Non-Linear Addressing Scheme for a Reconfigurable Look-Up Table

Elvio Dutra [1,2], Weiler Finamore [3], Leandro Indrusiak [4], Manfred Glesner [2]
Instituto de Estudos Avançados – Divisão de Geointeligência (http://www.ieav.cta.br/geointeligencia/) [1]
Technische Universität Darmstadt – Mikroelektronische Systema (http://www.mes.tu-darmstadt.de/team/mitarbeiter.en.jsp) [2]
Pontifícia Universidade Católica do Rio de Janeiro - Centro de Estudos em Telecomunicações (http://www.cetuc.puc-rio.br) [3]
The University of York - Department of Computer Science (http://www-users.cs.york.ac.uk) [4]

***ABSTRACT* — Look-Up Tables (LUT) are generic microelectronic blocks used in many different applications. They are very useful for interpolation: a desired function is sampled, these samples are stored, and the values in between are interpolated using Taylor's approximations. This paper presents a reconfigurable LUT with a non-linear sampling scheme used in the FPGA implementation of a Gaussian noise generator for developing and testing telecommunications Systems, such as the Link-BR2 protocol used for V/UHF data telecommunications between aircrafts. We sample a given transformation function *g(x)* in a non-linear pattern: whenever *g(x)* varies abruptly, more sampling points are stored on the LUT. The first advantage of this non-linear addressing scheme is to save memory size in the LUT. By changing internal LUT parameters, we can also apply the non-linear addressing scheme to other transformation functions. This leads to different kind of noise generators, maintaining the non-linear addressing scheme and the consequent advantage of memory saving.**

***Keywords* — Reconfigurable Look-Up Table, FPGA implementation, Non-linear addressing.**

I. INTRODUCTION

Look-Up Tables (LUT) are very common microelectronic blocks, used at a very broad range of applications. They can be utilized to speed up complex and slow calculations by means of storing pre-computed values on it, allowing one to achieve very high-speed designs [3]. They can also be used on flexible noise generators, in which one can change the probability density functions (pdf) of the generated noise just changing the content stored inside the LUT [2].

The main contribution of this work is to present a generic and versatile LUT block for microelectronic designs, using a non-linear addressing scheme. The proposed block can be used in a very wide range of applications and domains. It has the advantage of storing a desired function in a very memory saving scheme.

The proposed design also allows the reconfiguration of a given application on the fly, storing different functions, sampled on different non-linear schemes. This advantage can be very useful on projects that map more than one function, in a small memory space: only one LUT block can perform different functions, saving silicon area.

Élvio Carlos Dutra e Silva Júnior, elvio.dutra@ieav.cta.br, Phone +55 - 12 - 39475345, Fax +55 - 12 - 39441177; lsi@cs.york.ac.uk, Phone +44 - 1904 - 325570; glesner@mes.tu-darmstadt.de, Phone +49 - 6151 - 164537; weiler@cetuc.puc-rio.br, Phone +55 - 21 - 35271148.
This work was financed by the Brazilian Aeronautics Command (COMAER), through the Portaria R-126/GC1, and by the Technische Universität Darmstadt (TUD), through the Microelectronic Systems Institute.

The reconfigurable LUT proposed on this work was applied as the interpolation unit of a Gaussian noise generator used for test and design of telecommunication systems such as the Link-BR2 protocol, designed by EMBRAER (http://www.embraer.com/) under the coordination of CISCEA (http://www.ciscea.gov.br/) and technical supervision of IEAv (http://www.ieav.cta.br/). This protocol was designed to be the backbone of an airborne V/UHF data telecommunications system.

The remaining of the paper is organized as follows: Section II describes the LUT structure, the interpolation function domain and the approximation scheme used. Section III presents the external configuration table that configures the LUT parameters in order to correctly address and sample differently functions; this section also shows the transformation function *g(x)* used in the reconfigurable LUT of our study case: a Gaussian noise generator. Section IV selects which order of Taylor's approximation was used in the selected study case. Section V presents the FPGA implementation of the reconfigurable LUT using Xilinx System Generator, detailing the proposed non-linear addressing scheme designed to limit the maximum error produced by the LUT around the poles of the transformation function. This section also improves the efficiency by using half secant approximations. Section VI shows the flexibility of this design using the reconfigurable LUT in the interpolation of two different functions, using different sampling schemes. Section VII closes this article with a summary of the achieved results and a flavour of future works.

II. LUT STRUCTURE

The LUT designed on this article maps an input x to a desired output $g(x)$. The input x is a fixed point, signed, two's complement, 15 bits wide with binary point position equal to 14. Therefore x varies from -1.00000000000000 to +0.99993896484375. The LUT can be used in $g(x)$ functions with for differently wide its domains. For example, in the case of a domain (-C,+D), where D > C, we just have to scale the input from the interval (-D,+D) to (-1,+1), and neglect the values on the interval (-D,-C).

The LUT stores on RAM blocks two values pre-calculated: the ordinate $g(x)$ itself and its derivate $g'(x)$. For an input x, it interpolates $g(x)$ using the Taylor's approximation (1):

$$g(x) = \sum_{j=0}^{\infty} \frac{1}{j!} g^{j}(x_0)(x - x_0)^j \quad (1)$$

Since the values stored inside the RAM blocks are non-uniformly spaced, it is not possible to simply *slice* some most significant bits (MSB) of *x* for calculating the RAM addresses as usually done on LUT interpolation. In our case, it is necessary to determine which segment of the RAM is related to the input *x* according to the inferior and superior limits in table 2, select the MSB, subtract a constant value K given on table 2, and calculate the difference between the input *x* and the biggest value smaller than it stored inside the RAM. The block schematic of the proposed non-linear LUT subsystem is presented on Fig. 6 at the end of this text. The error introduced by the improved LUT is shown in Fig. 7, still bounded by the defined limit of 2^{-15} for all range of input values.

III. EXTERNAL CONFIGURATION TABLE

All configuration parameters of the LUT block; including the content of its memories, are previously calculated and imported to a FPGA. This design was implemented using System Generator, on a Xilinx Spartan-3 development kit from Avnet (XC3S2000-FG676 Spartan 3 FPGA).

The External Configuration Table has two inputs related to the sampling scheme of the desired function *g(x)*, as can be seen on Table 1: the Sampling Limit Point, which defines the boundaries of a sampling region, and the Resolution, related to the distance between sampled points (for example, for a resolution equal to 10, the distance between sampled points is equal to 2^{10}). Based on this inputs, we calculate the configuration parameters of the LUT, which are given on table 2 at the end of this text.

TABLE 1 – EXTERNAL CONFIGURATION TABLE (INPUT VALUES)

Region	Sampling Limit Point	Resolution
1	-1,00000000000000	15
2	-0,99770000000000	14
3	-0,99540000000000	13
4	-0,99030000000000	12
5	-0,98050000000000	11
6	-0,95900000000000	10
7	-0,91410000000000	9
8	-0,82040000000000	8
9	-0,64070000000000	7
10	-0,34380000000000	6
11	-0,12510000000000	5
12	0,00000000000000	5
13	0,12490000000000	6
14	0,34360000000000	7
15	0,64050000000000	8
16	0,82020000000000	9
17	0,91400000000000	10
18	0,95890000000000	11
19	0,98040000000000	12
20	0,99010000000000	13
21	0,99530000000000	14
22	0,99760000000000	15
23	0,99993896484375	15

In our study case, the non-linear LUT is used on a flexible noise generator to interpolate a transformation functions *g(x)* responsible for changing the *pdf* of a source uniformly distributed noise into a signal with a desired *pdf* (in our case, into a Gaussian distributed noise). By changing the transformation function of a noise generator, we can generate different distributed noise signals. The source uniformly distributed noise can be implemented on a wide gamma of ways [9, 10], both digital (e.g. pseudo-random number generators) and analog. The uniformly distributed noise generator used in our case studies was published on [3].

The implementation of a linear LUT to interpolate a desired transformation function *g(x)* is straightforward and easily found on literate [8, 11]. Usually the function *g(x)* and its derivates are linearly sampled and stored inside RAM blocks. In this case, the addressing scheme is constructed with the MSB (Most Significant Bits) of the input. Here, this scheme was modified in order to compensate the very non-linear behaviour of *g(x)* around its poles, and to maintain the approximation error bounded.

As mentioned, the source input signal *x* in our study case has a uniformly distributed *pdf* function [4], described by (2).

$$f_x(x)=\begin{cases}\frac{1}{2}, for -1\le x\le +1\\ 0, otherwise\end{cases} \qquad F_x(x)=\begin{cases}\frac{1}{2}x+\frac{1}{2}, for -1\le x\le +1\\ 0, otherwise\end{cases} \tag{2}$$

Our goal is to produce normally distributed noise, with mean value equal to zero and variance equal to σ_y. This means that the LUT's output should result on signals with a *pdf* function given by (3).

$$f_y(y)=\frac{1}{\sqrt{2\pi}\sigma_y}e^{\frac{-y^2}{2\sigma_y^2}} \qquad F_y(y)=\frac{1}{2}\left(1+erf\left(\frac{y}{\sqrt{2}\sigma_y}\right)\right) \tag{3}$$

To deduce the transformation function *g(x)*, we use equation 4.36 from [2] and the *pdf* equations above (4).

$$F_y(y)=\frac{1}{2}\left(1+erf\left(\frac{y}{\sqrt{2}\sigma_y}\right)\right)\Rightarrow erf\left(\frac{y}{\sqrt{2}\sigma_y}\right)=2F_y(y)-1\Rightarrow$$

$$\Rightarrow \frac{y}{\sqrt{2}\sigma_y}=erf^{-1}(2F_y(y)-1)\Rightarrow y=\sqrt{2}\sigma_y erf^{-1}(2F_y(y)-1) \tag{4}$$

Using the identity $x=F_y(y)\Leftrightarrow y=F_y^{-1}(x)$, it results (5):

$$y=\sqrt{2}\sigma_y erf^{-1}(2F_y(y)-1)\Rightarrow F_y^{-1}(x)=\sqrt{2}\sigma_y erf^{-1}(2x-1)\Rightarrow \tag{5}$$

$$\Rightarrow F_y^{-1}\left(\left\{\frac{1}{2}(x+1)\right\}\right)=\sqrt{2}\sigma_y erf^{-1}\left(2\left\{\frac{1}{2}(x+1)\right\}-1\right)\Rightarrow F_y^{-1}\left(\left\{\frac{1}{2}(x+1)\right\}\right)=\sqrt{2}\sigma_y erf^{-1}(x)$$

Comparing (5) with equation 4.36 from [2], give us the desired transformation function *g(x)* in (6):

$$g(x)=F_y^{-1}(F_x(x))=F_y^{-1}\left(\frac{1}{2}(x+1)\right)\Rightarrow g(x)=\sqrt{2}\sigma_y erf^{-1}(x) \tag{6}$$

As can be seen in Fig. 1, the transformation function *g(x)* has two poles located at x = -1 and x = +1. Both the

uniformly distributed input signal and the domain of *g(x)* go from -1 to +1. The range of this function ideally goes from -∞ to +∞, what is expected since the output is an unlimited normally distributed signal.

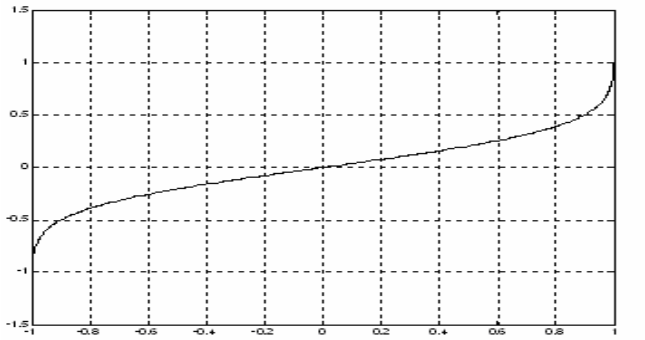

Fig. 1 - Transformation Function *g(x)*

IV. TAYLOR'S APPROXIMATION

The *g(x)* transformation function (6) deduced in the previous section has a very strong non-linear behaviour around its poles. Normal approaches using linear interpolation techniques would lead to very high approximation errors for inputs around those regions, or it would require very big RAM for storing many linearly spaced samples $g(x_0)$. Another approach would use a chain of multipliers in a higher Taylor's approximation order (7), but this solution would require a very big implementation area (a very long chain of multipliers and very long tables for storing $g(x_0)$ and many order of its derivates) in order to get low error approximations, although such approach would introduce an unreasonable latency that would impact the overall system performance.

$$g(x) = \sum_{j=0}^{\infty} \frac{1}{j!} g^{j}(x_0)(x - x_0)^j \qquad (7)$$

We face a trade-off between the interpolation error produced by the LUT and the memory size required to construct it. In addiction, every time we increase the order of the Taylor's approximation we need an extra multiplier, which increases not only the complexity of the LUT but also its latency. So, as we increase the order of the Taylor's Approximation, we have to deal with three disadvantages: (1) larger memory required to store one more derivate order of the transformation function; (2) increased arithmetic resource usage and (3) increased latency due to the cascading of one more multiplier. In the other hand, it has the advantage of decreasing the error by approximating the transformation function *g(x)* by a curve, instead of a segmented line as it occurs in zero and first order Taylor's Approximations.

Fig. 2 shows the transformation function (a curve ploted wit a continuous line), its zero order Taylor's approximation (marked with circles), its first order (marked with x), second order (market with crosses), third order (marked with stars), fourth order (marked with squares), and fifth order (marked with diamonds). Fig. 3 uses the same identification tags to show the error obtained by each approximation order.

After analyzing the data on Table 3, we concluded that the first order Taylor's approximation scheme is the best compromise between hardware costs and approximation error. We obtain the biggest marginal improvement by increasing from zero to first order Taylor's approximation: the average error decreases 11 times. This approximation order has also the lower hardware cost: only one multiplier and one extra LUT for the first derivate are needed.

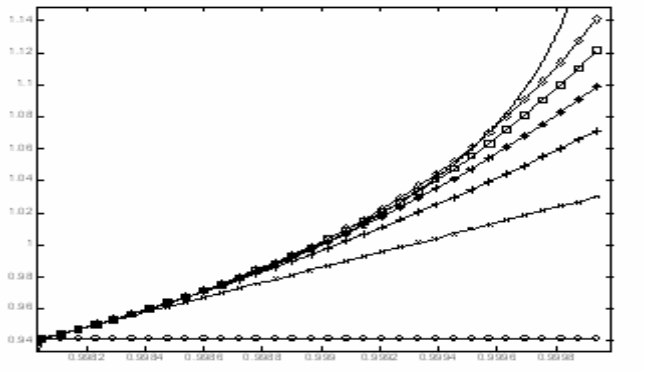

Fig. 2 - Zero to Fifth Order Taylor's Approximation

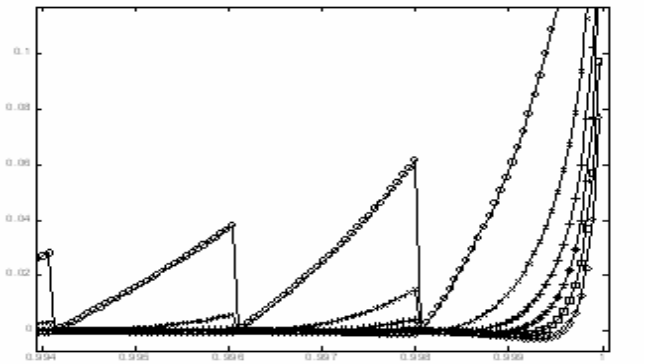

Fig. 3 - Zero to Fifth Order Taylor's Approximation Error

TABLE 3 - ZERO TO FIFTH ORDER TAYLOR'S APPROXIMATION ERROR

Order	Peak Error	Average Error
Zero	0.27705274	1.0282369221 x 10-3
First	0.18849472	8.6443005413 x 10-5
Second	0.14720625	4.0277456890 x 10-5
Third	0.11928469	2.3411098682 x 10-5
Fourth	0.09700337	1.3598830895 x 10-5
Fifth	0.07722615	6.5296863738 x 10-6

V. LUT-BASED DESIGN

Taking into account the trade-off described in the previous section, a LUT-based implementation of a transformation function using first order Taylor's approximation scheme was realized. The LUT was designed using Xilinx System Generator [5], which is a library of parametrical IP cores integrated into Matlab/Simulink, allowing for system level design entry and simulation. Fig.

20 at the end of this text shows the block schematic implementation within System Generator. Its input is a uniformly distributed noise in the range from -1 to +1 - 2^{-15}, due to the 15 bit input size.

Three main data paths can be seen in the schematic of Fig. 20: The data path of memory block *RAM1,* which stores the values of the transformation function $g(x_0)$; the *RAM2,* which stores the values of the first derivate $g'(x_0)$; and the third data path which uses the *Slice2* block to calculate the difference between the x_0 values present at the input of the LUT and the corresponding biggest $g(x_0)$ value smaller than it stored inside *RAM1*.

Memory Size Optimization

The depth of RAM blocks on the LUT is defined according to the maximum allowed interpolation error: the larger the memory size, the smaller the interval between adjacent sampling points and smaller the interpolation error. Fig. 4 plots the error for each possible input value using different memory sizes. The horizontal line represents a boundary: it is desirable that the error values stay below that limit for all possible inputs. This limit is equal to 2^{-15} = 3.0518 x 10^{-5}. The input values are 15 bits long and any error lower that 2^{-15} does not decrease the quality of the LUT. Note that the error reaches its maximum for inputs near the poles.

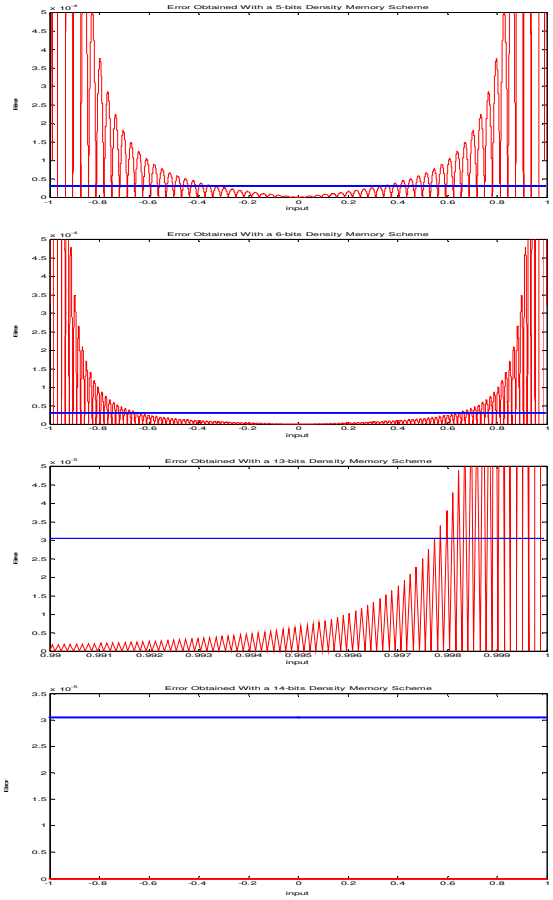

Fig. 4 - Error using 2^5, 2^6, 2^{13} and 2^{14} memory positions

In order to maintain the error below 2^{-15}, we need to increase the memory size, which means increasing the density of sample points between 0 and +1. Fig. 4 shows the error obtained in a structure with 2^5, 2^6, 2^{13} and 2^{14} memory positions. As the size of the memory increases, the error decreases and stays below the boundary limit for a larger share of input values. On the third graph there is a zoom in the x-axis, showing only values from 0.99 to 1, while the other graphs show them for the entire domain (-1, +1).

The maximum error limit is respected only by using 2^{14} memory positions (bottom graph on Fig. 4). In this case we reach an error equal to zero for all possible inputs. In this extreme case there is no linear interpolation, but a one-to-one mapping of all possible input values. In such case there is no need for a memory to store the first derivate g'(x), neither a multiplier nor an adder are necessary for the interpolation.

At first glance, it seems that the memory size of the LUT was found: a RAM memory with 2^{14} = 16384 positions eliminates completely any possible error, but this size is too large for implementation on a FPGA. The proposed solution uses a non-uniformly spaced memory scheme. It uses less memory positions for input values around zero (where the error is smaller), and more memory positions for points around the pole (where the error is bigger), saving memory space. This approach is described in table 2 and graphically presented on Fig. 5. This graph depicts how the density of sampling points increases as the input reaches the pole region, while the table 4 details how the data stored inside RAM1 was defined.

TABLE 4 - NON-UNIFORMLY SPACED MEMORY SCHEME.

Size	Start	End	Inferior	Superior	MSB	K
16	0	1	0.0000000	0.1249389	4	0
32	4	10	0.1250000	0.3436889	5	2
64	22	40	0.3437500	0.6405639	6	13
128	82	104	0.6406250	0.8202514	7	54
256	210	233	0.8203125	0.9140014	8	159
512	468	490	0.9140625	0.9589233	9	393
1024	982	1003	0.9589843	0.9804077	10	884
2048	2008	2027	0.9804687	0.9901733	11	1888
4096	4056	4076	0.9902343	0.9953002	12	3916
8192	8154	8172	0.9953613	0.9976196	13	7993
16384	16346	16383	0.9976806	1.0000000	14	16166

Size: number of memory positions; Start: lower memory position used; End: higher memory position used; Inferior and Superior: Identifies the limit range of the input for which the error stays below the limit of 2^{-15}; MSB: Number of most significant bits taken from the input signal to construct the addresses, excepting the signal bit; K: Constant added to MSB to construct the addresses

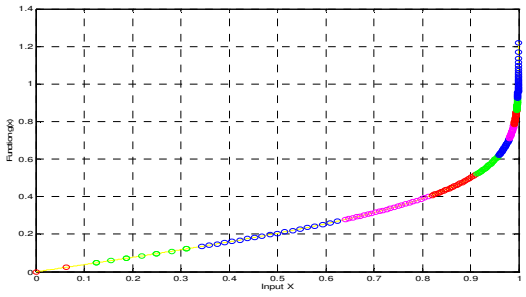

Fig. 5 - Function g(x) sampled according the to Table 2

The RAM blocks store the 213 memory positions defined

by Table 2. Since the values stored inside RAM1 and RAM2 are not uniformly spaced anymore, it is not possible to use the MSB to address its contents. For calculating the addresses, first it is necessary to determine which segment includes the input value (according to inferior and superior limits in Table 2), select the MSB, and subtract a constant value K (6^{th} and 7^{th} columns of table 2). The third path calculates the difference between the value *x* presented at the input of the LUT, and the corresponding biggest $g(x_0)$ value smaller than it stored inside RAM1 by taking the less significant bits of the input. The block schematic of the improved LUT subsystem is presented on Fig. 6 (at the end of this work). The interpolation error obtained with the improved LUT is shown in Fig. 6, bounded by the threshold limit (2^{-15}) for all input values.

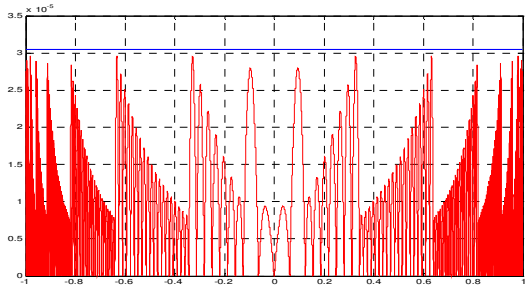

Fig. 6 - Error obtained with non-linear addressing scheme

Secant Cross Point Choice

The LUT scheme presented in the subsection above approximates the curve $g(x)$ by a sequence of straight line segments, which are secants to the curve at their extremities. Due to the fact that the sampling points are not uniformly spaced, it allows the minimization of the total amount of memory used for a given error limit. One further improvement in the organization of the memory blocks can be achieved by changing the cross point position of these line segments. This implies in using the same *RAM1* memory content $g(x_0)$, but decreasing the $g'(x_0)$ values stored at *RAM2*.

Fig. 7 shows six different cases where the error is plotted for different types of secant (from top to bottom: 100%, 50%, 90%, 80%, 85% and 82.5% cross point positions). The first case (100%) shows the error produced by a full secant scheme with the cross points at the extremities of each segment, like the one presented on the previous subsection. In this case (100%), the error is equal to zero at the border of each secant, and it reaches a maximum just in the middle. The second graph presents a secant that crosses the curve $g(x)$ in the middle (50%) of the distance between two sampling points. In this case, the error is equal to zero at the beginning of each secant and in the point that corresponds to 50% of the distance to the next sampling point. The third graph shows the error in the case that the secant crosses $g(x)$ at 90% of the distance between the sampling points; here the error becomes bigger before the 90% crossing point. Changing the crossing points to 80%, 85% and 82.5%, the difference between the errors before and after the crossing point decreases, reaching equilibrium when the crossing point is placed at 82.5%.

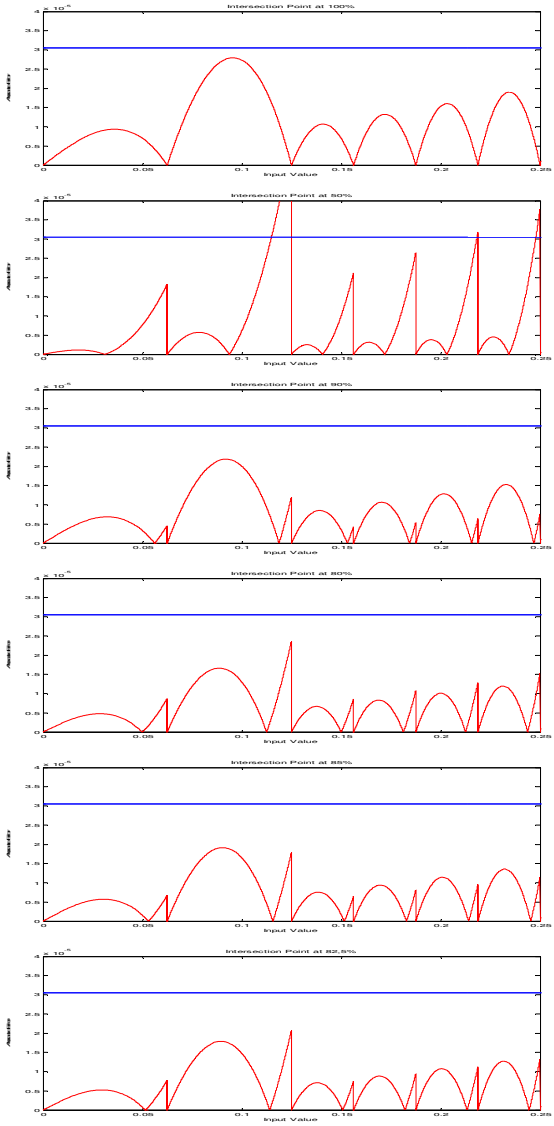

Fig. 7 - Secant and Half-Secant Approximation Scheme

Fig. 8 shows the error for all possible LUT input values with an 82.5% half-secant. When comparing this graph with Fig. 6, it is possible to see that the error is decreased approximately by a factor of two. Note that this improvement is achieved without changing the size of the RAM, only the contents of *RAM2* are changed.

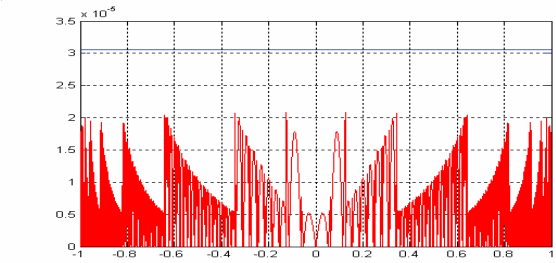

Fig. 8 - Error for non-linear addressing and half-secant approximation

The updated LUT subsystem was validated using two different generators for the uniformly distributed noise, one based on a chaotic system [3] and the other on a random number generator (RNG). Fig. 9 shows respectively, from top to bottom, the *pdf* of the uniformly distributed noise generated by the chaotic system [1], its normally distributed noise presented at the output of the LUT structure [1], the *pdf* of the uniformly distributed noise generated the RNG and its corresponding normally distributed noise.

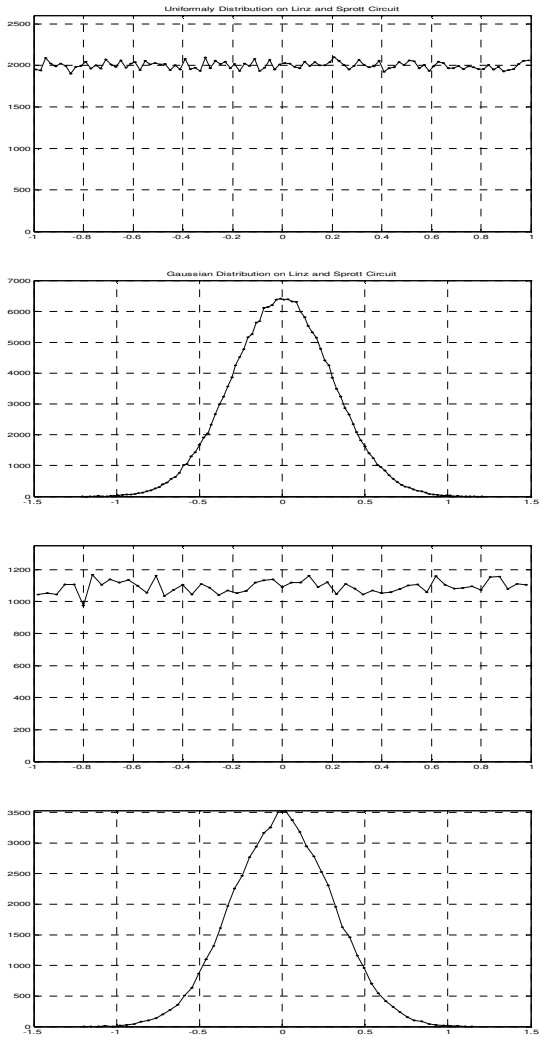

Fig. 9 - LUT-based transformation of uniformly distributed into normally distributed noise

VI. LUT RECONFIGURATION

The sections above presented the study case of a reconfigurable interpolation LUT with non-linear addressing scheme for the generation of a Gaussian noise. In this case, the LUT was configured by table 2 (External Configuration Table) which mapped the transformation function presented on (6).

We can change the input values of table 1 and reconfigure the LUT in order to sample (6) according different non-linear addressing schemes. It can be also reconfigurable to map other transformation functions $g(x)$ using different non-linear addressing schemes that suits better each function. This flexibility can be used, for example, in the design of noise generators that produces outputs with probability distribution functions different from the Gaussian noise explained on the last subsection. This reconfiguration could be archived on the fly, with no extra hardware cost.

To illustrate this advantage, we use the proposed reconfigurable LUT in the interpolation of two different equations: the cubic function $g(x) = x^3$ (fig 10) and the exponential function $g(x) = e^x$ (Fig. 11). These equations were selected as mathematical examples, and they have no correlation with the noise generation application explained on last section.

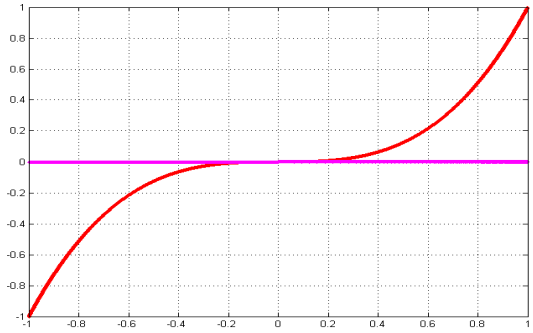

Fig. 10 – Cubic Function

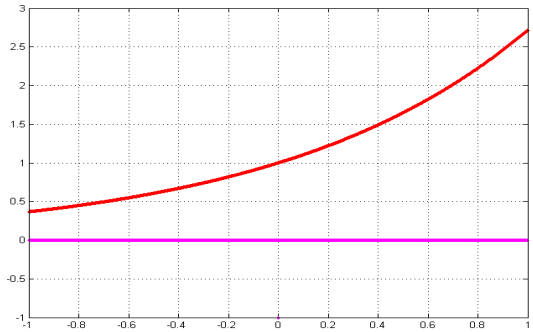

Fig. 11 – Exponential Function

Each function was sampled according to 2 different non-linear patterns. For them both, we calculate new values for table 2, based on new sampling intervals and resolutions of table 1. Note that these intervals can be symmetric to the origin, or not.

Fig. 12 shows the error obtained after applying a first order Taylor's approximation to the cubic function, and Fig. 13 shows the error for the exponential function. The upper graphs of Fig. 12 and 13 use the same sampling intervals used on the study case presented on section V (compare to the Fig. 6). But the lower graphs of these 2 figures use sampling intervals different from the one used on section V, and also different one from another. On Fig. 12, the smaller resolution occurs in the interval (-0.6,+0.3), and in the interval (-0.5,+0.0) on the Fig. 13.

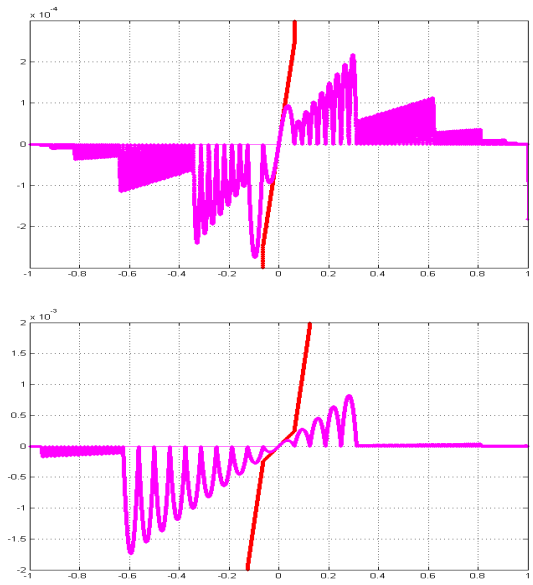

Fig. 12 – Approximation Error on Cubic Function

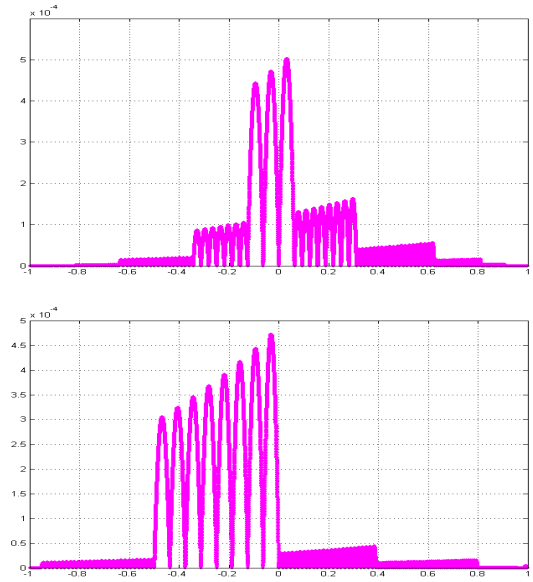

Fig. 13 – Approximation Error on Exponential Function

VII. Conclusion

This paper identified a flexible approach to the design of noise generators. It relies on transformation functions that allow the generation of noise signals with different probability density functions using as source a uniformly distributed noise. A special case of transformation function *g(x)* was presented, allowing the generation if Gaussian noise. The mathematical background used to derive this particular function was also presented. This mathematical procedure can be used to derive transformation functions that are able to generate noise with different probability density functions, such as Weibull or Lognormal. The transformation functions are implemented on Look-Up Tables within reconfigurable logic devices, so the reconfiguration of the noise generator involves mainly the replacement of the contents of the Look-Up Tables.

This paper also explored specific features of transformation functions with poles using techniques for non-linear LUT addressing and half-secant approximation of the transformation function, achieving insignificant interpolation error with moderate size of memory. The final system was implemented in a Xilinx XCV800 device, using Xilinx ISE 6.2i, as seen on Table 3

TABLE 5 – XILINX ISE 6.2I SYNTHESIS INFORMATION

Property	Value
Device Part Type	XCV 800
Package Type	HQ 240
External GCLKIOBs	1 out of 4, or 25 %
External IOBs	49 out of 166, or 29%
BLOCKRAMs	2 out of 28, or 7%
Slices	371 out of 9408, or 3%
GCLKs	1 out of 4, or 25%
Average Connection Delay	2.016 ns
Maximum Pin Delay	7.267 ns
Clock Period Constrain	100 ns
Maximum Frequency	90.228 MHz
Power Consumption	18 mW
Junction Temperature	25 C
Typical Package Resistance	0.208 ohms
Typical Package Inductance	13.2 nH
Typical Package Capacitance	2.25 pF

The non-linear addressing scheme presented here requires 213 memory positions and bounds the error to a minimum value. Just to compare, if a linear addressing scheme with 256 positions were used, 3.27 % of all possible LUT output values would stay beyond the limit boundary of 2^{-15}, and 50 outputs would produce an unacceptably high absolute error grater than 0.01.

VIII. References

[1] Dutra, Elvio. Analysis, Design and FPGA-Implementation of Chaotic Systems as Alternative for Gaussian Noise Generation. Darmstadt, Alemanha, 2004.

[2] Dutra, Élvio; Indrusiak, Leandro; Glesner, Manfred. Non-Linear Addressing Scheme for a Lookup-Based Transformation Function in a Reconfigurable Noise Generator. In 18th Symposium on Integrated Circuits and Systems Design (SBCCI), ACM, 2005.

[3] McLoone, Máire; McCanny, John. Rijndael FPGA Implementation Utilizing Look-Up Tables. In Journal of VLSI Signal Processing 34, 261 – 275, Kluwer Academic Publishers, 2003.

[4] Indrusiak, Leandro; Dutra, Élvio; Glesner, Manfred. Advantages of the Linz-Sprott Weak Nonlinearity on the FPGA Implementation of Chaotic Systems: a Comparative Analysis. In: International Symposium on Signals, Circuits and Systems (ISSCS), Iasi, 2005.

[5] Papoulis, Athanasios: Probability, Random Variables and Stochastic Processes. McGraw-Hill, 1991.

[6] Xilinx Inc.: System Generator for DSP version 3.1 – Quick

[7] Start Guide, Introductory Tutorials and Reference Guid

[8] Gribbon, KT and Johnston, CT and Bailey, DG. A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation. Image and Vision Computing Journal, pages 408 - 413.

[9] Lee, D.U. and Luk, W. and Villasenor, J.D. and Cheung, P.Y.K. A Gaussian noise generator for hardware-based simulations. IEEE Transactions on Computers, volume 53, number 12, pages 1523 – 1534, 2004.

[10] Addabbo, T. and Alioto, M. and Fort, A. and Rocchi, S. and Vignoli, V. Uniform-distributed noise generator based on a chaotic circuit. Instrumentation and Measurement Technology Conference, 2006. IMTC 2006. Proceedings of the IEEE.

[11] Lehmann, T.M. and Gonner, C. and Spitzer, K. Survey: Interpolation methods in medical image processing. IEEE Transactions on Medical Imaging, volume 18, number 11, pages 1049 – 1075. 1999.

TABLE 2 – EXTERNAL CONFIGURATION TABLE (OUTPUT VALUES)

Inf. Lim. Precise	Sup. Lim. Corrected	B	D	S	Z	S	Desloc.	AddLog	R0si	#	R0sf	#
-1,00000000000000	-0,99774169921875	-14	15	9	15	7	16384	0	-1,00000000000000	1	-0,99768066406250	39
-0,99768066406250	-0,99542236328125	-13	14	9	14	1	8192	19	-0,99755859375000	40	-0,99536132812500	58
-0,99536132812500	-0,99029541015625	-12	13	9	13	2	4096	38	-0,99511718750000	59	-0,99023437500000	79
-0,99023437500000	-0,98052978515625	-11	12	9	12	3	2048	58	-0,98974609375000	80	-0,98046875000000	99
-0,98046875000000	-0,95904541015625	-10	11	9	11	4	1024	78	-0,97949218750000	100	-0,95898437500000	121
-0,95898437500000	-0,91412353515625	-9	10	9	10	5	512	99	-0,95703125000000	122	-0,91406250000000	144
-0,91406250000000	-0,82037353515625	-8	9	9	9	6	256	121	-0,91015625000000	145	-0,82031250000000	168
-0,82031250000000	-0,64068603515625	-7	8	9	8	7	128	144	-0,81250000000000	169	-0,64062500000000	191
-0,64062500000000	-0,34381103515625	-6	7	9	7	8	64	167	-0,62500000000000	192	-0,34375000000000	210
-0,34375000000000	-0,12506103515625	-5	6	9	6	9	32	188	-0,31250000000000	211	-0,12500000000000	217
-0,12500000000000	-0,00006103515625	-4	5	9	5	10	16	202	-0,06250000000000	218	0,00000000000000	219
0,00000000000000	0,06243896484375	-4	5	9	5	10	16	202	0,06250000000000	220	0,06250000000000	220
0,06250000000000	0,31243896484375	-5	6	9	6	9	32	185	0,09375000000000	221	0,31250000000000	228
0,31250000000000	0,62493896484375	-6	7	9	7	8	64	143	0,32812500000000	229	0,62500000000000	248
0,62500000000000	0,81243896484375	-7	8	9	8	7	128	39	0,63281250000000	249	0,81250000000000	272
0,81250000000000	0,91009521484375	-8	9	9	9	6	256	-193	0,81640625000000	273	0,91015625000000	297
0,91015625000000	0,95697021484375	-9	10	9	10	5	512	-682	0,91210937500000	298	0,95703125000000	321
0,95703125000000	0,97943115234375	-10	11	9	11	4	1024	-1684	0,95800781250000	322	0,97949218750000	344
0,97949218750000	0,98968505859375	-11	12	9	12	3	2048	-3711	0,97998046875000	345	0,98974609375000	365
0,98974609375000	0,99505615234375	-12	13	9	13	2	4096	-7786	0,98999023437500	366	0,99511718750000	387
0,99511718750000	0,99749755859375	-13	14	9	14	1	8192	-15958	0,99523925781250	388	0,99755859375000	407
0,99755859375000	0,99987792968750	-14	15	9	15	7	16384	-32322	0,99761962890625	408	0,99987792968750	445

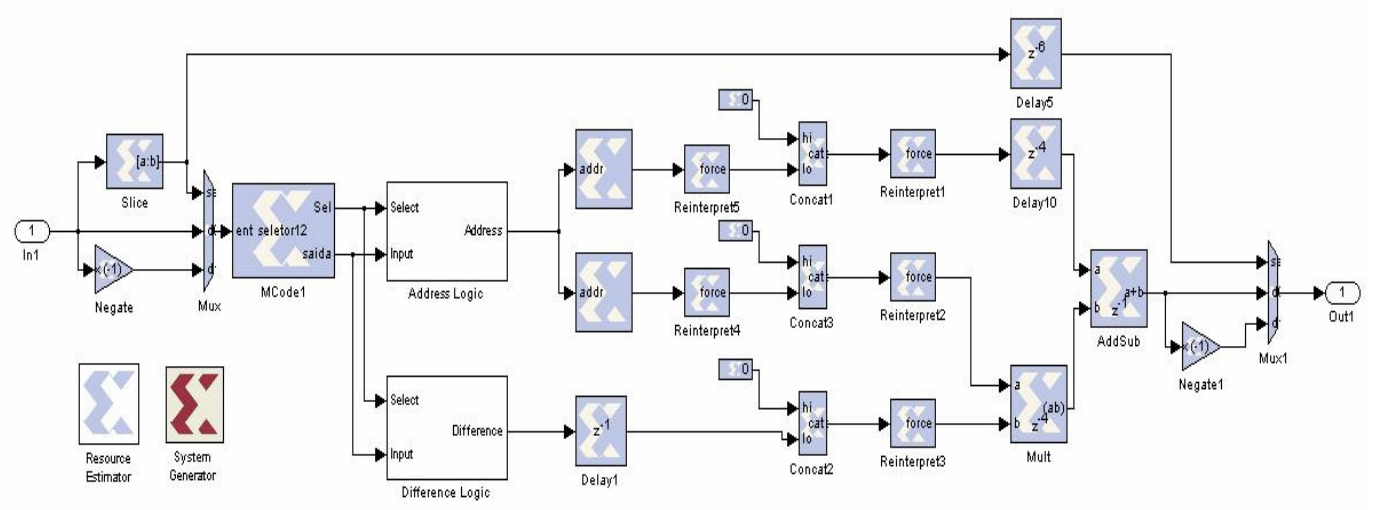

Fig. 20 - LUT schematic with non-linear addressing scheme