

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



**FEUP**

# **Jitter Extraction Tool**

**Luís Cruz**

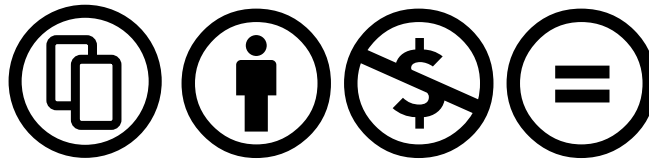
PROVISIONAL VERSION

Master in Electrical and Computer Engineering

Supervisor: José Carlos Alves (Prof.)

March, 2011

© Copyright by Luís Cruz, 2011  
Some rights reserved



## You are free:

---



**to Share** — to copy, distribute and transmit the work.

## Under the following conditions:

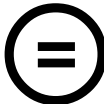
---



**Attribution** — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



**Noncommercial** — You may not use this work for commercial purposes.



**No Derivative Works** — You may not alter, transform, or build upon this work.

## With the understanding that:

---

**Waiver** — Any of the above conditions can be **waived** if you get permission from the copyright holder.

**Other Rights** — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

**Notice** — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

# Contents

<b>1</b>	<b>Software Selection</b>	<b>1</b>
1.1	Existent Solutions . . . . .	1
1.2	Programming Language Requisites . . . . .	1
1.3	Python Programming Language . . . . .	2
1.4	Python Configuration . . . . .	3
1.5	Conclusion . . . . .	3
<b>2</b>	<b>Jitter Extraction Tool</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	Q-scale on discrete events . . . . .	5
2.3	Q-scale Linearization . . . . .	6
2.4	extract_jitter tool . . . . .	7
2.4.1	Linearization . . . . .	7
2.4.2	Multi-thread . . . . .	7
2.4.3	How to use . . . . .	8
2.4.4	Results . . . . .	8
2.5	Conclusion . . . . .	9
	<b>References</b>	<b>13</b>



# List of Figures

2.1	Different Values for $A$ resulting on different jitter values . . . . .	6
2.2	Gaussian Jitter decomposition using <code>extract_jitter</code> tool . . . . .	10
2.3	Dual Gaussian Jitter decomposition using <code>extract_jitter</code> tool . . . . .	10
2.4	Dual-Dirac Gaussian Jitter decomposition using <code>extract_jitter</code> tool . . . . .	10
2.5	Periodic and Gaussian Jitter decomposition using <code>extract_jitter</code> tool . . . . .	11
2.6	Square and Gaussian Jitter decomposition using <code>extract_jitter</code> tool . . . . .	11



# List of Tables

2.1	Total Jitter Expected . . . . .	9
2.2	Total Jitter obtained for BER=1e-14 . . . . .	9
2.3	Total Jitter obtained for BER=1e-12 . . . . .	9





# Abbreviations and Symbols

BER	Bit Error Ratio (Number of errors divided by the number of transmitted bits)
BERT	Bit Error Ratio Tester
CAD	Computer-Aided Design
CASE	Computer-Aided Software Engineering
CDF	Cumulative Density Function
CDR	Clock and Data Recovery
CSV	Comma Separated Values
DDJ	Data Dependent Jitter
DJ	Deterministic Jitter
EDF	Empirical Distribution Function
IC	Integrated Circuits
ISI	Inter-symbol interference
PCS	Phase Changing Speed
PDF	Probability Density Function
PLL	Phased-locked Loop
RMS	Root Mean Square
RJ	Random Jitter
SerDes	Serializer/Deserializer
TIE	Time Error Interval
TJ	Total Jitter



# Chapter 1

## Software Selection

### 1.1 Existent Solutions

The development of new Integrated Circuits (IC) starts by developing a behavioral model of the circuit, allowing the system engineers to identify the correct platforms/solutions that must be implemented.

During the platform specification, system engineers define the blocks that must be implemented as well as the block parameter's like: gain, frequency response, bandwidth, maximum response delay, etc. The correctness of the block specification will be dictated by the accuracy of the behavioral models, at this point system engineers don't use languages like spice or even verilog-A, because it will require unnecessary work that will reduce the ability to choose between different platform solutions.

System engineers tend to use MATLAB or even ADS (Advanced Design System) tools, providing an abstraction layer to the real implementation allowing the development of accurate behavioral models. Such tools require a depth knowledge of the proprietary programming languages supported, requiring from engineers extensive training before starting the behavioral model creation. The use of proprietary languages reduces the number of engineers that have the skills to work as system engineers. MATLAB and ADS licenses are expensive reducing even more the number of engineers that have access to them. The ideal solution will be to have the ADS/MATLAB engine on a form of free programming language allowing a bigger number of engineers to understand and work at system level.

### 1.2 Programming Language Requisites

There are different programming languages suitable to use on the tool development, Octave is one of them. Octave is a MATLAB clone that supports the majority of MATLAB instructions, but as MATLAB the number of users is restricted and it will require also a previous extensive training. Octave isn't fast [1] and has limited support, reducing the ability to produce an interesting tool.

The programming language that will be use on the software tool development will have to respect the following requisites:

- Free
- Relatively fast
- Built-in signal processing functions
- Built-in plot functions
- Built-in probability functions
- Built-in functions for "electronic engineers"
- Support for interactive debugging
- Support for multi-threading
- Support for graphical interface
- Easy to use

### 1.3 Python Programming Language

Scientific groups are trying to use open source software to address the new projects, the use of Python for Scientific purposes has been increasing. Due to this fact new libraries are being added to python making it a really scientific tool.

Python is very well spread with good support forums on the internet, it's easier to find documentation and tutorials. One of the major advantages is related with portability, python doesn't need to be compiled, the same code will work in Windows, Linux and MacOS, reducing the inter operability problems, the user only needs to instal the python interpreter on their machine to be able to run python code.

Python packages extend the base python functions giving to the end user a powerfully engine to produce scientific tools. Numpy and scipy packages extend python to support: statistics, numerical integration, linear algebra, fourier transforms, signal processing, image processing, special functions, powerful N-dimensional array object, random number capabilities, between others. Numpy and scipy were build in c language ensuring good performance. matplotlib Package is a python 2D/3D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Pyrex package is a language specially designed for writing Python extension modules. It's designed to bridge the gap between the nice, high-level, easy-to-use world of Python and the messy, low-level world of C. Python also has a command line interpreter, making it user friendly. Graphical user interface development can be done using PyQt package, that has a substantial set of GUI widgets. multiprocessing is a package

that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using subprocesses instead of threads. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. There is the possible to perform calculations on the graphics card, pygtk package is the responsible for such capability.

Python is very easy to use with object oriented support. The variable manipulation doesn't require a type definition, such type will be dynamically allocated.

## 1.4 Python Configuration

On this project python 2.6 version was used due to the code stability and bigger support on the internet. The reader can be asking why it wasn't used python 3.x, this version it's a complete different distribution with a different engine and orientation. Python 2.6 was selected due to the extensive support and packages availability. Python(x,y) was used to provide a complete and stable environment removing the need to install the different packages one-by-one.

Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces, Eclipse integrated development environment and Spyder interactive scientific development environment. Python(x,y) can also be defined as a scientific-oriented Python Distribution based on Qt and Eclipse, its purpose is to help scientific programmers used to interpreted languages (such as MATLAB) or compiled languages (C/C++ or Fortran) to switch to Python.

Python(x,y) supports interactive debugging, allowing the user to place breakpoints across the code under development as the possibility to dynamically check the variables value. Python(x,y) interactive mode provides the necessary support aid during the development phase.

## 1.5 Conclusion



## Chapter 2

# Jitter Extraction Tool

### 2.1 Introduction

Jitter decomposition is very important to predict the system behavior for long bit sequences. Usually this type of analysis are performed over IC samples using expensive laboratorial equipment. The intent of this work is to present a tool capable of doing it but free.

extract\_jitter tool uses the Q-scale method to extract the deterministic and random jitter components. The user just need to provide a jitter histogram file on a csv (comma separated values) file format to gen\_hist.exe executable file to obtain a pdf file with the jitter extracted components.

### 2.2 Q-scale on discrete events

As described on chapter ?? Q-scale can be expressed as:

$$Q(x) = \sqrt{2} \operatorname{erf}^{-1} [1 - \operatorname{BER}(x) \cdot A] \quad (2.1)$$

Total jitter is determined by integrating the probability density function (PDF) separately from left and right to determine the symmetric cumulative density function (CDF). The width of this curve at the specified BER (or confidence interval) gives the total jitter. Meaning that  $\operatorname{BER}(x) = \operatorname{CDF}(x) = \int_{x'=-\infty}^{x'=\infty} \operatorname{PDF}(x') dx'$ . Of course  $\operatorname{CDF}(x)$  is purely theoretical, but it can however be calculated using the EDF (Empirical Distribution Function), summing the jitter histogram from the left extreme to the desired value of  $x$ .

$$\operatorname{EDF}(x = h_i) = \frac{\sum_{k=0}^{k=i-1} H_k}{\sum_{k=0}^{k=N-1} H_k} = \frac{1}{P_{total}} \sum_{k=0}^{k=i-1} H_k \quad (2.2)$$

Now, for the purpose of calculating  $Q(x)$  and keeping with the tradition of previous jitter discussions, it's necessary to calculate the left and right sides of the Q-scale, since the designer is interested in both variations in timing jitter, before and after the mean timing value. The first thing

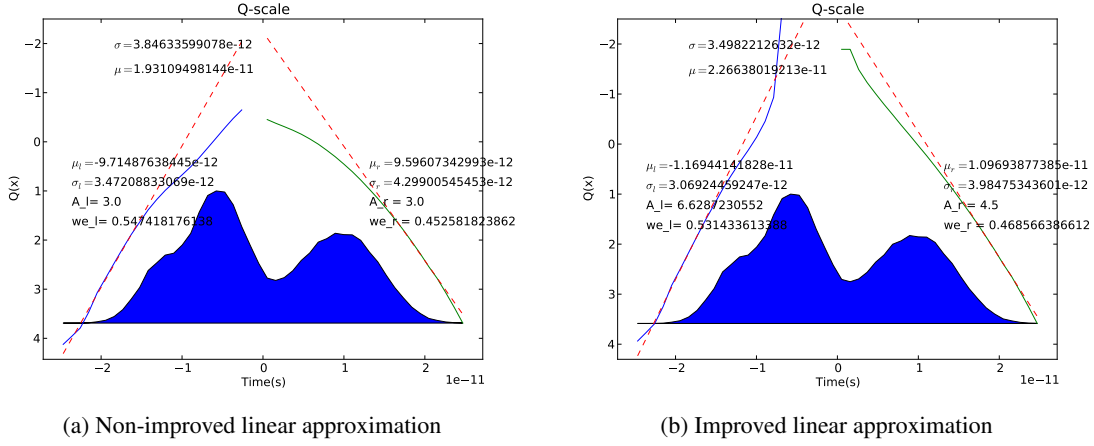


Figure 2.1: Different Values for  $A$  resulting on different jitter values

to do is to obtain the mean position value and then calculate the respective BER:

$$BER_l(x) = EDF_l(x = hi) = \sum_{k=0}^{k=i_{mean}-1} H_k \quad (2.3)$$

$$BER_r(x) = EDF_r(x = hi) = \sum_{k=N-1}^{k=i_{mean}} H_k \quad (2.4)$$

In practice, these two functions are joined at the median of the histogram (the bin containing the median, or the bin which 50% of the total population is in that bin or those with lower index). Consequently, 50% of the total population also falls within that bin and those bins with higher index.

### 2.3 Q-scale Linearization

Jitter decomposition tool is based on Q-scale method using a linear approximation to extract random and deterministic jitter values. Since  $Q(x) = \frac{x-\mu}{\sigma}$ , the relation between  $Q(x)$  and  $x$  can be given by a line, with  $slope = \frac{1}{\sigma}$  and  $y - intercept = \frac{-\mu}{\sigma}$ . The intent of `extract_jitter` tool is to find such values.

During the linearization process the variable  $A$  will be changed to reduce the error between  $Q(x)$  and the obtained linear approximation. The designer must be aware that a wrong value of  $A$  can result on wrong values for random and deterministic jitter extraction values. On Figure 2.1 it's clearly seen the difference on the extracted jitter values. These differences are caused by a bad linearization, because  $Q(x)$  is far from a straight line (the designer should only take into consideration values where  $Q(x) > 0$ ).

`extract_jitter` tool divides the jitter histogram in two parts (left and right) and then processes the data of each part independently. It's important to note that each piece has its own coefficient,  $A_r$  or  $A_l$ , allowing a correct linearization of each side. The use of two independent coefficients is



mathematically accurate because each side has their own deterministic jitter, different coefficients allows the consideration of different DJ profiles on each side.

In terms of linearization it's also interesting to notice that to correctly extract the jitter components it's necessary to consider only low probability events, because Q-scale method assumes that RJ follows a gaussian distribution and DJ can be defined by two dirac functions, if the designer uses high probability values the obtained results will be certainly wrong since the idea behind this method is no longer valid. The golden value can be obtained following this approach: Let's consider 15 as the maximum number of consecutive equal bits transmitted, using a PRBS function to generate the random bits results on an event probability of:  $\frac{1}{2^{15}} = 3.05e^{-5}$ . Mapping it into the Q-scale, results on  $Q(3.05e^{-5}) = 4.008$ . In conclusion to obtain accurate jitter values the provided jitter distribution function must generate  $Q(x)$  function with values bigger than 4.008.

## 2.4 extract\_jitter tool

### 2.4.1 Linearization

The linear approximation is the tool core, since everything will depend of it. The idea was to change the coefficient  $A$  limiting the minimum and maximum values, using the Newton's method to find the next  $A$  value that reduces the quadratic error between the obtained line and  $Q(x)$ , the error is only considered for  $Q(x) > 0$ . The Newton's method can have convergence issues, this was addressed by defining a maximum number of iterations, after that number the linear approximation stops and it's used the  $A$  value that returns the lower error. The algorithm also stops when the error is bellow than 0.001.

The number of interactions needed to find a coefficient that reduces the error of the linear approximation was reduced by using the second order derivation function (when possible).  $A_{i+1}$  can be defined as:

$$A_{i+1} = A_i - \frac{\frac{dQ(x)}{dx}}{\frac{d^2Q(x)}{dx^2}} \quad (2.5)$$

Proper linearization was also ensured by ensuring that the linearization error differences are only considered when  $Q(x)$  has values lower than zero, this ensures that  $Q(x)$  can be described as the convolution of two dual-Dirac functions and gaussian distribution.

### 2.4.2 Multi-thread

One of the major advantages of using a programming language to create this software tool, is the possibility to use threads and queues. Since the jitter extraction can be divided in two parts, the jitter extraction can run on each side independently. Based on this fact the software also divides the work in two threads, one calculates the jitter of the left side and the other calculates the right side.

Current processors have multi-threading capabilities, the introduction of multi-thread native support on `extract_jitter` tool allows the designers to obtain jitter decomposition values faster, making use of the capabilities of current processors.

### 2.4.3 How to use

Jitter extraction software can be found under: , the user need to have a software capable of opening rar files, like 7-Zip, to be able to start working. Next it's necessary to provide the jitter histogram. The `extract_jitter.rar` file also provides three examples that can be found under `examples` directory. The user can create their own file, to do that it's just necessary to follow this steps:

- Create a file called `<file_name>.csv`
- Inside `<file_name>.csv` separate the histogram hits by new lines
- Each histogram hit (line), should have the time and hits value separated by comma: time, hits
- When the histogram was completely described it's just necessary to save and close the file
- On the command line and inside `extract_jitter` directory perform the following command:  
`gen_hist.exe <file_name>.csv`
- The jitter tool will create a file named `<file_name>_Qscale.pdf` containing the jitter extraction values

In order to make the software more generic there is no indication of the total jitter value for the desired BER, but the user can obtain this value through:

$$TJ(BER) = Q_{rBER} * \sigma_r * we_r + |Q_{lBER} * \sigma_l| * we_l + |\mu_r - \mu_l| \quad (2.6)$$

Where:

$$Q_{rBER} = \sqrt{2} \operatorname{erf}^{-1} [1 - BER \cdot A_r] \quad (2.7)$$

$$Q_{lBER} = \sqrt{2} \operatorname{erf}^{-1} [1 - BER \cdot A_l]$$

### 2.4.4 Results

Tables 2.1, 2.2 and 2.3 summarize the obtained results using different types of jitter with different number of acquisitions. Jitter distribution functions were obtained trough the application of the math formulas with the application of the convolution when needed, resulting on accurate expected jitter values allowing an accurate comparison between the expected and obtained results.

Jitter Type	DJ real	RJ Real	TJ(1e-14)real	Tj(1e-12)real
Gaussian only	0	4.000	61.2	56.24
Dual Gaussian	0	4.472	68.42	62.88
Dual Dirac Gaussian	10	3.000	55.9	52.18
Periodic Gaussian	16	2.000	46.6	44.12
Square Gaussian	17	2.500	55.25	52.15

Table 2.1: Total Jitter Expected

Jitter Type	DJ Obtained	RJ Obtained	TJ(1e-14)Obtained	Error(%)
Gaussian only	0.514	4.174	65.88	17.14
Dual Gaussian	0.761	4.840	76.61	21.84
Dual Dirac Gaussian	9.379	3.098	58.3	11.73
Periodic Gaussian	10.963	2.344	48.13	9.09
Square Gaussian	10.926	2.876	56.53	8.4

Table 2.2: Total Jitter obtained for BER=1e-14

Jitter Type	DJ Obtained	RJ Obtained	TJ(1e-12)Obtained	Error(%)
Gaussian only	0.514	4.174	61.29	8.97
Dual Gaussian	0.761	4.840	70.79	12.58
Dual Dirac Gaussian	9.379	3.098	54.61	4.66
Periodic Gaussian	10.963	2.344	45.34	2.77
Square Gaussian	10.926	2.876	53.11	1.83

Table 2.3: Total Jitter obtained for BER=1e-12

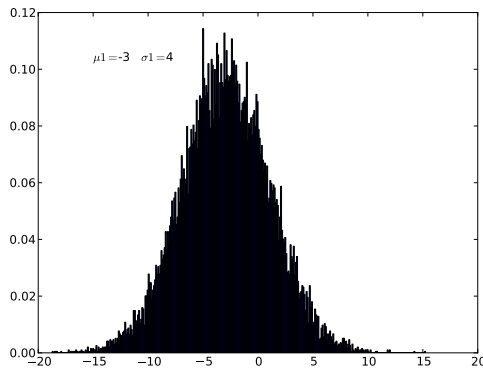
The jitter histograms with lower probability values tend to result on more accurate values after jitter decomposition, this fact is expected due to the intrinsic characteristics of the dual-Dirac method.

Figures 2.2, 2.3, 2.4, 2.5 and 2.6 contain the waveforms used to obtain the results of tables 2.1, 2.2 and 2.3.

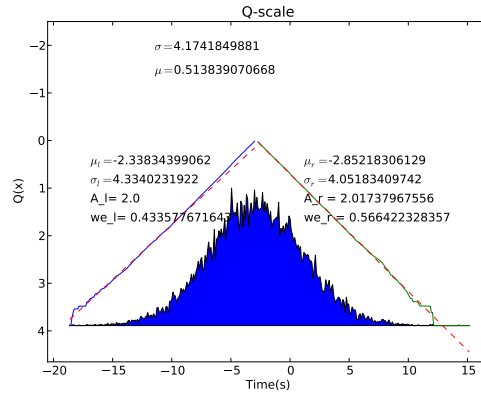
## 2.5 Conclusion

The `extract_jitter` tool is very easy to use and provides good results. It's always necessary to understand that  $DJ_{\delta\delta}$  is always lower than  $DJ_{pp}$ , but  $TJ_{\delta\delta}$  is higher than  $TJ_{pp}$  for low BER. This tool allows the designers to predict the IC behavior on an early stage or even to reduce the cost with a tool to extract jitter.

The accuracy of the results increases with the increase of the  $Q(x)$  maximum value, this is mainly due to the reason that for low probability values the influence of deterministic jitter is negligible, but at higher probability values, the Q-scale technique doesn't provide accurate results,

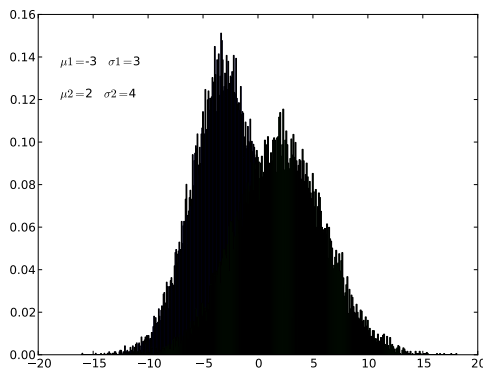


(a) Gaussian jitter distribution

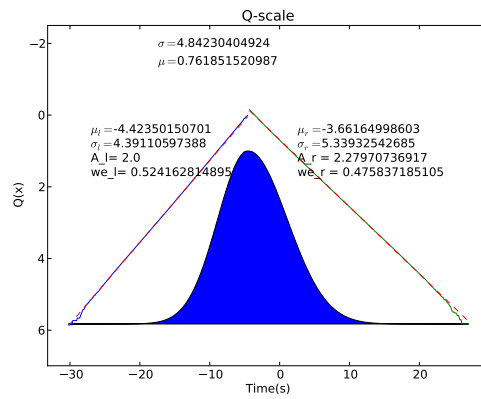


(b) Q-scale

Figure 2.2: Gaussian Jitter decomposition using extract\_jitter tool

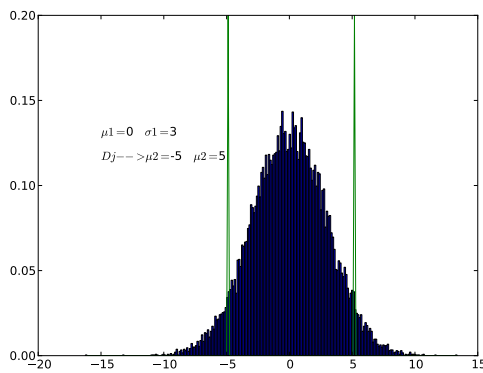


(a) Dual Gaussian jitter distribution

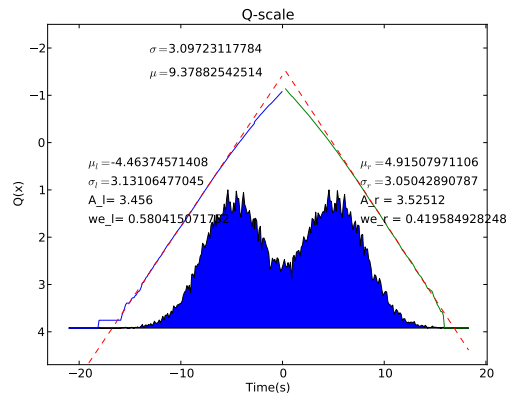


(b) Q-scale

Figure 2.3: Dual Gaussian Jitter decomposition using extract\_jitter tool



(a) Dual-Dirac Gaussian jitter distribution



(b) Q-scale

Figure 2.4: Dual-Dirac Gaussian Jitter decomposition using extract\_jitter tool

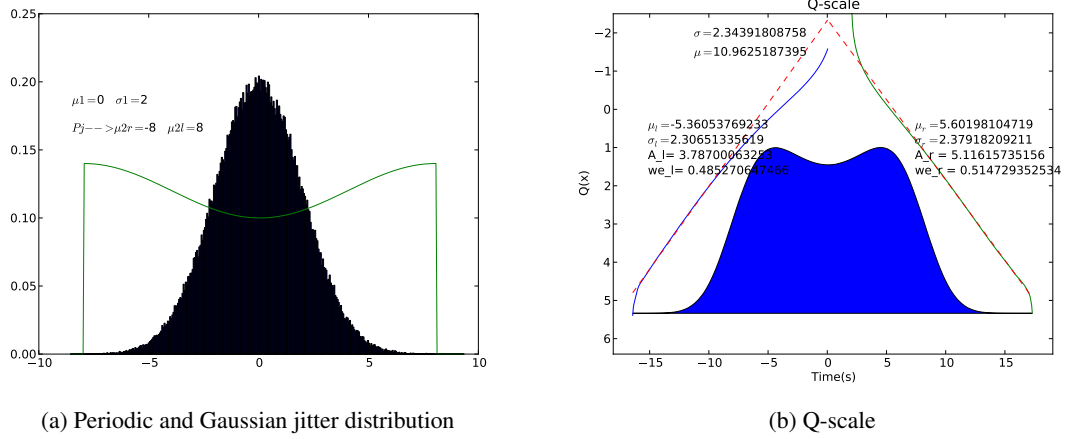


Figure 2.5: Periodic and Gaussian Jitter decomposition using extract\_jitter tool

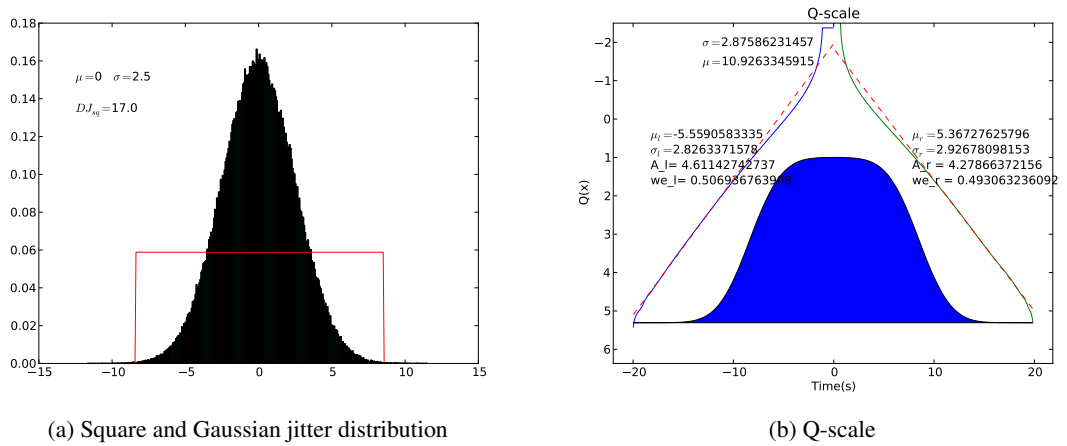


Figure 2.6: Square and Gaussian Jitter decomposition using extract\_jitter tool

even when the linearization is correctly performed, since deterministic jitter is mixed with gaussian jitter resulting on super overestimated random jitter values (by nature Q-scale overestimates random jitter).

The availability of this software on early stages of the design allows the designers to have an idea of the overall jitter on the system allowing them to take preventive actions increasing the robustness of the IC.

# References

- [1] Brian Guilfoos Judy Gardiner Stanley Ahalt Ashok Krishnamurthy Jose Unpingco Alan Chalker Andy Warnock Juan Carlos Chaves, John Nehrbass and OH Siddharth Samsi Ohio Supercomputer Center, Columbus. "a high-level scripting languages productivity and performance evaluation". *HPCMP Users Group Conference (HPCMP-UGC'06)*, 2006. [http://www.osc.edu/research/cse/projects/octave\\_python.pdf](http://www.osc.edu/research/cse/projects/octave_python.pdf).