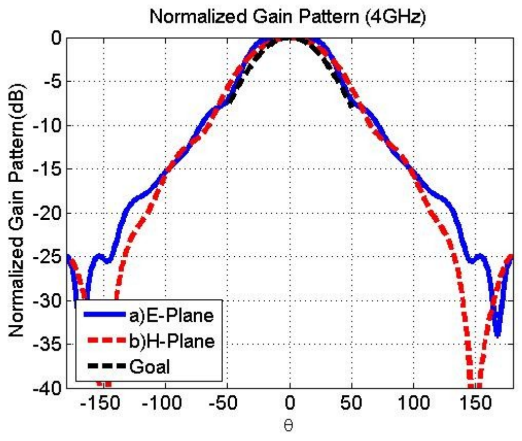
(a) 1GHz



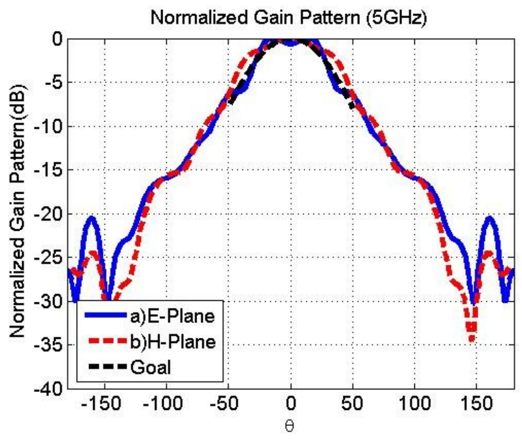
(b) 2GHz



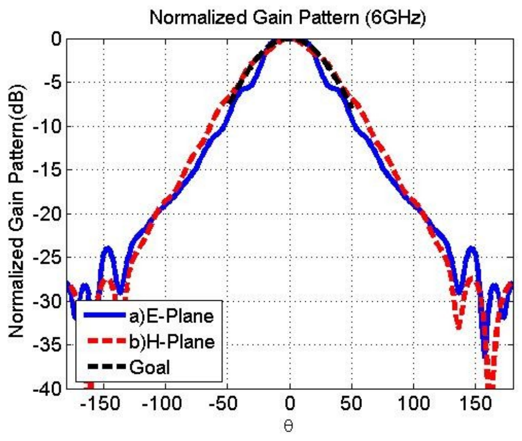
(c) 3GHz



(d) 4GHz

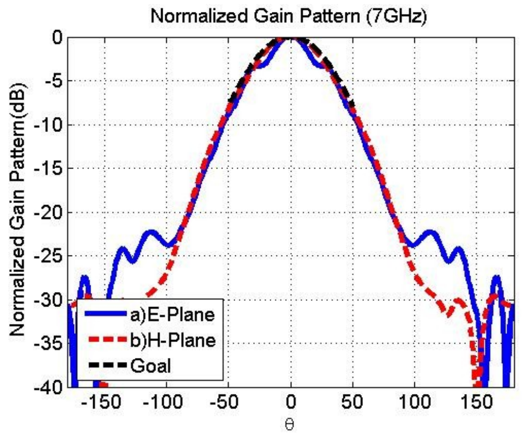


(e) 5GHz

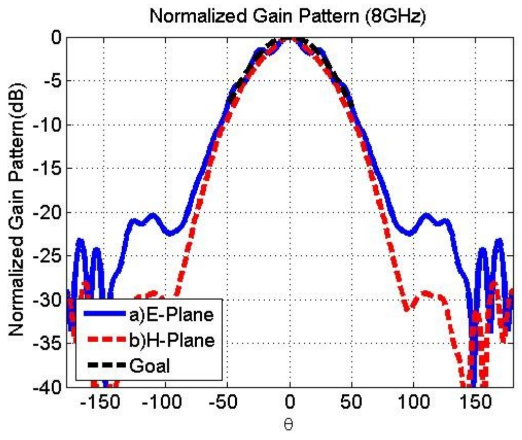


(f) 6GHz

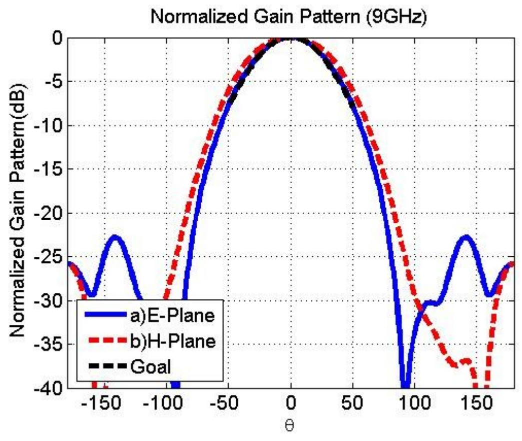
Figure 4.23 Normalized patterns for new optimized four-layer DRA with modified F_{pc} (1)



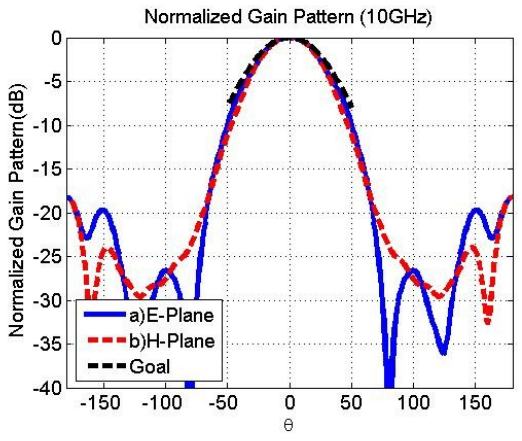
(g) 7GHz



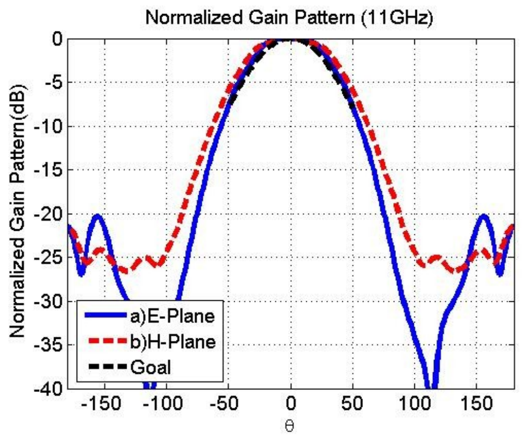
(h) 8GHz



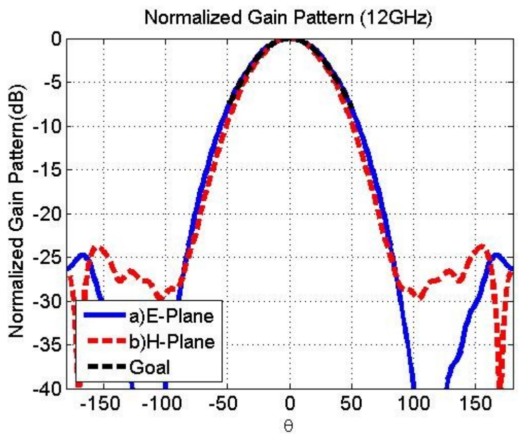
(i) 9GHz



(j) 10GHz

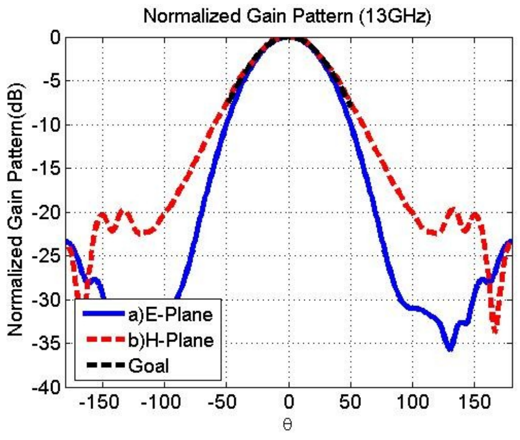


(k) 11GHz

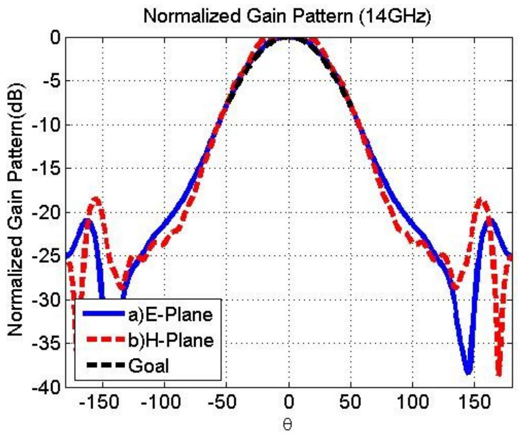


(l) 12GHz

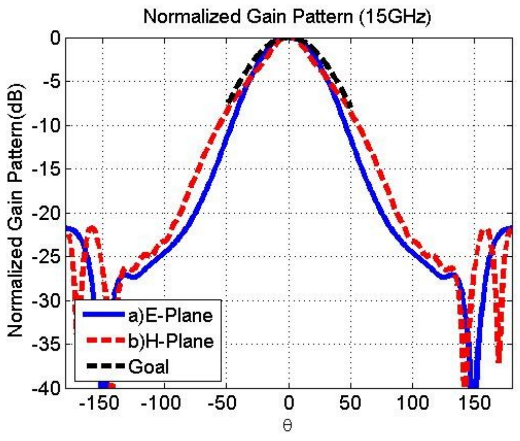
Figure 4.24 Normalized patterns for new optimized four-layer DRA with modified F_{pc} (2)



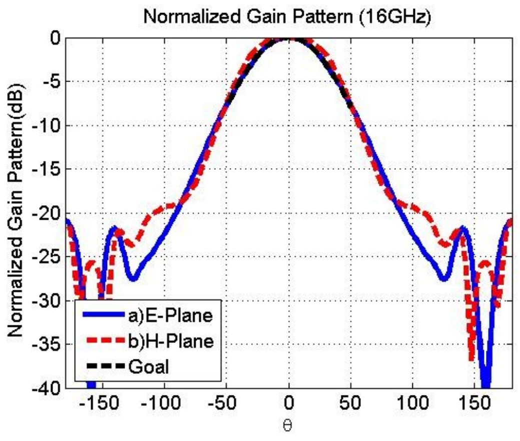
(m) 13GHz



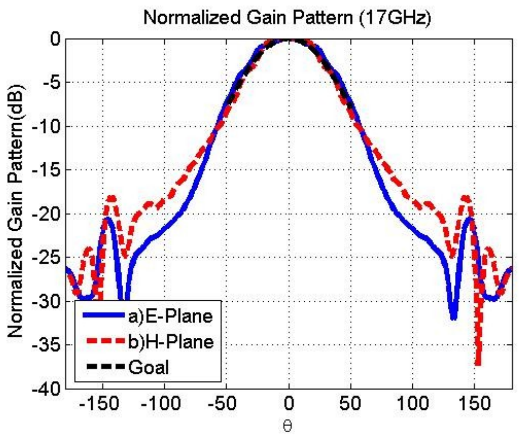
(n)14GHz



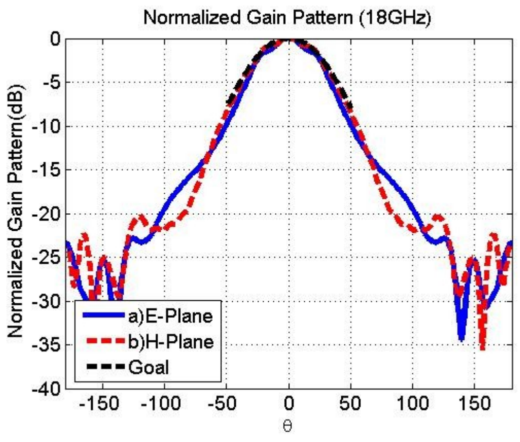
(o) 15GHz



(p) 16GHz



(q) 17GHz



(r) 18GHz

Figure 4.25 Normalized patterns for new optimized four-layer DRA with modified F_{pc} (3)

4.4 Optimized Three-Layer DRA Design with Practical Feed Structure

In the previous optimizations, in order to obtain the true radiation from the dielectric body and ignore any undesired influence from the feed structure, all the designs are excited waveport. But waveort doesn't practically exist, and the reliability of the optimized three-layer DRA performances needs further verification. The reliability can be verified by feeding the optimized three-layer DRA with a practical feeding structure mentioned in the chapter 2 and comparing the performances.

4.4.1 Feeding by 30 Degrees Pyramid and 10 Degrees Metal Triangles

The geometry of the optimized three-layer DRA with a practical launcher section has shown in the figure 4.26 and it is formed by 30 degrees pyramid with 10 degrees metal isosceles triangles and matched to 100 ohms port. The pyramid angle is small to achieve a longer taper launcher section that can provide a better energy transition form the feed to the waveguide. The angle of isosceles triangle metals is also small to insure the impedance matching. Taper resistive strips are connected to the end of metals to reduce reflection from the end of the feeding to reduce undesired reflection from the end of the metals.

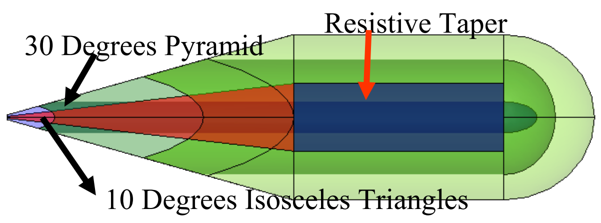
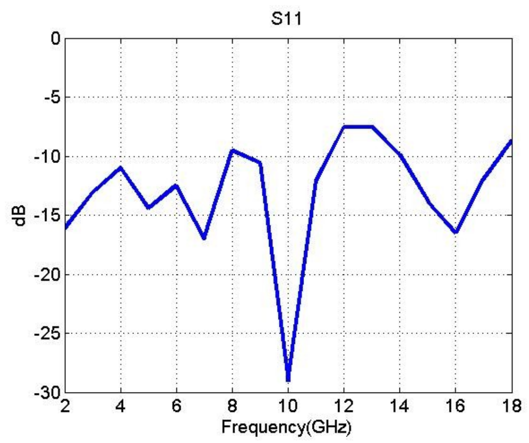


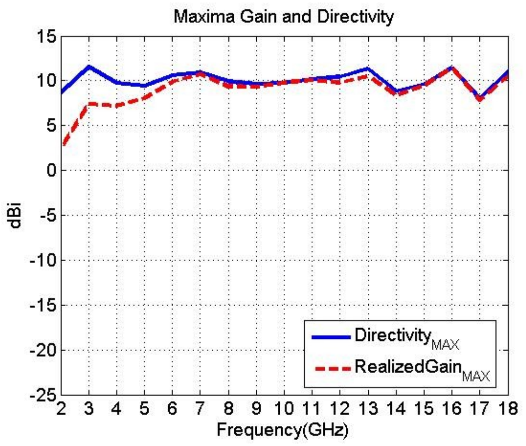
Figure 4.26 Geometry of the optimized three-layer DRA with a practical feed

The simulation results of three-layer DRA with the practical feed structure are shown in the Figure 4.27 to 4.32. The S_{11} (Figure 4.27(a)) performances are below -7.5dB from 2 to 16GHz. The realized gain (Figure 4.27(b)) is flat and not losing significantly at lower frequency caused by resistive taper. All patterns (Figure 4.28-4.30) properly match to the results with waveport excitation and reaches 50 degrees HPBW (Figure 4.27(d)) from 3 to 16GHz. Though the phase center performances show some oscillations, the results are still fairly stable and close to the results with ideal feeding.

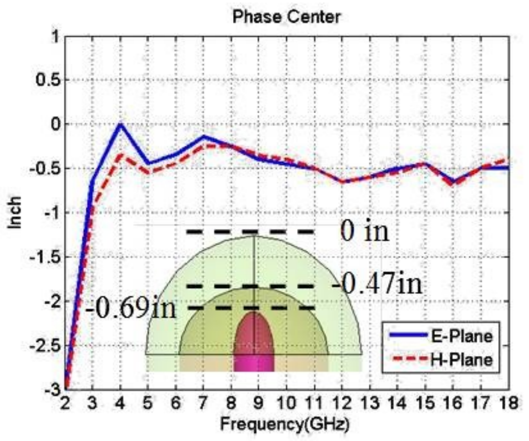
However, there are some differences occurred in pattern and gain results above 16GHz because the launcher section is not long enough to ensure most of the high frequency energy is guided inside the inner layer. As shown in the Figure 4.32, the field plots illustrate that the high frequency energy doesn't constraint tightly from the feed to the in the inner layer and affect the pattern performances. Moreover, because the field at 2GHz extends to the end of feeding structure, the discontinuity at the end of metal creates strong diffraction and affects the patterns. As shown in the field plot (Figure 4.31(a)), one can clearly point out the strong field distribution at the end of the metal. The issues can be solve by prolong the launcher section by choosing smaller angle of pyramid structure and extending the length of the metal triangles.



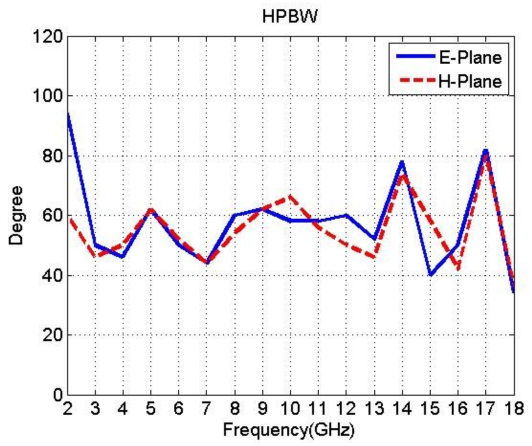
(a) S_{11}



(b) Maxima Directivity and Gain

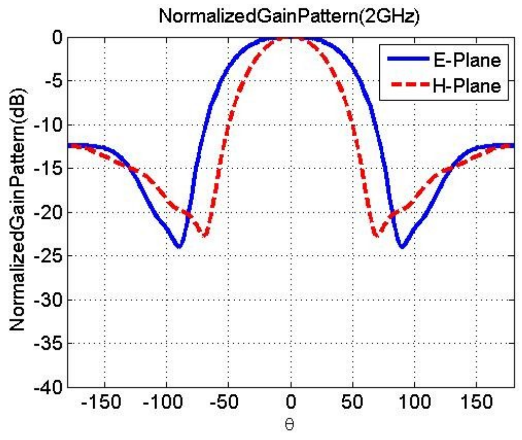


(c) Phase Center

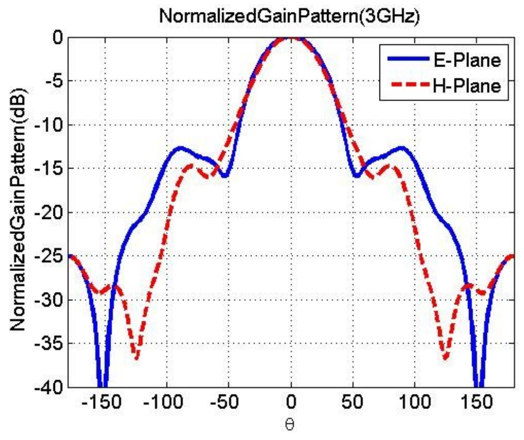


(d) HPBW

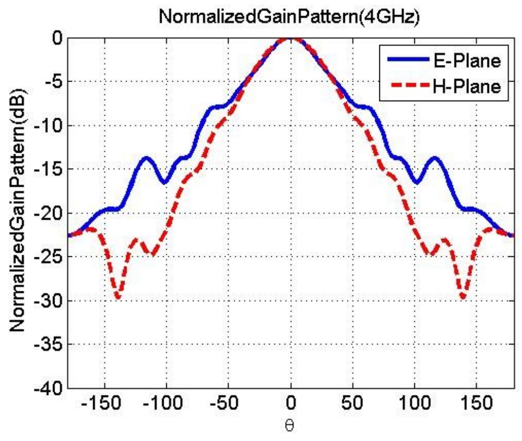
Figure 4.27 Simulation results the optimized three-layer DRA with a practical feed



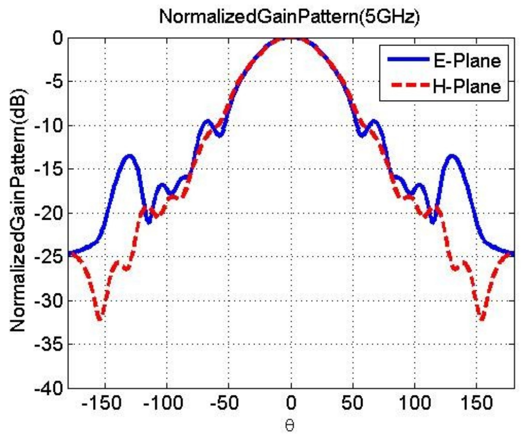
(a) f = 2GHz



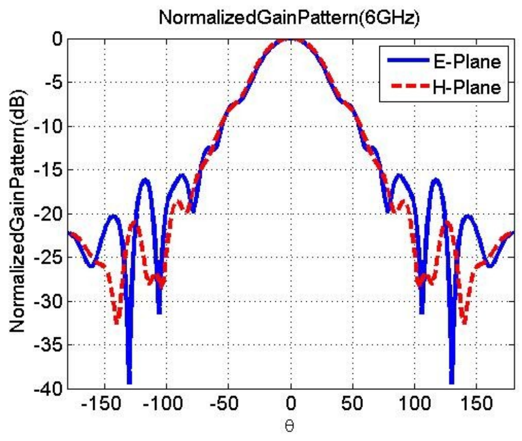
(b) f = 3GHz



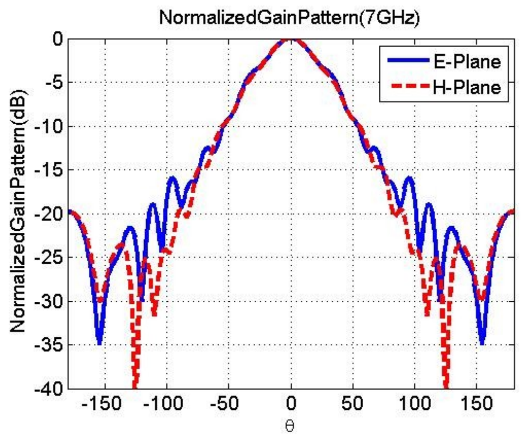
(c) f = 4GHz



(d) f = 5GHz

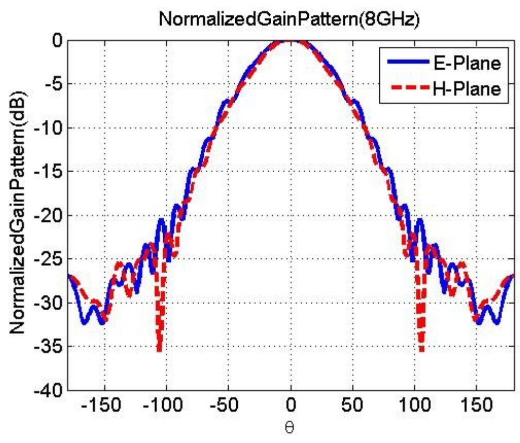


(e) f = 6GHz

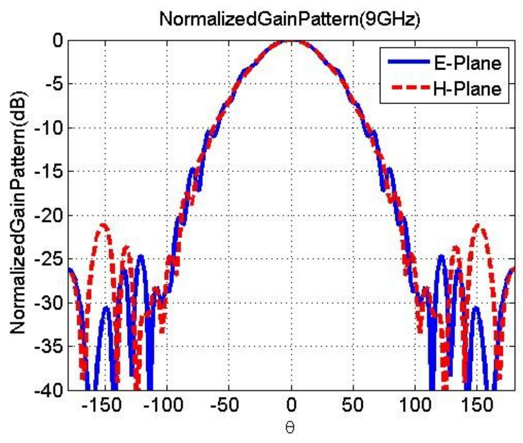


(f) f = 7GHz

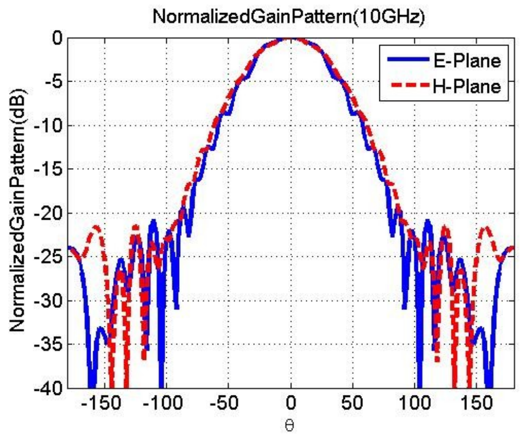
Figure 4.28 Normalized Patterns of optimized 3-layer DRA design with practical feed (1)



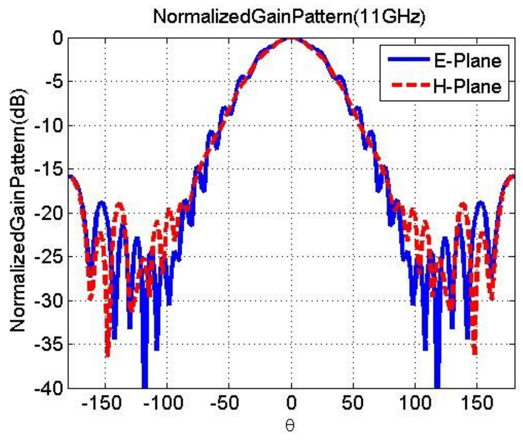
(g) f = 8GHz



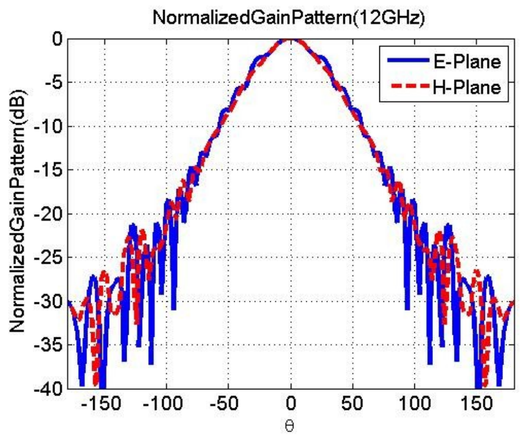
(h) f = 9GHz



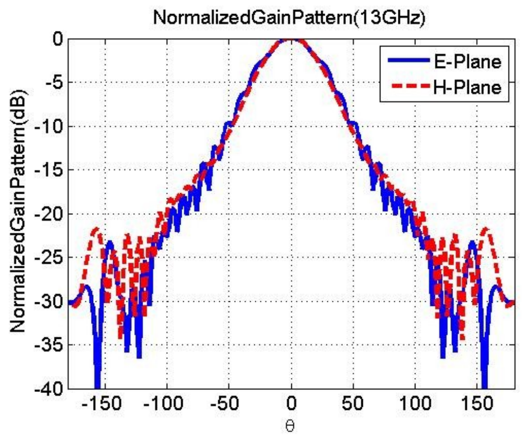
(i) f = 10GHz



(j) f = 11GHz

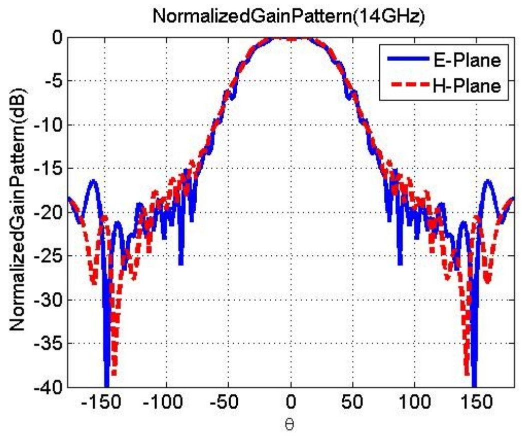


(k) f = 12GHz

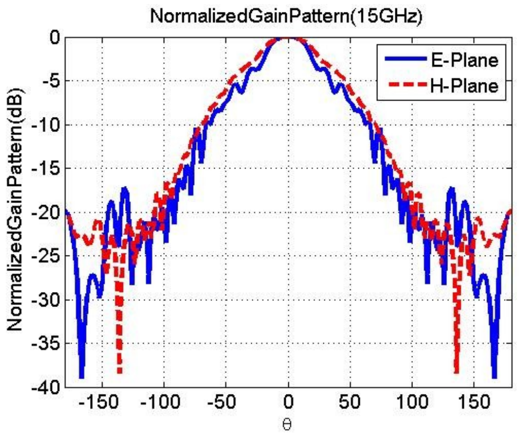


(l) f = 13GHz

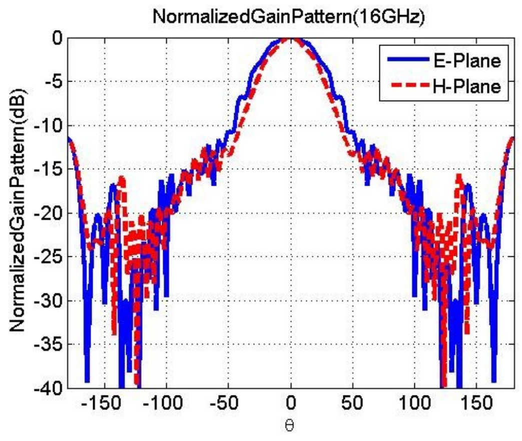
Figure 4.29 Normalized Patterns of optimized 3-layer DRA design with practical feed (2)



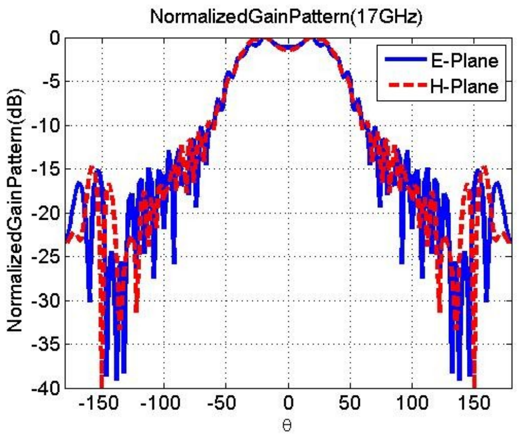
(m) f = 14GHz



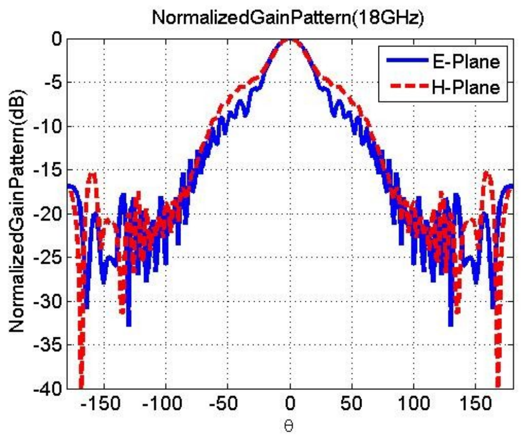
(n) f = 15GHz



(o) f = 16GHz

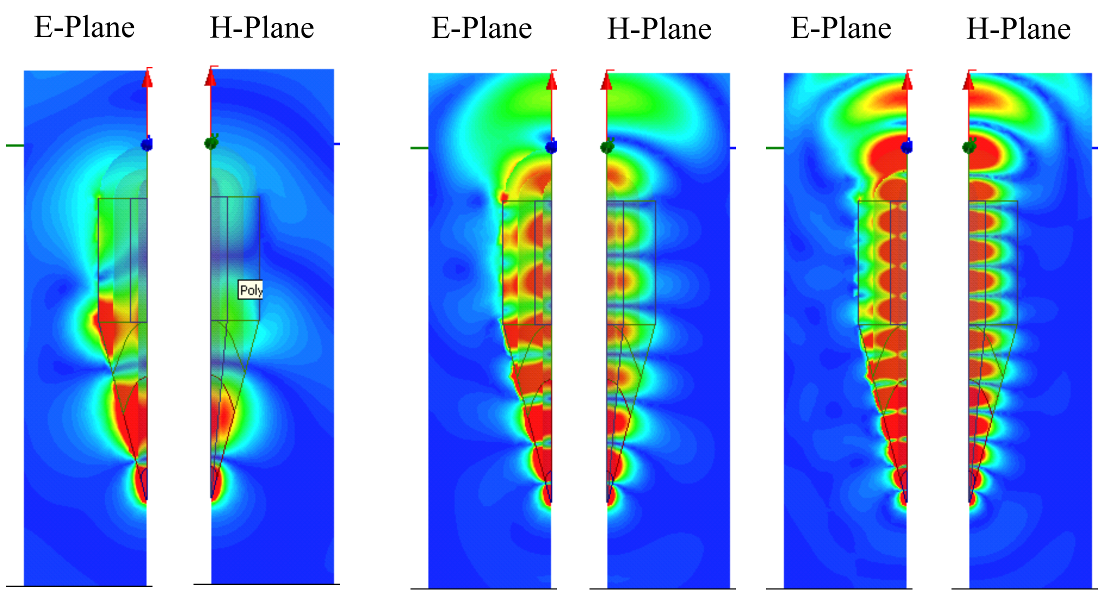


(p) f = 17GHz

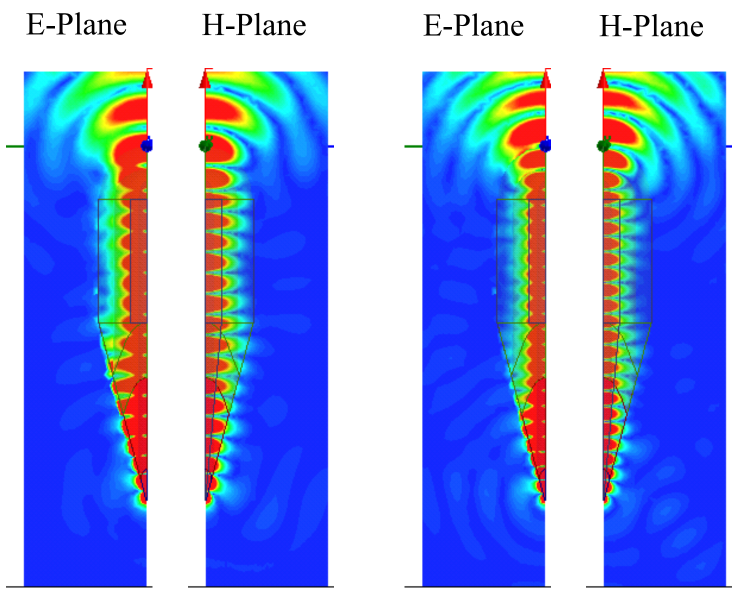


(q) f = 18GHz

Figure 4.30 Normalized Patterns of optimized 3-layer DRA design with practical feed (3)

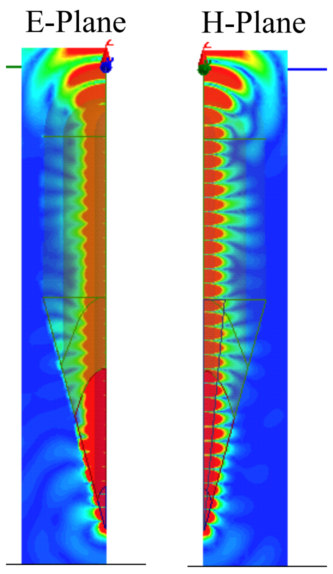


(a) f = 2GHz (b) f = 4GHz (c) f = 6GHz

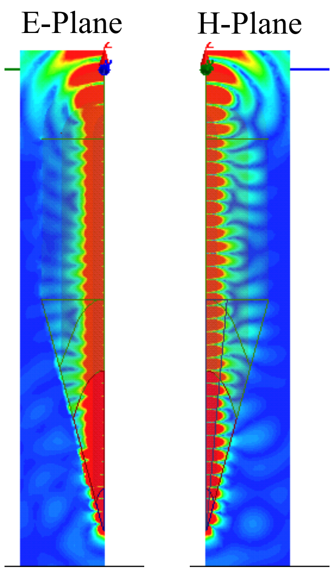


(d) f = 8GHz (e) f = 10GHz

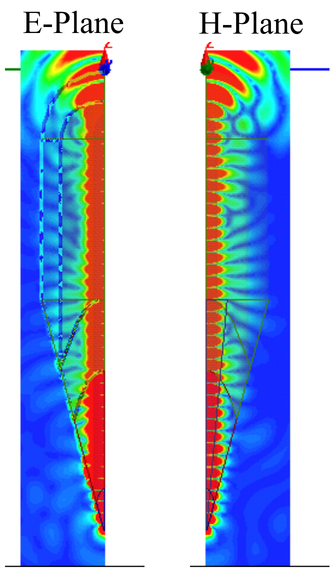
Figure 4.31 E-filed plot on E and H plane of the realistic feed with optimized 3-layer DRA design (1)



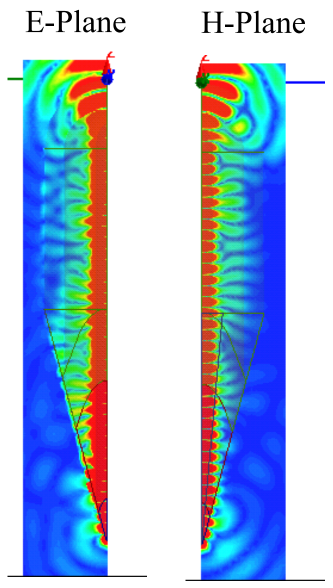
(f) f = 13GHz



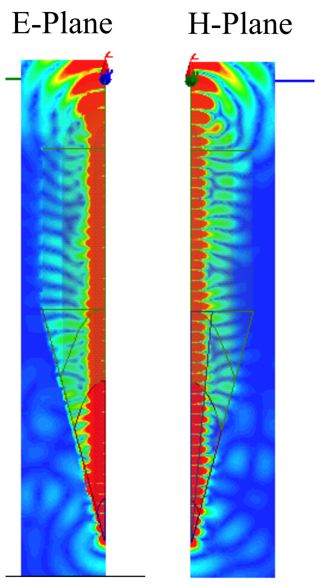
(g) f = 14GHz



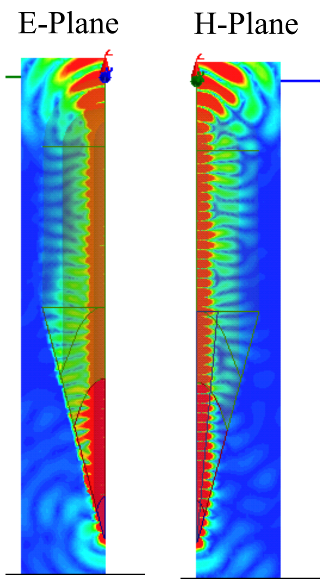
(h) f = 15GHz



(i) f = 16GHz



(j) f = 17GHz



(k) f = 18GHz

Figure 4.32 E-filed plot on E and H plane of the realistic feed with optimized 3-layer DRA design (2)

4.4.2 Feeding by 10 Degrees Pyramid and 10 Degrees Metal Triangles

As demonstrated, because the energy at higher frequency didn't convert well form the launcher section into the waveguide section, higher order modes are excited in the waveguide section. And the radiation patterns are strongly affected by the excitation of high order modes. In order to convert the energy better from the launcher section to the waveguide section, one can be achieved by prolonging the launcher section.

Figure 4.33 shows the geometry of the optimized three-layer DRA with prolonged launcher section. The prolonged launcher section is formed by a 10 degrees pyramid structure and 10 degrees isosceles triangles and the total length of the launcher section is 5in.

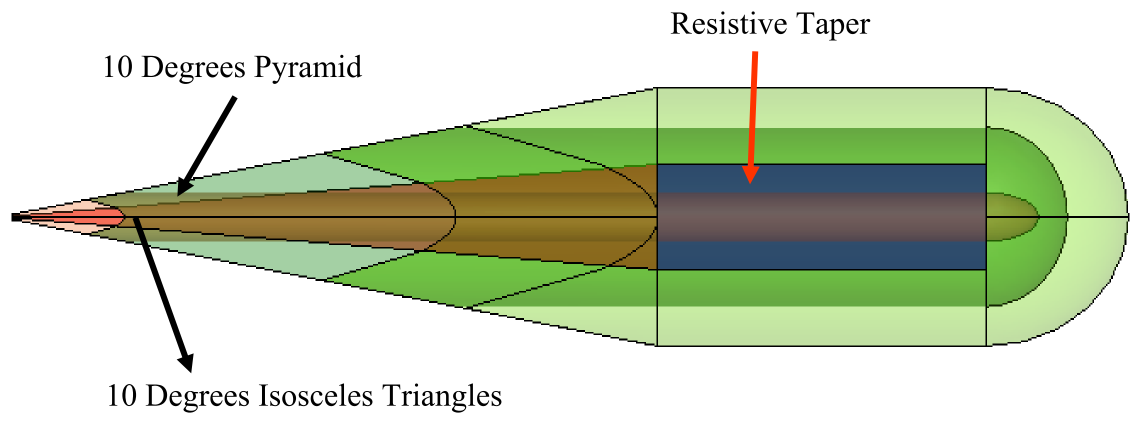
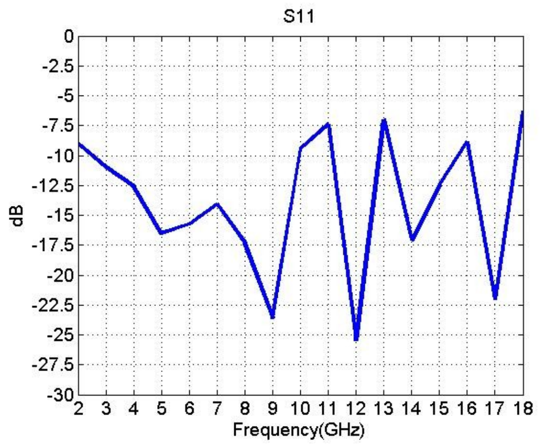


Figure 4.33 Geometry of optimized three-layer DRA with prolonged launcher section

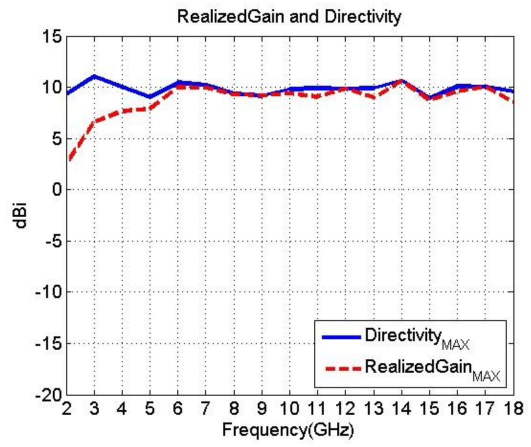
The simulation results of three-layer DRA with the prolonged feed structure are shown in the Figure 4.34. The S_{11} (Figure 4.34(a)) performances have more oscillations because the impedance is varied when the angle of pyramid structure changes. But, overall, the S_{11} are lower than -7.5dB from 2 to 18GHz and the impedance can be tuned by changing the angle of the triangle metals. The realized gain (Figure 4.34(b)) is still flat and not losing significantly at lower frequency caused by resistive taper. Compared to the shorter launcher feeding structure, the phase center (Figure 4.34(c)) is more stable at higher frequency, and the pattern beamwidth (Figure 4.34(d)) have less oscillation over the operation frequency band.

Figure 4.35 to 4.37 show the normalized patterns of this design, and it demonstrates broad, consistent, and similar to that waveport excitation from 3 to 17GHz. The pattern at 18GHz becomes more frequency dependent and narrow beamwidth because the field is still not perfectly guided as the waveport excitation. Figure 4.38 shows the E-field distributions at 18GHz of three different feed, and it demonstrates that the fields are guided better with a longer launcher. Even though it still has stronger undesired field existing in the waveguide section, the field distribution of longer launcher is more similar to the waveport excitation. Hence, by properly choosing the length of launcher section, it can achieve a better mode transition from launcher to waveguide section and avoid the excitation of higher order modes.

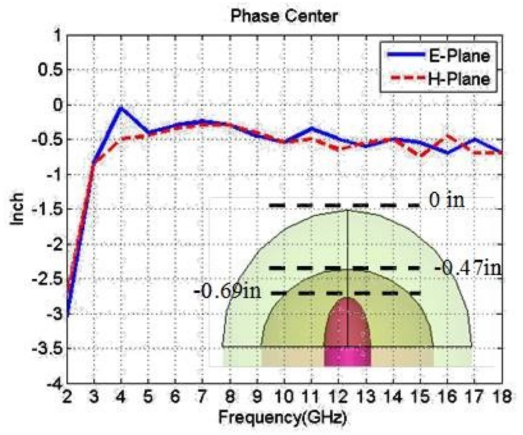
But the radiation patterns at lower frequencies (Figure 4.35(a)-(d)) are still affected by the scattering field from the end of the launcher structure. In order to further mitigate the undesired influence from the end of the launcher section, the discontinuity point for the triangle metals would need to be adjusted.



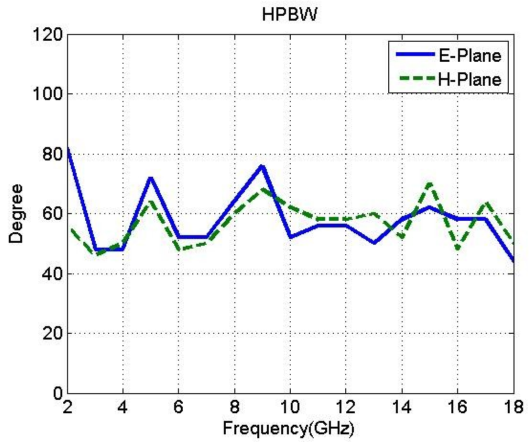
(a)S11



(b) RealizedGain and directivity

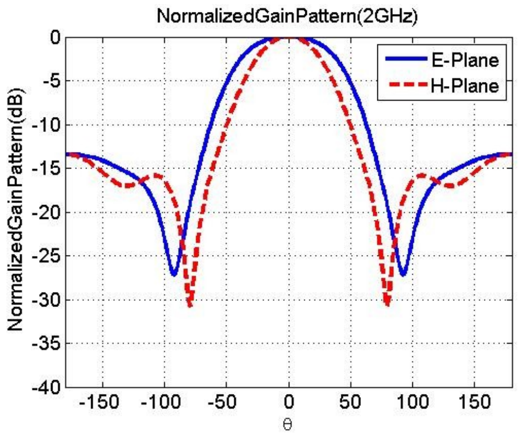


(c)Phase Center

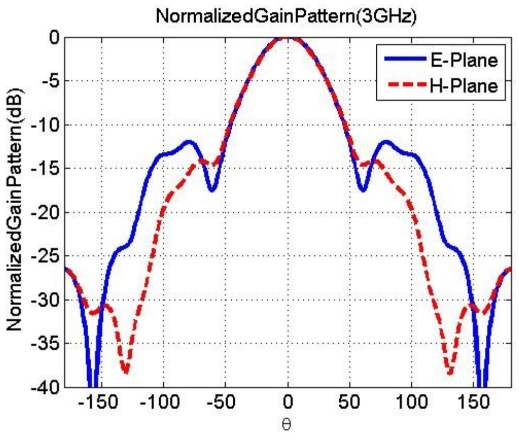


(d) HPBW

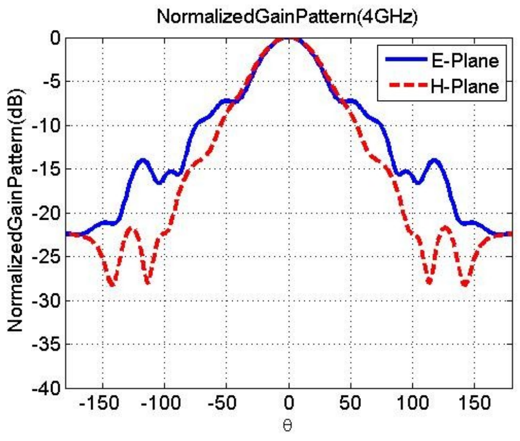
Figure 4.34 Simulation results of optimized three-layer DRA with prolonged feed



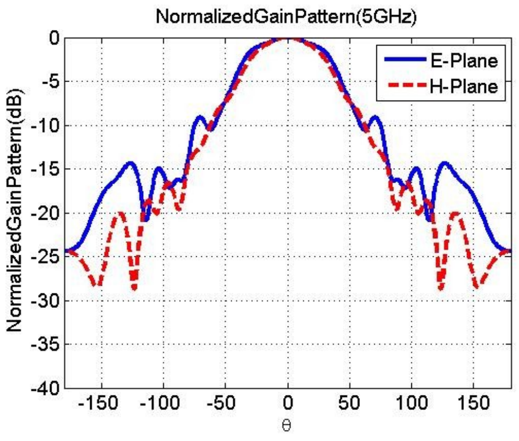
(a)2GHz



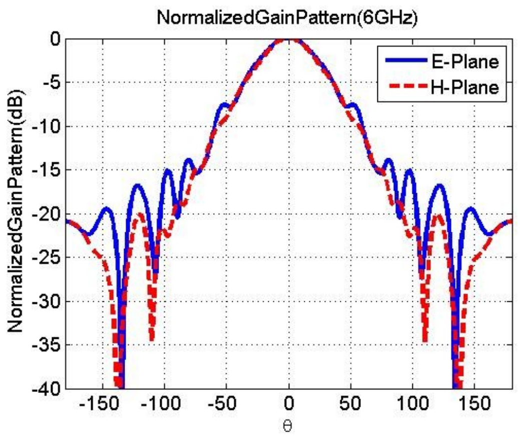
(b)3GHz



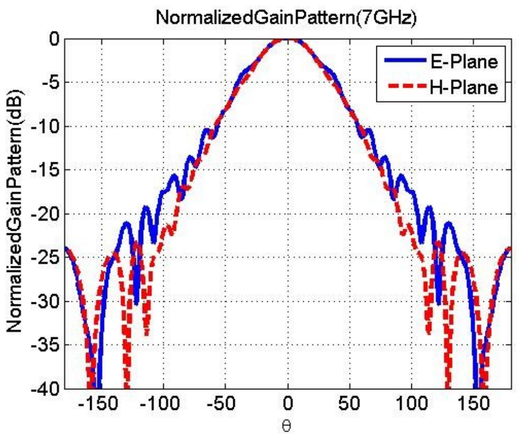
(c)4GHz



(d)5GHz

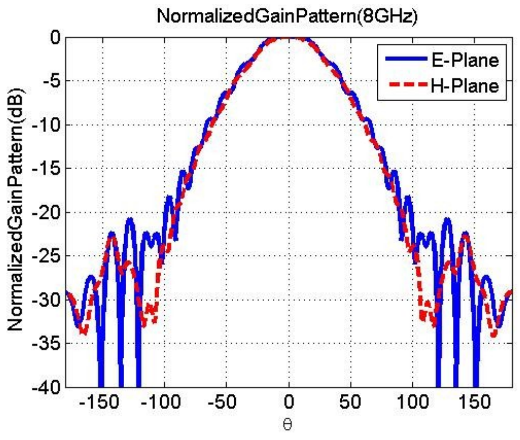


(e)6GHz

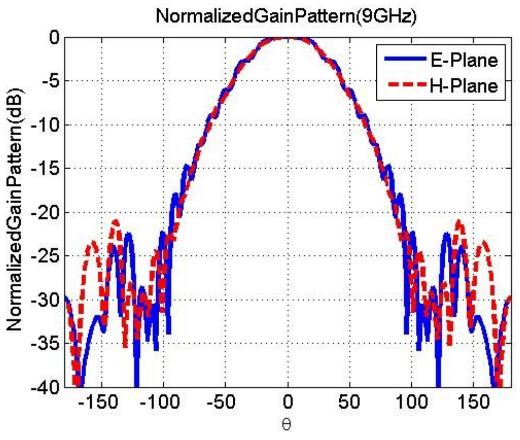


(f)7GHz

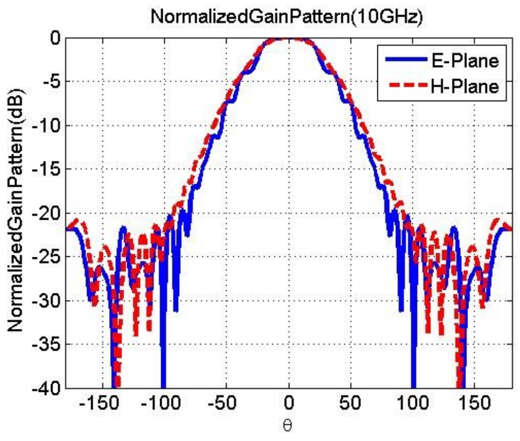
Figure 4.35 Normalized patterns of optimized three-layer DRA with prolonged feed (1)



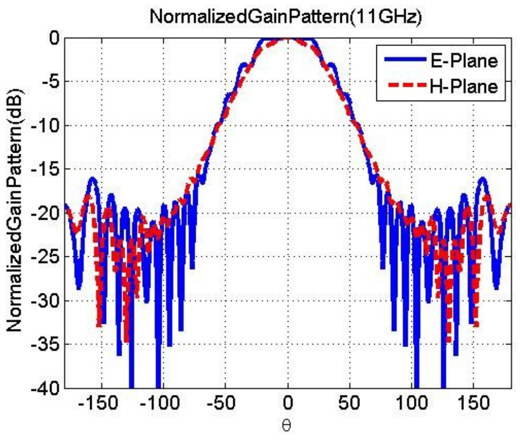
(g) 8GHz



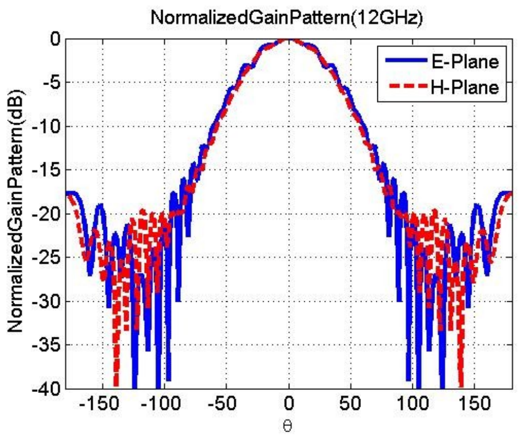
(h) 9GHz



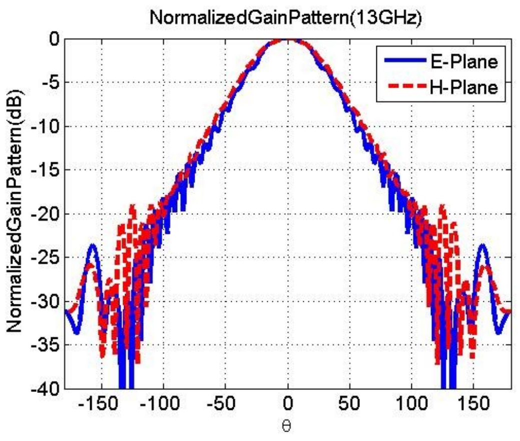
(i) 10GHz



(j)11GHz

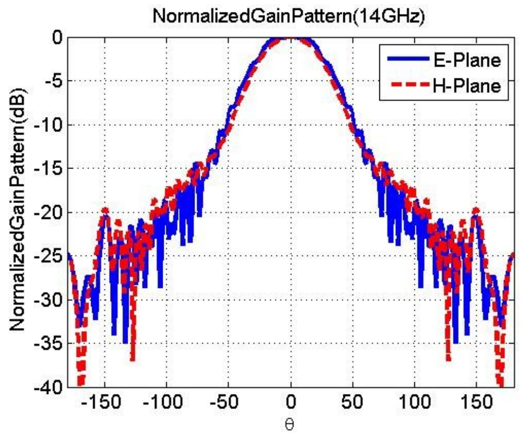


(k) 12GHz

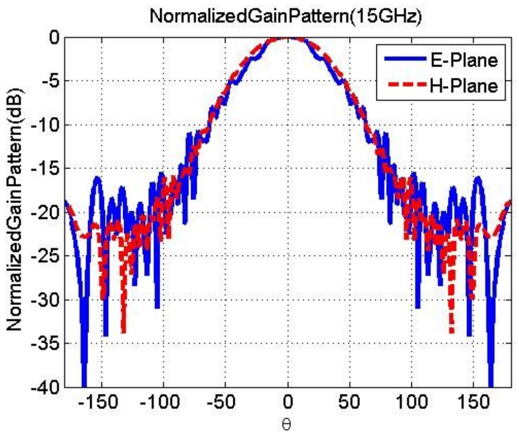


(l) 13GHz

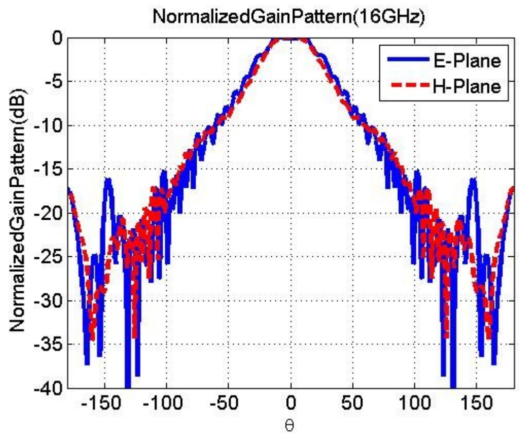
Figure 4.36 Normalized patterns of optimized three-layer DRA with prolonged feed (2)



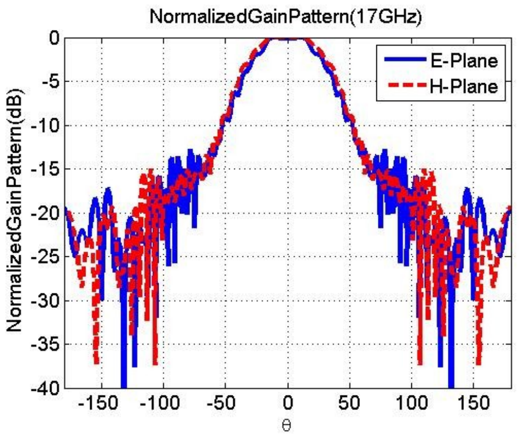
(m) 14GHz



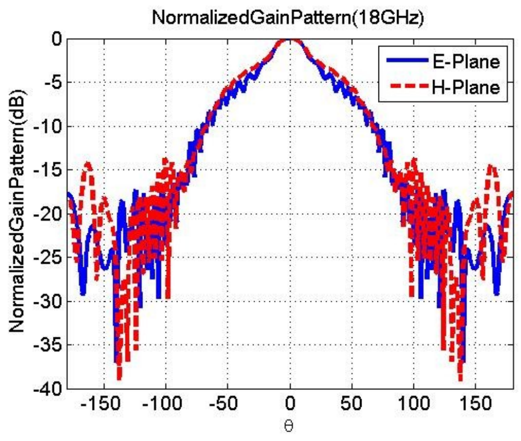
(n) 15GHz



(o) 16GHz

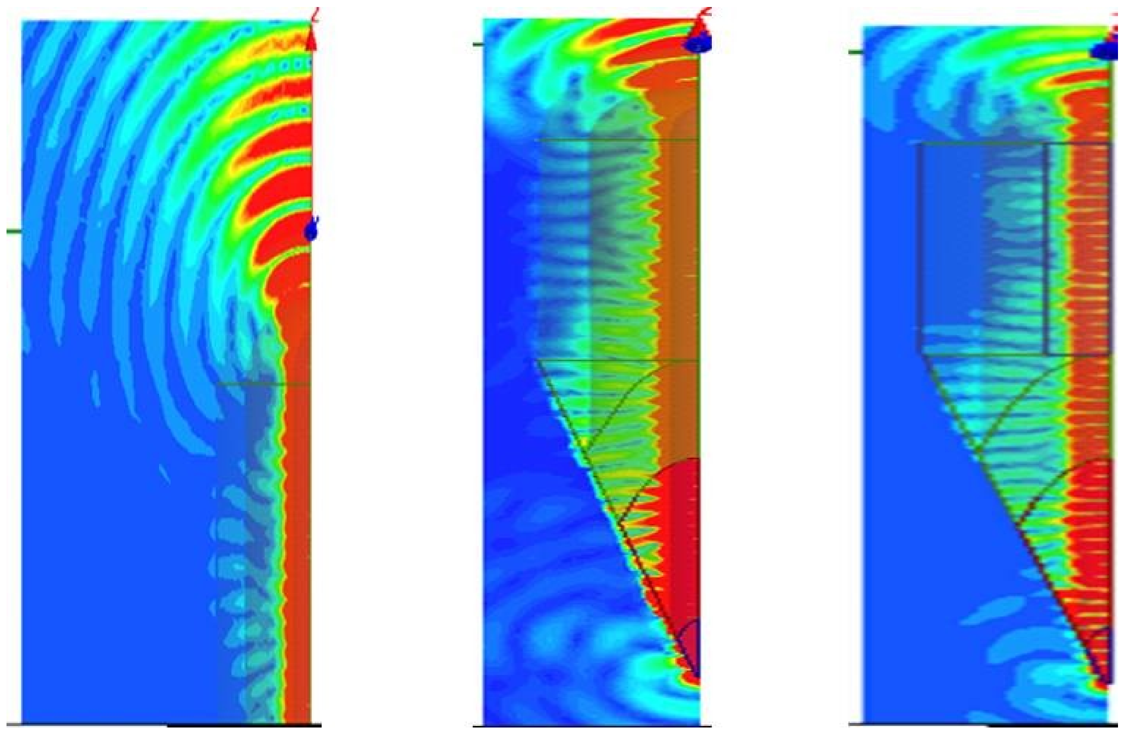


(p) 17GHz



(q) 18GHz

Figure 4.37 Normalized patterns of optimized three-layer DRA with prolonged feed (3)



(a) Waveport (b)Shorter launcher (c) Longer launcher

Figure 4.38 E-field plot at 18GHz

4.4.3 Extending the Length of Triangle Metals

As shown before, there are strong scattering fields at the discontinuity between metal and the resistive taper, and the strong diffractions would results in pattern variation at lower frequencies. In order to resolve this issue, we can reduce the scattering field at the connection by extending the length of the metals.

Figure 4.39 shows the geometry of the optimized three-layer DRA with a practical feed structure, and the feed structure is formed by a 10 degrees pyramid structure and 10 degrees isosceles triangles. Moreover, the metals length is 0.25in longer than the previous design.

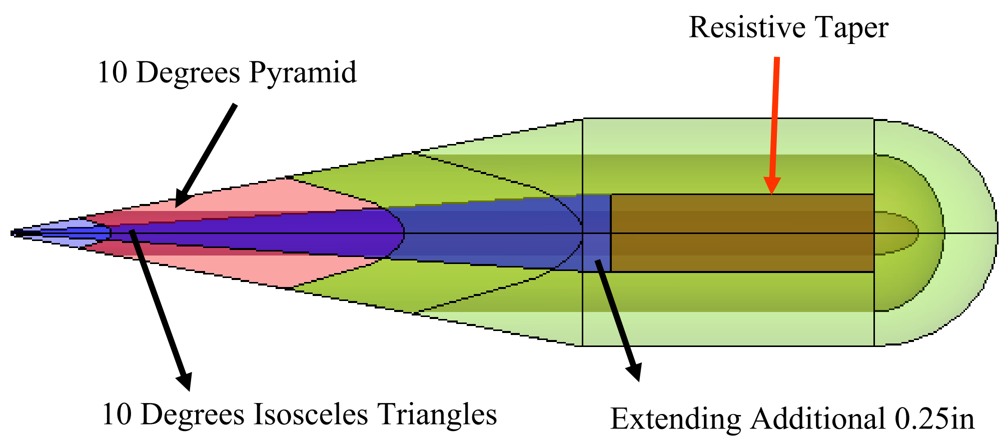
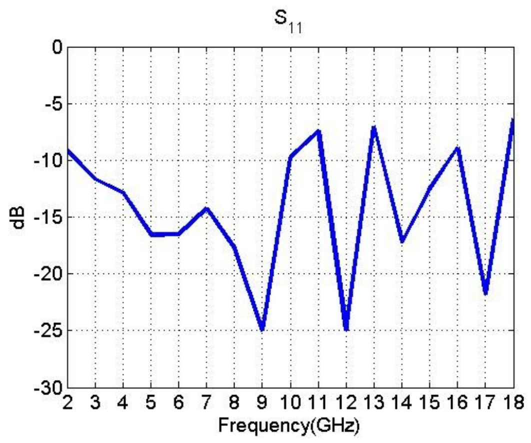


Figure 4.39 Optimized three-layer DRA with adjusted launcher

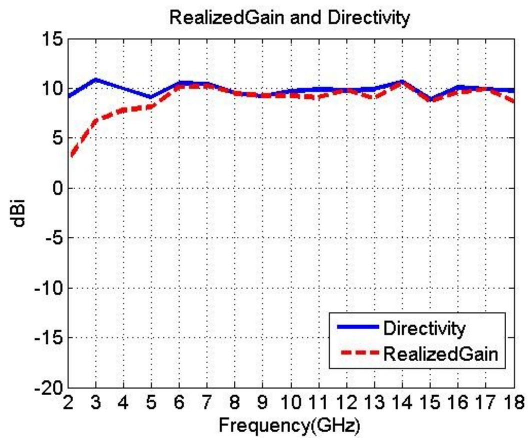
The simulation results of three-layer DRA with the extended metal length are shown in the Figure 4.40. The S_{11} (Figure 4.40(a)) performances are similar to the previous design, and demonstrate that the additional length of the metal wouldn't significantly change the input impedance. The realized gain (Figure 4.40(b)) is still flat. Compared to the feeding structure without extending the metal length, the phase center (Figure 4.40(c)) is more stable at lower frequency, and the pattern beamwidth(Figure 4.40(d)) are more consistent from 2 to 18GHz.

Figure 4.41 to 4.43 show the normalized patterns of this design, and it demonstrates broad and stationary patterns from 3 to 18GHz. By comparing the normalized gain patterns to the goal pattern, one can point out that the patterns are conformed to the target patterns well from 3 to 18GHz. Moreover, by comparing the patterns between longer metals design to the shorter design, it's clear that the sidelobe level at lower frequencies are lower. Figure 4.44 – 4.45 show the E-filed distributions between two different metal length at 2 and 3GHz, and it's demonstrates that the diffractions the discontinuity are weaker with longer metal length. By further adjusting the length of the triangle metals, the sidelobe levels can be reduced more.

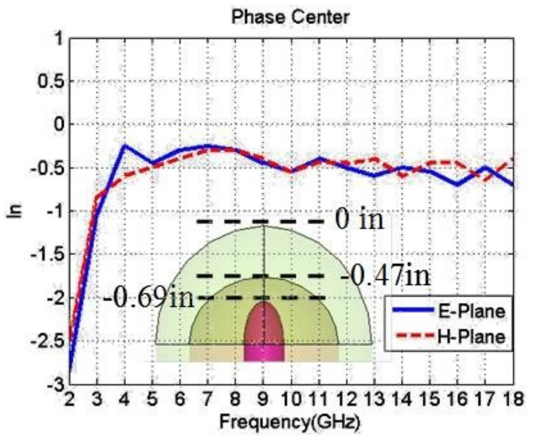
Because the feeding structure is not optimized yet, some performance differences between the practical feed structure and the waveport excitation are expected. After all, the results are fairly similar on the performances of S_{11}, phase center, gain, and patterns respectively between two different feeding structures. Hence, the reliability of the optimized design from GA optimization procedure has been proven, and the optimized performances can be achieved by properly optimizing the angle of pyramid launcher, the angle of isosceles triangles, and the starting point of resistive taper strips, the difference can be resolved.



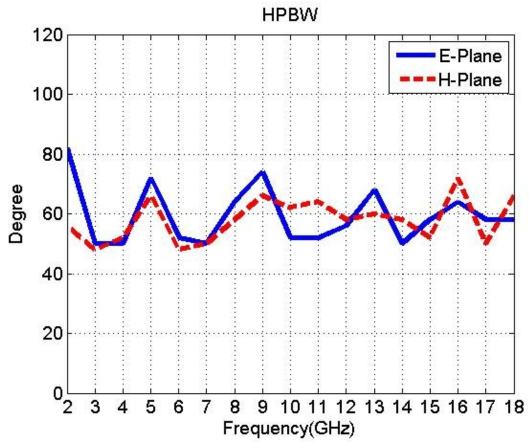
(a) S_{11}



(b) Realizedgain and directivity

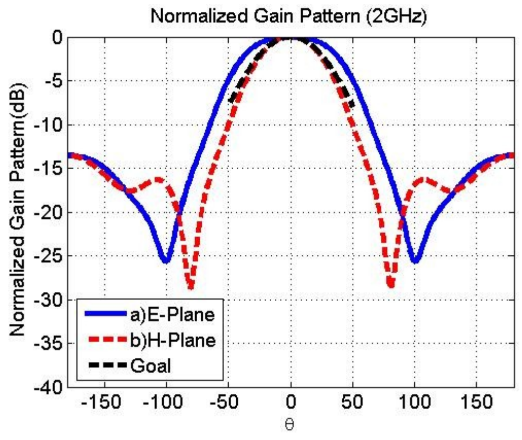


(c) Phase center

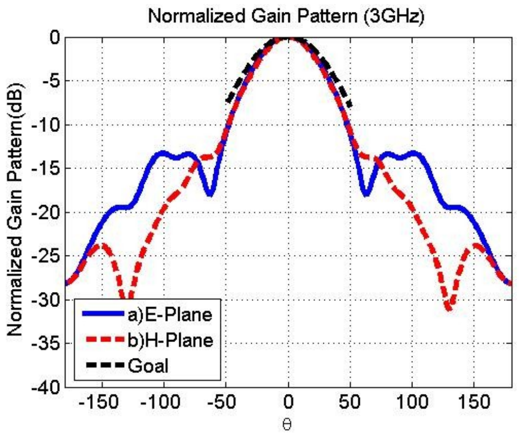


(d)HPBW

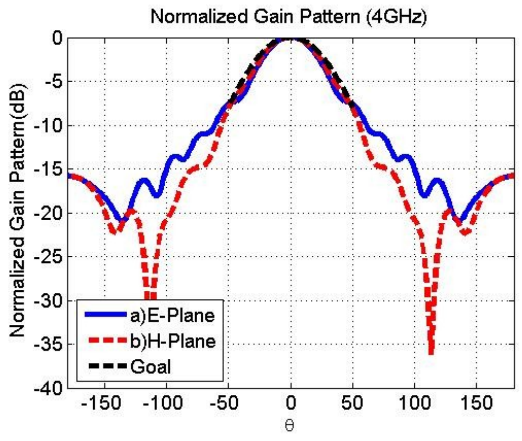
Figure 4.40 Simulation results of optimized three-layer DRA with adjusted launcher



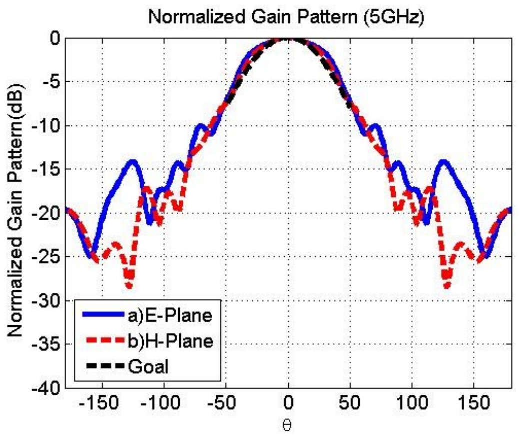
(a) 2GHz



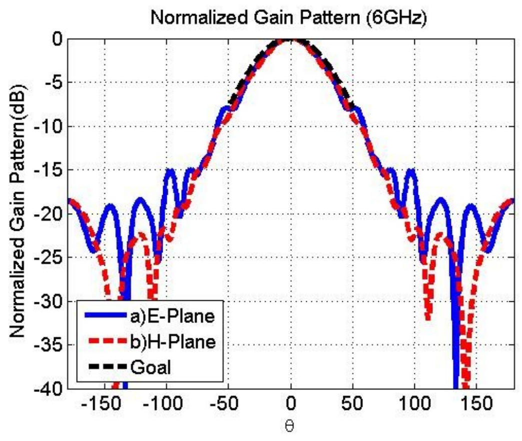
(b) 3GHz



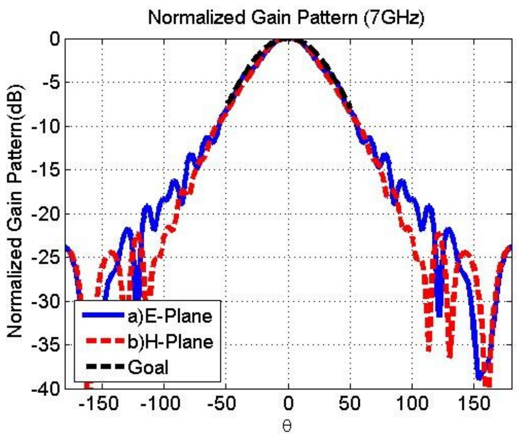
(c) 4GHz



(d) 5GHz

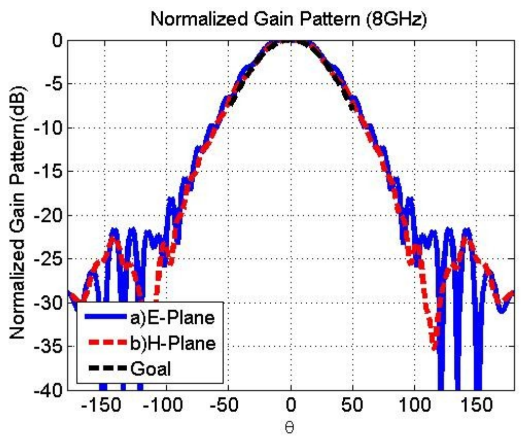


(e) 6GHz

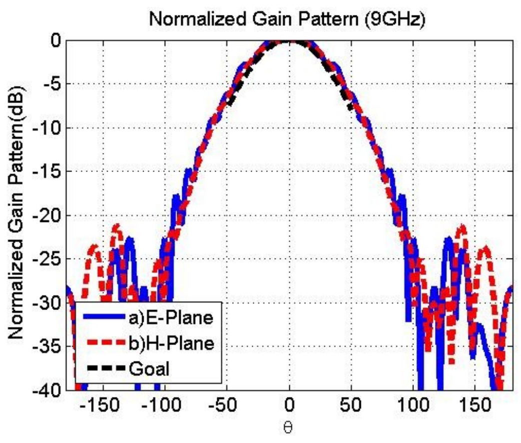


(f) 7GHz

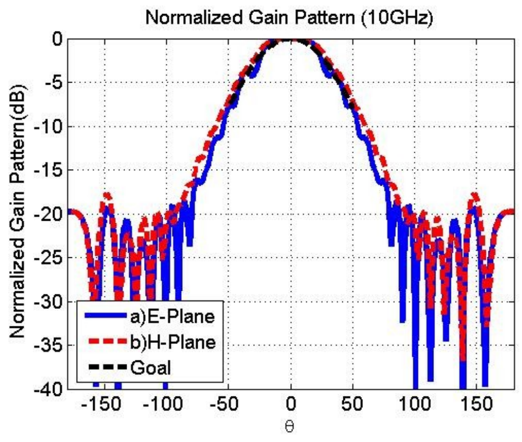
Figure 4.41 Normalized patterns of optimized three-layer DRA with adjusted launcher (1)



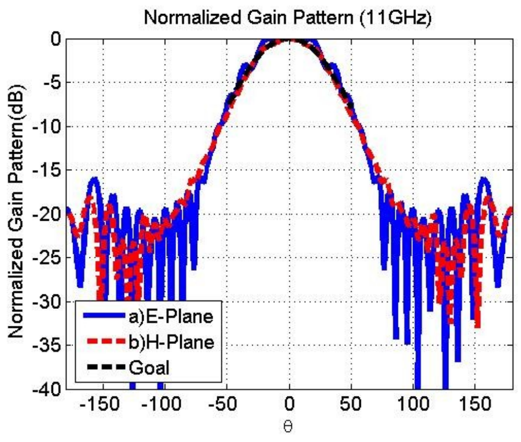
(g) 8GHz



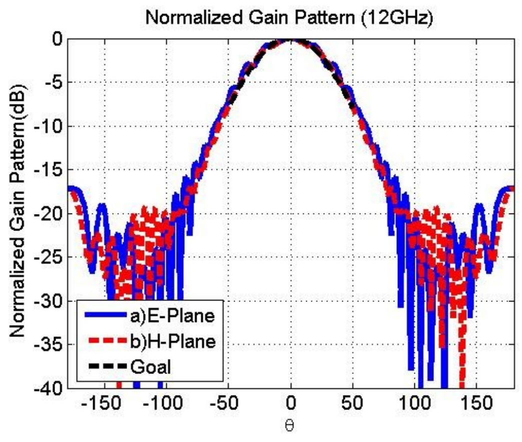
(h) 9GHz



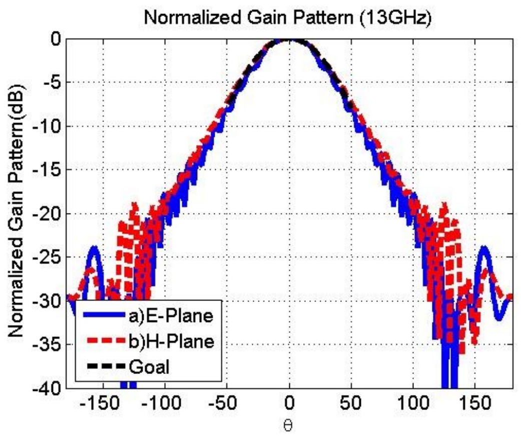
(i) 10GHz



(j) 11GHz

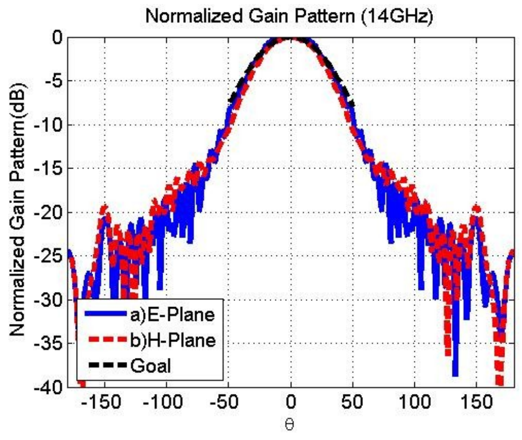


(k) 12GHz

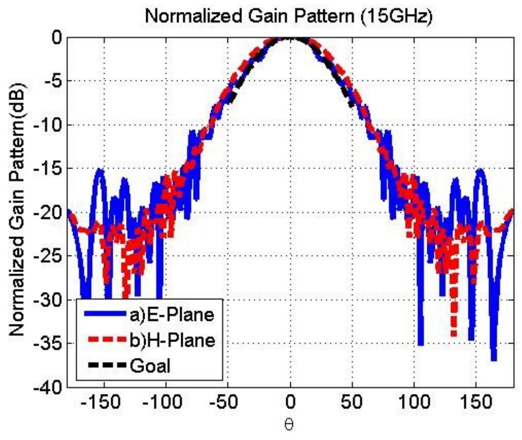


(l) 13GHz

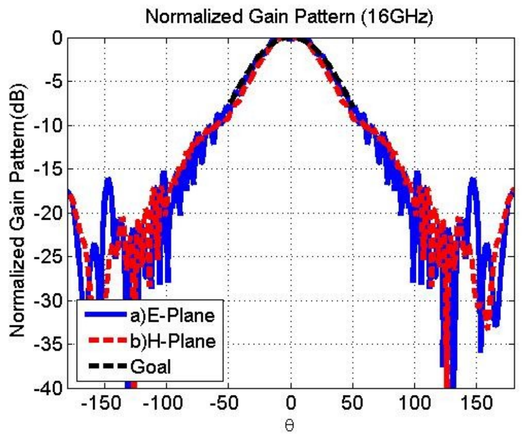
Figure 4.42 Normalized patterns of optimized three-layer DRA with adjusted launcher (2)



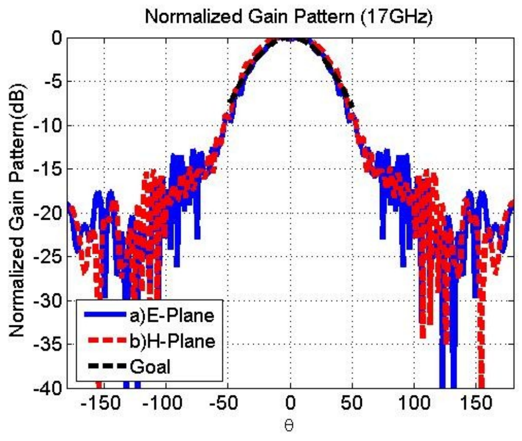
(m)14GHz



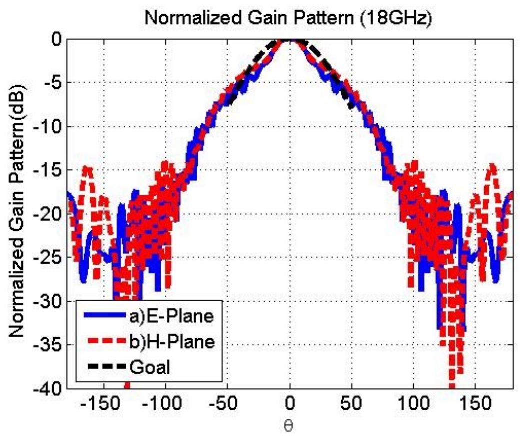
(n)15GHz



(o) 16GHz

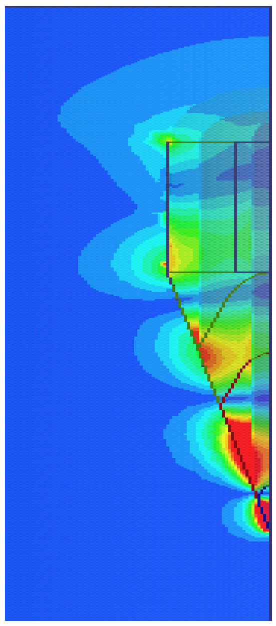


(p) 17GHz



(q)18GHz

Figure 4.43 Normalized patterns of optimized three-layer DRA with adjusted launcher (3)


(a) Shorter metal length

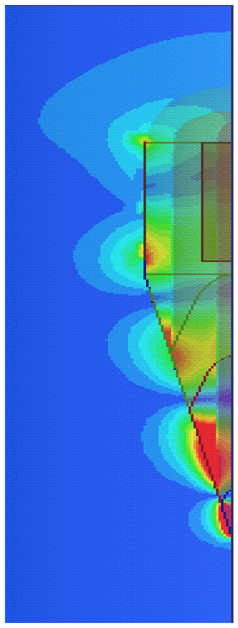
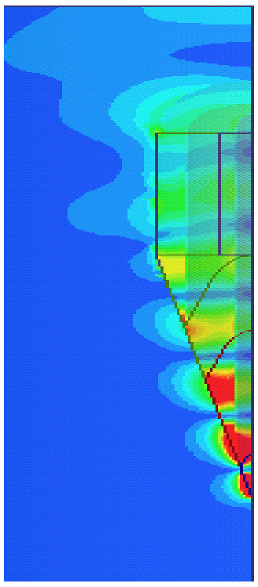

(b) Longer metal length

Figure 4.44 E-field plot at 2GHz


(a) Shorter metal length

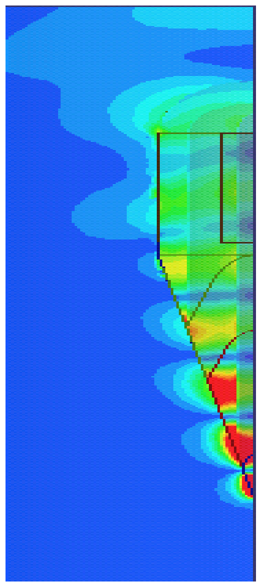

(b) Longer metal length

Figure 4.45 E-field plot at 3GHz

CHAPTER 5

Conclusion

In this thesis, the design of multi-layer DRA has been presented. The operation principles of the automatic design method have been presented. The optimization program has been developed and it can effectively optimize a multi-layer DRA that reaches the design requirement. Furhermore, the reliability of the automatic optimization method has been proven.

By combining the commercial code HFSS with an external GA optimization program, the automatic design optimization method for UWB multi-layer DRA has been developed and presented in the thesis. With the assistance of the automatic optimization method, a three-layer DRA with 2 to 18GHz bandwidth and a four-layer DRA with 1 to 18GHz bandwidth have been optimized. Both optimized designs reach the performance requirements on gain, phase center, pattern, and beamwidth over the operation frequency band. Moreover, by observing the convergence plots of all the optimization designs, all the optimizations achieve the optimized design in 10 generations. Therefore, it's proven that the automatic optimization method can effectively optimize multi-layer DRA with different performance requirements.

Besides, by comparing the optimized design parameters of both three-layer and four-layer DRA optimizations, it shows that the dielectric constant difference between adjacent layers is always small to reduce undesired reflection from waveguide section to radiation section. Moreover, the axial ratios for the inner layers prefer to be larger to postpone the radiation point for high frequency, and it can ensure the stable phase center over operation band. In addition, by comparing the dimension between three-layer and four-layer design, it shows that the diameter of the four-layer design is 0.4in wider than the three-layer design because the four-layer design needs larger electrical dimension to enhance the performance at 1GHz. And, as expected, the four-layer design achieves better S_{11} and phase center performance than the three-layer case below 3GHz. As a result, all the design parameters optimized by the automatic design method are reasonable, and the optimization method developed in the thesis is trustable.

The fitness function used in the thesis is aimed for statically stable over the operation band. Moreover, because the phase center and S11 performances of a multi-layer at higher frequencies are most likely to be stable, the fitness function might be biased when it searches for statically stable over the whole band. In order to improve the low frequency performance, the fitness might need to separate the whole operation frequency range into several bands and enforce the performance at lower frequencies more.

Though the launcher section is not included in the optimization procedure, three important design parameters have been discussed. First, it’s very important to properly choose the angle of the pyramid structure because the angle controls the mode transition between launcher to waveguide section. Normally, a smaller angle can achieve better transition, avoid the excitation of higher order modes, and reduce the pattern variation, but it also increase the size of the antenna and change the input impedance. Therefore, it’s important to choose an angle that meets the balance between antenna size and mode transition. Second, because the combination of the pyramid structure and triangle metals determines the input impedance, it’s crucial to select a good combination that can match to 100 ohms well over the operation frequency band. Third, since the scattering field at the discontinuity between metals and resistive tapers increase the sidelobe level at lower frequencies. Hence, the discontinuity point also needs to be well selected to minimize the undesired influences.

The reliability of the optimized results has been verified by exciting the optimized three-layer design with a practical feed structure and comparing the results between waveport and a practical feed structure. Even though there are still some minor performance losses because the practical launcher section is not optimized, the overall performance between two different feed are fairly similar. By further optimizing the launcher section, the optimized multi-layer DRA design can be completed.

Chapter 6

Future Work

An effective optimization procedure that automatically designs a UWB multi-layer DRA has been presented by combining external GA program with EM analysis software. Optimizations for different antennas are also achievable by properly changing fitness function. Hence, for the future application of GA optimization program, it's important to extend the GA optimization procedure to be compatible for different kinds of commercial codes. And the optimization efficiency can be further enhanced by properly combining optimizing antenna with suitable EM software.

For the current multi-layer DRA design optimization, it's clear that the fitness function doesn't enforce the performance to reach the requirement at lower frequencies because the fitness function tries to find the statistically stable performance over the whole operation band and .neglects the minor defects at lower frequencies. Hence, the fitness function should add addition enforcements at lower frequency to achieve better optimized performances.

For the future design of multi-layer DRA, the most important issue is to complete the launcher section design since a perfect launcher structure is the only part that is not optimized so far. By further optimizing the angle of pyramid structure and the shape of isosceles triangle metals, it should able to complete the desired feeding structure. Besides, a new feed design is also important since the current pyramid shape feeding structure is difficult to fabricate especially when the number of layers increases for wider bandwidth. Several planer wide-band structures, such as bowtie antenna and log-periodic antenna, have the potential to be a feeding structure for a UWB multi-layer DRA but these design haven't been studied yet. If new planer feeding structure can be discovered, the complexity of the fabrication procedure will be greatly reduced.

Reference

[1] K. Siwiak, “Ultra-wideband radio: Introducing a new technology,” in *Proc. 2001 Spring Vehicular Technology Conf.*, 2001, pp. 1088–1093

[2] T. Milligan, “A Design Study for the Basic TEM Horn Antenna,” *IEEE Antennas and Propagation Magazine*, vol.46, no.1, pp. 86-92, Feb. 2004.

[3] Kaczmarski, K.J.; Laxpati, S.R.; , "Design and development of a square coaxial waveguide-fed horn antenna with equal E - and H - plane beamwidth," *Antennas and Propagation Society International Symposium, 2007 IEEE* , vol., no., pp.5672-5675, 9-15 June 2007

[4] Zhongxiang Shen; Chao Feng; , "A new dual-polarized broadband horn antenna," *Antennas and Wireless Propagation Letters, IEEE* , vol.4, no., pp. 270- 273, 2005

[5] Barbano, N., "Phase center distributions of spiral antennas," *WESCON/60 Conference Record* , vol.4, no., pp. 123- 130, Aug 1960

[6] Kwan-Ho Lee; Chi-Chih Chen; Lee, R.; , "Development of UWB, dual-polarized dielectric horn antenna (DHA) for UWB applications," *Antennas and Propagation Society International Symposium, 2004. IEEE* , vol.3, no., pp. 2931- 2934 Vol.3, 20-25 June 2004

[7] R.B. Watson and C.W. Horton, “The radiation Pattern of Dielectric Rod – Experiment and Theory,” J. of Applied Physics, vol 19, pp.661-670, 1948

[8] Ando, T.; Yamauchi, J.; Nakano, H.; , "Demonstration of the discontinuity-radiation concept for a dielectric rod antenna," *Antennas and Propagation Society International Symposium, 2000. IEEE* , vol.2

[9] Qiu Jing-hui; Wang Nan-nan; , "Optimized dielectric rod antenna for millimeter wave FPA imaging system," *Imaging Systems and Techniques, 2009. IST '09. IEEE International Workshop on* , vol., no., pp.147-150, 11-12 May 2009

[10] Jae-Young Chung; Chi-Chih Chen; , "Two-Layer Dielectric Rod Antenna," *Antennas and Propagation, IEEE Transactions on* , vol.56, no.6, pp.1541-1547, June 2008

[11] Chi-Chih Chen; Kishore Rama Rao; Lee, R.; , "A new ultrawide-bandwidth dielectric-rod antenna for ground-penetrating radar applications," *Antennas and Propagation, IEEE Transactions on* , vol.51, no.3, pp. 371- 377, March 2003

[12] Ala, R.; Sadeghzadeh, R.; Kazemi, R.; , "Two-layer dielectric rod antenna for far distance," *Antennas and Propagation Conference (LAPC), 2010 Loughborough* , vol., no., pp.149-152, 8-9 Nov. 2010

[13] K.-H. Lee, C.-C. Chen, and R. Lee, "UWB dual-linear polarization dielectric horn antennas as reflector feeds," *IEEE Trans. Antennas Propag.*, vol. 55, pp. 798–804, 2007.

[14] Yeh, C., and Fred I.; "The essence of dielectric waveguides", *New York: Springer Verlag*, 2008.

[15] Salema, Carlos, Carlos Fernandes, and Rama Kant. "Solid dielectric horn antennas". *Artech House,* 1998.

[16] Richter, J.; Muller, M.; Schmidt, L.-P.; , "Measurement of phase centers of rectangular dielectric rod antennas," *Antennas and Propagation Society International, Symposium, 2004. IEEE* , vol.1, no., pp. 743- 746 Vol.1, 20-25 June 2004

[17] H. -T. Chou, C. -W. Liu, W. -J. Liao, "Optimum Horn Antenna Design based on an Integration of HFSS Commercial Code and Genetic Algorithms for the Feed Application of Reflector Antennas," *ACES Journal*, Volume 25**,** Number 2 **,** 2010

[18] Frenzel, J.F.; , "Genetic algorithms," *Potentials, IEEE* , vol.12, no.3, pp.21-24, Oct 1993

[19] Johnson, J.M.; Rahmat-Samii, V.; "Genetic algorithms in engineering electromagnetics," *Antennas and Propagation Magazine, IEEE* , vol.39, no.4, pp.7-21, Aug 1997

[20] R.T. Marler and J.S. Arora; "Survey of multi-objective optimization methods for engineering", *Structural and Multidisciplinary Optimization, Springer Berlin / Heidlberg*, vol.26, issue 6, pp. 3690395, 2004

Appendix A

BCB Code of Genetic Algorithm

```
#include <vcl.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TGA *GA;
time_t t;
//---------------------------------------------------------------------------
__fastcall TGA::TGA(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
AnsiString Setup = "C:\\Optima\\temp\\Setup.txt";    // Store Basic setting
AnsiString MAXDATA = "C:\\Optima\\temp\\Max.txt";    // Store maximum of each Parameters
AnsiString MINDATA = "C:\\Optima\\temp\\Min.txt";    //   ;   min          ;
AnsiString BITDATA = "C:\\Optima\\temp\\Bit.txt";    //  Bit of each parameters ( for binary )
AnsiString GENDATA = "C:\\Optima\\temp\\Gen.txt";    //Generation counting!
AnsiString READY = "C:\\Optima\\temp\\READY.txt";    //Generation counting!

AnsiString BinaryT = "C:\\Optima\\temp\\BT.tmp";
AnsiString DecimalT = "C:\\Optima\\temp\\MT.tmp";
AnsiString poplistB = "C:\\Optima\\temp\\pop\\Binary\\p";   // temp data
AnsiString poplistD = "C:\\Optima\\temp\\pop\\Decimal\\p";
AnsiString poplistR = "C:\\Optima\\temp\\pop\\RealNum\\p";
AnsiString FITNESS = "C:\\Optima\\temp\\FITNESS\\FIT.txt";

AnsiString poplistBG = "C:\\Optima\\temp\\pop\\Binary\\BGen\\Gen";
AnsiString poplistRG = "C:\\Optima\\temp\\pop\\realnum\\RGen\\Gen";
AnsiString poplistDG = "C:\\Optima\\temp\\pop\\Decimal\\DGen\\Gen";
```

```
int parameters, pop, gen;       //GA Parameters
float mu, cro;                  //GA Parameters
int format;  // format=2 for Binary , 10 for deciaml format

int wait=0;
void __fastcall TGA::BinaryCClick(TObject *Sender)
{
   parameters = StrToInt(Parameters->Text);
   pop = StrToInt(Population->Text);
   Label1->Visible = true;
   Label2->Visible = true;
   Label3->Visible = true;
   DecimalC->Enabled = false;
   Max->Enabled = true;
   Min->Enabled = true;
   Bits->Enabled = true;
   format = 2;
   Save->Enabled = true;
   MemoAdd->Click();
}
//---------------------------------------------------------------------------
void __fastcall TGA::Timer1Timer(TObject *Sender)
{
   //Check if all the required blocks are filled
   if( Parameters->Text == "" || Generation->Text == "" || Population->Text == "" || Crossover->Text == ""
|| Mutation->Text =="" )
   {
      BinaryC->Enabled = false;
      DecimalC->Enabled = false;
   }
   else
   {
      BinaryC->Enabled = true;
      DecimalC->Enabled = true;
      Timer1->Enabled = false;
   }
}
//---------------------------------------------------------------------------
void __fastcall TGA::MemoAddClick(TObject *Sender)
{
   for ( int i = 1; i <= parameters ; i++ )
   {
      Max->Lines->Add("M"+IntToStr(i)+":");
      Max->Lines->Add(" ");
      Min->Lines->Add("m"+IntToStr(i)+":");
      Min->Lines->Add(" ");
      if( format == 2 )
      {
         Bits->Lines->Add("B"+IntToStr(i)+":");
         Bits->Lines->Add(" ");
      }
   }
}
```

```
//---------------------------------------------------------------------------
void __fastcall TGA::SaveClick(TObject *Sender)
{
   Max->Lines->SaveToFile(MAXDATA);
   Min->Lines->SaveToFile(MINDATA);
   Bits->Lines->SaveToFile(BITDATA);
   BackProcess->Lines->Add("Format:");
   BackProcess->Lines->Add(IntToStr(format));
   BackProcess->Lines->Add("Parameter Num:");
   BackProcess->Lines->Add(Parameters->Text);
   BackProcess->Lines->Add("Generations:");
   BackProcess->Lines->Add(Generation->Text);
   BackProcess->Lines->Add("Populations:");
   BackProcess->Lines->Add(Population->Text);
   BackProcess->Lines->Add("Crossover:");
   BackProcess->Lines->Add(Crossover->Text);
   BackProcess->Lines->Add("Mutation:");
   BackProcess->Lines->Add(Mutation->Text);
   BackProcess->Lines->SaveToFile(Setup);

   Max->Enabled = false;
   Min->Enabled = false;
   Bits->Enabled = false;
   Start->Enabled = true;
   Save->Enabled = false;
}
//---------------------------------------------------------------------------
void __fastcall TGA::Timer2Timer(TObject *Sender)
{
  //Random number generator #1

  int randN = 10000;
  if( BackProcess2->Lines->Capacity < randN )
  {
     srand((unsigned) time(&t));
     for( int i = 1 ; i <= 3 ; i++)
     {
        int div;
        float ddiv;
        div = pow(10,i);
        ddiv = div/(2*1.47);
        if( (rand()*rand()+rand()) % div > ddiv )
        {
           BackProcess2->Lines->Add(IntToStr(1));
        }
        else
        {
           BackProcess2->Lines->Add(IntToStr(0));
        }
     }
  }
  if( BackProcess10->Lines->Capacity < randN )
  {
     srand((unsigned) time(&t));
```

```
        BackProcess10->Lines->Add( rand()%1000 );
        BackProcess10->Lines->Add( (rand()+1)%1000 );
        BackProcess10->Lines->Add( (rand()+2)%1000 );
        BackProcess10->Lines->Add( (rand()+3)%1000 );
        BackProcess10->Lines->Add( (rand()+4)%1000 );
    }
}
//---------------------------------------------------------------------------
void __fastcall TGA::Timer3Timer(TObject *Sender)
{
    //Random number generator #2

    int randN = 10000;
    if( BackProcess2->Lines->Capacity < randN )
    {
        srand((unsigned) time(&t));
        for( int i = 1 ; i <= 2 ; i++)
        {
            int div;
            float ddiv;
            div = pow(10,i);
            ddiv = div/(2*1.47);
            if( (rand()*rand()*rand()+rand()) % div > ddiv )
            {
                BackProcess2->Lines->Add(IntToStr(1));
            }
            else
            {
                BackProcess2->Lines->Add(IntToStr(0));
            }
        }
    }
    if( BackProcess10->Lines->Capacity < randN )
    {
        srand((unsigned) time(&t));
        BackProcess10->Lines->Add( (rand()-1)%1000 );
        BackProcess10->Lines->Add( (rand()-2)%1000 );
        BackProcess10->Lines->Add( (rand()-3)%1000 );
        BackProcess10->Lines->Add( (rand()-4)%1000 );
        BackProcess10->Lines->Add( (rand()-5)%1000 );
    }
}
//---------------------------------------------------------------------------
void __fastcall TGA::StartClick(TObject *Sender)
{
    // this subroutine is used to generate the population! (first generation)
    AnsiString *temp;
    BackProcess->Clear();   // clear the MEMO; memo provides a place for data processing
    temp = new AnsiString[parameters];
    if( format == 2 )     // binary set
    {
        for( int p = 1 ; p <= pop ; p++ )    // generate enough population
        {
            for( int i = 0 ; i < parameters ; i++ )  // generate enough parameters in each population
```

```
      {
        temp = new AnsiString[parameters];    // binary set '100001'..... whatever :)
        for( int k = 0 ; k < StrToInt(Trim(Bits->Lines->operator [](2*i+1))) ; k++ )
        {
          if( BackProcess2->Lines->operator [](0) != "" )
          {
            *(temp+i) = *(temp+i) + BackProcess2->Lines->operator [](0);
            BackProcess2->Lines->Delete(0);
          }
          else if ( BackProcess2->Lines->operator [](0) == "" && BackProcess10->Lines->operator
[](0) != ""  )
          {
            if( (StrToInt(BackProcess10->Lines->operator [](0)) + rand())%1000 > 500 )
            {
              *(temp+i) = *(temp+i) + "1";
            }
            else
            {
              *(temp+i) = *(temp+i) + "0";
            }
            BackProcess10->Lines->Delete(0);
          }
          else
          {
            srand((unsigned) time(&t));
            if( rand()*rand() % 1000 > 500 )
            {
              *(temp+i) = *(temp+i) + "1";
            }
            else
            {
              *(temp+i) = *(temp+i) + "0";
            }
            Sleep(500);
          }
        }
       BackProcess->Lines->Add(*(temp+i));
       *(temp+i) = "";
      }
     BackProcess->Lines->SaveToFile(poplistB+IntToStr(p)+".txt");   // Save each population to file for
futher use
     BackProcess->Clear();    // Memo clear!
    }
  }
  TransToR->Click();
  delete[] temp;
  BackProcess->Clear();
  BackProcess->Lines->SaveToFile(READY);
}
//---------------------------------------------------------------------------
void __fastcall TGA::TransToRClick(TObject *Sender)
{
  // Reset the basic parameters
  float *max, *min;
```

```
    BackProcess->Clear();
    BackProcess->Lines->LoadFromFile(Setup);
    format = StrToInt(BackProcess->Lines->operator [](1));
    parameters = StrToInt(BackProcess->Lines->operator [](3));
    pop = StrToInt(BackProcess->Lines->operator [](7));
    max = new float[parameters];
    min = new float[parameters];
    BackProcess->Lines->LoadFromFile(MAXDATA);
    for( int i = 0 ; i < parameters ; i++ )
    {
       *(max+i) = StrToFloat(Trim(BackProcess->Lines->operator [](2*i+1)));
    }
    BackProcess->Lines->LoadFromFile(MINDATA);
    for( int i = 0 ; i < parameters ; i++ )
    {
       *(min+i) = StrToFloat(Trim(BackProcess->Lines->operator [](2*i+1)));
    }

    // this subroutine is used to transfer the Pop to the real number
    AnsiString temp;
    float count;
    if( format == 2 )  // binary set
    {
       for( int p = 1 ; p <= pop ; p++ )
       {
          BackProcess->Lines->LoadFromFile(poplistB+IntToStr(p)+".txt");
          for( int pa = 1 ; pa <= parameters ; pa++ )
          {
             temp = BackProcess->Lines->operator [](0);
             count = 0;
             for( int l = 0 ; l < temp.Length() ; l++ )
             {
                if( temp.SubString(l+1,1) == "1" )          // transfer the '1001' -> '9' (binary to Decimal)
                {
                   count = count + pow(2,temp.Length()-l-1);
                }
             }
             BackProcess->Lines->Delete(0);
             count = count / (pow(2,temp.Length())-1);
             // Normalize the parameter
             count = count * (*(max+pa-1)-*(min+pa-1)) + (*(min+pa-1));
             // Set to the region of required
             BackProcess->Lines->Add(FloatToStr(count));
          }
          BackProcess->Lines->SaveToFile(poplistR+IntToStr(p)+".txt");
             //Save it!
       }
    }
    Timer4->Enabled = true;
    delete[] max, min;
}
//---------------------------------------------------------------------------
void __fastcall TGA::Timer4Timer(TObject *Sender)
{
    // This timer is used to check if the FITNESS file is Created !!
    // During the Waiting period, the GA program will be stupor!
    if(!FileExists(FITNESS))
    {
       GA->Enabled = false;
       Label4->Caption = "Waiting for calculating fitness number";
    }
    else
    {
        GA->Enabled = True;
        Timer4->Enabled = false;
        Algorithm->Click(); //GA
    }
}
//---------------------------------------------------------------------------
int tournament = 4;
void __fastcall TGA::AlgorithmClick(TObject *Sender)
{
    BackProcess->Clear();
// Back up the population of the last generation and DO GENETIC ALGORITHM
    BackProcess->Lines->LoadFromFile(Setup);           //Reload data
    format = StrToInt(BackProcess->Lines->operator [](1));
    parameters = StrToInt(BackProcess->Lines->operator [](3));
    pop = StrToInt(BackProcess->Lines->operator [](7));

    cro = StrToFloat(BackProcess->Lines->operator [](9));
    mu = StrToFloat(BackProcess->Lines->operator [](11));

    if(FileExists(GENDATA))
    {
       BackProcess->Lines->LoadFromFile(GENDATA);
       gen = StrToInt(BackProcess->Lines->operator [](0));            //genertation
    }
    else
    {
       gen = 1;
       BackProcess->Clear();
    }
    BackProcess->Clear();                        // change generation data
    BackProcess->Lines->Add(IntToStr(gen+1));
    BackProcess->Lines->SaveToFile(GENDATA);
    BackProcess->Clear();

    int *sel, *sel1, posM, prob;                 // parasmeters for selection method
    float *sel2, probf;                    // used for identify the crossover and mutation paarameter
    AnsiString *sel3, *sel4;         // string processing parameter

    sel = new int[2];
    sel1 = new int[tournament];
    sel2 = new float[tournament];
    sel3 = new AnsiString[4];
    sel4 = new AnsiString[2];

    if( format == 2 )
    {
       int *bit;
        // Backup the old data !
       bit = new int[parameters];
       // how many parameters in a population
       BackProcess->Lines->LoadFromFile(BITDATA);

       for( int i = 0 ; i < parameters ; i++ )
       {
          *(bit+i) = StrToFloat(Trim(BackProcess->Lines->operator [](2*i+1)));
          // reload the BITS information
       }
       // how many bits contain in a parameter !

       BackProcess->Lines->Clear();
       for( int p = 1 ; p<= pop ; p++ )              // back up the binary data for each generation
       {
          BackProcess1->Lines->LoadFromFile(poplistB+IntToStr(p)+".txt");
         // Store the old generation
          BackProcess->Lines->Add("-----Population : " + IntToStr(p) + " -----");
          for( int pa = 1 ; pa <= parameters ; pa++ )
          {
             BackProcess->Lines->Add(BackProcess1->Lines->operator [](pa-1));
          }
       }
       BackProcess->Lines->SaveToFile(poplistBG+IntToStr(gen)+".txt");
       BackProcess->Clear();
       BackProcess1->Clear();

       for( int p = 1 ; p<= pop ; p++ )
       //back up the real number data for each generation
       {
          BackProcess1->Lines->LoadFromFile(poplistR+IntToStr(p)+".txt");
          // Store the old generation
          BackProcess->Lines->Add("-----Population : " + IntToStr(p) + " -----");
          for( int pa = 1 ; pa <= parameters ; pa++ )
          {
             BackProcess->Lines->Add(BackProcess1->Lines->operator [](pa-1));
          }
       }
       BackProcess->Lines->SaveToFile(poplistRG+IntToStr(gen)+".txt");
       BackProcess->Clear();
       BackProcess1->Clear();

       // doing selection!
       for( int p = 1 ; p <= pop/2 ; p++ )
 // Population should be even!
       {
// Select two population and find the better one (twice); Select the better one
          for( int j = 0 ; j < 2 ; j++ )
// in these two; Do the procedure twice and find the both parents
          {
 // Choose 1 from 2 Selection; do it twice to generate the parents.
             for( int i = 0 ; i < tournament ; i++ )
```

```
            {
               if( BackProcess10->Lines->Capacity < 100 )
// if run out of the random sequence ! generate it!
               {
                  for( int r = 0 ; r< 10 ; r++ )
                  {
                     srand((unsigned) time(&t));
                     BackProcess10->Lines->Add( (rand()+rand()) % 1000);
                     BackProcess10->Lines->Add( (rand()*rand()) % 1000);
                     BackProcess10->Lines->Add( (rand()*rand()+rand()) % 1000);
                     BackProcess10->Lines->Add( (rand()+rand()+rand()) % 1000);
                     BackProcess10->Lines->Add( (rand()) % 1000);
                     Sleep(750);
                  }
               }

               (*(sel1+i)) = StrToInt(BackProcess10->Lines->operator [](0));
               (*(sel1+i)) = ((*(sel1+i))%pop)+1;                          // Select #1

               BackProcess10->Lines->Delete(0);

               if( i != 0 )
               {
                  for ( int k = 0 ; k<= i-1 ; k++ )
                  {
                     if(  (*(sel1+i)) ==  (*(sel1+k)) )
                     {
                        i--;
                        k = i;
                     }
                  }
               }
            }
            BackProcess->Lines->LoadFromFile(FITNESS);       // load the fitness file
            posM = 0;
            for( int i = 0 ; i < tournament; i++ )
            {
               *(sel2+i) =  StrToFloat(BackProcess->Lines->operator []((*(sel1+i))-1));
 // load the fitness value of the selection
               if( i!=0 )
               {
                  if( (*(sel2+posM)) > (*(sel2+i)) )
                  {
                     posM = posM;
                  }
                  else
                  {
                     posM = i;
                  }
               }
            }
            (*(sel+j)) = (*(sel1+posM)) ;
         }
```

```
        BackProcess->Lines->LoadFromFile(poplistB+IntToStr( (*(sel+0)) ) +".txt");      // load file
contained the gene of the selection
        BackProcess1->Lines->LoadFromFile(poplistB+IntToStr( (*(sel+1)) ) +".txt");     // Same  ;

        int tran;

        for( int i = 0 ; i < parameters ; i++ )
        {
           if( BackProcess10->Lines->Capacity < 100 )
// if run out of the random sequence ! generate it!
           {
              for( int r = 0 ; r< 10 ; r++ )
              {
                 srand((unsigned) time(&t));
                 BackProcess10->Lines->Add( (rand()+rand()) % 1000);
                 BackProcess10->Lines->Add( (rand()*rand()) % 1000);
                 BackProcess10->Lines->Add( (rand()*rand()+rand()) % 1000);
                 BackProcess10->Lines->Add( (rand()+rand()+rand()) % 1000);
                 BackProcess10->Lines->Add( (rand()) % 1000);
                 Sleep(750);
              }
           }
           prob = StrToInt( BackProcess10->Lines->operator [](0) );
           probf = prob;
           probf = probf/1000;
           BackProcess10->Lines->Delete(0);  // delete used random number

           if( (StrToFloat(BackProcess10->Lines->operator [](0))/1000) < cro )
 // do crossover       ( if crossover )
           {
              BackProcess10->Lines->Delete(0);
// delete used random number
              *(sel3+0) = BackProcess->Lines->operator [](0);
 // load the gene information
              *(sel3+1) = BackProcess1->Lines->operator [](0);

              BackProcess->Lines->Delete(0);
              BackProcess1->Lines->Delete(0);

              tran = (StrToInt(BackProcess10->Lines->operator [](0))%((*(bit+i))+1));    // create the part of
crossover
              BackProcess10->Lines->Delete(0);  // delete used random number

              *(sel4+0) =  (*(sel3+0)).SubString(1,tran) ;
 // Create new gene   #CROSSOVER #1
              *(sel4+1) =  (*(sel3+1)).SubString(tran+1,(*(bit+i))-tran);
              (*(sel3+2)) = (*(sel4+0)) + (*(sel4+1));

              *(sel4+0) =  (*(sel3+1)).SubString(1,tran) ;
// Create new gene    #CORSSOVER #2
              *(sel4+1) =  (*(sel3+0)).SubString(tran+1,(*(bit+i))-tran);
              (*(sel3+3)) = (*(sel4+0)) + (*(sel4+1));

           }
```

```
            else
            {
               BackProcess10->Lines->Delete(0);
               (*(sel3+2)) = BackProcess->Lines->operator [](0);
               (*(sel3+3)) = BackProcess1->Lines->operator [](0);
               BackProcess->Lines->Delete(0);
               BackProcess1->Lines->Delete(0);
            }

            if( (StrToFloat(BackProcess10->Lines->operator [](0))/1000) < mu )
// mutation   #1
            {
               BackProcess10->Lines->Delete(0);
               tran = (StrToInt(BackProcess10->Lines->operator [](0))%(*(bit+i)));
               tran = tran+1;
               if( (*(sel3+2)).SubString(tran,1) == "1" )
               {
                  (*(sel3+2)).Delete(tran,1);
                  (*(sel3+2)).Insert("0",tran);
               }
               else
               {
                  (*(sel3+2)).Delete(tran,1);
                  (*(sel3+2)).Insert("1",tran);
               }
            }

            if( (StrToFloat(BackProcess10->Lines->operator [](0))/1000) < mu )
// mutation   #2
            {
               BackProcess10->Lines->Delete(0);
               tran = (StrToInt(BackProcess10->Lines->operator [](0))%(*(bit+i)));
               tran = tran+1;  // tran should between 1-genes
               if( (*(sel3+3)).SubString(tran,1) == "1" )
               {
                  (*(sel3+3)).Delete(tran,1);
                  (*(sel3+3)).Insert("0",tran);
               }
               else
               {
                  (*(sel3+3)).Delete(tran,1);
                  (*(sel3+3)).Insert("1",tran);
               }
            }
            BackProcess->Lines->Add( (*(sel3+2)) );
            BackProcess1->Lines->Add( (*(sel3+3)) );
         }


         BackProcess->Lines->SaveToFile(poplistBG+IntToStr(2*p-1)+".dll");
         BackProcess->Clear();

         BackProcess1->Lines->SaveToFile(poplistBG+IntToStr(2*p)+".dll");
         BackProcess1->Clear();
```

```
        }
        delete[] bit;

        for( int p = 1 ; p <= pop ; p++ )
        {
           BackProcess->Lines->LoadFromFile(poplistBG+IntToStr(p)+".dll");
           BackProcess->Lines->SaveToFile(poplistB+IntToStr(p) +".txt");
           DeleteFile(poplistBG+IntToStr(p)+".dll");
           BackProcess->Clear();
        }
     }
//-------------------------------
     if(FileExists(FITNESS))
     {
        BackProcess->Lines->LoadFromFile(FITNESS);
        BackProcess->Lines->SaveToFile("C:\\Optima\\temp\\FITNESS\\FIT"+IntToStr(gen)+".txt");
        DeleteFile(FITNESS);
     }
     TransToR->Click();
     delete[] sel, sel1, sel2, sel3, sel4;
     BackProcess->Lines->SaveToFile(READY);
}
//---------------------------------------------------------------------------
```

Appendix B

BCB Code of Master Program

```
#include <vcl.h>
#include <iostream>
#include <cmath>
#pragma hdrstop
#include "Unit1.h"
#include "math.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
AnsiString Setup = "C:\\Optima\\temp\\Setup.txt";    // Store Basic setting
AnsiString poplistR = "C:\\Optima\\temp\\pop\\RealNum\\p";
AnsiString poplistB = "C:\\Optima\\temp\\pop\\Binary\\p";
AnsiString FITNESS = "C:\\Optima\\temp\\FITNESS\\FIT.txt";
AnsiString FITNESS1 = "C:\\Optima\\temp\\FITNESS\\FIT";
AnsiString READY= "C:\\Optima\\temp\\ready.txt";
AnsiString Result = "C:\\Optima\\temp\\Result\\temp.csv";
AnsiString Result_1 = "C:\\Optima\\temp\\Result\\temp1.csv";
AnsiString Result_2 = "C:\\Optima\\temp\\Result\\temp2.csv";
AnsiString Result_3 = "C:\\Optima\\temp\\Result\\temp3.csv";
AnsiString Result_a = "C:\\Optima\\temp\\Result\\tempa.csv";
AnsiString Result_1a = "C:\\Optima\\temp\\Result\\temp1a.csv";
AnsiString Result_2a= "C:\\Optima\\temp\\Result\\temp2a.csv";
AnsiString Result_3a = "C:\\Optima\\temp\\Result\\temp3a.csv";
AnsiString ResultS = "C:\\Optima\\temp\\Result\\temp";
AnsiString ResultS_1 = "C:\\Optima\\temp\\Result\\temp1";
AnsiString ResultS_2 = "C:\\Optima\\temp\\Result\\temp2";
AnsiString ResultS_3 = "C:\\Optima\\temp\\Result\\temp3";
AnsiString ResultS_a = "C:\\Optima\\temp\\Result\\tempa";
AnsiString ResultS_1a = "C:\\Optima\\temp\\Result\\temp1a";
AnsiString ResultS_2a = "C:\\Optima\\temp\\Result\\temp2a";
AnsiString ResultS_3a = "C:\\Optima\\temp\\Result\\temp3a";
AnsiString VBS = "C:\\Optima\\temp\\VBS\\";
int pop, parameterN;
AnsiString *name;
int Gen = 1, PopC = 1, reportN = 4;   //Report Number!!
float fitness_pattern =0 ;  // commute the data
int layer = 6;
float  Er_i, Er_r, Lambda_r, H_r;  // number of layers ( need to be optimaized later on )
float  fitness_height, Height_limit = 5;

void __fastcall TForm1::B1Click(TObject *Sender)     // Initial Setting (Asking the parameter name!
{
   VBSMemo->Lines->LoadFromFile(Setup);     // loading the setup from file
   pop = StrToInt(VBSMemo->Lines->operator [](7));  //population
   parameterN = StrToInt(VBSMemo->Lines->operator [](3));   //Number of parameter
   name = new AnsiString[parameterN];     // Array that store name of parameter
   VBSMemo->Lines->Clear();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::B2Click(TObject *Sender)     //VBS Creating
{              // pop=1;

   int datap = 6;
   int layers = 4;
   float datapf;
   AnsiString datap2, datap3;
   for ( int vbsC = 1; vbsC <= 2; vbsC++ )
   {
   for ( int i = 1 ; i <= pop ; i++ )     // generate n-VBscript
   {
      VBSMemo->Lines->Clear();
      VariName->Lines->Clear();
      VariName->Lines->LoadFromFile(poplistR+IntToStr(i)+".txt");
 //load the parameter
      fitness_H->Lines->LoadFromFile(poplistB+IntToStr(i)+".txt");
      // how many globe and local
      // say i need r value, h value, er value
      // this three should be calculated by the radnom unmber i have.!
      // globe variable $Er
      float *Er, *radius, *h, *ra, *height, *rax;
      Er = new float[layers];
      radius = new float[layers];
      ra = new float[layers];
      rax = new float[layers];
      h = new float[1];
      height = new float[1];

      *(Er+layers-1) = 2.2;  //fixed er!

      for( int j = 1 ; j  < layers ; j++ )
```

```
        {
          if( j == 1)
          {
            *(Er+layers-1-j) = (*(Er+layers-1-j+1)) + StrToFloat(VariName->Lines->operator [](j-1))-0.2;
// load the parameter!
          }
          else
          {
            *(Er+layers-1-j) = (*(Er+layers-1-j+1)) + StrToFloat(VariName->Lines->operator [](j-1));
          }
          VBSMemo->Lines->Add(fitness_H->Lines->operator [](j));
        }

        for( int j = 0; j<layers ; j++ )
        {
           *(radius+j) = StrToFloat(VariName->Lines->operator [](layers-1+j));
        }
        for( int j = 0; j<layers ; j++ )
        {
           *(ra+j) = StrToFloat(VariName->Lines->operator [](2*layers-1+j));
        }
        for( int j = 0; j<layers ; j++ )
        {
           *(rax+j) = StrToFloat(VariName->Lines->operator [](3*layers-1+j));
        }
        *(h+0) = 2.5;
        VariName->Lines->Clear();
        fitness_H->Lines->Clear();
        VBSMemo->Lines->Clear();

        //VBS Generator ! (XD)

        for ( int j =0; j<= InitialMemo->Lines->Capacity ; j++ )         //Initial VBS
        {
          if( j!= 10 || vbsC == 1)
          {
            VBSMemo->Lines->Add(InitialMemo->Lines->operator [](j));
          }
          else
          {
         // ShowMessage("a");
            datap2 = InitialMemo->Lines->operator [](j);
            datap3 = datap2.Delete(datap2.Length()-2,1);
            datap2 = datap3.Insert("2",datap3.Length()-1);
            VBSMemo->Lines->Add(datap2);
                //      ShowMessage(datap2);
          }
        }

        for ( int l = 0 ; l < (layers)*4+1 ; l++ )                        // Variable VBS
        {
          if ( l < layers )                         //globe variable
          {
            for ( int j =0; j< VariableMemoG->Lines->Capacity ; j++ )
          {
            if( j < VariableMemoG->Lines->Capacity-1 )
            {
              VBSMemo->Lines->Add(VariableMemoG->Lines->operator [](j));
            }
            else
            {
              AnsiString t1 = VariableMemoG->Lines->operator [](j);
 // until the last line
              int j1 = t1.Pos("$");
              AnsiString t3 = t1.Delete(j1,1);
              AnsiString t2 = t3.Insert("$Er"+IntToStr(l+1),j1);
// put the parameter name
              j1 = t2.Pos("Variable");
              t1 = t2.Delete(j1,8);

              datap = 6;
              datap2 = FloatToStr( (*(Er+l) )).SubString(datap,1);
              for( int k = 6 ; k>0; k-- )
              {
                if( datap2.SubString(k,1)== "0" )
                {
                    datap--;
                }
                else
                {
                  k =0;
                }
              }
              t2 = t1.Insert(FloatToStr( (*(Er+l) )).SubString(1,datap),j1);    // insert the value
              VBSMemo->Lines->Add(t2);
              VariName->Lines->Add(FloatToStr( (*(Er+l) )).SubString(1,datap));
            }
          }
        }

        if ( l>=layers )                              // local variable
        {
          for ( int j =0; j< VariableMemo->Lines->Capacity ; j++ )
          {
            if( j < VariableMemo->Lines->Capacity-1 )
            {
              VBSMemo->Lines->Add(VariableMemo->Lines->operator [](j));
            }
            else
            {
              AnsiString t1 = VariableMemo->Lines->operator [](j);
              int j1 = t1.Pos("$");
              AnsiString t3 = t1.Delete(j1,1);
              AnsiString t2 ;
              if( l < 2*layers )
              {
                t2 = t3.Insert("r"+IntToStr(l-(layers-1)),j1);
              }
```

```
else if( l < 3*layers && l > 2*layers-1)
{
   t2 = t3.Insert("ra"+IntToStr(l-(2*layers-1)),j1);
}
else if( l < 4*layers && l > 3*layers-1)
{
   t2 = t3.Insert("rx"+IntToStr(l-(3*layers-1)),j1);
}
else
{
    t2 = t3.Insert("height",j1);
}
j1 = t2.Pos("Variable");
t1 = t2.Delete(j1,8);
if( l != layers*4 )
{
   datap = 6;
   if( l< 2*layers)
   {
     datap3 = FloatToStr( (*(radius+l-layers) ));
   }
   if( l<3*layers && l > 2*layers-1)
   {
     datap3 = FloatToStr( (*(ra+l-layers*2) ));
   }
   if( l<4*layers && l > 3*layers-1)
   {
     datap3 = FloatToStr( (*(rax+l-3*layers) ));
   }
   for( int k = 6 ; k>0; k-- )
   {
      if( datap3.SubString(k,1) == "0" )
      {
          datap--;
      }
      else
      {
         k =0;
      }
   }
   if( l < 2*layers )
   {
      t2 = t1.Insert( datap3.SubString(1,datap)+" in",j1);
   }
   else
   {
      t2 = t1.Insert( datap3.SubString(1,datap),j1);
   }
   VariName->Lines->Add(datap3.SubString(1,datap));
}
else
{
   datapf = 0;
   for( int m = 0 ; m < layers ; m++ )
```

```
                        {
                          datapf = datapf + StrToFloat(VariName->Lines->operator [](layers+m));
                          *(height+0) = datapf * StrToFloat(VariName->Lines->operator [](layers*2+m));
                          if ( m==0 )
                          {
                            *(h+0) = datapf * StrToFloat(VariName->Lines->operator [](layers*2+m));
                          }
                          else
                          {
                            if ( *(h+0) <  *(height+0) )
                            {
                              *(h+0)  =  *(height+0);
                            }
                          }
                          //ShowMessage(FloatToStr(*(h+0)));
                        }
                        datap = 6;
                        for( int k = 6 ; k>0; k-- )
                        {
                          if( FloatToStr( (*(h+0)) ).SubString(k,1) == "0" )
                          {
                              datap--;
                          }
                          else
                          {
                            k =0;
                          }
                        }
                        t2 = t1.Insert( FloatToStr( (*(h+0)) ).SubString(1,datap)+" in",j1);
                        VariName->Lines->Add( FloatToStr( (*(h+0))).SubString(1,datap));
                      }
                      VBSMemo->Lines->Add(t2);
                    }
                  }
                }

              }
              delete[] Er, radius, h ;

              for ( int j =0; j< SaveAndRunMemo->Lines->Capacity ; j++ )          //Initial VBS (save and run
        commend)
              {
                if( j< SaveAndRunMemo->Lines->Capacity-1 )
                {
                  if(vbsC==1)
                  {
                    VBSMemo->Lines->Add(SaveAndRunMemo->Lines->operator [](j));
                  }
                  else
                  {
                    if( j== SaveAndRunMemo->Lines->Capacity-3 || j== SaveAndRunMemo->Lines->Capacity-6)
                    {
                      datap2 = SaveAndRunMemo->Lines->operator [](j);
                      datap3 = datap2.Delete(datap2.Length()-2,1);
```

```
            datap2 = datap3.Insert("2",datap3.Length()-1);
            VBSMemo->Lines->Add(datap2);
        }
        else
        {
            VBSMemo->Lines->Add(SaveAndRunMemo->Lines->operator [](j));
        }
    }
}
else
{
        AnsiString t1;
        AnsiString t2;
        int j1;
        for ( int rep = 1; rep <= reportN; rep++ )
        {
            t1 = SaveAndRunMemo->Lines->operator [](j);
            j1 = t1.Pos(".csv");                    // report export (XY plot) !

            if ( vbsC == 1 )
            {
            if(rep == 1)
            {
                t2 = t1.Insert(ResultS,j1);         // direction of file to save
            }
            else if (rep ==2)
            {
                t2 = t1.Insert(ResultS_1,j1);
            }
            else if (rep ==3)
            {
                t2 = t1.Insert(ResultS_2,j1);
            }
            else if (rep ==4)
            {
                t2 = t1.Insert(ResultS_3,j1);
            }
            }

            else
            {
            if(rep == 1)
            {
                t2 = t1.Insert(ResultS_a,j1);         // direction of file to save
            }
            else if (rep ==2)
            {
                t2 = t1.Insert(ResultS_1a,j1);
            }
            else if (rep ==3)
            {
                t2 = t1.Insert(ResultS_2a,j1);
            }
            else if (rep ==4)
            {
              t2 = t1.Insert(ResultS_3a,j1);
            }
            }
            j1 = t2.Pos("*");
            t1 = t2.Delete(j1,1);
            t2 = t1.Insert(IntToStr(rep),j1);
            VBSMemo->Lines->Add(t1);
          }
      }
  }
  for ( int j =0; j< DeleteMeshMemo->Lines->Capacity ; j++ )          //Initial VBS (clear result)
  {
      AnsiString t1;
      AnsiString t2;
      int j1;

      if( j == 0 )
      {
        VBSMemo->Lines->Add(DeleteMeshMemo->Lines->operator [](j));
      }
      else
      {
        if( j>=1 && j<=layers )  //layers !!
        {
          t1 = DeleteMeshMemo->Lines->operator [](j);
          j1 = t1.Pos("?");
          t2 = t1.Delete(j1,1);
          if( j == layers )
          {
            t1 = t2.Insert(VariName->Lines->operator [](j-1).SubString(0,3),j1);
          }
          else
          {
            t1 = t2.Insert(VariName->Lines->operator [](j-1),j1);
          }
          VBSMemo->Lines->Add(t1);
        }
        else if ( j == layers+1 )
        {
          t1 = DeleteMeshMemo->Lines->operator [](j);
          j1 = t1.Pos("?");
          t2 = t1.Delete(j1,1);
          t1 = t2.Insert(VariName->Lines->operator [](4*4),j1);
          VBSMemo->Lines->Add(t1);
        }
        else
        {
          t1 = DeleteMeshMemo->Lines->operator [](j);
          j1 = t1.Pos("?");
          t2 = t1.Delete(j1,1);
          t1 = t2.Insert(VariName->Lines->operator [](j-2),j1);
          VBSMemo->Lines->Add(t1);
        }
```

```
            }
        }
        if( vbsC == 1 )
        {
            VBSMemo->Lines->SaveToFile(VBS+"G"+IntToStr(Gen)+"P"+IntToStr(i)+".vbs");
            VBSMemo->Lines->Clear();
        }
        else
        {
            VBSMemo->Lines->SaveToFile(VBS+"G"+IntToStr(Gen)+"P"+IntToStr(i)+"a.vbs");
        }

    }
    }
    if(FileExists(Result))
    {
        DeleteFile(Result);
    }
    if(FileExists(Result_1))
    {
        DeleteFile(Result_1);
    }
    if(FileExists(Result_2))
    {
        DeleteFile(Result_2);
    }
    if(FileExists(Result_3))
    {
        DeleteFile(Result_3);
    }
    if(FileExists(Result_a))
    {
        DeleteFile(Result_a);
    }
    if(FileExists(Result_1a))
    {
        DeleteFile(Result_1a);
    }
    if(FileExists(Result_2a))
    {
        DeleteFile(Result_2a);
    }
    if(FileExists(Result_3a))
    {
        DeleteFile(Result_3a);
    }
    Timer1->Enabled = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)       //run script
{                                                            // run script
    WinExec(("Wscript.exe
"+VBS+"G"+IntToStr(Gen)+"P"+IntToStr(PopC)+".vbs").c_str(),SW_SHOWMINIMIZED);
```

```
   Timer2->Enabled = true;
   Timer1->Enabled = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer2Timer(TObject *Sender)
{
   if(FileExists(Result_3))        //waiting for results!
   {

      WinExec(("Wscript.exe
"+VBS+"G"+IntToStr(Gen)+"P"+IntToStr(PopC)+"a.vbs").c_str(),SW_SHOWMINIMIZED);
          //check result
      Timer4->Enabled = true;
      Timer2->Enabled = false;
   }
   else
   {
      Label1->Caption = "Running";
   }
}
//---------------------------------------------------
void __fastcall TForm1::Timer4Timer(TObject *Sender)
{
   if(FileExists(Result_3a))        //waiting for results!
   {                                                    //check result
      B3->Click();
      Timer4->Enabled = false;
   }
   else
   {
      Label1->Caption = "Running";
   }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::B3Click(TObject *Sender)
{
   VBSMemo->Lines->Clear();         // pre-assign the parameters
   Memo1->Lines->Clear();
   double *freq, *value;
   int cap, pos;
   AnsiString temp, temp1;
   double sdv, avg, avg1;
   float S11 = -12;  // hard limit -> all s11(f) < -8
   float dir_l = 10;

   VBSMemo->Lines->LoadFromFile(Result);  //loading the s11 results
   Memo1->Lines->LoadFromFile(Result_a);
   for( int i = 1 ; i < Memo1->Lines->Capacity; i++ )
   {
      VBSMemo->Lines->Add(Memo1->Lines->operator [](i));
   }
   VBSMemo->Lines->Delete(0);            //Delete the declare line
   cap = VBSMemo->Lines->Capacity;      //calculate how many datas
   freq = new double[cap];
```

```
value = new double[cap];
double *sum;                        //sum is the fitness value
sum = new double[1];                 // pre-assume that fitness value is 0
*(sum) = 0;
sdv =0, avg=0, avg1=0;
//s11
for( int i = 0; i< cap; i++ )
{
   temp = VBSMemo->Lines->operator [](i);  //Data Process of excel freq, value
   pos = temp.Pos(",");
   *(freq+i) = StrToFloat(temp.SubString(1,pos-1));  // Get freq
   *(value+i) = StrToFloat(temp.SubString(pos+1,temp.Length()-pos));  //Get value

   if(  *(value+i) < S11 )
   {
      sdv = sdv+S11;
   }
   else
   {
      sdv = sdv + (*(value+i)) ;
   }
}
avg = sdv/cap;  //mean
for( int i = 0; i<cap; i++ )
{
   if( *(value+i) < S11 )
   {
       avg1 = avg1+ pow((S11-avg),2);
   }
   else
   {
        avg1 = avg1+ pow((*(value+i)-avg),2);
   }
}
sdv = sqrt( (avg1/(cap-1)) );
*(sum) = -1*avg/(sdv+1);
VBSMemo->Lines->SaveToFile(ResultS+"G"+IntToStr(Gen)+"P"+IntToStr(PopC)+"_s11.csv");

// Phase Center Calculating
B4->Click();
*(sum) = *(sum) * fitness_pattern;
B5->Click();
*(sum) = *(sum) * fitness_pattern;
Fitness->Lines->Add(FloatToStr((*(sum))));        // output the final fitness value! (ori)
Fitness->Lines->SaveToFile("c:\\optima\\b_fit.txt");
delete[] freq,value, sum;
DeleteFile(Result);
DeleteFile(Result_1);
DeleteFile(Result_2);
DeleteFile(Result_3);
DeleteFile(Result_a);
DeleteFile(Result_1a);
DeleteFile(Result_2a);
DeleteFile(Result_3a);
```

```
    PopC++;

    if(PopC>pop)
    {
       PopC = 1;
       Gen++;
       if(FileExists(READY))
       {
          DeleteFile(READY);
       }
       Timer3->Enabled = true;
       Fitness->Lines->SaveToFile(FITNESS);
 // throw out the signal for GA to do the generation caculation
       Fitness->Lines->Clear();
       fitness_H->Lines->Clear();
    }
    else
    {
       Timer1->Enabled = true;
    }

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer3Timer(TObject *Sender)
{
    if(FileExists(READY))           // waiting for creating new script!
    {
       B2->Click();
       Timer3->Enabled = false;
    }
    else
    {
       Label1->Caption = "Waiting GA";
    }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::B4Click(TObject *Sender)
{
    Memo1->Lines->Clear();
    float f_max, f_min;  //frequency range
    float f_max_a1=5, f_min_a1=1;  //frequency range
    float f_max_a2=18, f_min_a2=6;  //frequency range
    float theta = 1,  f_step = 1;
    float theta_s = 0, theta_e = 60;
    float temp, temp1, temp2, temp3, temp4, avg=0, avg1=0,sdv=0;
    float lambda, wavenumber, center_p, center_s;
    fitness_pattern = 0;
    Variant XL,v0,v1,vcell;

    AnsiString tmp;
    for( float h = 1; h <=2 ; h++ )
    {
       for( int count = 1; count <= 2; count++ )
       {
        if( count == 1 )
        {
           XL=Variant::CreateObject("excel.application");
           XL.OlePropertySet("Visible",false);
           XL.OlePropertyGet("Workbooks").OleProcedure("Open", Result_2.c_str() );
           v0=XL.OlePropertyGet("Sheets","temp2");      // sheet name : temp2
           v1=v0.OlePropertyGet("Cells");
           f_min = f_min_a1;
           f_max = f_max_a1;
        }
        else
        {
           XL=Variant::CreateObject("excel.application");
           XL.OlePropertySet("Visible",false);
           XL.OlePropertyGet("Workbooks").OleProcedure("Open", Result_2a.c_str() );
           v0=XL.OlePropertyGet("Sheets","temp2a");      // sheet name : temp2
           v1=v0.OlePropertyGet("Cells");
           f_min = f_min_a2;
           f_max = f_max_a2;
        }
        for( float i = f_min; i<= f_max; i = i+ f_step )
        {
           lambda = 12/i;
           wavenumber = 2*M_PI/lambda;
           center_s = 0, temp4 = 0;
           for( center_p = 0; center_p<=4; center_p = center_p+0.1 )
           {
              temp =0, temp1 = 0, temp2 = 0, temp3 = 0;
              for( float j = theta_s; j<=theta_e ; j=j+theta )
              {
                 //Get the content of the Cell located at row 2 and column 2
                 vcell=v1.OlePropertyGet("Item", 2 + (j-theta_s)/theta, 2+(h-1)*((f_max-
f_min)/(f_step)+1)+(i-f_min)/f_step);                    tmp=vcell.OlePropertyGet("Value");  //store that
content to ansistring "tmp"

                 if( j == 0 )
                 {
                    temp = StrToFloat(tmp);
                 }

                 temp1 = StrToFloat(tmp) - temp;
                 temp2 =  wavenumber*center_p*(  1 - cos( j*M_PI/180)   ) - temp1;
                 temp3 = temp3 + temp2*temp2;
              }
              if( center_p == 0 )
              {
                 temp4 = temp3;
                 center_s = center_p;
              }
              else
              {
                 if( temp4 < temp3 )
                 {
                    temp4 = temp4;
                    center_s = center_s;
                  }
                  else
                  {
                    temp4 = temp3;
                    center_s = center_p;
                   // ShowMessage("a");
                  }
                }
              }
              Memo1->Lines->Add(FloatToStr( center_s ));
            }
            XL.OleProcedure("Quit");
            XL=Unassigned;
        }
    }
    Memo1->Lines->SaveToFile(ResultS+"G"+IntToStr(Gen)+"P"+IntToStr(PopC)+"_PhaseCenter.csv");
    for( int i = f_min-f_min ; i<= 1+2*(f_max_a2-f_min_a1)/f_step ; i=i+f_step )
    {
      if( (i != 0) && (i != 1+(f_max_a2-f_min_a1)/f_step) )
      {
        avg = avg + StrToFloat(Memo1->Lines->operator [](i));
      }
    }
    avg = avg/((2*(f_max_a2-f_min_a1)/f_step));
    for( int i = f_min-f_min ; i<= 1+2*(f_max_a2-f_min_a1)/f_step ; i=i+f_step )
    {
      if( (i != 0) && (i != 1+(f_max_a2-f_min_a1)/f_step) )
      {
        avg1 = avg1+ pow(( StrToFloat(Memo1->Lines->operator [](i)) -avg),2);
      }
    }
    sdv = sqrt( avg1/((2*(f_max_a2-f_min_a1)/f_step)));
    fitness_pattern = exp(-1*sdv);

    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::B5Click(TObject *Sender)
{
    Memo1->Lines->Clear();
    float f_max=18, f_min=1;  //frequency range
    float f_max_a1=5, f_min_a1=1;  //frequency range
    float f_max_a2=18, f_min_a2=6;  //frequency range
    float theta = 1,  f_step = 1;
    float theta_s = 0, theta_e = 50;
    float temp, avg=0, avg1=0, sdv=0;
    float HPBW;
    fitness_pattern = 0;

    Variant XL,v0,v1,vcell;
    AnsiString tmp;
    for( int count = 1; count <=2 ;count++ )
    {
      for( float h = 1; h<=2; h++ )
      {
        if(count ==1)
        {
              XL=Variant::CreateObject("excel.application");
              XL.OlePropertySet("Visible",false);
              XL.OlePropertyGet("Workbooks").OleProcedure("Open", Result_3.c_str() );
              v0=XL.OlePropertyGet("Sheets","temp3");                v1=v0.OlePropertyGet("Cells");
              f_min = f_min_a1;
              f_max = f_max_a1;
        }
        if(count ==2)
        {
              XL=Variant::CreateObject("excel.application");
              XL.OlePropertySet("Visible",false);
              XL.OlePropertyGet("Workbooks").OleProcedure("Open", Result_3a.c_str() );
              v0=XL.OlePropertyGet("Sheets","temp3a");                v1=v0.OlePropertyGet("Cells");
              f_min = f_min_a2;
              f_max = f_max_a2;
        }
        for( float i = f_min; i<= f_max; i = i+ f_step )
        {
              HPBW = 0;
              temp =0, avg=0, avg1=0, sdv=0;
              for( float j = theta_s; j<=theta_e ; j=j+theta )
              {
                //Get the content of the Cell located at row 2 and column 2
                vcell=v1.OlePropertyGet("Item", 2 + (j-theta_s)/theta, 2+(h-1)*((f_max-f_min)/(f_step)+1)+(i-
f_min)/f_step);
                tmp=vcell.OlePropertyGet("Value");
                temp = StrToFloat(tmp);
                avg = StrToFloat(Trim(Pattern_C->Lines->operator [](j)));
                avg1 = temp-avg;
                if( abs(avg1) > 0.2)
                {
                  if( abs(avg1) > HPBW )
                  {
                    HPBW = abs(avg1);
                  }
                  else
                  {
                    HPBW = HPBW;
                  }
                }
              }
              Memo1->Lines->Add(FloatToStr( HPBW ));
        }
        XL.OleProcedure("Quit");
        XL=Unassigned;
       //e ShowMessage(FloatToStr(HPBW));
      }
    }

    Memo1->Lines->SaveToFile(ResultS+"G"+IntToStr(Gen)+"P"+IntToStr(PopC)+"_HPBW.csv");
```

```
   avg = 0;
   avg1 = 0;
   temp = 0;
   for( int i = f_min-f_min ; i< 2*((f_max_a2-f_min_a1)/f_step+1) ; i=i+f_step )
   {
      avg = avg + StrToFloat(Memo1->Lines->operator [](i));
   }
   avg = avg/(2*(1+(f_max_a2-f_min_a1)/f_step));
  //   ShowMessage(FloatToStr(avg));
   for( int i = f_min-f_min ; i<= 1+2*(f_max_a2-f_min_a1)/f_step ; i=i+f_step )
   {
      avg1 = avg1+ pow(( StrToFloat(Memo1->Lines->operator [](i)) -avg),2);
   }
   sdv = sqrt( (avg1/((2*(1+(f_max_a2-f_min_a1)/f_step)))) );
   Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------
```