Hindawi Publishing Corporation
Journal of Engineering
Volume 2014, Article ID 489765, 10 pages
http://dx.doi.org/10.1155/2014/489765

Research Article

Design and Implementation of Radar Cross-Section Models on a Virtex-6 FPGA

B. U. V. Prashanth

Department of Electronics and Communications Engineering, Anurag College of Engineering, Hyderabad, Telangana 501301, India

Correspondence should be addressed to B. U. V. Prashanth; shiva.prashanth81@gmail.com

Received 11 August 2014; Revised 6 October 2014; Accepted 8 October 2014; Published 5 November 2014

Academic Editor: Lucian Dascalescu

The simulation of radar cross-section (RCS) models in FPGA is illustrated. The models adopted are the Swerling ones. Radar cross-section (RCS) which is also termed as echo area gives the amount of scattered power from a target towards the radar. This paper elucidates the simulation of RCS to represent the specified targets under different conditions, namely, aspect angle and frequency. This model is used for the performance evaluation of radar. RCS models have been developed for various targets like simple objects to complex objects like aircrafts, missiles, tanks, and so forth. First, the model was developed in MATLAB real time simulation environment and after successful verification, the same was implemented in FPGA. Xilinx ISE software was used for VHDL coding. This simulation model was used for the testing of a radar system. The results were compared with MATLAB simulations and FPGA based timing diagrams and RTL synthesis. The paper illustrates the simulation of various target radar cross-section (RCS) models. These models are simulated in MATLAB and in FPGA, with the aim of implementing them efficiently on a radar system. This method can be generalized to apply to objects of arbitrary geometry for the two configurations of transmitter and receiver in the same as well as different locations.

1. Introduction

Radar cross-section is used to describe the amount of scattered power from a target towards the radar, when the target is illuminated by RF energy. The intensity of backscattered energy that has the same polarization as the radar's receiving antenna is used to define the target RCS [1]. RCS is used as means of discrimination. In simulations, the control parameters in the resulted RCS model are optimally tuned using an algorithm so as to show the full potential of the proposed RCS model. The algorithm along with mathematical calculations computes and plots Swerling statistical models.

The methodology adopted to design the simulation of RCS model is as follows:

(a) development of RCS models for simple and complex objects;

(b) generation of the Swerling Random Sequences;

(c) verification of results in MATLAB R-2013 real time simulation software;

(d) implementation of RCS models in FPGA using VHDL code by Xilinx ISE v12.1ISE;

(e) application of simulated models to a radar system.

Here the RCS models are developed for simple and complex objects such as ellipsoid, sphere, and cylinder using MATLAB which calculates the backscattered RCS for a perfectly conducting sphere and Spherical Bessel functions are computed using series approximation and recursion. Then Swerling Random Sequences are generated for exponential and Chi-square, degree 4. Then all the above results are simulated in MATLAB and implemented on FPGA using VHDL code using XILINX software. Finally the simulated models are applied to the radar system.

2. Generation of Swerling Random Sequences

Swerling Target Models are models of probability density function of the radar backscatter from a complex target for rotating surveillance radar [2]. Table 1 illustrates this concept

Table 1: Swerling models.

Probability density function of power	Scan-to-scan	Pulse-to-pulse
Exponential	1	2
Chi-square, degree 4	3	4

for scan-to-scan and pulse-to-pulse probability density function of power for exponential and Chi-square, degree 4.

In this paper we use primarily two methods to generate Swerling models; they are as follows:

(1) exponential power distribution,

(2) 4th degree Chi-square power distribution.

Probability density functions are simulated to verify if random variables are following the distribution. The radar cross-section of complex objects is given by Swerling models. The characteristic plot of Swerling random variables obtained in real time simulation environment is the probability distribution function for RCS fluctuation of complex objects [3].

The exponential power distribution is as defined by the following equation:

$$p_x(x) = \begin{cases} \frac{1}{\mu} e^{-x/\mu}, & x \geq 0, \\ 0, & x < 0, \end{cases} \quad (1)$$

where $x = -\mu \ln u$ and further $p_x(x)$ = probability density function, u = random number, μ = mean RCS, and x = RCS.

The exponential power distribution method to generate a Swerling model is observed in the Agilent spectrum analyzer and the simulation results of exponential power distribution [4] are obtained, which are illustrated in the following:

$$p_x(x) = \begin{cases} \frac{4x}{\mu^2} e^{-2x/\mu}, & x \geq 0, \\ 0, & x < 0, \end{cases} \quad (2)$$

where $x = -\mu/2 \ln(u1u2)$, $p_x(x)$ = probability density function, u = random number, μ = mean RCS, and x = RCS.

3. Generation of Uniformly Distributed Random Numbers

We use linear feedback shift registers for generating our 32-bit uniformly distributed random number. The term uniformly distributed random number makes it clear that the exponential distributions and the Chi-square distributions are derived by these uniformly distributed random numbers and the methods are stated in the algorithm designed along with the mathematical formulations in VHDL syntax as illustrated below. A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.

The IP core generator used in XILINX is CORDIC IP core generator. It is a trigonometric Sine-Cosine generator. Libraries cannot be used as they are only for simulation and cannot be used for synthesis purposes [5].

The following are the steps to obtain the received power from the target for the generated RCS models.

(a) First generate random numbers using VHDL code.

(b) Those random numbers are taken as u.

(c) Scale the random number between 0 and 1 using divider (if 16-bit random number is generated we need to divide by 65,536 to get in range 0-1).

(d) Input the values to $\tanh^{-1}$ to calculate logarithm.

(e) Multiply ln values with RCS average.

(f) The RCS models generated are used to get the received power from a target.

The algorithm for random variable generation in VHDL is designed as follows, as the description of the designed algorithm helps to ground what techniques are adopted.

(a) Define a signal as a random number.

(b) Initialize the clock signal.

(c) Initialize the control signal as SIGNAL cntr: std_logic_vector(1 down to 0):= (others => '0');

(d) Define the exponential signal as signal exp_rv: std_logic_vector(20 down to 0);

(e) Declaration for TANH_INV component core instantiation is as follows:

(i) signal tanh_den: std_logic_vector(9 downto 0):= ((8) => '1',others => '0');

(ii) signal tanh_num: std_logic_vector(9 downto 0):= ((9) => '1',others => '0');

(iii) signal temp_num: std_logic_vector(8 downto 0):= (others => '0');

(iv) signal temp_num2: std_logic_vector(9 downto 0):= (others => '0');

(v) signal tanh_out: std_logic_vector(15 downto 0);

(vi) define the inputs, phase out and clock for tanhinv component instantiation

```
component tanhinv
port (
x_in: IN std_logic_VECTOR(9 down to 0);
y_in: IN std_logic_VECTOR(9 down to 0);
phase_out: OUT std_logic_VECTOR(15 down to 0);
clk: IN std_logic);
```

(f) Random Number Generation Algorithm section in VHDL is as follows:

```
variable rand_temp: std_logic_VECTOR(7 downto 0):=(others => '0');
variable temp: std_logic:='0';
begin
if(rising_edge(clk2))then
temp:=rand_temp(7) xnor rand_temp(5) xnor rand_temp(4) xnor rand_temp(3);
rand_temp(7 downto 1):= rand_temp(6 downto 0);
```

rand_temp(0):=temp;
rnd_num <= rand_temp;

(g) Initialize the output buffer as follows:

(a) OBUF_inst: OBUF
(b) generic map (

DRIVE => 12,
IOSTANDARD => "DEFAULT",
SLEW => "SLOW")

port map (

O => dac_clk, -- Buffer output (connect directly to top-level port)

I => clk -- Buffer input

);

(h) Define the digital to analog control signals as follows:

(a) Select Non zero Stuff, Baseband as:
MOD_DAC<="00";
(b) Select Data Rate bandwidth 24 TO 100MSPS as:
DIV_DAC<="01";
(c) Reset the digital to analog converter signal

reset_dac<='0'; RESET SIGNAL;

sleep_dac<='0';

(i) Process for clock calculation

variable temp:std_logic_vector(1 downto 0):= (others => '0');

temp:= cntr + '1';

cntr <= temp;

if(cntr = "11") then

clk2 <= not (clk2);

cntr <= "00";

(j) Convert the generated random numbers into 1's complement format

(a) tanh_den(7 downto 0) <= rnd_num(7 downto 0);
(b) temp_num(7 downto 0) <= rnd_num(7 downto 0);
(c) temp_num2 <= "1000000000" - temp_num;
(d) tanh_num(8 downto 0) <= (not(temp_num2(8 downto 0))) + "000000001";

(k) Define TANH_INV IP CORE

tanh_ip: tanhinv

port map (

x_in => tanh_den,
y_in => tanh_num,
phase_out => tanh_out,
clk => clk);

(i) Take Natural logarithm of random number

ln_out(15 downto 1) <= not(tanh_out(14 downto 0)) + "000000000000001";

exp_rv1 <= ln_out * rcs_avg;

(m) Convert the output into 2's complement format

(n) exp_rv <= (not(exp_rv1)) + "00000000000000000001";

(o) The generated random numbers are taken as u.

(p) Scale the Random Number between 0 and 1 using divider (if 16-bit random number is generated, then divide by 65,536 to get in range 0-1).

Input the values to $\tan^{-1} h$ to calculate logarithm and multiply the natural log ln values with RCS average.

The RCS models generated are used to get the Received power from a target.

4. FPGA Implementation

The block diagram in Figure 1 illustrates the FPGA implementation [6].

The Chip Scope Pro tool is used to model the XILINX ISE design on to the Hardware FPGA-Virtex-6. The bit stream file generated in XILINX ISE is ported on the Virtex-6 FPGA kit. The digital to analog converter takes the digital data from Virtex-6 FPGA and converts it into analog format and displays it on the oscilloscope. Chip Scope Pro tool solution helps minimize the amount of time required for downloading and debugging. Further it is described how the target RCS influences the radar detection performance which constitutes the representation of control parameters of RCS. The FPGA implementation is done through both compile time and run time parameters which include normalization range, distance threshold, and fade cycle length which are involved in registers, which are user defined and also can be changed during the run time. These control parameters create a user defined environment in which the response obtained from FPGA can adapt itself to this environment both statically at compile time and dynamically at run time. The streaming data obtained from this FPGA solution response is further processed onto the host computer. The FPGA implementation is modular in nature in which the design consideration can be further modified based on the future requirements in the FPGA implementations.

The FPGA implementation is verified in simulation using XILINX ISE and in hardware on an innovative integration Virtex-6 FPGA hardware board running at frequency of 200 MHz.

The following procedure was performed for FPGA hardware.

(1) The host P.C sends the input RCS models to the hardware in the form of input first-in first-out (FIFO) over the FMC Bridge-PCI Express Bus.

(2) The input is read from the first-in first-out (FIFO) and the fade cycle length is calculated with all other

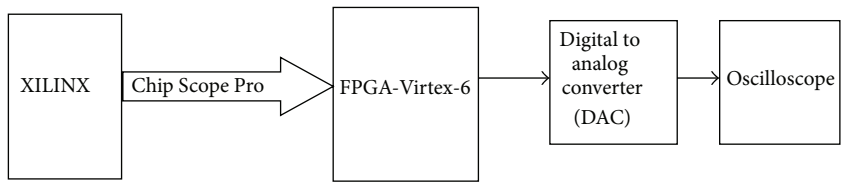

Figure 1: FPGA implementation.

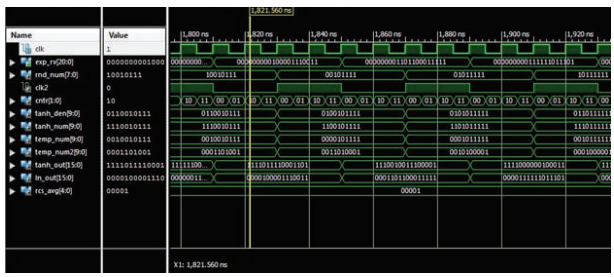

Figure 2: Simulations results for exponential distribution simulation on Virtex-6 FPGA.

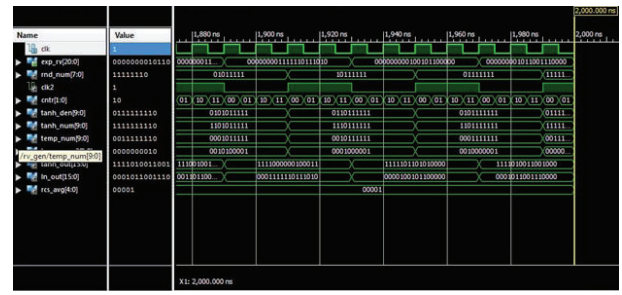

Figure 3: Simulations results for FPGA 4th degree Chi-square distribution simulation on Virtex-6 FPGA.

parameters such as normalization range and distance threshold coordinates are updated in the registers.

(3) The contents of the registers are read back to the host for postanalysis verification.

The advantages of using the target RCS on FPGA hardware are as follows.

The XILINX Virtex-6 implementation is estimated as very much faster than equivalent "C" language program running on a 3 GHz Intel Xeon processor. The FPGA implementation is further implemented in radar pulse deinterleaving which is basically utilized as an electronic warfare application. The other applications of this FPGA implementation involve the streaming with the data evolved.

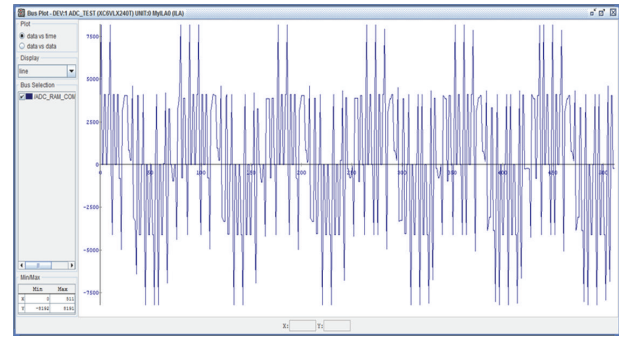

Figure 4: Bus Plot of ADC for the FPGA device XC6VLX240T.

5. Results Obtained in FPGA

The simulation is carried out using Model-Sim software. The target device selected is Virtex-6. The functional and timing simulations were carried out. The output is obtained after 156 clock cycles. The maximum frequency obtained is 35.750 MHz's. In this case the target device selected is xc6vlx240t--3ff324. The simulation time is 2,000,000 μs. In this the model based design approach is provided for an integrated workflow [6]. This model based design speeds up the algorithm development with a unified design environment and automates the manual steps in FPGA implementation to enable shorter iteration cycles from the XILINX environment [5].

Figures 2 and 3 illustrate the simulation results for different objects with respect to the object names as seen in these figures such as clock (clk1 and clk2), exp_rv, rcs_avg, rnd_num,tanh_den, tanh_num, temp_num, tanh_out, and ln_out along with various instances and processes for FPGA based exponential distribution as well as for 4th degree Chi-square. Further the synthesis results are determined by a parameter Speed Grade which is equal to −3 for exponential distribution simulation. In this case the target device selected is xc6vlx240t--3ff324 and also the other parameters of the synthesized design summary such as:

minimum period: 28.77 ns (maximum frequency: 34.750 MHz);

minimum input arrival time before clock: no path found;

maximum output time required after clock: 4.123 ns;

maximum combinational path delay: no path found.

The analog to digital converter simulated in a XILINX based chip scope pro analyzer above is a 12-channel ADC elucidated in Figure 4, in which the data port is a 12-channel ADC_RAM_Component. The above simulation is done through the JTAG chain which consists of two devices which are device 0 consisting of SYSTEM_ACE_COMPONENT and device 1 consisting of ADC_RAM_Component (from channel 0 to channel 12) where in this case the sample buffer is full. The plot in Figure 3 is a data versus time plot in which the x-axis

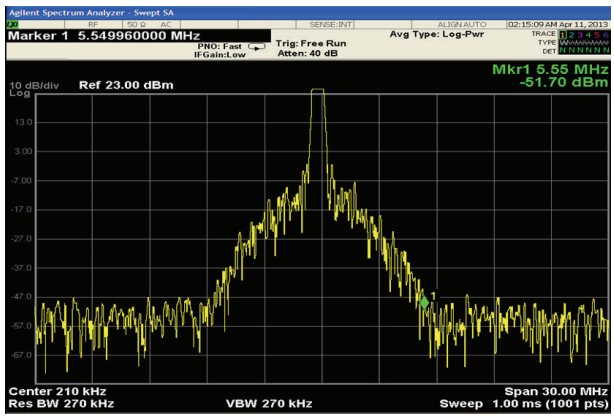

Figure 5: Exponential power distribution.

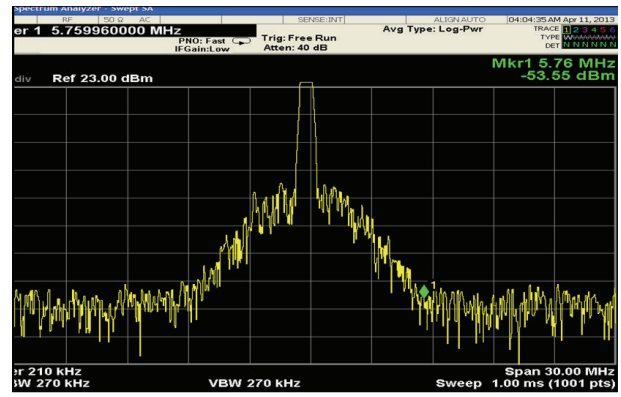

Figure 6: 4th degree Chi-square power distribution Swerling model.

represents the minimum 0 to maximum 511 units, and y-axis represents minimum −8192 to maximum 8191 units.

In the working hardware the RCS models are received directly on FPGA from chip scope pro analyzer which is a pulse measurement module in the form of analog to digital converter simulated as shown in Figure 4 with a multiplexer control and the streaming output data is streamed continuously to the hardware processing unit at the frequency of 5 microseconds and 10 microseconds via a USB-FPGA interface module 1.15x as illustrated in Figure 7 to Figure 10 for both 4th degree Chi-square power distribution and exponential models. The FPGA implementation is capable of keeping the RCS models at the rate that is expected to be received from the environment. Working procedure for both 5 microseconds and 10 microseconds exponential and 4th degree Chi-square models is as follows.

(1) Signal generator: amplitude = 1.499 V and frequency = 1 MHz.

(2) Function generator: amplitude = 1.5 Vp-p (square wave) and frequency = 20 MHz or sampling frequency.

(3) Duty cycle = 50%.

(4) Power supply: 5 V, 0.33 mA.

Figure 5 illustrates the exponential power distribution, with a free run trigger in the Agilent spectrum analyzer with the attenuation of 40 dB. Figure 4 also elucidates the radar signal simulation with the processing environment to generate a Swerling model. The average type of power used is the log power. The video bandwidth of the spectrum analyzer is given as 270 kHz and the resolution bandwidth is 270 kHz; the center frequency is given as 210 kHz with span of 30 MHz. There is a sweep of 1 ms with 1001 pts. Further the intermediate frequency (IF) gain is low.

Figure 6 illustrates the 4th degree Chi-square power distribution Swerling model, with a free run trigger in the Agilent spectrum analyzer with the attenuation of 40 dB. The figure also elucidates the radar signal simulation with the processing environment to generate a Swerling model. The average type of power used is the log power. The video

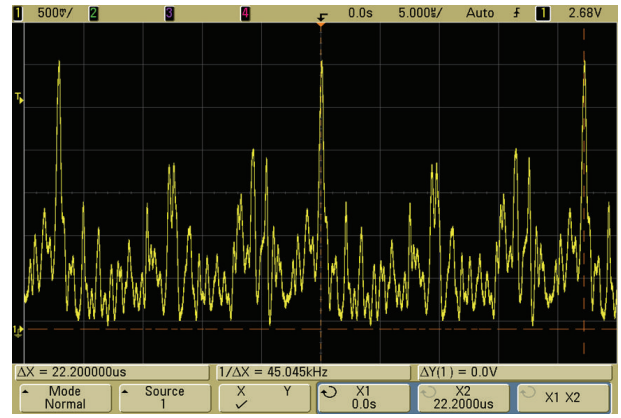

Figure 7: For the 5 μs frequency for exponential model.

bandwidth of the spectrum analyzer is given as 270 kHz and the resolution bandwidth is 270 kHz; the center frequency is given as 210 kHz with span of 30 M Hz. There was a sweep of 1 ms with 1001 pts. The probability density function of power as illustrated in Table 1 is for the individual echo powers with radar cross-section proportional to each other.

As seen from Figures 7 and 8, the echo power in Figure 6 represents the exponential probability density function of power which is equal to −51.7 dBm with the marker frequency of 5.55 MHz, while the echo power in Figure 7 represents the 4th degree Chi-square probability density function of power which is equal to −53.55 dBm with the marker frequency of 5.76 MHz.

The plot shown in Figure 7 illustrates the exponential model operated at 5 μs frequency. The multiplexer controlled streaming output data is streamed continuously to the hardware processing unit at the frequency of 5 microseconds.

The plot shown in Figure 8 illustrates the exponential model operated at 10 μs frequency. The multiplexer controlled streaming output data is streamed continuously to the hardware processing unit at the frequency of 10 microseconds.

Further as illustrated in Figures 9 and 10 for 5 μs and 10 μs, frequency for 4th degree Chi-square power distribution is via a USB-FPGA interface module 1.15xs as illustrated.

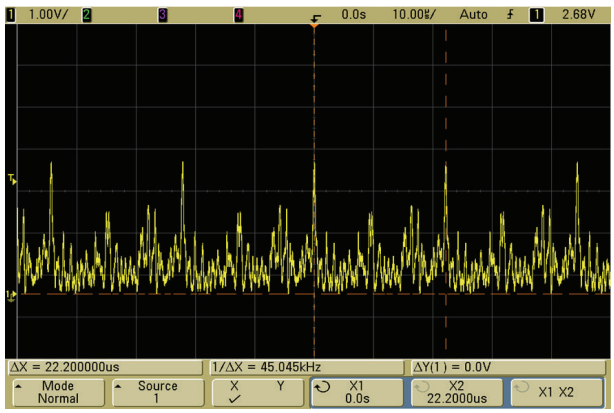

Figure 8: For 10 μs frequency exponential model.

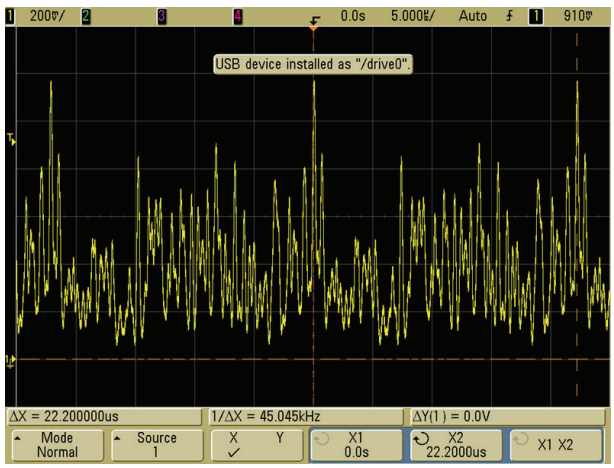

Figure 9: For 5 μs frequency, 4th degree Chi-square power distribution.

6. RCS Variation for Two Point Scatters in MATLAB

The algorithms in MATLAB real time simulation environment are illustrated in Sections 6.1 to 6.4 with the detailed mathematical expressions with the MATLAB based syntax of the RCS of different objects and calculation steps, as the description of the designed algorithms helps to ground what techniques are adopted.

6.1. RCS Dependency on Aspect Angle. The RCS variation for two point scatterers separated by a distance, with the aspect angle, is calculated in MATLAB. A function is created to calculate and plot the variation in RCS with respect to aspect angle. The function is shown below. This function demonstrates the effect of aspect angle on RCS:

function [rcs] = rcs_aspect_angle (scat_spacing, freq).

Point scatterers are separated by scat spacing meter. Initially the two scatters are aligned with radar line of sight. The aspect angle is changed from 0 degrees to 180 degrees and the equivalent RCS is computed according to the mathematical expressions in MATLAB based syntax illustrated below. Plot

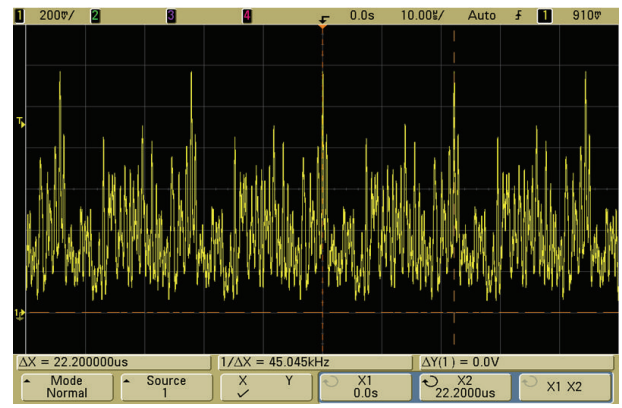

Figure 10: For 10 μs frequency, 4th degree Chi-square power distribution.

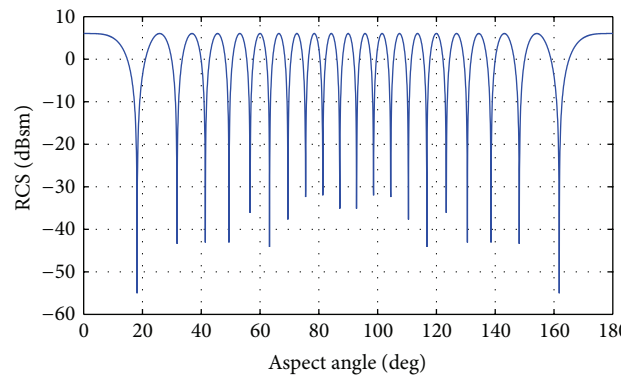

Figure 11: Variation in RCS of two isotropic scatterers with aspect angle.

of RCS versus aspect is generated as shown in Figure 11. Consider

```
eps = 0.00001;
wavelength = 3.0e+8 / freq;
Compute aspect angle vector
aspect_degrees = 0.:.05:180.;
aspect_radians = (pi/180).* aspect_degrees;
Compute electrical scatterer spacing vector in wavelength units
elec_spacing = (2.0*scat_spacing / wavelength). * cos(aspect_radians);
Compute RCS
(rcs = RCS_scat1 + RCS_scat2);
Scat1 is taken as phase reference point
rcs = abs(1.0 + cos((2.0 * pi).* elec_spacing) + i * sin((2.0 * pi).* elec_spacing));
rcs = rcs + eps;
rcs = 20.0*log10(rcs); % RCS in dBsm.
```

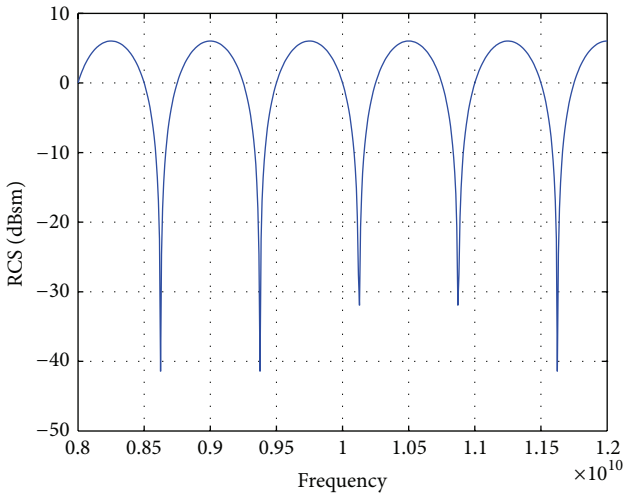

Figure 12: Variation in RCS of two isotropic scatterers with frequency.

6.2. RCS Dependency on Frequency. The MATLAB simulation as seen in Figure 12 for RCS variation with frequency of incident RF wave is done using functional algorithm given as follows:

function [rcs] = rcs_frequency (scat_spacing, frequ, freql).

This algorithm along with mathematical expressions MATLAB based syntax demonstrates the dependency of RCS on wavelength:

```
eps = 0.0001;
freq_band = frequ - freql;
delfreq = freq_band / 500.;
index = 0;
for freq = freql: delfreq: frequ
index = index +1;
wavelength(index) = 3.0e+8 / freq;
end
elec_spacing = 2.0 * scat_spacing./ wavelength;
rcs = abs ( 1 + cos((2.0 * pi).* elec_spacing) + i *
sin((2.0 * pi).* elec_spacing));
rcs = rcs + eps;
rcs = 20.0*log10(rcs); % RCS ins dBsm.
```

6.3. Maximum Radar Range Dependency on RCS. The algorithm illustrated along with mathematical expressions MATLAB based syntax for calculating the maximum radar range is given as follows:

```
ps = 400;
g = 190^3.5;
l = ((3*10e+8)/(17*10e+9));
rcs = 0.1:0.01:0.5;
pmin = 10^(−13.5);
```

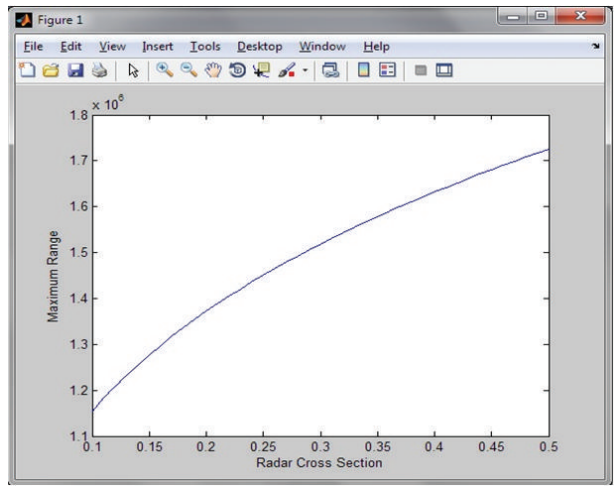

Figure 13: Maximum radar range dependency on RCS.

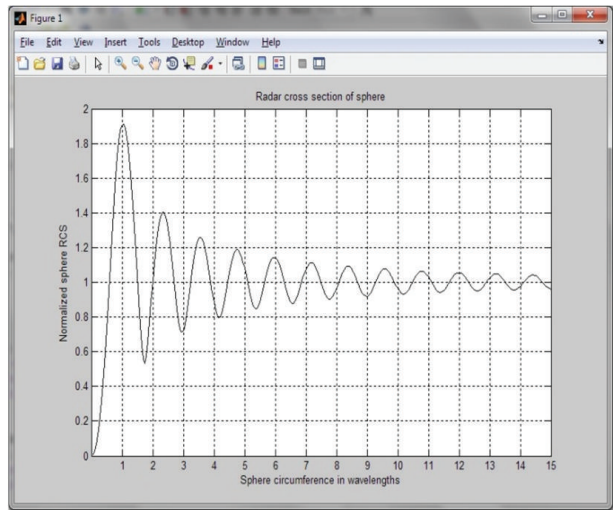

Figure 14: RCS of sphere.

```
numerator = ps*(g^2)*(l^2)*rcs;
denominator = pmin*64*(pi^3);
Rmax = sqrt(sqrt(numerator/denominator));
```

Figure 13 illustrates the maximum radar range dependency on RCS.

6.4. RCS of Simple Objects

6.4.1. RCS of the Sphere. The algorithm calculates the backscattered RCS for a perfectly conducting sphere. Spherical Bessel functions are computed using series approximation and recursion as illustrated below. The simulation is illustrated in Figure 14. Trace of Power Scattering Matrix and Module of Polarization Elliptically are constant because at any orientation angle the shape of the sphere remains constant. Amplitudes of cross-polarization are equal for the dipole. For sphere depolarization value is zero which indicates that the

symmetric scattering centre for a sphere is almost equal to 1. Consider

```
eps = 0.00001;
index = 0;
kr limits are [0.05–15] implies 300 points
for kr = 0.05:0.05:15
index = index + 1;
sphere_rcs = 0. + 0.∗i;
f1 = 0. + 1.∗i;
f2 = 1. + 0.∗i;
m = 1.;
n = 0.;
q = −1.;
initially set del to huge value
del =100000+100000∗i;
while(abs(del) > eps)
q = −q;
n = n + 1;
m = m + 2;
del = (2.∗n−1) ∗ f2 / kr-f1;
f1 = f2;
f2 = del;
del = q ∗ m /(f2 ∗ (kr ∗ f1−n ∗ f2));
sphere_rcs = sphere_rcs + del;
rcs(index) = abs(sphere_rcs);
sphere_rcsdb(index) = 10. ∗ log10(rcs(index));
```

6.4.2. RCS of the Ellipsoid. Consider

```
eps = 0.00001;
sin_phi_s = sin(phi) ^2;
cos_phi_s = cos(phi) ^2;
Generate aspect angle vector
theta = 0.:.05:180.0;
theta = (theta.∗ pi)./ 180.;
if(a ~= b & a ~= c)
rcs = (pi ∗ a^2 ∗ b^2 ∗ c^2)./ (a^2 ∗ cos_phi_s.∗
(sin(theta). ^2) + ⋯
b^2 ∗ sin_phi_s.∗ (sin(theta). ^2) + ⋯
c^2.∗ (cos(theta). ^2)). ^2;
else
if(a == b & a ~= c)
rcs = (pi ∗ b^4 ∗ c^2)./ ( b^2.∗ (sin(theta). ^2) + ⋯
c^2.∗ (cos(theta). ^2)). ^2;
else
```

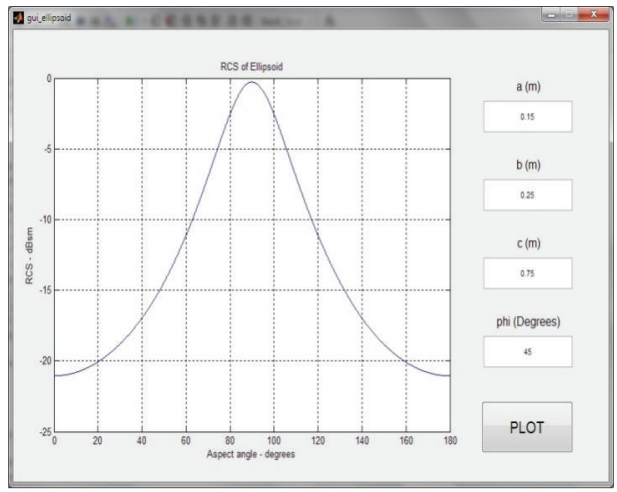

Figure 15: Graphical User Interface for RCS of ellipsoid.

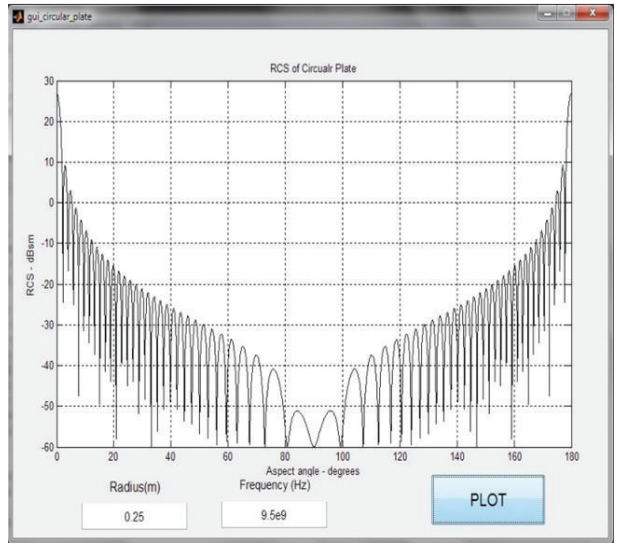

Figure 16: GUI for RCS of circular plate.

```
if (a == b & a ==c)
rcs = pi ∗ c^2;
rcs_db = 10.0 ∗ log10(rcs);
```

Figure 15 illustrates the GUI for RCS of ellipsoid.

6.4.3. RCS of the Circular Plate. The algorithm below calculates and plots the RCS of a circular flat plate of radius r. The simulation results are obtained in Figure 16. Consider

```
eps = 0.000001;
Compute wavelength
lambda = 3.e+8 / freq; % X-Band
index = 0;
for aspect_deg = 0.:.1:180
index = index +1;
```

```
aspect = (pi /180.) * aspect_deg;
Compute RCS using the equation below
if (aspect == 0 | aspect == pi)
rcs_po(index) = (4.0 * pi^3 * r^4 / lambda^2) + eps;
rcs_mu(index) = rcs_po(1);
else
Compute RCS using the equation below
x = (4. * pi * r / lambda) * sin(aspect);
val1 = 4. * pi^3 * r^4 / lambda^2;
val2 = 2. * besselj(1,x) / x;
rcs_po(index) = val1 * (val2 * cos(aspect)) ^2 + eps;
Compute RCS using the equation below
val1m = lambda * r;
val2m = 8. * pi * sin(aspect) * (tan(aspect) ^2);
rcs_mu(index) = val1m / val2m + eps;
rcsdb_po = 10. * log10(rcs_po);
rcsdb_mu = 10 * log10(rcs_mu);
angle = 0:.1:180;
```

6.4.4. RCS of the Cylinder. The following algorithm illustrates the RCS of the cylinder and the simulation results are shown in Figure 17. The dipole is also cylindrically symmetrical. Consider

```
eps = 0.00001;
wavelength = 3.0e+8 / freq;
Compute aspect angle vector
aspect_degrees = 0.:.05:180.;
aspect_radians = (pi/180).* aspect_degrees;
Compute electrical scatterer spacing vector in wavelength units
elec_spacing = (2.0 * scat_spacing / wavelength).* cos(aspect_radians);
Compute RCS (rcs = RCS_scat1 + RCS_scat2)
Scat1 is taken as phase reference point
rcs = abs(1.0 + cos((2.0 * pi).* elec_spacing) + i * sin((2.0 * pi).* elec_spacing));
rcs = rcs + eps;
rcs = 20.0*log10(rcs); % RCS in dBsm
```

6.5. RCS of Complex Objects. The radar cross-section of complex objects is given by Swerling models. The MATLAB algorithm for the RCS and probability density of RCS is shown below and the simulation is illustrated in Figure 18.

This algorithm along with mathematical calculations based on MATLAB based syntax computes and plots Swerling statistical models:

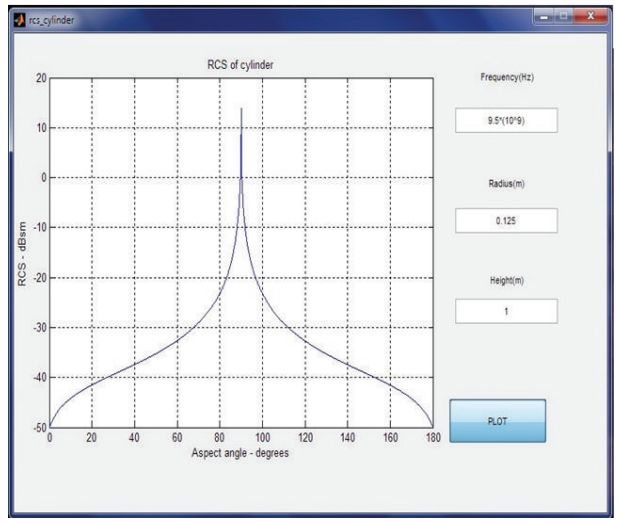

Figure 17: GUI for RCS of cylinder.

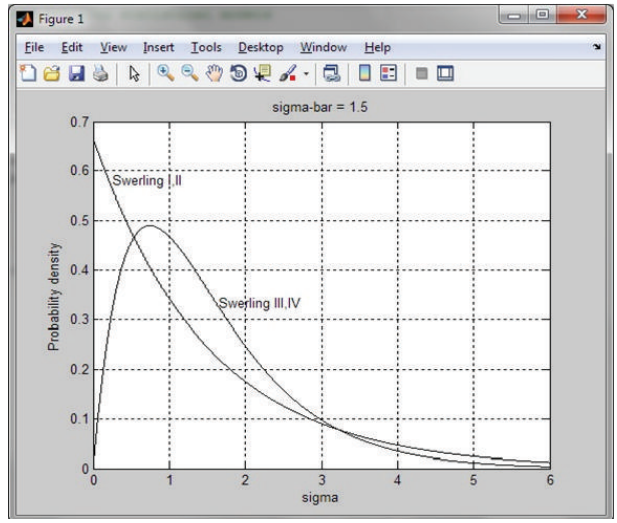

Figure 18: GUI for RCS of complex objects.

```
sigma_bar = 1.5;
sigma = 0:0.001:6;
sigma_bar = 1.5;
swer_3_4 = (4. / sigma_bar^2).* sigma.* exp(−2. * (sigma./ sigma_bar));
t.*exp(-(t. ^2)./2.
swer_1_2 = (1. /sigma_bar).* exp( -sigma./ sigma_bar);
```

7. Conclusions

Radar cross-section is simulated to represent the specified targets under different conditions, namely, aspect angle and frequency. This model can be used for the performance evaluation of radar. RCS models are developed for various targets like simple objects to complex objects. The RCS models were developed in MATLAB and successfully verified through the

simulation results and after that the same were implemented on FPGA [7]. Xilinx ISE software is used for VHDL coding. This simulation model is used for the testing of radar system. The final results obtained are found to be matching for FPGA and MATLAB real time simulation environment. Here the RCS models are designed for simple and complex objects. Then Swerling Random Sequences are generated. Then all the above results are simulated in MATLAB and implemented on FPGA using VHDL code using XILINX software. Finally the simulated models are applied to the radar system to evaluate the performance of the radar. Shaping is very important in warships by manipulating the geometry above the waterline for the naval targets. The appropriate approach is to design the structures above the waterline with minimum echo through shaping and then apply radar absorbing materials to the remaining problem areas.

Conflict of Interests

The author has no conflict of interest regarding the publication of this paper.

References

[1] J. M. Headrick and M. I. Skolnik, "Over-the-horizon radar in the HF band," *Proceedings of the IEEE*, vol. 62, no. 6, pp. 664–673, 1974.

[2] J. M. Hendrick, "HF over-the-horizon radar," in *Radar Handbook*, McGraw-Hill, New York, NY, USA, 2nd edition, 1990.

[3] L. Sevgi, "Stochastic modelling of target detection and tracking in surface wave HF radars," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 11, no. 3, pp. 167–181, 1998.

[4] L. Sevgi, A. Ponsford, and H. C. Chan, "An integrated maritime surveillance system based on high-frequency surface-wave radars, Part 1. Theoretical background and numerical simulations," *IEEE Antennas and Propagation Magazine*, vol. 43, no. 4, pp. 28–43, 2001.

[5] B. U. V. Prashanth, P. A. Kumar, and G. Sreenivasulu, "Design & implementation of floating point ALU on a FPGA processor," in *Proceedings of the International Conference on Computing, Electronics and Electrical Technologies (ICCEET '12)*, pp. 772–776, Kumaracoil, India, March 2012.

[6] B. U. V. Prashanth, "Implementation of FIR filter and FFT systems on a STRATIX-III FPGA processor," *International Journal of Computer Applications*, vol. 39, no. 3, pp. 16–19, 2012.

[7] B. U. V. Prashanth, C. Padmini, and S. Rajendar, "Design and Implementation of a Floating Point ALU on a STRATIX-III FPGA," *International Journal of Computer Applications*, vol. 55, no. 2, pp. 48–50, 2012.