WORK PROJECT REPORT
Cern Summer Student Program 2010

GEORGIA THEANO PAPADAKIS

National Technical University of Athens,Greece

Project: **STATIC AND DYNAMIC TESTING OF A 14-BIT ADC-CHARACTERIZING THE ADCs**

Department: BE
Group: CO
Supervisors: Javier Serrano
Juan David Gonzalez Cobas
Samuel Iglesias Gonsalvez

CERN-EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
1211 GENEVA-SWITZERLAND

September 3, 2010

1.Introduction

By the time I arrived at Cern , the BE/CO group had just constructed a new 14-bit resolution ADC (FMC ADC 100M 14b 4cha). An ADC is a device that converts an analog signal into a digital one. The goal of my main project was to use Python for developing algorithms that would measure all the noise and distortion quantities related to ADCs,in order to characterize the new ADC. Consequently,my work would be separated in two parts; getting familiar with Python in order to use it as a tool for developing algorithms and ,also, getting familiar with signal processing procedures in order to be able to handle the output signals coming from the ADC.

2.Basic Parameters of an ADC

The transfer function of an ADC is shown below.

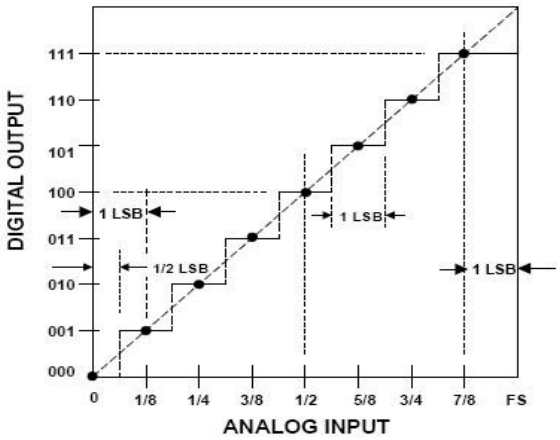

Figure 1-The ideal transfer function of an ADC

The basic parameters that have to be presented are;

- The number of bits M. An ADC with M number of bits has 2^M digital values to represent an analog signal.
- FSR stands for full-scale range and is the range of analog values that the device can represent.
- LSB stands for least significant bit and is the minimum change in voltage required to guarantee a change in the output code level and is given by the formula:

$$LSB=\frac{FSR}{2^M} \qquad (1)$$

3.Plan of the Project

By studying about the testing methods of ADCs,I found out that there are a lot of different specifications and a lot of different methods. I was asked to make a plan of how exactly I would test the new ADC and write a short tutorial about it. After some research I was led to the conclusion that the most practical way to test an ADC like the "FMC ADC 100M 14b 4cha" was to supply it with a full scale amplitude sine wave input, and to use the digitized samples of it (that have values from 0 to 2^{14} ,for a bipolar ADC) to measure non-linearities and specific noise and distortion quantities. The method is divided in two parts:Static and Dynamic testing. So,at the end,I also had two algorithms;one for each kind of testing.

3.1. Static Testing

The static testing of ADCs contains the measurements of two basic parameters: DNL and INL.

- DNL (differential non linearity) error is defined as the difference between an actual step width of the transfer function of the ADC (which is shown above) and the ideal value of 1LSB.
- INL (integral non linearity) is described as the deviation, in LSB or percent of full-scale range , of an actual transfer function from a straight line.

To measure those two quantities,using the given sine wave digitized samples,it is necessary to reconstruct the probability density function of the digitized sine wave ,as a histogram ,and compare it to the ideal probability density function of a sine wave .Once the histogram is made,the DNL and INL errors can be calculated using the following formulas:

DNL error; $$DNL(n)=\frac{h(n)_{ACTUAL}}{p(n)M_T} \qquad (2)$$

INL error ; $INL_j = \sum_{n=1}^{j} DNL_j$ (3)

where h(n) is the number of occurrences per bin in the output histogram and p(n) is the probability density of the sine wave. M_T is the number of samples.

3.1.1.Static Testing Algorithm Details

The algorithm I developed can be found at the end of this document .This paragraph's goal is to describe the main structure of the algorithm. Before presenting the structure,I should note some ascertainments I had made:First of all,the number of samples M_T should be very big;much bigger than the number of possible digital values that the ADC can represent. This wasn't possible to be done,so I used sine wave samples that I generated myself using a small sine wave generator routine that was simple to develop.

Also,the number of samples should always guarantee an integer number of sine wave periods. In addition,the sine wave' s frequency and the sampling frequency should not be correlated in any way in order to receive as many different digital values as possible,for the histogram to be reliable,according to IEEE Standard.

The algorithm that I developed for the static testing is simple;I used a vector in which I appended the digitized-sine-wave values that I made by using the samples I generated. To digitize the samples I developed a small routine:I corresponded every analog value to a digital one by comparing each analog value to every element of a vector called ”analog” ,whose elements are integer multiples of LSB. Then,I appended to a new vector the digital values that where stored in another vector,proportional to “analog”, called “digital” which contains the corresponding digital values .By using that new vector, I made a histogram in order to measure the number of occurrences per bin,meaning the number of times that a specific digital value was observed. I also used another vector in which I appended the theoretical values of the probability density function(p(n) in formula (2)) per bin. p(n) should always look like the figure below. Then I used the formulas (2) and (3) to calculate a vector of DNL and INL values. My algorithm also calculates the maximum deviation of the transfer function,which is another way to measure INL.

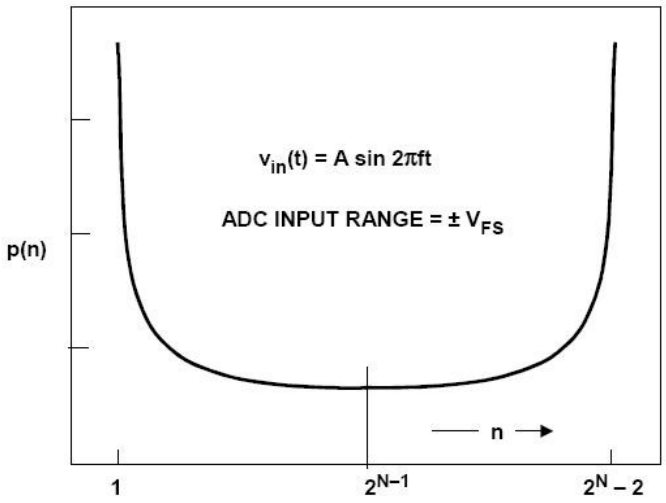

Figure 2-probability density function of a sine wave

3.1.2.Static Testing Results

Below,there is an execution of my Static Testing algorithm. As an input I used a sine wave with 0.314Hz frequency,5Volts amplitude and the sampling rate of the ADC is 1000Hz .

```
give number of bits :8
give me fsr(volts): 10
please give amplitude(volts):5
please give the sinewave's frequency(Hertz):0.3.1415164576
please give phase:0
please give sampling rate(Hertz):1000
please give number of samples:100000
if the ADC is bipolar then press 1,else press 0 :1
the number of occurencies per bin are  3927, 1634, 1252, 1061,  933,...,  1095,1295, 1682, 4056
the lsb is : 0.0390625
  >>> dnl
 array([ 2.55229473e-01,   1.48077630e-01,   1.07063086e-01,   1.13477963e-01,   1.42539189e-01,
        1.87508411e-01])
>>The maximum deviation of the transfer function is : 0.255229473128 LSBs. Which means : 0.0996990129405 %FSR.
```

The algorithm also produces the plot of both the ideal histogram and the histogram made out of the digitized samples. For the input described above,this plot is shown below.

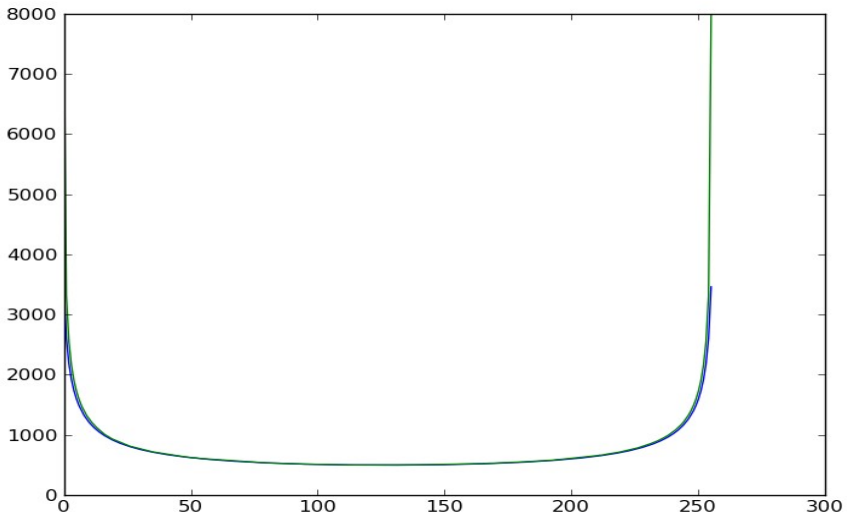

Figure 3-The theoretical and the calculated number of occurrences per bin

In this example,as it should,DNL and INL are calculated in LSBs. INL is also calculated as a percentage of FSR. The value of INL is in the correct order of magnitude and,also,the value of it is quite satisfying. It is to be expected to have DNL and INL errors even in the case that a pure sine wave is used as an input because,as it can be seen in the routine above,the way my “software ADC” works is exactly the same as the way that a real ADC works where the non-linearities are caused by the inside – imperfections of the ADC:meaning that it isn't easy to obtain all the different digital values that the ADC can represent:that would need a huge number of samples or a very high sampling frequency but that doesn't always guarantee a finite executing time so a compromise is made. An interesting observation is that, as the sampling frequency and the number of bits increase,the DNL and,consequently,the INL errors decrease(as they should).

3.2. Dynamic Testing

The dynamic testing contains the calculation of the following noise and distortion-related parameters;SNR,THD,SINAD and ENOB.

- SNR (signal to noise ratio) shows how much a signal has been corrupted by noise. The theoretical formula to calculate it is:

$$SNR_{THEORETICAL}=6.02*M+1.76 \text{ (dB)} \quad (4)$$

and the formula that shows the relation between the signal's power and the noise is:

$$SNR = S - NOISE\ FLOOR - 10\log_{10}(M_T/2) \text{ (dB)} \quad (5)$$

- THD (total harmonic distortion) is defined as the ratio of the signal to the root-sum-square (rss) of a specified number harmonics of the fundamental signal .The number of the harmonics that will be included in the calculation depends on the case. In this 14-bits resolution ADC,I decided to take into account as many number of harmonics as my algorithm can detect,based on the real position of the frequencies of the harmonics that should be an integer multiple of the sine wave 's frequency ,and always confirming that there is a local maximum at that frequency(see FFT figure below). So THD is calculated by the following formula:

$$THD = 20\log\left(\sqrt{10^{[V_2/20]2}+10^{[V_3/20]2}+..+10^{[V_{10}/20]2}}\right)=10\log\left(10^{[V_2/20]2}+10^{[V_3/20]2}+...+10^{V_n/202}\right) \quad (6)$$

where $V_2,V_3,...,V_n$ are expressed in dB.

- SINAD (signal to noise and distortion) is a measure of the quality of a signal and is defined as:

$$SINAD = \frac{P_{signal}+P_{noise}+P_{distortion}}{P_{noise}+P_{distortion}} \quad (7)$$

In the procedure of dynamic testing of the ADCs, SINAD is going to be measured by using the following formula:

$$SINAD = 20\log\left(\sqrt{10^{[-SNR/20]^2}+10^{[-THD/20]^2}}\right) \text{ (dB)} \quad (8)$$

• ENOB (effective number of bits) represents the quality of a digitized signal .All real signals contain a certain amount of noise. If the AD Converter is able to represent signal levels below the system noise floor, the lower bits of the digitized signal only represent system noise and do not contain useful information. ENOB specifies the number of bits in the digitized signal above the noise floor. An often used definition for it is:

$$ENOB = \frac{SINAD - 1.76 + factor}{6.02} \tag{9}$$

where all values are given in dB. The quantity "factor" equals to:

$$factor = 20\log\left(\frac{full\ scale\ amplitude}{real\ amplitude}\right) \tag{10}$$

and is a correction factor that "normalizes" the ENOB value to full-scale range regardless of the actual signal amplitude. Full scale amplitude is half the FSR for a bipolar ADC.

3.2.1.Dynamic Testing Algorithm Details

The algorithm I developed for the Dynamic-testing can also be found at the end of this document. To calculate the quantities mentioned above,I decided to take the Fourier Transform of the digitized samples of the "perfect" sine wave,to convert all the quantities into dB units and work in frequency domain. This time,I used real samples from the ADC to test my algorithms. Firstly,I used a vector in which I appended the values of the digitized samples. I decided to use also a window function ,and multiply it with the digitized sine wave and then take the FFT of that product. I did that to make the important parts of the FFT clear in order to discriminate the signal,noise and distortion from each other more easily. I used as a window function the Bartlett function but I could also use the Hamming,rectangular or even some other window function. I should note here that in order to obtain spectrally pure results,the FFT data window must contain an exact integral number of sine wave cycles,otherwise spectral leakage would occur. After that,the noise and distortion quantities had to be calculated.

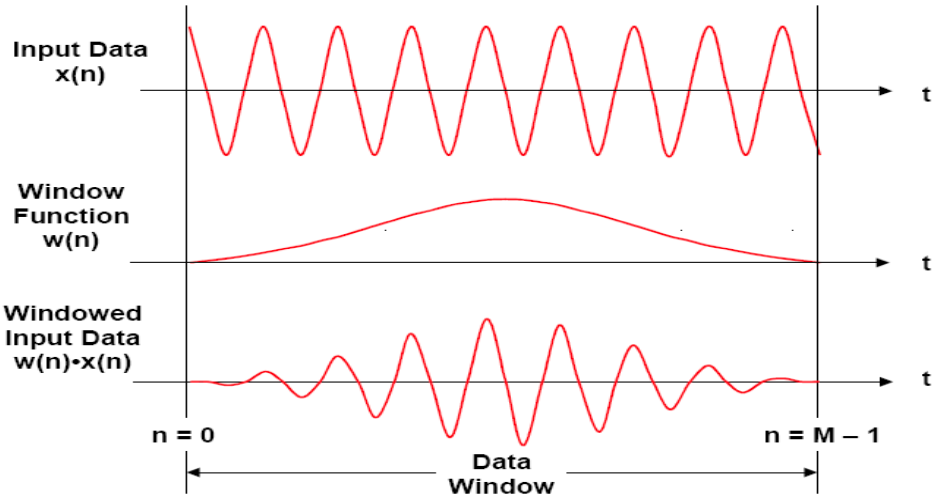

Figure 4-The use of the window function

Firstly,I developed a routine to detect the peaks of the harmonics. To do that,I used two vectors : one to append the results of the FFT of the multiplication of the window function and the digitized-sine wave and one to keep the corresponding frequencies. The condition I used in order to recognize a value as a peak or not was double: firstly,the algorithm searches for local maxima of the FFT. Then,it recognizes them as distortion only if the corresponding to those peaks frequency values are close to a value that equals to an integer multiple of the sine wave' s frequency."How"close depends on the number of samples. The distortion was then measured as the rss value of the magnitudes of those detected peaks. This is how THD could be calculated by using equation (6).

Following,the noise was now easy to measure:any component of the signal that is smaller than the mean value of the distortion is recognized as noise. So,the rss value of those components of the initial vector is the noise. Of course,the pure sine wave 's component is excluded. This is how SNR is being calculated by using equation (5).

The calculation of ENOB and SINAD was than easy to do by using equations (8),(9),(10).

3.2.2.Dynamic Testing Results

Below there is an execution of the Dynamic Testing Algorithm that I developed.

```
GIVE ME THE NUMBER OF BITS :14
GIVE ME THE INPUT AMPLITUDE :5
GIVE ME THE FULLSCALE AMPLITUDE :5
GIVE ME THE NAME OF THE FILE:samples_2.dat
NUMBER OF SAMPLES ARE(-SIZE OF a IS) :  8192
THE SIZE OF a EQUALS TO:  2 TO THE  13.0
SIZE OF hhalf IS: 4096
the frequency of the sinewave is  1562500
maximum of hhalf is  -9.64327466553e-16
[ 0.08        0.08000054  0.08000217 ...,  0.08000217  0.08000054  0.08      ]
THE NOISE FLOOR IS : -105.588459196
******************************** SNR ***********************************************
SNR IS (quick and dirty) : 69.4648597164
SNRtheoretical IS : 86.04
******************************** THD***********************************************
the REAL position of distortion is:  [3125000, 4687500, 10937500]
the REAL distortion is:  [-87.859262900247728, -81.114456373310162, -89.413998243431095]
THD IS : -80.2808572929
****************************** SINAD *********************************************
SINAD IS : 69.1190942732
****************************** ENOB **********************************************
ENOB IS : 11.1892183178
```

The samples used for this execution are real samples taken from the new 14-bit resolution ADC. As it can be seen from those results,there is a difference between the theoretical value of SNR,calculated using equation (4) and the real one calculated using equation (5).All of the values presented above are reasonable,although not very satisfying,according to specifications found on ADC datasheets. The algorithm also produces a plot of the FFT and this is shown below.

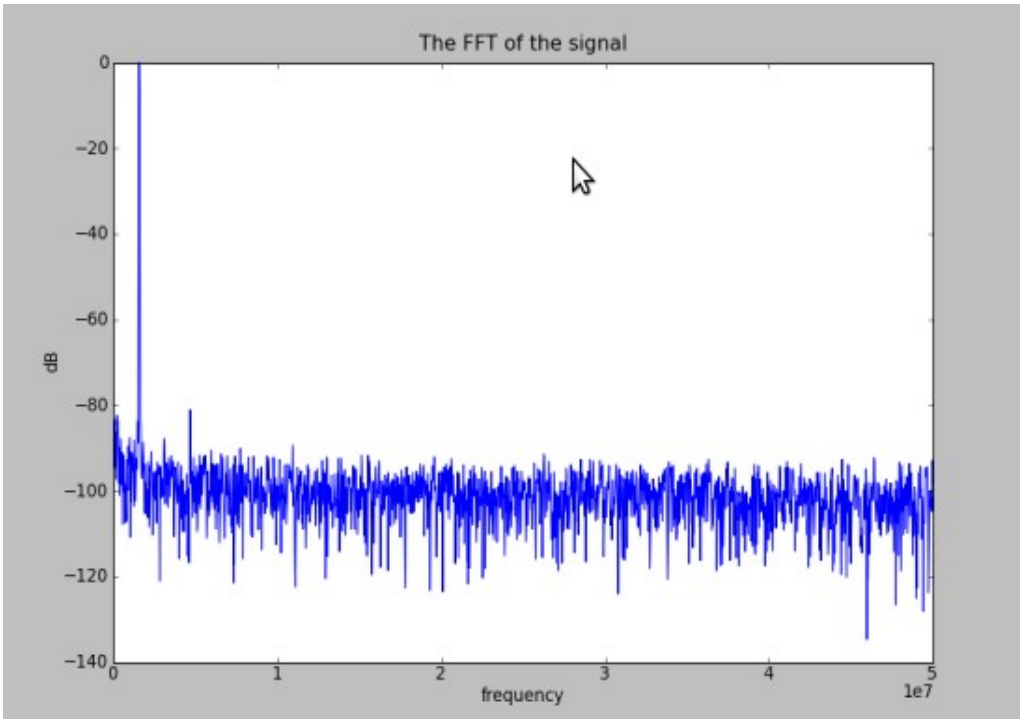

Figure 4-The Fourier Transform of the Sine wave

The highest peak in the figure above represents the sine wave' s frequency component. Some of the other peaks represent the distortion caused by the harmonics and the rest of the signal represents the noise floor.

ANNEX

The algorithms

A. Static Testing Algorithm

```
from math import *
from matplotlib import pyplot as plt
from numpy import *
#**********************************CREATING THE DISTORTED SINEWAVE*******************************#

def make_sinewaveforunipolar (n,sampl_rate,fr,ph,amp,fullsr):
    #produces the values of the sinewave#
     b=[]
     maj=[]
     for i in range (n):
      # b.append(fullsr/2.0+amp*sin(2.0*pi*fr*i/sampl_rate+ph*2.0*pi))
         b.append(fullsr/2+amp*sin(2*pi*fr*i/sampl_rate+ph*2*pi)+0.8*amp*sin(4.44*pi*fr*i/sampl_rate+ph*2*pi)
     +0.09*amp*sin(6.66*pi*fr*i/sampl_rate+ph*2*pi))
     return b

def make_sinewaveforbipolar (n,sampl_rate,fr,ph,amp,fullsr):
   #produces the values of the sinewave#
   b=[]
   for i in range (n):
        b.append(amp*sin(2.0*pi*fr*i/sampl_rate+ph*2.0*pi)+0.02*amp*sin(4.44*pi*fr*i/sampl_rate+ph*2*pi)
        +0.09*amp*sin(6.66*pi*fr*i/sampl_rate+ph*2*pi))
   return b

#*****************************************ANALOG TO DIGITAL*************************************#

def adc_transferunipolar(v, fsr, bits):

      #MAKING THE TRANSFER FUNCTION FOR UNIPOLAR ADC
      a=2.0**bits
      lsb=float(fsr/a)
      digital=[]
      analog=[]
      for i in range (int(a)):
         b=lsb*i
         analog.append(b)
         digital.append(i)

      #AND NOW THE CORRESPONDENCE!!!

      i=0
      if (v>=analog[0]) and (v<=analog[int(math.pow(2,bits)-1)]):
        while i<=a-1:
           if (v>=analog[i]) and (v<analog[i+1]):
              d=(digital[i])
           i=i+1
      elif (v>analog[int(math.pow(2,bits)-1)]) :
         d=( digital[int(math.pow(2,bits)-1)])
      else:
         d=(digital[0])

      return d
```

```
def adc_transferbipolar(v, fsr, bits,vfs):

    #MAKING THE TRANSFER FUNCTION FOR BIPOLAR ADC
    a=2.0**bits
    lsb=float(fsr/a)
    digital=[]
    analog=[]
    for i in range (int(a)):
       b=-vfs+lsb*i
       analog.append(b)
       digital.append(i)

    #AND NOW THE CORRESPONDENCE!!!

    i=0
    if (v>=analog[0]) and (v<=analog[int(math.pow(2,bits)-1)]):
      while i<=a-1:
         if (v>=analog[i]) and (v<analog[i+1]):
            d=(digital[i])
         i=i+1
    elif (v>analog[int(math.pow(2,bits)-1)]) :
       d=( digital[int(math.pow(2,bits)-1)])
    else:
       d=(digital[0])

    return d


#*************************************MAKING THE HISTOGRAM*********************************#

def histo(bits,samples):
    samples=array(samples)
    c=[]
    for i in range (2**bits):
       c.append(0)
    i=0


    for j in range (2**bits):
       for i in range(samples.size):
          if (samples[i]==j):
             c[j]=c[j]+1

    return c

#*********************************************THE INPUTS ******************************************#

b=input("give number of bits :")
fsr=input("give me fsr(volts): ")
a=input("please give amplitude(volts):")
f=input("please give the sinewave's frequency(Hertz):")
p=input("please give phase:")
sr=input("please give sampling rate(Hertz):")
s=input("please give number of samples:")
bipolar=input("if the ADC is bipolar then press 1,else press 0 :")




############################################## EXECUTION ######################################
```

```
fs=fsr/2

samples=[]                                  #making the sinewave's samples
if (bipolar==0):
   samples=make_sinewaveforunipolar(s,sr,f,p,a,fsr)
elif (bipolar==1):
   samples=make_sinewaveforbipolar(s,sr,f,p,a,fsr)
samples=array(samples)

dig_samples=[]       #the convertion
if (bipolar==0):
   for i in range (s):
      dig_samples.append(adc_transferunipolar(samples[i], fsr, b))
elif (bipolar==1):
    for i in range (s):
       dig_samples.append(adc_transferbipolar(samples[i], fsr, b, fs))
dig_samples=array(dig_samples)
#to be sure that the frequency of the sinewave and the sampling frequency are not correlated,i need dig_samples to have
#a lot of different from each other values
e=[]
e=histo(b,dig_samples)
#print ("the number of occurencies per bin are "),e[:]
print ("                                    ")
e=array(e)
ala=0                                       #this variable shows the number of different values
for i in range (e.size):
         if (e[i]!=0):
                ala=ala+1
h_first=e[0]
last=e.size
h_last=e[last-1]

#*************************CALCULATING THE ESTIMATED AMPLITUDE*******************************#
lsb=fsr/(2.0**b)
print("the lsb is :"),lsb
print ("                                    ")
g=(math.pi/2)
help1=float(s+h_first+h_last)
help2=float((s/help1))
w=float((g*(help2)))
k=float(math.sin(float(w)))
amp=float(fs/k)
print("the  estimated amplitude is "),amp
print ("                                    ")
#############################################################################################################

#************************************CALCULATING THE p(n)*************************************#

p=[]
twobits = 2.0**b
twobits1 = 2.0**(b-1)
pirecip  = 1.0/math.pi
for i in range(1,int(twobits)+1):
      p.append(pirecip*( math.asin(2*fs*(i  -twobits1)/(amp*twobits)) - math.asin(2*fs*(i-1-twobits1)/(amp*twobits)) ))
p=array(p)
print ("                                    ")

#############################################################################################################
```

```
#**********************************CALCULATING h(n)theoretical*************************************#

h=[]
for i in range(p.size):
   h.append(p[i]*s)
#print("                                  ")
h=array(h)
#print("the theoretical number of samples is :"),sum(h)
#**********************************CALCULATING DNL ERROR*************************************#
dnl=[]
help4=0.0
help5=0.0
for i in range (0,2**b):
      help4=(e[i]/h[i])
      help5=help4-1.0
      dnl.append(help5)
dnl=array(dnl)
alala=0
for i in range ((2**b)-2):
          if (dnl[i]!=-1.0):
                  alala=alala+1   #this variable calculates how many different values dnl vector has

print ("T H E  DNL  E R R O R S   A R E   :"),dnl[:]
#**********************************CALCULATING INL ERROR*************************************#
inl=0
for i in range (2**b-2):
      inl=inl+dnl[i]
#########################################################################################################

rang=range(p.size)
plt.plot(rang,h,rang,e)
plt.show()
```

B. Dynamic Testing Algorithm

```
from math  import *
from numpy import *
from matplotlib import pyplot as plt

#****************************************** OPEN THE FILE ***************************************#
num_of_bits=input("GIVE ME THE NUMBER OF BITS :")
real_amplit=input("GIVE ME THE INPUT AMPLITUDE :")
full_scale_amplitude=input("GIVE ME THE FULLSCALE AMPLITUDE :")
name=raw_input("GIVE ME THE NAME OF THE FILE:")
f=open(name)
a=[]
for line in f:
   value=int(line)
   a.append(value)

#**********************************************MAKING THE FFT************************************ #
a=a-mean(a)
a=array(a)
asize=a.size
x=log2(asize)
print ("NUMBER OF SAMPLES ARE(-SIZE OF a IS) : "),asize
print ("THE SIZE OF a EQUALS TO:  2 TO THE "),x
```

```
b=abs(fft.fft(a))
maxb=max(b)
d=b/maxb
e=log10(d)
f=20*e
h=fft.fftshift(f)
m=h.size

#************************I 'LL MAKE A COUPLE:VECTOR h AND VECTOR freq ****************************#
freq=[]
for i in range (m):
   freq.append(-50000000+i*100000000/m)

# **********************I'LL MAKE ANOTHER COUPLE:Vector hhalfh AND VECTOR freq_shift***************** #
hhalf=[]
freq_shift=[]
for i in range(m/2,m):
   freq_shift.append(freq[i])
   hhalf.append(h[i])
hhalf=array(hhalf)
freq_shift=array(freq_shift)
ser=hhalf.size
print ("SIZE OF hhalf IS:"),ser

#******SEARCH FOR THE POSITION OF THE MAXMUM OF hhalf=>EXPECTING THE INPUT FREQUENCY******#
for i in range (ser):
      maximum=hhalf[i]
      for j in range (ser):
      if hhalf[j]>maximum:
          maximum=hhalf[j]
          sinefreq=freq_shift[j]
print("the frequency of the sinewave is "),sinefreq
print("maximum of hhalf is "),maximum
#****************************************WINDOW FUNCTION*****************************************#
nbart=asize/2
d=hamming(nbart)
print d[:]

gerog=a[0:(asize/2)]
gerog=array(gerog)
d=array(d)
mult=d*gerog
mult=array(mult)
mult1=abs(fft.fft(mult))
maxmult1=max(mult1)
mult2=mult1/maxmult1
mult3=log10(mult2)
mult4=20*mult3
mult5=fft.fftshift(mult4)
multsize=mult5.size

freq_shift1=[]
for i in range (gerog.size):
          freq_shift1.append(-50000000+i*100000000/nbart)
for i in range (gerog.size):
          if freq_shift1[i]==0:
                    flag=i
fourievector=mult5[flag:multsize]
freqvector=freq_shift1[flag:multsize]


# **********CALCULATION OF THE DISTORTION WHEN THERE IS ALSO PROCESS GAIN SO THE**********#
```

```
#**************************HARMONICS ARE #A LITTLE SPREAD ************************************#
#**************THE ARRAYS THAT ARE GOING TO BE USED ARE: fourievector and freqvector *****************#

position=[]
distortion=[]
for i in range (fourievector.size):
#I TRY TO FIND THE POSITION OF THE HARMONICS BY SEARCHING IN freq_shift VECTOR FOR
#FREQUENCIES CLOSE TO THE:f=n*sinefreq
      if (freqvector[i]%sinefreq==0) :#if you find a harmonic
                if x%2==0 :
                  magnitude=[]
                  helpfreq=[]
                  if i>=2**(((x-2)/2)-1) and i<=fourievector.size-2**(((x-2)/2)-1)-1:
                        for j in range (int(-2**(((x-2)/2)-1)),int(2**(((x-2)/2)-1))):
                           magnitude.append(fourievector[i+j])
                           helpfreq.append(freqvector[i+j])
                   elif  i<2**(((x-2)/2)-1):
                              for j in range (0,int(i+2**(((x-2)/2)-1))):
                                 magnitude.append(fourievector[j])
                                 helpfreq.append(freqvector[j])
                   else:
                              for j in range (int(i-2**(((x-2)/2)-1)),fourievector.size):
                                 magnitude.append(fourievector[j])
                                 helpfreq.append(freqvector[j])
                            magnitude=array(magnitude)
                            helpfreq=array(helpfreq)
                            maxmag=max(magnitude)
                            sizemag=magnitude.size
                            for w in range(sizemag):
                                if magnitude[w]>=maxmag:
                                   maxmag=magnitude[w]
                                   position.append(helpfreq[w])
                                   distortion.append(magnitude[w])

                 else:
                          magnitude=[]
                          helpfreq=[]
                          for j in range(int(-2**(((x-3)/2)-1)),int(2**(((x-3)/2)-1))):
                              magnitude.append(fourievector[i+j])
                              helpfreq.append(freqvector[i+j])
                          magnitude=array(magnitude)
                          helpfreq=array(helpfreq)
                          maxmag=max(magnitude)
                          sizemag=magnitude.size
                          for w in range(sizemag):
                             if magnitude[w]>=maxmag:
                                maxmag=magnitude[w]
                                position.append(helpfreq[w])
                                distortion.append(magnitude[w])

#**********I ASSUMED THAT WHEN THERE IS A SIGNAL LEACAGE,ONLY THE HIGHEST PEAK IS ***********
#*************A HARMONIC AND THE REST OF THE PEAKS ARE NOTHING,NOT EVEN NOISE***************#

distortion=array(distortion)
distsize=distortion.size
position=array(position)
meandistortion=mean(distortion)

realdistortion=[]
realposition=[]
noise1=[]
for i in range(distsize):
```

```
    if (distortion[i]>=meandistortion) and (position[i]>sinefreq):
        realdistortion.append(distortion[i])
        realposition.append(position[i])


#********************************* C A L C U L A T I O N  O F  N O I S E ==> SNR *************************#

for i in range (ser):
    if hhalf[i]<meandistortion:
        noise1.append(hhalf[i])


noise1=array(noise1)
noisesize=noise1.size
noise2=[]
for i in range(noisesize):
    if noise1[i]>-140.0:
        noise2.append(noise1[i])

print ("THE NOISE FLOOR IS :"),mean(noise2)
snr=maximum-mean(noise2)-10*log10(asize/2)    # there is also the process gain:10*log10(asize/2) #
snr_theoret=6.02*num_of_bits+1.76
print("******************************** SNR ********************************************")
print ("SNR IS (quick and dirty) :" ),snr
print ("SNRtheoretical IS :"),snr_theoret
#*************************************************************************************************#
#************************  C A L C U L A T I O N  O F  D I S T O R T I O N ==> THD ***********************#

print("******************************** THD*******************************************")
print("the REAL position of distortion is: "),realposition[:]

print("the REAL distortion is: "),realdistortion[:]
realposition=array(realposition)
realdistortion=array(realdistortion)
thd1=[]

for i in range (realdistortion.size):
    thd1.append(realdistortion[i]/20.0)
thd1=array(thd1)
thd2=thd1*2
thd2=array(thd2)
thd3=10**(thd2)
thd4=0
for i in range (realdistortion.size-1):
    if thd3[i]!=1 :
        thd4=thd4+thd3[i]
thd=10*log10(thd4)
print ("THD IS :"),thd
#*************************************************************************************************#
#********************************* C A L C U L A T I O N  O F  SINAD ***********************************#

print("*******************************SINAD ********************************************")
sinad1=-(snr)/10.0
sinad2=10**(sinad1)
sinad3=-abs(thd)/10.0
sinad4=10**(sinad3)
sinad5=sinad2+sinad4
sinad=-10*log10(sinad5)
print("SINAD IS :"),sinad

#*************************************************************************************************#
```

```
#********************************** C A L C U L A T I O N  O F  ENOB ********************************#
print("**************************** ENOB ***********************************************")
factor=20*log10(full_scale_amplitude/real_amplit)
enob=(sinad-1.76+factor)/6.02
print ("ENOB IS :"),enob
#*****************************************************************************************************#

#************************************** PLOTTING ***************************************************#
freqvector=array(freqvector)
fourievector=array(fourievector)
plt.plot(freqvector,fourievector)
plt.title("The FFT of the signal")
plt.xlabel("frequency")
plt.ylabel("dB")
plt.show()
```

References

-ANALOG-DIGITAL CONVERSION-TESTING DATA CONVERTES

-ANALOG DEVICES : MT -003 TUTORIAL

-Advanced A/D and D/A Conversion Techniques and Their Applications -7th European Workshop on ADC Modelling and Testing

-IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters

-Waveform Recorder Testing: IEEE Standard 1057 and You [Thomas E. Linnenbrick]

-IEEE TRASNACTIONS ON INSTRUMENTATION AND MEASUREMENT ,VOL 57,NO 2,FEBRUARY 2008- Preliminary Considerations on ADC Standard Harmonization [Sergio Rapuano ,Member ,IEEE]

-Digital Signal Processing (4th Edition) :John G. Proakis ,Dimitris K. Manolakis

-Beginning Python (From Novice to Professional) : Magnus Lie Hetland

-http://www.maxim-ic.com/app-notes/index.mvp/id/2085

-http://www.maxim-ic.com/app-notes/index.mvp/id/729

Acknowledgments

I would like to thank Juan David Gonzalez Cobas and Samuel Iglesias Gonsalvez for being great teachers and,also, Maciej Fimiarz for his valuable explanations and for being so willing to help me.