Innovative Systems Design and Engineering
ISSN 2222-1727 (Paper) ISSN 2222-2871 (Online)
Vol.4, No.8, 2013

Development of FPGA Based System for Neutron Flux Monitoring in Fast Breeder Reactors

M.Sivaramakrishna, Dr. P.Chellapandi, IGCAR, Dr.S.V.G.Ravindranath (BARC),
IGCAR, Kalpakkam, India
(sivarama@igcar.gov.in)

Abstract
The project aims to calculate the frequency of the neutron flux by monitoring the signal from neutron detector from shutdown to full power over 10 decades. This neutron flux signal is input to the FPGA based MODULE. A mathematical relationship has been established between the neutron flux (frequency of the neutrons) and the area under the signal. Variable amplitude and occurrence have been accounted for. White noise has also been added and tested for. VHDL has been used to simplify the otherwise complicated logic gate design. Mathematical modeling has been used as it is the most accurate of the available methods.
Index Terms -- Neutron flux monitoring, area, pulses

1. Need of the system

Currently, neutron flux is monitored in all states of the reactor by Neutron flux monitoring systems. The system consists of several sets of detectors and instrument channels. For smooth transition from one set of instrument channel to other, interlocks with auto inhibition in safety logic are provided. In each channel, there is a trade-off between response time and accuracy. Volume of electronics involved is very high.
In addition, the pulses from the neutron detector are not periodic. Hence counting techniques do not result in accurate prediction of frequency. It can also be seen that as the frequency goes up the pulses over lap. Hence estimating the power using pulse counting can't predict the power correctly.
Currently, the detector is operated in pulse and Campbell modes. Even in Campbell mode, there is a trade-off between the accuracy and response time and linearity is obtained only for 4 decades.
FPGAs are currently the most user friendly and economically viable option for logic circuit design. They can be programmed to match user's requirements.
The work aims to find a relation between the frequency of the neutron flux signal and a mathematical function. The code designed will be able to calculate frequency for signals with constant amplitude, random amplitude, random occurrence and signals with noise from the samples supplied by the analog to digital convertor (ADC) connected to the FPGA.

2. Data and assumptions

- The pulse width varies. However, a width of 100 ns is a good estimate. The signal rise time varies from 5 ns to 20 ns. The fall time varies from 50 ns to 120 ns
- The amplitude of the signal is varying 0.7 uA to 1.3 uA
- The individual signals might overlap resulting in a single larger pulse.
- The signals will be affected by noise, which is also random in nature.
- The occurrence of signals follows a Poisson distribution.
- There is almost no overlap of signals for a frequency of less than 10^4 Hz. Beyond this, pulses will overlap. Due to this, the standard deviation will be proportional to the neutron flux, as per campbell theorm (which is applicable to statistical random occurring, discrete overlapping incidents).
- At very high frequencies, the pulse over lap fully to give average DC current.

To solve the problems such as range, accuracy with the existing techniques of neutron detector instrumentation, new techniques are investigated such as calculation of the area under the signal pattern (Curve).
Presently software simulation is completed to find out the relation between the above frequency parameters and the incident neutron flux. Hardware simulation is being carried out. Finally FPGA based simple embedded systems will be made for need of real time high computation required.

3. APPROACH

3.1 INITIAL STAGES

At first, sample signals were generated to mimic the neutron flux signals in terms of rise time, fall time, overlap, noise etc. From this signal mathematical functions such as average, variance and area were calculated.
While considering the overlap, a linear relationship was taken. For each decade starting from 10^4 Hz, a 20% overlap was considered upto 10^9 Hz.

Initially samples were run for constant overlap. After the appropriate mathematical function was obtained, it was extended to random amplitude and to noise.
Functions considered:
1) Average
We can see that for average, a straight line relation is obtained with frequency in case of both no overlap and overlap in signals.
There is no discrepancy from the straight line at any frequency. However, calculation with such precision will require very high sampling rates at the order of GHz. Hence, we search for a better option

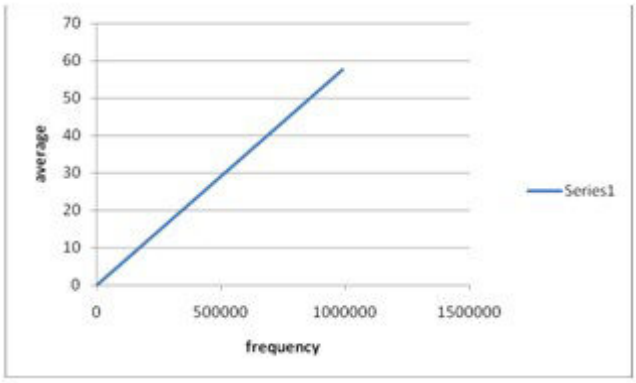

Fig 1: plot of average value of neutron flux vs. frequency

2) Variance
For variance, at lower frequencies and no overlap, we get a straight line relationship. However as the frequency and overlap increase, there is an exponential variation.
We also see that the rise of variance with frequency is sudden and large making it harder to distinguish between the higher frequencies. Hence, this is not the best method to follow as accuracy will be low.

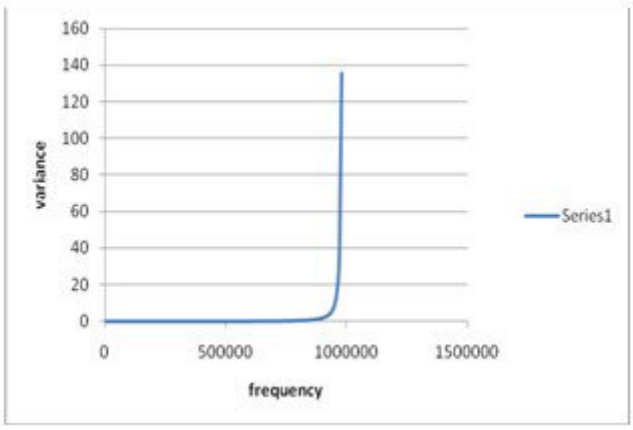

Fig 2: plot of variance of neutron flux vs. frequency

3) Area under curve
For area under curve we see that like variance and average, there is a straight line relationship at lower frequency and in the absence of overlap. However, with increase in frequency and overlap, the area under curve increases polynomially.
We can see that at each level of overlap, the frequencies follow a linear pattern. Overall, when we look at the curve, we see the increase is more gradual and a more distinguishable pattern is observed.

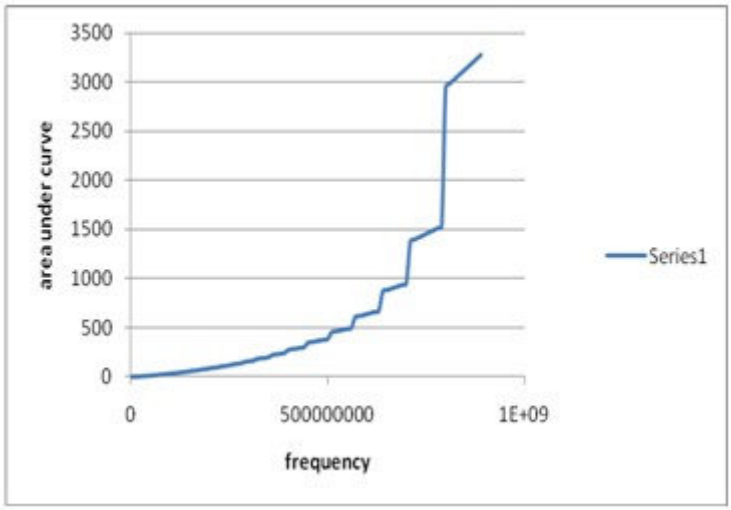

Fig 3: plot of area under curve of neutron flux vs. frequency

A clearer pattern is observed, when the curve is spilt into different portions and more values are taken in each range.
Hence, we decide to go ahead with area under curve as it is best suited for the situation.
Now we tried the pattern for random amplitude.
Initially we taook only 3 amplitudes at 0.7, 1 and 1.3 times the constant amplitude considered (1V)

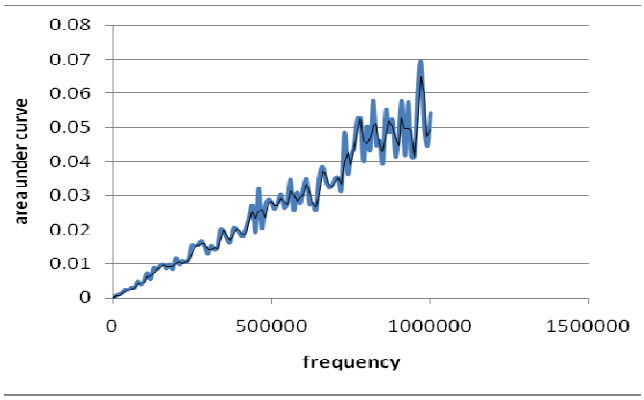

Fig 4. Area under curve vs. frequency (for amplitude of 0.7, 1, 1.3 V)

We see that due to the randomness in amplitude, the pattern immediately disappears and is replaced by a random waveform.

However, if we take this further and completely the randomize the amplitude to all values between 0.7 to 1.3 (a 3 sigma Poisson variation), we notice that the amplitudes average out to give a linear pattern

This linear pattern is observed at all frequencies. However the error is lower at higher frequencies because more the pulses, higher are the chances of the averaging of the values to the mean level.

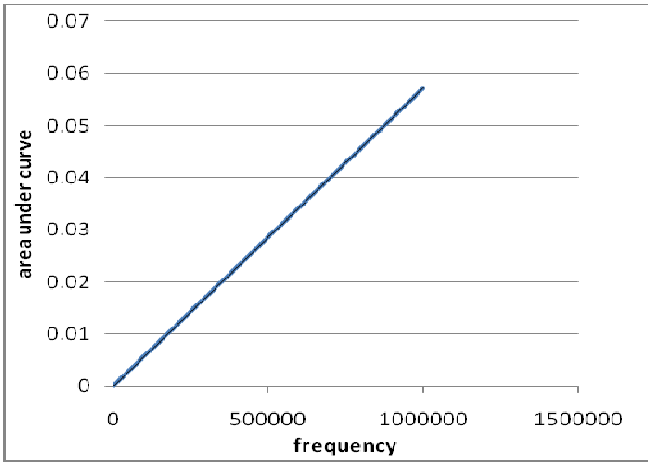

Fig 5. Area under curve vs. frequency (for completely randomized amplitude 0.7-1.3 V)

3.2 PATTERNS OBSERVED

Once area under curve was finalized as the function to be considered, more tests were run with multiple values in all ranges. Pattern was identified for each range of frequency and overlap.

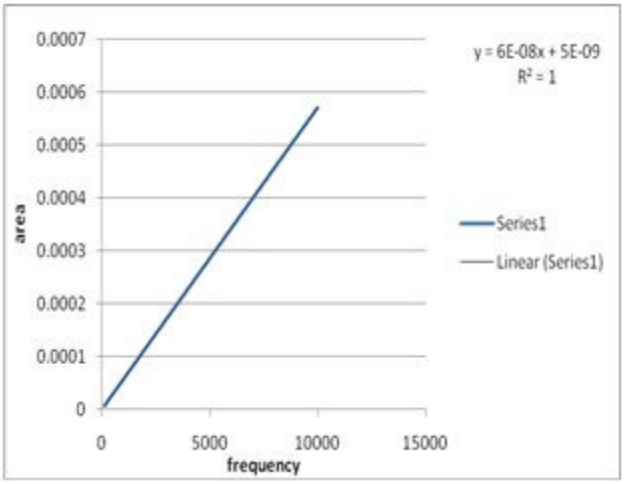

Fig 6. Area under curve vs. frequency (0 to 10^4 Hz)

Below are the plots obtained for each range. In the curve, the equation corresponding and the variation of points from the plot are mentioned.

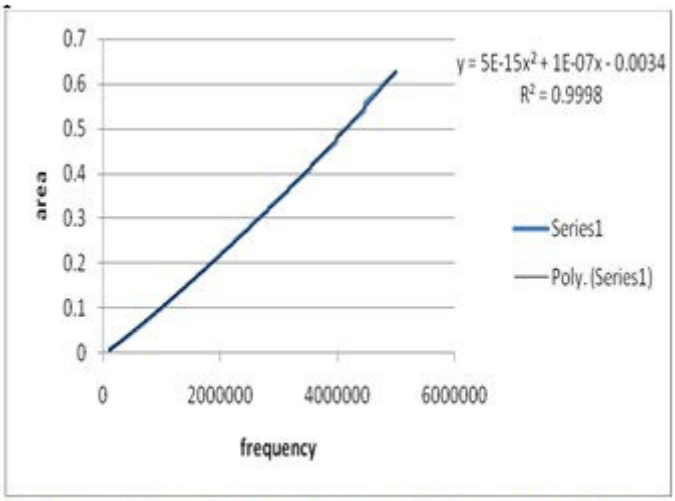

Fig 7. Area under curve vs. frequency (10^4 to 5*10^6 Hz)

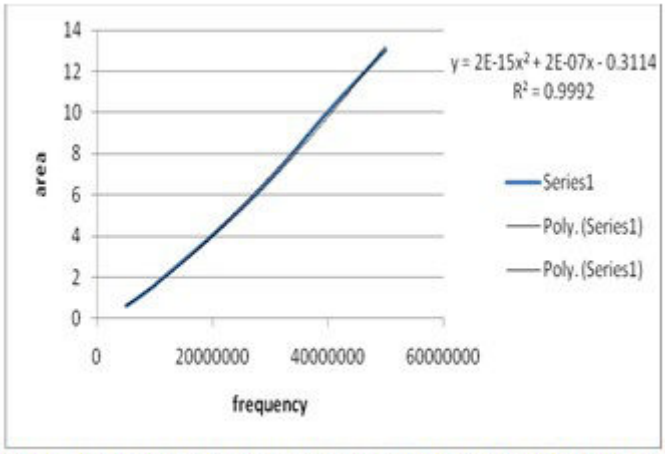

Fig 8. Area under curve vs. frequency (5*10^6 to 5*10^7 Hz)

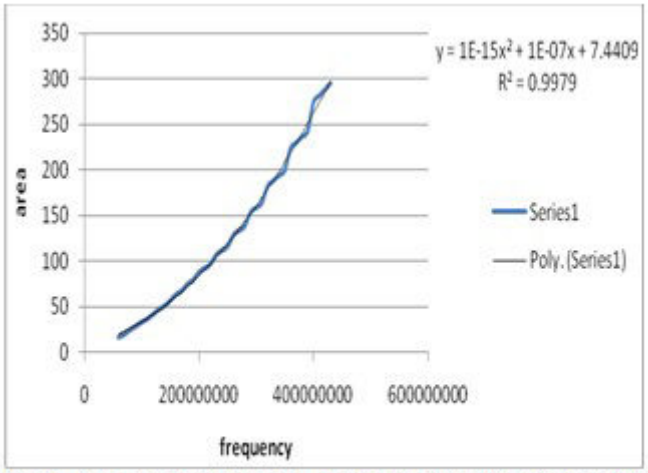

Fig 9. Area under curve vs. frequency (5*10^7 to 4.4*10^8 Hz)

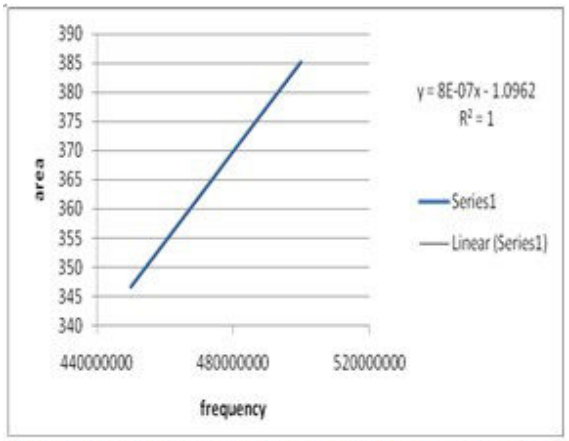

Fig 10. Area under curve vs. frequency (4.5*10^8 to 5*10^8 Hz)

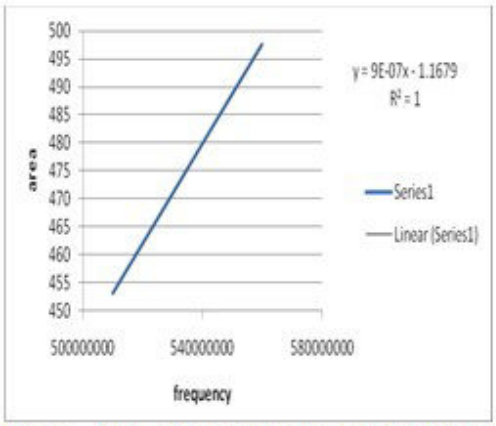

Fig 11. Area under curve vs. frequency (5.1*10^8 to 5.6*10^8 Hz)

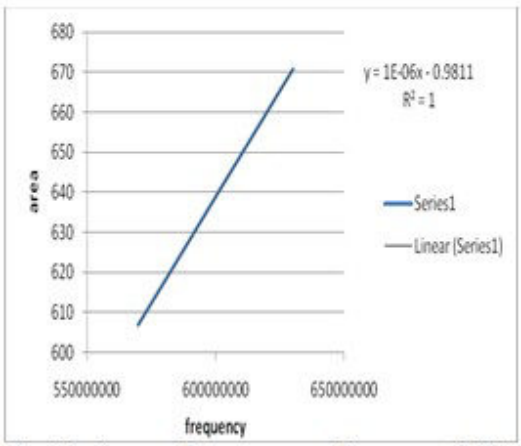

Fig 12. Area under curve vs. frequency (5.7*10^8 to 6.3*10^8 Hz)

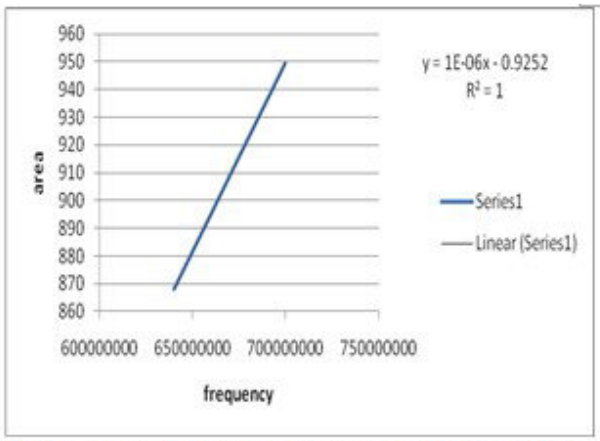

Fig 13. Area under curve vs. frequency (6.4*10^8 to 7*10^8 Hz)

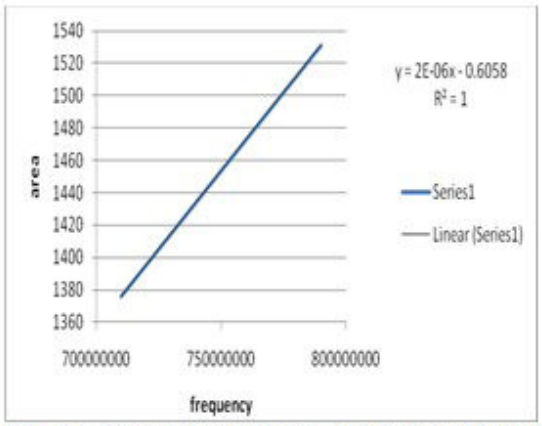

Fig 14. Area under curve vs. frequency (7.1*10^8 to 7.9*10^8 Hz)

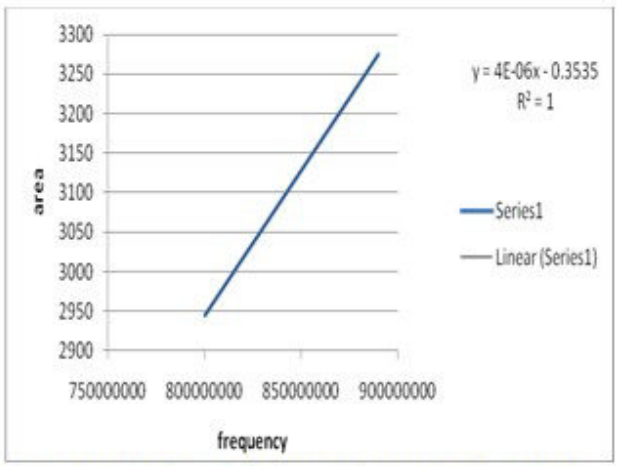

Fig 15. Area under curve vs. frequency (8*10^8 Hz onwards)

3.3 DEVELOPMENT OF CODE

After all patterns were identified, code was developed on VHDL for calculation of frequency from sampled signal.

Value will be read from a text file containing 10^7 values per second. The output frequency will be stored in another text file. A signal of '1' is output if frequency is below threshold else '0' is output.

The VHDL code for identifying mathematical, for sample signal generation, test values and for frequency calculation is in Appendix.

On changing the values in the places commented in the code, sample signal is generated for any frequency and corresponding value calculated by VHDL simulation is obtained.

4. RESULTS

Below are the results obtained from the simulation of the VHDL simulation.

Table 1 shows the value of frequency used for sample signal generation on matlab and the corresponding value obtained from VHDL for constant amplitude, random amplitude and occurrence and signals with noise. We see error increases slightly with very high overlap and frequency and is more in case of overlap with noise.
For constant amplitude, 80 more samples were taken, distributed equally in every decade, and tested to check deviation.The average error comes to 2.50699%.
Below is the plot of obtained value vs. average value. We see the pattern is almost linear and there is very slight deviation in pattern

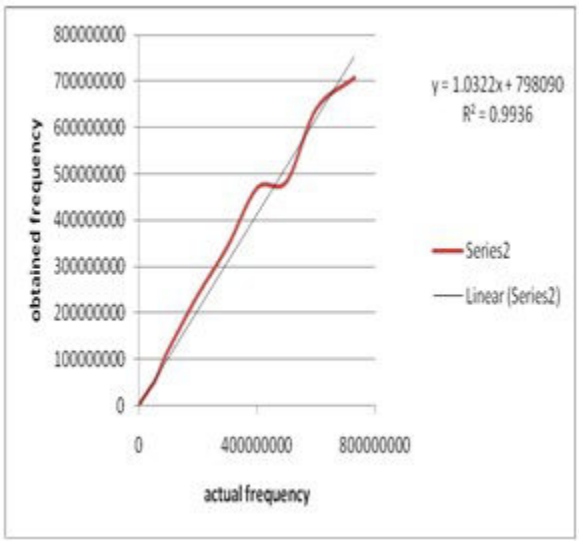

Fig 16. Obtained frequency vs. actual frequency

5. CONCLUSIONS

We see that the frequency calculation for constant amplitude is almost accurate with very small error. The error falls and then increases again at very high frequency.
For random amplitude, the error is larger at lower frequencies but reduces significantly at larger frequencies as the amplitudes average out.
For signals with noise, the error is lower at lower frequencies but is high at higher frequencies. However, here we have considered the noise to be mixed with the signal. In practicality, however, the noise will be separated out first. Hence error will be significantly lower. In this project, noise signals were analysed using the same pattern as constant amplitude case. If patterns for this are analysed like for the other cases, the error can be decreased. Due to lack of time this was not attempted.

BIBLIOGRAPHY

1. Perry, Douglas 1998, *VHDL by Examples,* 3rd edition, Singapore: Mc GrawHill
2. Bhasker, Jayaram, 1998, *VHDL Primer*, New Jersey: P T R Prentice Hall
3. Ashden, Peter J, 1990, *The VHDL Cookbook,* Ashenden Designs
4. Chu, Pong P, 2008, *FPGA prototyping by VHDL examples*, New Jersey: John Wiley & Sons Inc.
5. http://www.cs.umbc.edu/portal/help/VHDL/math_real.vhdl
6. http://www.velocityreviews.com/forums/t22430-random-number-generator.html
7. http://verificationguild.com/dload/utils/vhdl/random1.vhd
8. http://www.freemodelfoundry.com/converters_vhdl.php
9. http://www.jjmk.dk/MMMI/Exercises/05_Counters_Shreg/No7_PWM_vs_SigmaDelta/index.htm
10. Singh, Om Pal, 2007, *Interfacing Analog to Digital Converters to FPGAs*

Table I: Values obtained for random test samples

Test Frequency	With Constant amplitude	% error	With Random amplitude	% error	With noise	% error
3	2.774	7.66	2.48	17.3	2.77	7.5
70	66.59	4.8	65.39	6.5	66.6	4.87
800	761.9	4.76	759.2	5.09	763	4.68
3300	3142.	4.78	3144	4.69	3140	4.75
22000	21703	1.34	20992	4.58	19300	12
864000	856085	0.915	823105	4.73	757000	12.4
6123400	5428491	11.3	5832320	4.75	4560000	25
49200000	45283660	7.96			36200000	26
729000000	706756500	3.05			644000000	11.6

Annexure:
VHDL CODE

1) Constant amplitude

```
--for constant amplitude

library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use ieee.math_real.all;

entity area is

  end area;

architecture freq_calc of area is
signal clk: std_logic;

begin
clockgen:process
begin
clk<='1';
wait for 1ns;
clk<='0';
wait for 1ns;
end process;

process
variable area:real:=0.0;
variable freq:real:=0.0;
FILE infile: TEXT is in "C:\test\samples_const_9.txt";   --enter file name with samples here
FILE outfile: TEXT is out "C:/test/freq9.txt";           --enter file name to store result here
variable in_val,out_val:line;
variable val:real;
variable a:real:=0.0;
variable b:real;
variable c:real;
variable d:real;
variable flag:integer:=0;
variable check: std_logic:='0';
```



```
begin
wait until clk'EVENT and clk='1';
while not(endfile(infile)) loop
check:='1';
readline(infile,in_val);                    -- each line from file being read
read(in_val,val);                           -- value from each line being read
area:=area+val*1.0E-7;                      -- area being calculated.
wait until clk'EVENT and clk='1';           --sampling time=10^-7 seconds
end loop;

if (area<0.00756) then                      --frequency calculation
freq:=(area-5.0E-9)/(6.0E-8);

elsif(area<0.625) then
a:=5.0E-15;
b:=1.0E-7;
c:=-0.0034-area;
d:=b**2-4.0*a*c;
freq:=(-b+sqrt(d))/(2.0*a);

elsif(area<13.1) then
 a:=2.0E-15;
 b:=2.0E-7;
 c:=-0.3114-area;
 d:=b**2-4.0*a*c;
freq:=(-b+sqrt(d))/(2.0*a);

elsif (area<306.0) then
 a:=1.0E-15;
 b:=1.0E-7;
 c:=7.4409-area;
 d:=b**2-4.0*a*c;
freq:=(-b+sqrt(d))/(2.0*a);

elsif (area<386.103) then
freq:=(area+1.0962)/(8.0E-7);

elsif (area<498.0) then
freq:=(area+1.1679)/(9.0E-7);

elsif (area<671.0) then
freq:=(area+0.9811)/(1.0E-6);

elsif (area<950.0) then
freq:=(area+0.9252)/(1.0E-6);

elsif (area<1532.0) then
freq:=(area+0.6058)/(2.0E-6);

else
freq:=(area+0.3535)/(4.0E-6);

end if;
wait until clk'EVENT and clk='1';

write(out_val,freq);                        --writing value into file
writeline(outfile,out_val);
wait ;
```



```
end process;
end freq_calc;
```

2) For randomized amplitude

```
--for randomly varying amplitude
--solving linear equation

library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity area is
  end area;

architecture freq_calc of area is
signal clk: std_logic;
begin
clockgen:process
begin
clk<='1';
wait for 1ns;
clk<='0';
wait for 1ns;
end process;
process
variable area:real:=0.0;
variable freq:real:=0.0;
FILE infile: TEXT is in "C:\test\samples_rand_8.txt";--enter file with ADC samples here
FILE outfile: TEXT is out "C:/test/freq_r_8.txt";          --enter file to store results here
variable in_val,out_val:line;
variable val:real;
variable check: std_logic:='0';
begin
wait until clk'EVENT and clk='1';
while not(endfile(infile)) loop
check:='1';
readline(infile,in_val);
read(in_val,val);
area:=area+val*(1.0E-7);                                   --area calculation
wait until clk'EVENT and clk='1';
end loop;

freq:=(area-2.0E-8)/(6.0E-8);                              --frequency calculation
wait until clk'EVENT and clk='1';
write(out_val,freq);
writeline(outfile,out_val);
wait;
end process;
end freq_calc;
```

3) For signals with noise

```
--for constant amplitude
--with noise

library ieee;
```



```
use ieee.std_logic_1164.all;
use std.textio.all;
use ieee.math_real.all;

entity area is
end area;

architecture freq_calc of area is
signal clk: std_logic;

begin
clockgen:process
begin
clk<='1';
wait for 1ns;
clk<='0';
wait for 1ns;
end process;

process
variable area:real:=0.0;
variable freq:real:=0.0;
FILE infile: TEXT is in "H:\test\result81.txt";          --enter file name with samples here
FILE outfile: TEXT is out "H:/test/a81.txt";             -- enter file to store results here
variable in_val,out_val:line;
variable val:real;
variable a:real:=0.0;
variable b:real;
variable c:real;
variable d:real;
variable flag:integer:=0;
variable check: std_logic:='0';
begin
wait until clk'EVENT and clk='1';
while not(endfile(infile)) loop
check:='1';
readline(infile,in_val);
read(in_val,val);
area:=area+val*1.0E-7;                                   --sampling interval of 10^-7 seconds
wait until clk'EVENT and clk='1';
end loop;

area:=area*2.0/3.0

if (area<0.00756) then                                   --frequency calculation
freq:=(area-5.0E-9)/(6.0E-8);

elsif(area<0.625) then
a:=5.0E-15;
b:=1.0E-7;
c:=-0.0034-area;
d:=b**2-4.0*a*c;
freq:=(-b+sqrt(d))/(2.0*a);

elsif(area<13.1) then
 a:=2.0E-15;
 b:=2.0E-7;
 c:=-0.3114-area;
```



```
 d:=b**2-4.0*a*c;
freq:=(-b+sqrt(d))/(2.0*a);

elsif (area<306.0) then
 a:=1.0E-15;
 b:=1.0E-7;
 c:=7.4409-area;
 d:=b**2-4.0*a*c;
freq:=(-b+sqrt(d))/(2.0*a);

elsif (area<386.103) then
freq:=(area+1.0962)/(8.0E-7);

elsif (area<498.0) then
freq:=(area+1.1679)/(9.0E-7);

elsif (area<671.0) then
freq:=(area+0.9811)/(1.0E-6);

elsif (area<950.0) then
freq:=(area+0.9252)/(1.0E-6);

elsif (area<1532.0) then
freq:=(area+0.6058)/(2.0E-6);

else
freq:=(area+0.3535)/(4.0E-6);

end if;
wait until clk'EVENT and clk='1';

write(out_val,freq);                                    --writing value into file
writeline(outfile,out_val);
wait ;

end process;
end freq_calc;
```