EasyChair Preprint

№ 598

# Fixed-point design methodology of the hyperbolic tangent function using lookup tables with performance evaluation on FPGA

Santiago Tomás Pérez Suárez



October 29, 2018

# Fixed-point design methodology of the hyperbolic tangent function using lookup tables with performance evaluation on FPGA

Santiago Tomás Pérez Suárez
Signals and Communications Department
University of Las Palmas de Gran Canaria (ULPGC)
Las Palmas de Gran Canaria, Spain
santiago.perez@ulpgc.es

## Abstract

The sigmoid functions, including the hyperbolic tangent, are widely used in artificial neural networks. Usually, artificial neural networks are implemented on a computer, in training and testing phases, using floating point arithmetic. In some situations it is necessary to improve the performance of neural networks, in such cases the implementations are designed for specific digital devices, using fixed point arithmetic. This strategy increases the speed, and reduces the hardware resources and consumed power. The sigmoid functions are non-linear systems because include exponentials and divisions operations, so they are the bottleneck of the artificial neuron design and cannot be implemented directly in fixed-point format. For this reason, several methods of approximation are used, mainly based on piecewise approximations. One of the most common technique is to use Look-up Tables, which store samples of the function. This method needs a lot of hardware resources, but it reaches the highest speed. The first objective, using lookup table approximations, is to explore the set of solutions and study the effect of the number of bits. This can only be achieved if an advanced design methodology is used for experimentation, a second objective is to expose this fast and flexible design method. Thirdly, the measure of functionality associated with the error is reviewed. A new functionality measure is proposed associated with the concept of signal to noise power relation. The new measure allows compare different approximations; besides, some linearities are observed against the number of bits. Afterwards, the implementations are developed on some Field Programmable Gate Arrays and the performances are evaluated: hardware resources, speed and consumed power. Again, some linear behaviours are observed. Finally, a quality factor is proposed, which includes functionality and physical performances, this factor allows comparison between different implementations.

**Keywords:** floating-point, fixed-point, design methodology, digital circuit, sigmoid function, hyperbolic tangent, lookup table, FPGA, signal to noise relation, Matlab, HDL

## 1. Introduction

The sigmoid [1-20], particularly the hyperbolic tangent [10, 21-26], are widely used as transfer functions in Artificial Neural Networks (ANN) [27-31]. Although, sometimes it is possible to use fully or piecewise linear functions [32], which facilitates hardware implementation. The artificial neural networks are usually implemented on a computer or microprocessor system in floating point arithmetic, for the training and testing phases. This type of physical supports is enough when the requirements of speed, power and volume are reached. But sometimes these systems are not fast enough, in such cases the ANN implementation must be developed with others technologies for reach real-time operation. It is also necessary to change the ANN support if the level of power consumed or the necessary volume are exceeded.

It is possible to use analog [10, 26, 33-37] or digital devices [38, 39]. Analog devices present behaviour deviations against variations in temperature, power supply and load impedance; besides, they present drifts with the aging. Digital devices do not present such abrupt changes with those parameters [40].

Floating point arithmetic can be used on digital devices, but these operations need a high number of clock cycles [41], great consumption of power and hardware resources [42-45]. However, floating point numerical representation has a wide range with a good resolution [46, 47]. It is possible to use fixed point versus floating point arithmetic, justified by its better performances. Fixed point increases the operating speed, the latency can be reduced to one clock cycle. The hardware resources and consumed power in fixed point are smaller than floating point. The drawback in fixed point is that the designer must care the range of the signals with the necessary resolution. This can be solved by signal studies with appropriate mathematical resources and strategies [48, 49]. It should be emphasized that in fixed point the data format is totally arbitrary, which improves the performances. On the other hand, the use of floating point requires the election of a standard [50], increasing the number of bits and decreasing physical performances. Using a non-standardized floating point representation complicates the design process.

For these reasons, this study is based on fixed point, allowing to the sigmoid functions be included in ANN for high performance operation. One possibility is to use an Application Specific Integrated Circuit (ASIC) [19, 51, 52]. The ASIC designs must be sent to a factory for manufacturing the device; have little occupied area, low power consumption and high speed. As inconveniences they present high price, difficult debugging and verification, great time to market, not allow reprogramming and great cost of non-recurring engineering.

On the other hand, can be used Digital Signal Processors (DSP) [39], which are cheaper than ASIC devices. The DSPs achieve higher clock rates than ASICs. Nevertheless, in DSPs the data rate is limited because the parallelism of the data and its format are restricted; and the pipeline is fixed. A Graphics Processing Unit (GPU), is a coprocessor specialized in image processing, can also be used [53, 54]. The GPUs have higher data rate than DSPs, they have multitude of arithmetic and logic units.

Finally, Field Programmable Gate Arrays (FPGA) can be used [55-57], which are formed by logic circuit blocks, connection lines, switch arrays, and input-output pins. Logic circuit blocks concentrate the computing capacity of the FPGA, and have different names according to the manufacturer. In fact, the internal architecture of the logical blocks depends on the manufacturer and the FPGA family devices. The great advantage of FPGAs versus ASICs is that they are programmable by the designer, without having to be sent to a factory. Others FPGAs advantages are that they have low cost of non-recurring engineering, minimum development time, ease debug and verification, shorter time to market, high data parallelism, flexible data format, and flexible pipelined [30, 58, 59]. Although, clock frequencies in FPGAs are not as high as in DSPs, with the above features FPGAs have greater data rate than DSPs. For few units, the price of FPGAs is lower than ASIC elements but higher than DSPs. Power consumption in FPGAs is higher than ASICs but lower than DSPs. For the previous reasons, FPGAs are suitable for prototype development whit high data rate. Anyway, the designs for FPGAs can be transferred to ASIC. For the reasons stated in this paragraph the modelling is presented for FPGA devices.

Besides, many companies offer printed circuit boards with FPGAs and auxiliary devices. These boards contain: analog to digital and digital to analog converters, memories, audio and video

devices, communication ports, etc. These boards avoid the tedious task of its design and manufacture; and most importantly, allow the creation of complete systems.

## 2. Sigmoid functions

Transfer functions used in ANNs are usually continuous and derivable, the derivability is a desirable requirement for ANN training algorithms. Normally, two sigmoid functions are used, which satisfy the conditions of continuity and derivability.

One of them is unipolar, simply called sigmoid, or *logsig* in Matlab notation. Its mathematical expression is given by equation 1, and its representation is in figure 1. The output of this function is restricted to the interval (0,+1).

$$y(x) = logsig(x) = \frac{1}{1+e^{-x}} \qquad (1)$$

The second function is bipolar and coincides with the hyperbolic tangent, called *tansig* in Matlab notation. Its mathematical expression is given by equation 2, other expressions with fewer exponents are possible; its representation is in figure 1. The output of this function is restricted to the interval (-1, +1).

$$y(x) = tansig(x) = \frac{e^{+x}-e^{-x}}{e^{+x}+e^{-x}} \qquad (2)$$

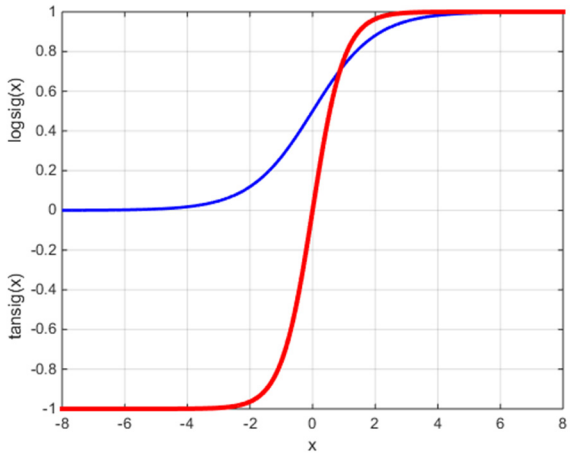


Figure 1. The unipolar and bipolar sigmoid functions.

These functions can have weighting factors in exponents, which changes the growth rate; although, these are the expressions that Matlab uses in ANNs, in training and testing, when they are selected. It can be easily demonstrated that equations 3 and 4 are satisfied, which shows that they are analogous functions. The approximation of one function can be obtained from the other approximated function. Moreover, in fixed-point arithmetic multiplying or dividing by two consists on shifting the bits in the most or least significant bit direction; besides, only the addition or subtraction operation is necessary.

$$tansig(x) = 2logsig(2x) - 1 \qquad (3)$$

$$logsig(x) = \frac{1}{2}\left[tansig\left(\frac{x}{2}\right) + 1\right] \quad (4)$$

## 3. State of the art of hardware implementation for sigmoidal functions

It should be emphasized that both functions are non-linear, because they include division and exponential operations, so they are the bottleneck of the artificial neuron design. The rest of the artificial neuron is formed by multiplications and sums. That is, the implementation of sigmoid functions is not immediate in fixed-point arithmetic and is usually approximated with some method.

One approach is using Look-up Tables (LUTs), which store samples of the function; this method needs a lot of hardware resources but gets a high speed [11, 20, 60-66]. The LUTs can be implemented with memory elements, or using logical resources. The two solutions can have different benefits. In the first case, the design optimization depends on the type of memories available in the device; in the second case, it is possible to develop logical simplifications. Other approximation form is the Piece Wise Lineal (PWL), which approaches each section with a straight line, in this case a multiplication and a sum are necessary [18, 19, 21, 38, 44, 45, 63, 66, 68, 69, 70].

It is also possible to approximate each section with polynomials, usually of grade two [70] or cubic [71], which increases the number of multipliers. Other authors use piecewise Taylor approximations [72, 73]. In all previous cases, it is possible to reduce the error by increasing the number of sections. Other approaches propose specific shape functions that consider the characteristics of the sigmoid functions; firstly where its derivative tends to a constant, and secondly its symmetry characteristics [4, 11, 21, 26]. Other authors use functions that look like sigmoid functions [9, 13, 74, 75], in such cases the error is bounded.

The comparison of these solutions is based on the functionality, which is measured with the error. There are several types of errors, and sometimes implementations are compared with different error values and error types [2, 47, 76, 77]. Once the functionality is fixed, there are three parameters to compare [78], which are the physical performances: the hardware resources, the consumed power and the speed of the system.

It should be noted that the effect of the number of bits in the representation is chosen discreetly or arbitrarily by authors [11, 19, 20, 60, 61, 66, 67, 69, 70, 71, 75, 79, 80, 81], usually based on the experience or previous works. At most, a scan is performed for a discrete set of bit numbers, without performing a systematic study of the effect of the number of bits on the functionality. A small number of bits can cause degradation of system functionality. An excessive number of bits may not improve functionality, but increase area and consumed power, and decrease speed.

Usually, authors extract area and speed, but almost never the consumed power [11, 19, 20, 45, 60-64, 66, 67, 70, 71, 75, 79, 80-83]; with the exception of [69]. Power is the most complicated feature to evaluate, it must be estimated with specific tools and a set of parameters must be fixed by the designer. Consumed power becomes important with the use of portable personal devices and the autonomy of the batteries [84, 85]. Many authors study the speed as the latency of the system, number of clock cycles required, but without estimating the maximum frequency [70, 71].

## 4. Objectives

The objective of this research is to fix a methodology for sigmoidal functions designed on digital programmable devices, in particular, the hyperbolic tangent. The approximation type will be based on LUTs implemented with logical elements.

There are many design methods [86], this development focuses on Simulink [87] of Matlab [88] using fixed-point arithmetic. This design flow is fast and flexible, allowing to check different architectures and the effect of the binary format in different points of the system; this makes possible to scan the number of bits in a systematic and extensive form.

A suitable parameter will be chosen to measure the functionality, which obviously will be associated with the representation error. This allows identifying different systems with similar functionality.

Once the systems have been chosen, which reach the functionality with the smaller sizes of fixed-point representation, from Simulink can be generated the project in a standard Hardware Description Language (HDL). One of them is Very High Speed Integrated Circuit Hardware Description Language (VHDL) [89, 90], and the other is Verilog [91, 92]. The generated project is formed by the digital implementation and files with input and output signals, to perform the necessary simulations. Besides, the physical performances of area, speed and power will be extracted for the chosen device. The speed will take into account the latency of the system, and also the maximum frequency.

It is also possible to design in Simulink for the two main FPGA providers, which are Altera [93] and Xilinx [94]; with their own tools, which are respectively DSP Builder [95] and System Generator [96]. In the design tools of these manufacturers it is possible to extract the performances for the chosen device [97, 98].

To show this design method the manufacturer Altera has been chosen, and the project has been generated in Verilog language. In this work it is assumed that ANN training is performed outside the device, this is called offline type [99], so it is not considered the approximation of the first derivative of the function. For example, for offline training can be used the Neural Network Toolbox [100] from Matlab.

## 5. The measure of functionality

Therefore, there are four parameters that can be evaluated in digital implementations: the functionality, the area, the power and the system speed. The intention is to analyse approximations with equal or similar functionality; afterwards, the three remaining parameters are contrasted. The first question is how to measure functionality, clearly associated with the error, but with different versions.

Let the function $y=f(x)$ be the one to be approximated, and let $ya=fa(x)$ be the expression of the approximation; the error is defined as $E(x)=ya-y=fa(x)-f(x)$, which generally has null mean value. The absolute error is defined as $E_{abs}(x)=|E(x)|$.

For comparing designs, the maximum value of the absolute error $|E(x)|_{max}$ can be used, as in [19, 62, 63, 69, 70, 71, 79]. Although, this is an important measure, a good approximation can have a high value error only in a small interval; anyway, it is important to take it in account.

Other measure is the mean value of the absolute error [19, 62, 63, 69, 70, 75, 79]; which measures an approximation on an interval, but high error values can be masked. Therefore, both measures can be combined, as in [16]. Occasionally, the square root of the mean value of the square error has been used [63, 75].

Previous measures are valid for comparing approximations of the same function; obviously, they are not valid for comparing approximations of different functions.

The relative error, defined as $E_{rel}(x)=(ya-y)/y=(fa(x)-f(x))/f(x)$ can also be used, no reference of it has been found. The relative error can be used for comparing same type functions, or different type, for the same input range. The relative error grows enormously when the value to be approximated tends to zero; moreover, it is not defined if the function value is zero and the approximation value is not zero. The absolute peak value, or average absolute value, of relative error can be considered, with the same previous observations. The relative error allows compare different function types, as it is shown in figure 1, because the numerator introduces the error and the denominator introduces the value of the function; besides, it can be expressed in per centum values.

To avoid uncertainty when the signal is zero, the error can be measured against the peak-to-peak function value (equation 5); as in [101-104], where it was used in per centum values. This value could be used to compare approximations of different functions.

$$E_{pp}(x) = \frac{E(x)}{y_{max} - y_{min}} \qquad (5)$$

Different authors use different error types, which makes comparison difficult. A relative measure is proposed, which is the quotient between function and error, which can also be called signal and noise, respectively. The sampled signals, input $x$ and output $y$, before quantizing and coding, are discrete time analog values; and also the generated noise signal. The signal to noise relation power, called Signal to Noise Ratio (SNR) [105], is defined in expression 6.

$$SNR = \frac{\sum_{i=0}^{k} y[i]^2}{\sum_{i=0}^{k} E[i]^2} \qquad (6)$$

The numerator is the energy of $(k+1)$ samples of the signal to be approximated, the denominator is the energy of $(k+1)$ samples of the corresponding error. Therefore, the equation 6 is the ratio of the energies of signal and error. The signals can be voltages or currents, and the load resistance can be 1 ohm; or $R$ ohms, which disappears in the quotient. The samples are produced every $\Delta t$ seconds, and the full interval is $T=(k+1)\Delta t$ seconds. The equation 6 coincides with the relation between the signal power and the noise power. This concept is common in analog and digital communications [106]. It should be emphasized that it is a relative measure between the function to be approximated and the generated noise. Often, this relation is expressed in decibels (equation 7) [105]; which differentiates small and close values, and makes possible to operate with smaller numbers for great values of SNR.

$$SNR(dB) = 10log_{10}(SNR) \qquad (7)$$

## 6. The model and its parameters

The Simulink block diagram for the approximation is shown in figure 2, where the arithmetic used is two's complement. The *1-D Lookup Table* block is the element that stores the samples for the approximation. In figure 2 the LUT stores 16 words, so its address bus has 4 bits; Simulink type is denoted as *ufix4*, 4 bits unsigned fixed-point. The words in the LUT have 1 sign bit, 1 bit for the integer part and 7 fractional bits, (*sfix9_En7*, 9 bits signed fixed-point 7 bits fractional); so in the output the values ±1 are representable, and the function is saturable. Besides, the *1-D Lookup Table* block does not allow using a sign bit without any bit for the integer part. The input format has 1 sign bit, 2 bits for the integer part and 6 fractional bits, (*sfix9_En6*, 9 bits signed fixed-point 6 bits fractional). Thus, the input values represented tend to [-4, +4] when the number of fractional bits grows. In fact, the represented input range is [-4, +4-$2^{-6}$]. The multiplier and adder in figure 2, together with constants *G* and *C*, perform a conversion of the input *x* to the LUT address bus. In this case, the conversion is from [-3.75, +3.75] to [0, +15]. The constants *G* and *C* can be represented without error under certain conditions, this will be explained below.

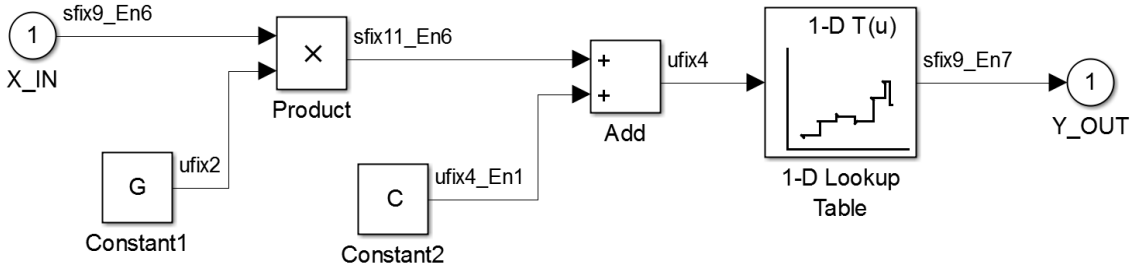


Figure 2. The Simulink approximation model for 16 words LUT.

Figure 3 shows the hyperbolic tangent function for [-4, +4] input range; also, the approximation and the error are shown. It should be noted that the LUT samples are evenly spaced in the input range. The SNR measured is 23.36 dB, according to expression 7.

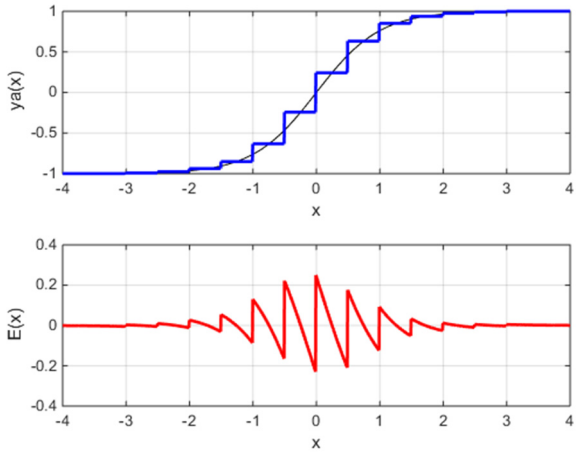


Figure 3. The hyperbolic tangent function, 16 words LUT output and error.

Once the model has been established, a set of parameters can be discussed: the format of the input signal, the number of words stored in the LUT, and the format of the words stored in the LUT. The number of words stored in the LUT is a power of 2; so this takes advantage of the address bus. Then, *M* words will be addressed by *n* address bits, such that $M=2^n$. The output format has 1 bit sign and 1 bit for integer part, as was explained previously. But the number of bits of the fractional part (*nbfo*) can be varied.

In the input the size of fractional bits (*nbfi*) can be varied, which affects the resolution. The question is how many bits to use for the input integer part (*nbii*). When the number of bits of the input fractional part is large the input range tends to [$-2^{nbii}$, $+2^{nbii}$); in fact, the range will be [$-2^{nbii}$, $+2^{nbii}$-$2^{nbfi}$]. Then, with *nbii* equal to 1 the representation range is [-2, +2-$2^{nbfi}$], with *nbii* equal to 2 the range is [-4, +4-$2^{nbfi}$], with *nbii* equal to 3 the range is [-8, +8-$2^{nbfi}$], etc. It is proposed to represent the function in an interval centred in the origin, and saturate the output to $\pm 1$ values outside that interval, since the function has two horizontal asymptotes. For this purpose, the SNR is measured when a sawtooth signal is introduced in the range [-16, +16]. Figure 4 shows the saturation error outside the range [-4, + 4], which is the input range with 2 bits for the integer part when the number of fractional bits grows indefinitely. That is, the saturation approximation is given by expression 8. The SNR obtained is 81.25 dB, which is a high value; the signal power is noise power multiplied by $1.33 \cdot 10^{+08}$. This is the maximum SNR for the input range [-16, +16] when the hyperbolic tangent is approximated in the interval [-4, +4].

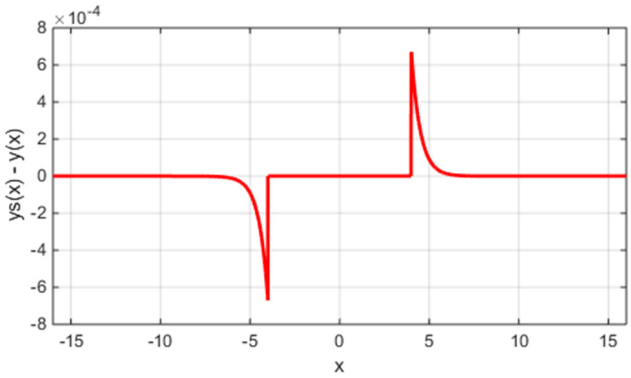


Figure 4. The saturation error outside the range [-4, + 4].

$$ys(x) = \begin{cases} -1 & x < -4 \\ tansig(x) & -4 \leq x \leq +4 \\ +1 & +4 < x \end{cases} \quad (8)$$

With the saturation out of the range [-2, +2], with 1 integer bit, the SNR for input range [-16, +16] is 46.61 dB; a high value but easily surmountable. The saturation out of the range [-8, +8], for 3 integer bits, the SNR for input range [-16, +16] is 150.73 dB, extremely high and unnecessary value. Besides, using 3 bits for the input integer part increases the bit size and the range to be approximated, disperses the samples, and input values reached are too much high for most scenarios. Therefore, the number of integer bits in the input is set to 2, the approximation will be performed within the range [-4, +4], and the circuit will saturate outside that range (equation 9). For measure the SNR of the approximation, the input signal will be a sawtooth signal in the range [-4, +4]. This coincides with the assumption that all values in the input for that range are equally likely.

$$ysa(x) = \begin{cases} -1 & x < -4 \\ ya(x) & -4 \leq x \leq +4 \\ +1 & +4 < x \end{cases} \quad (9)$$

Finally, the parameters to be varied will be: the number of fractional bits of the input signal (*nbfi*), the number of bits of the LUT address bus (*n*), and the number of fractional bits for stored words in the LUT (*nbfo*).

It is convenient to revise the final format of the input signal: 1 sign bit, 2 bits for the integer part and *nbfi* fractional bits. According to Simulink notation this is a *sfix(1+2+nbfi)_En(nbfi)* type. Similarly, the output signal has 1 sign bit, 1 integer bit and *nbfo* fractional bits; according to Simulink notation this is a *sfix(1+1+nbfo)_En(nbfo)*.

The conversion of the input signal to the LUT address is the expression 10; obviously, *A* (address) is an integer between 0 and $(2^n-1)$.

$$A = nearest\ enteger\ (Gx + C) \qquad (10)$$

The constants values *G* and *C* are given by expressions 11 and 12; where $x_{min}$ is -4, $x_{max}$ is +4 and *M* is $2^n$, where *n* is the number bits of LUT address. Since the extremes of *x* are entire powers of 2, and *M* is other integer power of 2, the constants can be expressed as powers of 2 and have an exact representation in fixed-point format.

$$G = \frac{M}{x_{max} - x_{min}} = 2^{(n-3)} \qquad (11)$$

$$C = (M-1)/2 = 2^{(n-1)} - 2^{(-1)} \qquad (12)$$

## 6.1 Scan of model parameters

For *n* equal to 4 (*M=16*) the number of fractional bits in input and output (*nbfi*, *nbfo*) was varied between 0 and 24. The SNR was measured in dBs, table 1 shows the results until 16 bits. The maximum SNR obtained is 23.36 dB, and the system gets this value with 6 input fractional bits and 7 output fractional bits. The maximum value of SNR could have been calculated, with signals of figure 3, by integration in time continuous domain. It should be emphasized that for 16 words LUT, were simulated 625 configurations. The fixed point output signal is stored in Matlab variables space in floating point format; afterwards, the SNR is measured in dBs. These simulations and measurements are very difficult to realize using a HDL.

| | | *nbfo* | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| *nbfi* | **0** | 13,75 | 14,01 | 15,33 | 14,62 | 14,65 | 14,85 | 14,91 | 14,93 | 14,93 | 14,93 | 14,93 | 14,93 | 14,93 | 14,93 | 14,93 | 14,93 | 14,93 |
| | **1** | 12,33 | 14,65 | 16,28 | 17,11 | 17,28 | 17,38 | 17,35 | 17,37 | 17,37 | 17,37 | 17,37 | 17,37 | 17,37 | 17,37 | 17,37 | 17,37 | 17,37 |
| | **2** | 13,35 | 16,04 | 18,91 | 20,46 | 20,79 | 20,94 | 20,91 | 20,94 | 20,94 | 20,94 | 20,94 | 20,94 | 20,94 | 20,94 | 20,94 | 20,94 | 20,94 |
| | **3** | 13,64 | 16,51 | 19,92 | 21,98 | 22,44 | 22,64 | 22,61 | 22,65 | 22,65 | 22,65 | 22,65 | 22,65 | 22,65 | 22,65 | 22,65 | 22,65 | 22,65 |
| | **4** | 13,73 | 16,59 | 20,23 | 22,44 | 22,94 | 23,15 | 23,13 | 23,17 | 23,17 | 23,17 | 23,17 | 23,17 | 23,17 | 23,17 | 23,17 | 23,17 | 23,17 |
| | **5** | 13,75 | 16,62 | 20,3 | 22,56 | 23,07 | 23,29 | 23,27 | 23,31 | 23,31 | 23,31 | 23,31 | 23,31 | 23,31 | 23,31 | 23,31 | 23,31 | 23,31 |
| | **6** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **7** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **8** | 13,75 | 16,63 | 20,33 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **9** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **10** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **11** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **12** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **13** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **14** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **15** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |
| | **16** | 13,75 | 16,63 | 20,32 | 22,60 | 23,11 | 23,33 | 23,31 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 | 23,36 |

Table 1. The SNR in dBs against input and output fractional bits.

Figure 5 shows the shape of SNR in dBs for 16 words LUT against input and output fractional bits. In other words, in order to reach the maximum SNR with the 16 words LUT, at least 6 input fractional bits and 7 output fractional bits are required, which is marked in figure 5. Increasing fractional bit numbers above these values do not increase the SNR.

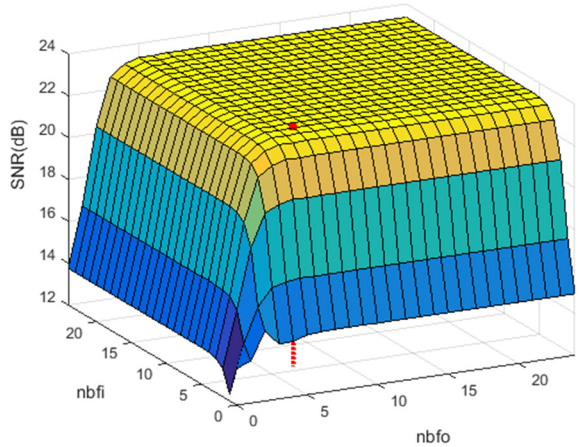


Figure 5. The shape of SNR in dBs for 16 words LUT.

Figure 6 shows the previous behaviour in two dimensions. For each value of input fractional bits (*nbfi*), the *SNR* in dB is plotted against the output fractional bits (*nbfo*). A saturation zone is observed, for a number of output fractional bits (*nbfo*) greater than 7; the *SNR* is constant for each value of *nbfi*. For *nbfo* less than 7 a transition zone is observed; which is linear for small values of *nbfo*; for large values of *nbfi* the slope is 3 dB per bit. For *nbfi* greater or equal than 6 the SNR values are similar. Analogous results are observed if the *SNR* in dB versus *nbfi* is represented for each value of *nbfo*.

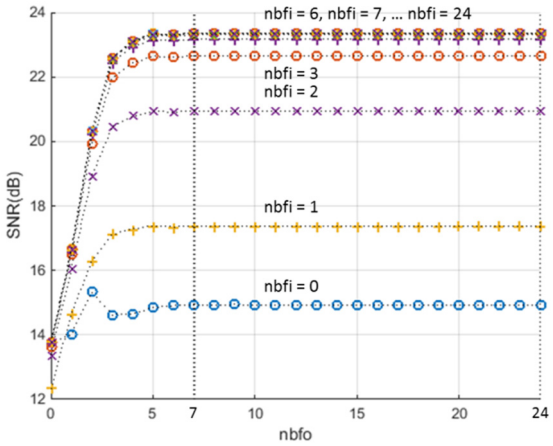


Figure 6. The two dimensions shape of SNR in dBs for 16 words LUT.

For the implementation the values 6 and 7 are chosen for *nbfi* and *nbfo* because the maximum SNR is obtained with the minimum number of bits, this system is shown in figure 2. From Simulink the Verilog project was generated for the Altera device EP2AGX260FF35I5 of Arria II GX family [107]. The project was compiled with Quartus II [108] and simulated with ModelSim-Altera Edition [109] and the Simulation Waveform Editor included in Quartus II. The schematic circuit is in figure 7; which shows the input and output of the function, the clock, the reset, and input and output enable signals. In short, only one FPGA implementation was generated, the most convenient case of the 625 Simulink simulations.

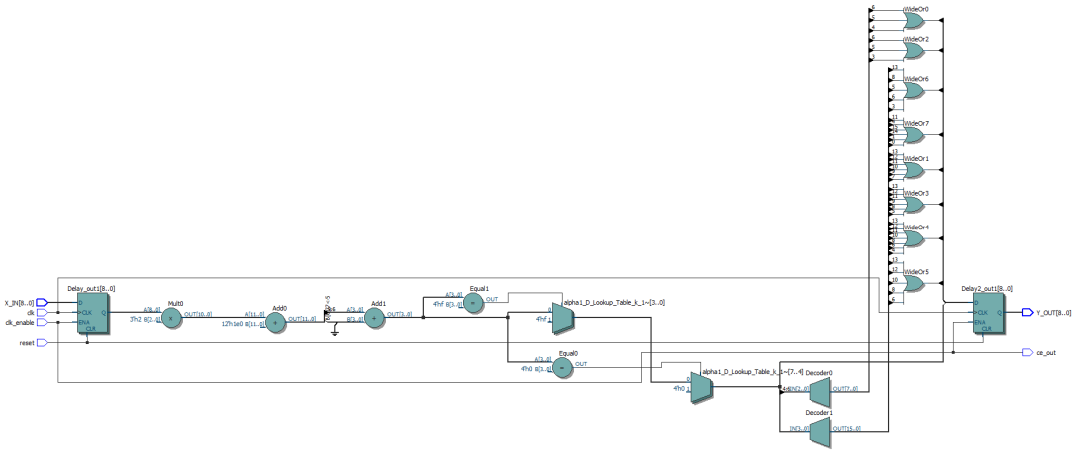

Figure 7. The schematic circuit of 16 words LUT.

The ModelSim simulation is shown in figure 8 for 25 MHz clock frequency. Only the input and output signals are shown in this figure; avoiding auxiliaries signals for simplification. The simulation input rate is $25 \cdot 10^{+6}$ values per second. The input and output registers, shown in Figure 7, set the latency of the system in 2 clock cycles. These registers, which are not shown in figure 2 for simplification, are necessary in Simulink for generating the clock signal in the project. The output Y_OUT of figure 8 is equal to the output signal in Simulink. This can be ensured because when the project is generated from Simulink, the input and output test signals (*testbench*) are also generated. The fixed point input signal is used in circuit simulation with ModelSim. On the other hand, the output signal of this simulator is compared with the Simulink fixed-point output signal, at the end the message "test completed passed" indicates that output signals have the same values. In this case, 100 points were generated for each interval on the x-axis, a total of 1600 discrete time values. The Verilog description circuit occupies 203 lines and the testbench 3540 lines, including blank and comments lines, this is difficult to generate using manual implementation.

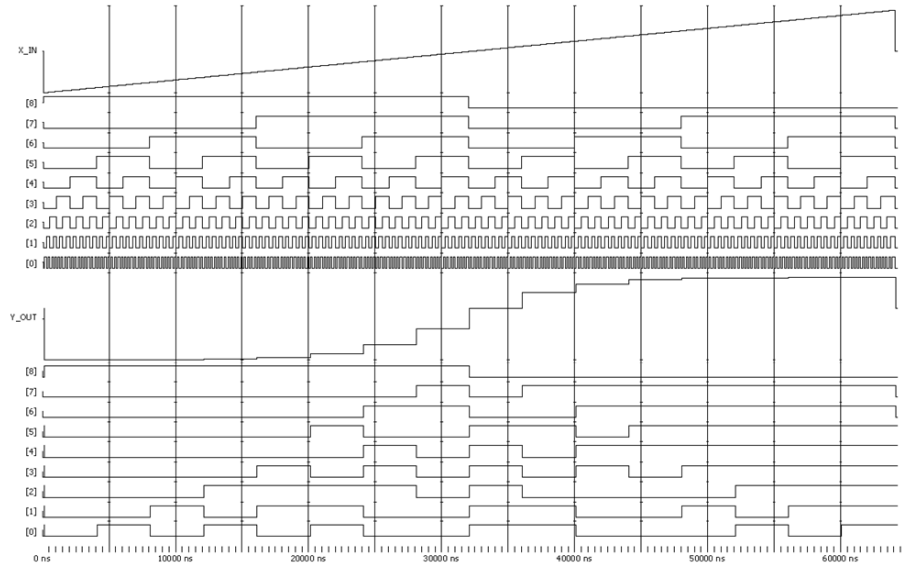

Figure 8. The circuit simulation with ModelSim of 16 words LUT.

In Quartus II hardware resources were estimated, for the previous device the area was evaluated as the number of Combinational Adaptive Lookup Tables (ALUTs); for this implementation were needed 8 ALUTs. With the TimeQuest Timing Analyzer of Quartus II it was found that the maximum operating frequency of the system was 909.09 MHz; for the worst case, which among other things, takes into account the temperature at which the device operates. Finally, with the PowerPlay Power Analyzer of the Quartus II, for a clock frequency of 25 MHz, a dynamic power of 0.80 mW and static power of 858.86 mW was obtained, that gives a total of 859.66 mW. Initially, only the dynamic power is interesting, which depends on the frequency; and not so much the static power, which depends on the continuous power supply. But the static power also increases with temperature; an increase of dynamic power, produces an increase of temperature, so this causes an increase of the static power. For this reason, the sum of the two powers was evaluated.

The power estimation was done with a fixed clock frequency, in order to compare the consumed power of different designs for the same data rate. This clock frequency must be less than the smallest of maximum frequencies of the compared designs. On the other hand, all power estimations were made for an ambient temperature of 25°C, without heatsink and no forced air flow. The power estimation was performed after loading the Value Change Dump File, which sets the form of change of the input signals. This is an input file for the power analyzer, which stores the change rates and static probabilities of signals. The Value Change Dump File was generated with the Simulation Waveform Editor, where the clock frequency was set to 25 MHz and the binary input signals were randomly varied.

## 7. Design flow

Figure 9 shows the design flow process. The fixed point Simulink model is shown in Figure 2, where the input and output registers are not shown. Between the floating point input signal and the fixed point model, a data type converter block exists and is not shown for simplification in figure 2. Similarly, a data type converter block exists at the output of the Simulink model. These converter blocks are not implemented in hardware, the input and output of the approximate function are in fixed point format. The Simulink model can be loaded, from Matlab, with the number of LUT address bits (*n*), the number of input fractional bits (*nbfi*) and the number of output fractional bits (*nbfo*).

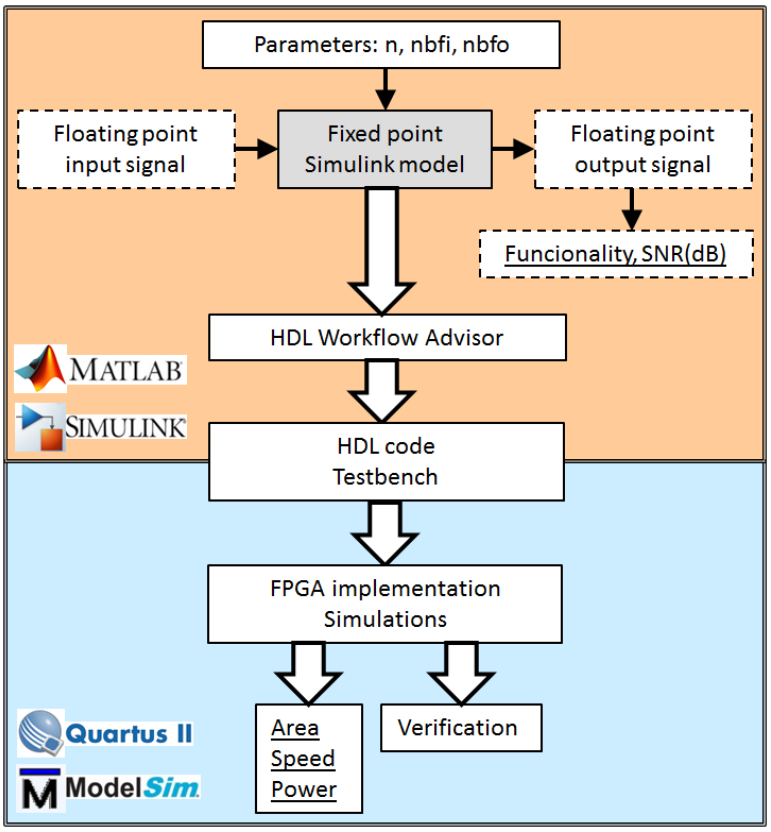


Figure 9. The design flow process from floating point to FPGA implementation.

The SNR is calculated with the exact floating point value of the hyperbolic tangent and the floating point values of the fixed point output signal. For the model with the desired functionality, and using HDL Workflow Advisor [110], it is generated the HDL project and the testbench. Not all Simulink blocks are supported by the HDL Workflow Advisor. In particular, only blocks of the HDL Coder library are implementable. But not all configurations of these blocks permit implementation, although they are simulable. Generating these files manually can be an impossible task, but this automatic generation can be performed in minutes. It should be noted that in HDL Workflow Advisor a synthesis tool is chosen, which has been installed previously in the computer; in this case Quartus II of Altera. Besides, the type of FPGA is chosen, and for the testbench allows set the clock, reset and enable signals. Once the HDL Workflow Advisor has completed the files generation, it is possible to implement the FPGA and obtain the hardware resources, the maximum frequency and the consumed power. With the Simulation Waveform Editor can be performed functional and timing simulations; also, testbench simulations can be performed with ModelSim. These last simulations take the input signal that was used in Simulink. On the other hand, ModelSim simulation compares its output with the Simulink output signal, which verifies the circuit operation. The implementation and simulations must not be dissociated, some simulations are necessary to set parameters for the power estimator. It would be possible to use Xilinx implementation software [98] since Matlab integrates Altera and Xilinx tools. It should be noted that in Quartus II it is possible to change the FPGA device, which avoids a new generation with HDL Workflow Advisor. Different designs are characterized by performances, underlined in figure 9: the functionality, measured as the SNR in dB; the area, hardware resources occupied in the FPGA; the speed, measured as the maximum frequency operation; and the consumed power for a certain operating frequency.

## 8. Experiments and results

In the previous section has been set: the model, the parameters, the design methodology and the measurement of performances. This section presents the results when the design parameters are varied. For this purpose the number of bits in LUT address bus (*n*) was varied until 16, the LUT reaches 64 kilowords. For each *n* value, the number of input and output fractional bits, *nbfi* and *nbfo* respectively, were varied from 0 to 24. A similar study to the previous section was done, the results of SNR were stored in matrices of 25 by 25 elements, similar behaviours were obtained to figure 5. For each *n* value, only the case of maximum SNR with the minimum numbers of bits was implemented, and its performances were evaluated. Figure 10.a shows the 16 responses obtained. Obviously, if the number of words in the LUT increases, the maximum SNR grows, but it is necessary to increase the number of input and output fractional bits; this tendency is observed in figure 10.b. In Simulink the number of simulations for generating figure 10.a was 10,000 (16x25x25), which were performed by running a loop for each *n* value.

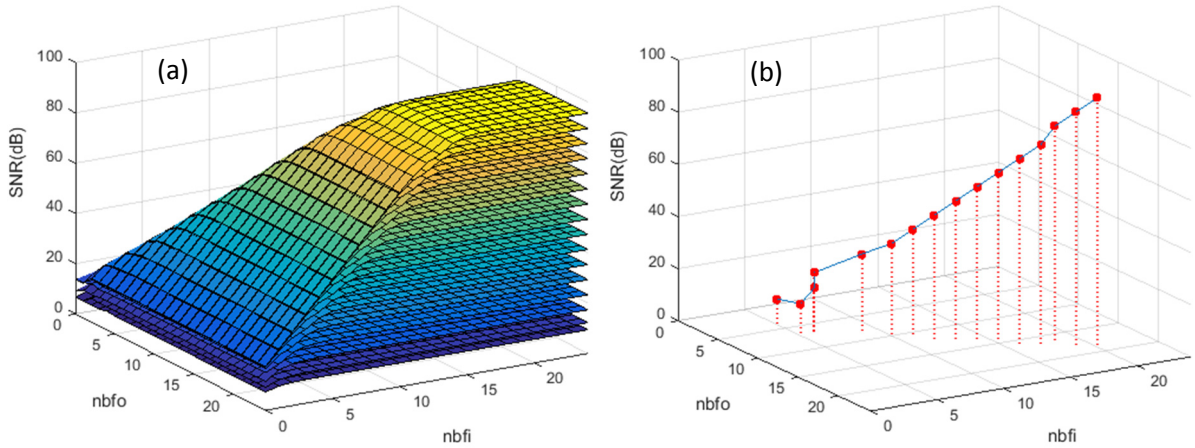


Figure 10. The SNR in dBs versus input and output fractional bits (a) and maximum SNR in dBs (b), for each number of bits in LUT address bus.

If it is desired to reach a SNR value, which coincides with a horizontal zone of figure 10.a for a *n* value, this *n* value and the smallest values of *nbfi* and *nbfo* would be taken. If it does not coincide and the horizontal plane of constant SNR intersects the curves, must be taken the minor *n*, and the lowest values of *nbfi* and *nbfo*; this involves minimizing the area. Anyway, this type of search can be done in the three-dimensional matrix where the SNR values are stored.

## 8.1 Measurements and results with no device dependency

Figure 11 shows the maximum SNR for each *n* value. The SNR is almost linear versus *n*, and increases 6 dB per bit. The SNR in equation 6 is multiplied by 4 when the number of words is doubled in the LUT.

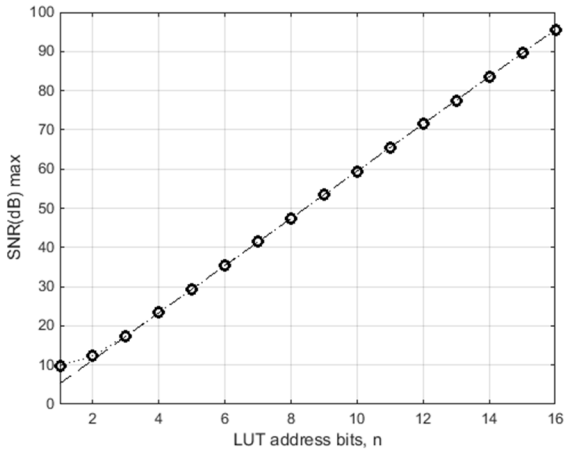

Figure 11. Maximum SNR in dBs for each *n* value

In figure 12 it is shown the input fractional bits versus the address bus size for the maximum SNR. In general, the number of input fractional bits is equal to the number of address bits plus four. This tendency indicates that it is necessary to increase one bit in the input when a bit is increased in the address bus; that is, the number of words in the LUT is doubled.

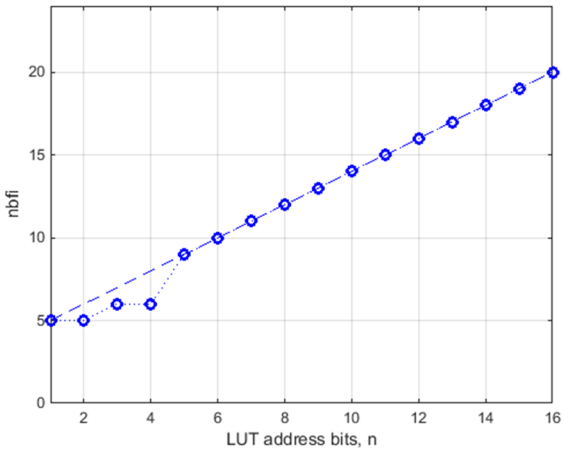

Figure 12. The input fractional bits against the address bus size for the maximum SNR.

Figure 13 shows the number of output fractional bits versus the size of the address bus, necessary to reach the maximum SNR, an almost linear tendency exists. In general, the number of bits required in the output is equal to the number of address bits plus four. It is necessary to increase one bit in the output when a bit is increased in the address bus; that is, the number of words is doubled in the LUT.

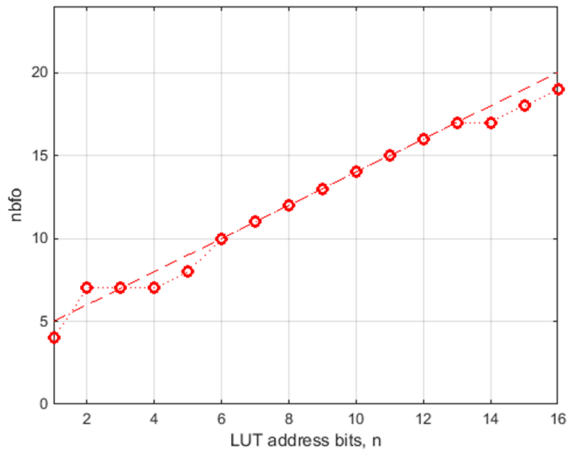


Figure 13. The Output fractional bits against the address bus size for the maximum SNR.

It should be emphasized that for 16 bits in the address bus the Verilog design size is 65,723 lines, and the testbench is 13,107,541 lines, including blank lines and comments; a hand coded design using a HDL would have been impossible to generate.

## 8.2 Measurements and results with device dependency

### 8.2.1 The Altera Arria II GX family

For this family, the device EP2AGX260FF35I5 was chosen [107]. One implementation was developed for each size of the LUT address bus; it was for the maximum SNR and minimum numbers of input and output fractional bits. The results of area, speed and power performances are shown below. It should be noted that the HDL Workflow Advisor does not allow to generate the project for one bit in the address bus, two words in the LUT; as this is a trivial case, the study from 2 to 16 bits in this bus is presented. The area versus the number of words in the LUT is shown in figure 14, which has a nearly linear shape. The area blocks of this device are Combinational ALUTs (Adaptive Lookup-Tables). It is true that area depends heavily on the number of words; but also depends on the word size. If the address bus increases in one bit, then the stored words are duplicated, but the words only increase in one bit. For this reason, the area depends heavily on the number of words but weakly on the number of bits. On average, 0.2 Combinational ALUTs per word stored in the LUT is needed.

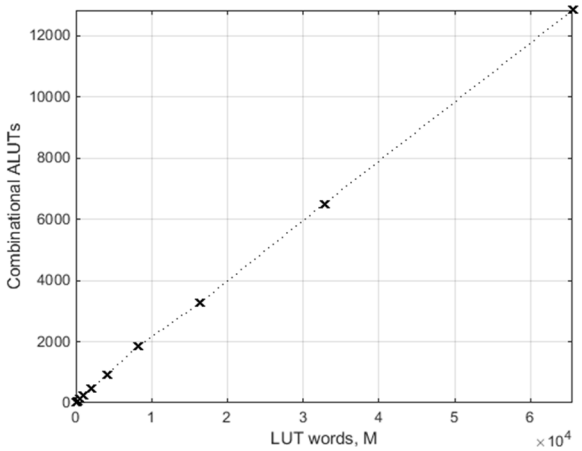


Figure 14. Hardware resources against the number of words in the LUT (Altera device EP2AGX260FF35I5 of Arria II GX family).

The maximum frequency versus *n*, for the maximum SNR, is shown in figure 15.a, which is not linear; is also not linear versus the number of words, which is not shown. The minimum allowed period for each case is represented in figure 15.b, it has almost linear behaviour by zones. In any case, a clear tendency is not observed for the speed.

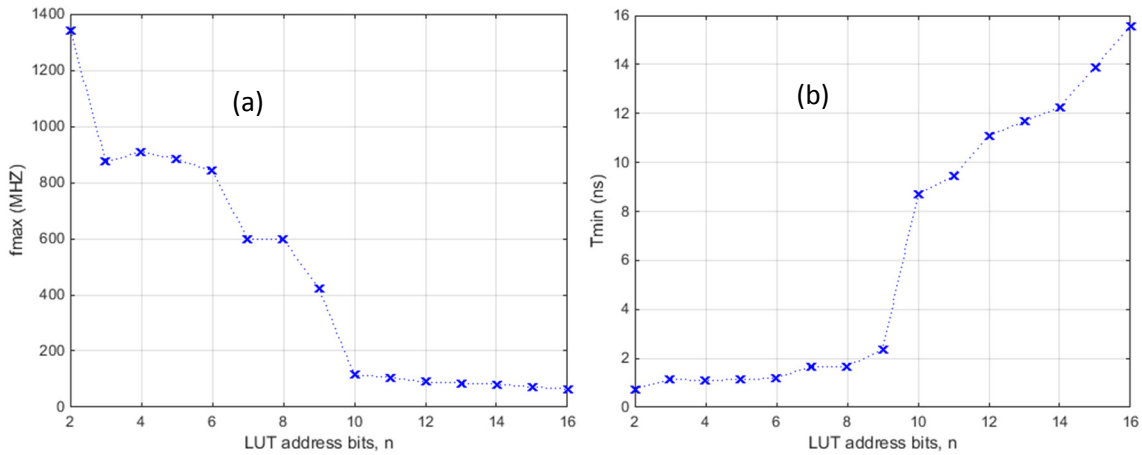


Figure 15. Maximum frequency (a) and minimum period (b) against the size of the LUT address bus for the maximum SNR (Altera device EP2AGX260FF35I5 of Arria II GX family).

Finally, the power was estimated for the previous cases (figure 16), the clock frequency was 25 MHz, smaller than the minimum in figure 15.a. The represented power is the sum of dynamic and the static powers, the behaviour is almost linear against the number of words (0.0026 mW/word).

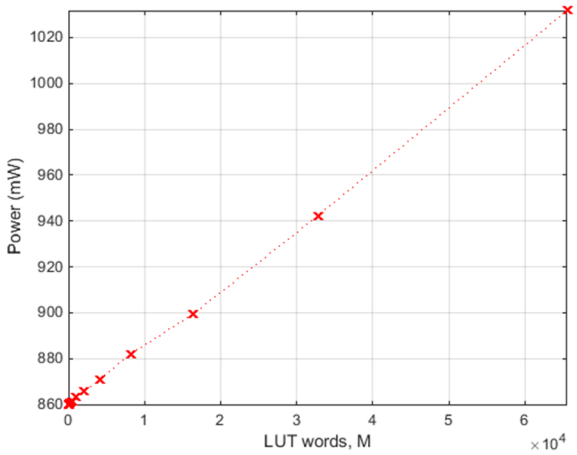


Figure 16. Consumed power against the number of words in the LUT for 25 MHz clock frequency (Altera device EP2AGX260FF35I5 of Arria II GX family).

Finally, it is possible to define a Quality Factor (expression 13), which includes the physical performances and functionality; the SNR obeys expression 6, not in dBs. In expression 13 the maximum frequency is expressed in hertz, the area is the number of Combinational ALUTs and the power is introduced in watts. The Quality Factor has an almost linear behaviour for *n* greater than 10, generally this value increases with *M*, according to figure 17. This factor can be used to compare different designs. For other systems the functionality must be measured with an appropriate parameter; for example, for a classifier may be the hit rate.

$$QF = \frac{f_{max} \cdot SNR}{Area \cdot Power} \qquad (13)$$

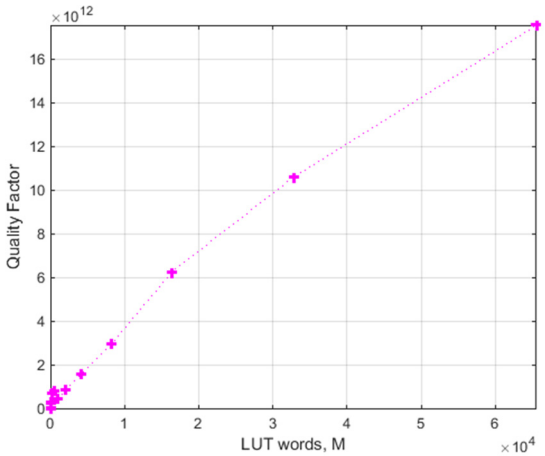


Figure 17. Quality Factor versus the number of words in the LUT (Altera device EP2AGX260FF35I5 of Arria II GX family).

### 8.2.2 The Altera Cyclone IV GX, MAX 10 and Stratix V families

The following tables show the results for other devices of different Altera FPGA families:
- the EP4CGX150DF31I7AD device of the Cyclone IV GX family [111].
- the 10M50SFE144I7G device of the MAX 10 family [112].
- the device 5SGXMBBR3H43I4 of the family Stratix V [113].

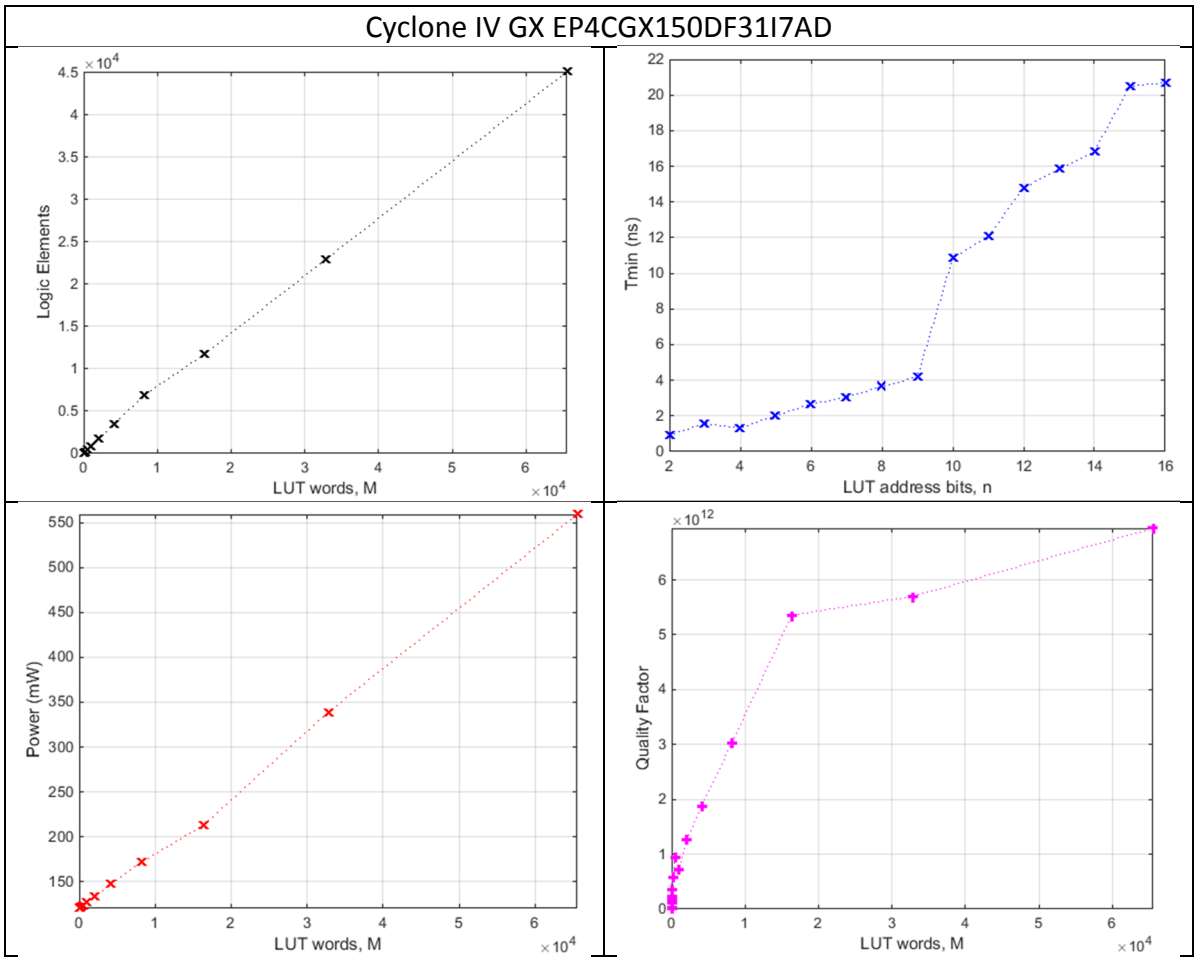


Table 2. Performances and Quality Factor for Altera device EP4CGX150DF31I7AD of Cyclone IV GX family.

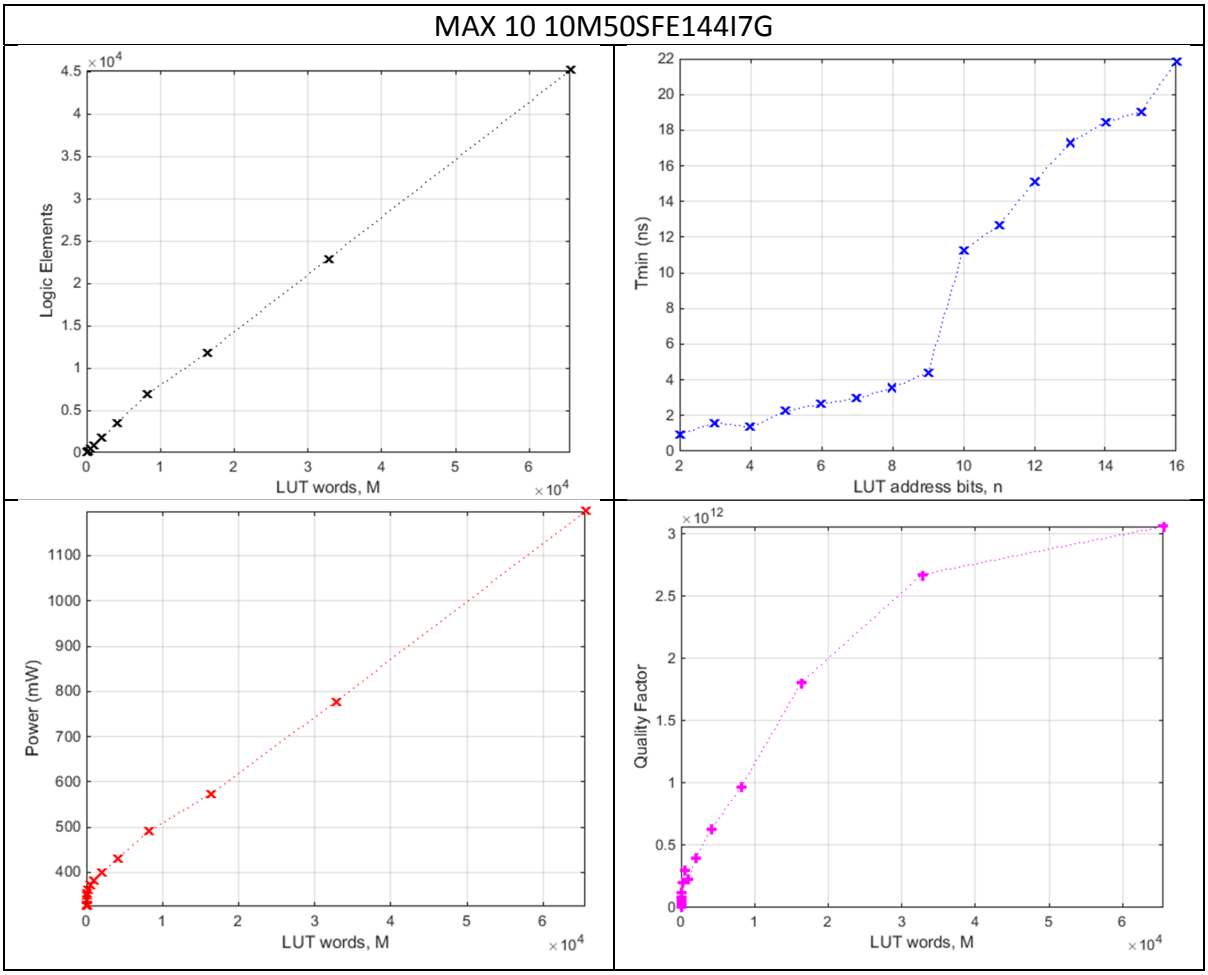


Table 3. Performances and Quality Factor for Altera device 10M50SFE144I7G of MAX 10 family.

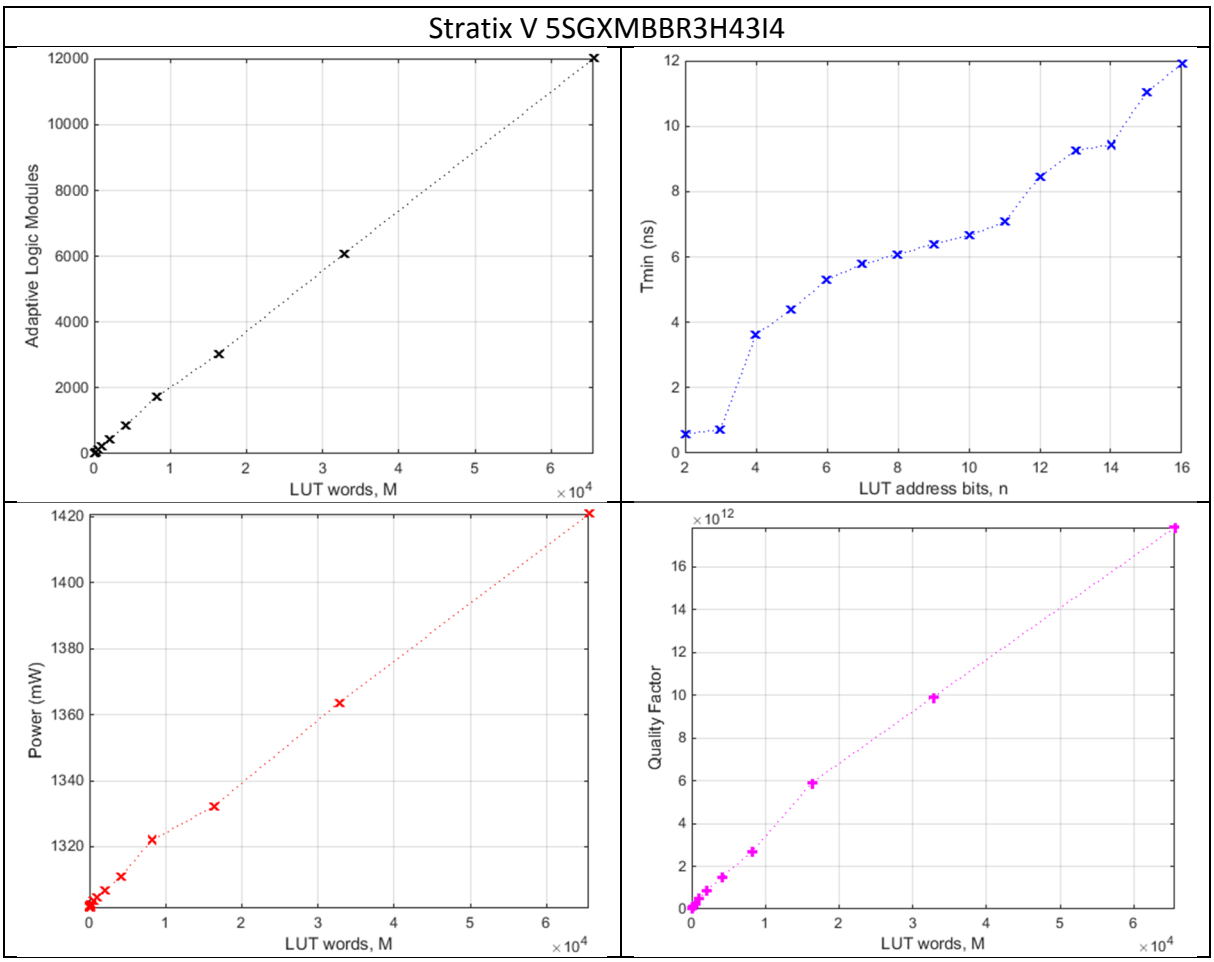


Table 4. Performances and Quality Factor for Altera device 5SGXMBBR3H43I4 of Stratix V family.

In the four devices the area is strongly linear versus the number of words in the LUT; for the speed is not observed a clear rule, vaguely the minimum period allowed in the clock signal is almost linear piecewise versus the number of bits of the LUT address bus; finally, the consumed power is strongly linear versus the number of words in the LUT. In all cases, the defined Quality Factor grows with the number of words; sometimes it is almost linear.

It should be noted that the logical blocks of the previous FPGAs, as indicated in the area figures, are:
- Combinational ALUTs (Adaptive Lookup-Tables), for Arria II GX family.
- Logic Elements, similar but not equal, for Cyclone IV GX and MAX 10 families,
- and Adaptive Logic Modules for the Stratix V family.

For the largest circuit, with 16 bits in the LUT address bus; the area for the devices is shown in Table 5. The Logic Elements of the Cyclone IV GX and MAX 10 families have similar behaviour for this design. Analogous, the Combinational ALUTs of the Arria II GX family and the Adaptive Logic Modules of the Stratix V are similar for implementing this design. For the area of this system, a Combinational ALUTs is equivalent to 3.5 Logic Elements; and an Adaptive Logic Module is equivalent to 3.8 Logic Elements.

| FAMILY | DEVICE | ELEMENT | Area for 16 bits in LUT address bus |
|---|---|---|---|
| Arria II GX | EP2AGX260FF35I5 | Combinational Adaptive Lookup-Tables | 12828 |
| Cyclone IV GX | EP4CGX150DF31I7AD | Logic Elements | 45108 |
| MAX 10 | 10M50SFE144I7G | Logic Elements | 45172 |
| Stratix V | 5SGXMBBR3H43I4 | Adaptive Logic Modules | 12009 |

Table 5. Hardware resources for 16 bits in the LUT address bus.

## 9. Conclusions and future lines

One of the objectives of this contribution is to show a fast and flexible design method for digital devices, which allows verifying different architectures and data formats for a system. In other words, the method allows exploring the space solutions. These advanced design techniques are embedded in Matlab for floating point models in files with Matlab extension [114] or for Simulink systems [115] and are connected to a digital synthesis tool. This method allows studying the effect of the number of bits in a wide range, which many authors only study in a discreet form by the limitation of the used method. Optimized systems can be transferred to FPGA or ASIC devices.

Altera and Xilinx have their own environments for designing as a block diagram in Simulink; these are DSP Builder [95] and System Generator [96] respectively, but handling fixed point format is more difficult.

The SNR has been introduced for measuring the functionality, this allows to compare the quality of an approximation of the sigmoid function (expression 1) to an approximation of the hyperbolic tangent (expression 2), for the same input range. On the other hand, the functionality estimation with the SNR in dBs allows observing linearities in performances.

Another possible study is to take advantage of the odd symmetry for the hyperbolic tangent, storing only half of the samples; this will save area, but it would be necessary to measure and compare the speed and power.

The developed approach uses intervals evenly spaced on the *x*-axis, which causes levels not evenly spaced on the *y*-axis. Some authors propose uniform intervals on the *y*-axis, which causes non-uniform spacing in the *x*-axis. These alternative designs are only somewhat more complex; for the same number of levels the approximation is improved, the reason is that there are more levels on the *x*-axis when the signal *y* changes more quickly and has the highest first derivative. But this strategy forces to introduce a set of comparators in the input, which increases area. In any case, this architecture should be studied, in order to establish its benefits and make comparisons; not only of the SNR but also the area, speed and power.

Other approaches are possible, using intervals or not, these can be implemented with the proposed method; on the same device for comparing the performances and Quality Factor. The evaluation of the area, speed and power are shown; but most authors do not estimate the power, which takes importance for the autonomy of batteries in mobile devices nowadays. Although power is the most complicated feature to evaluate. The delay of the systems depends on the latency and the clock period. In this case, the latency is two clock cycles, but with each cycle there is a valid value in the output; therefore, the data rate coincides with the clock frequency.

In the proposed model, when the size of the LUT address bus increases, large area is needed, solutions with a high number of words may become not realizable. The study of these solutions was performed to evaluate the method and to show the observed linearities. However, in the future, the increase of hardware resources in digital devices can allow using these implementations in totally parallelized ANNs.

Once the *tansig* function has been approached it is possible to approximate the *logsig* function using expression 4. According to this expression, the *logsig* function is approximated in [-8,+8] because the *tansig* function has been approximated in the interval [-4,+ 4]. On the other hand, the *logsig* function can be implemented directly using the same model with the appropriate settings. Furthermore, the *logsig* function approximation can take advantage of symmetry present around the point (0,0.5).

Actually, the HDL file and the testbench generated could be carried to other implementation tools, since they are implemented in generic code and do not use Altera primitives, but the project must be created and managed conveniently. In this way, it would be possible to repeat the experiments for Xilinx devices, but it is more convenient to generate the project from the HDL Workflow Advisor.

## Acknowledgments

Acknowledgment is expressed to Altera for donating software and licenses within its University Program.

## References

[1] M. Al-Nsour and H. S. Abdel-Aty-Zohdy, "Implementation of Programmable Digital Sigmoid Function Circuit for Neuro-Computing," in *40th Midwest Symposium on Circuits and Systems*, 1998.

[2] A. Armato, L. Fanucci, G. Pioggia and D. D. Rossi, "Low-error approximation of artificial neuron sigmoid function and its derivative," *Electronics Letters,* vol. 45, no. 21, pp. 1082- 1084, 2009.

[3] K. Basterretxea, “Recursive Sigmoidal Neurons for Adaptive Accuracy Neural Network Implementations,” in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2012)*, 2012.

[4] K. Basterretxea, J. M. Tarela and I. del Campo, “Approximation of sigmoid function and the derivative for hardware implementation of neural networks,” *IEE Proceedings-Circuits Devices and Systems, vol. 151, nº 1,* pp. 18-24, 2004.

[5] K. Basterretxea, J. Tarela and I. d. Campo, “Digital design of sigmoid approximator for artificial neural networks,” *Electronics Letters,* vol. 38, no. 1, pp. 35-37, 2002.

[6] I. d. Campo, R. Finker, J. Echanobe and K. Basterretxea, “Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons,” *Electronics Letters,* vol. 49, no. 25, p. 1598–1600, 2013.

[7] X. Chen, G. F. Wang, W. Zhou, S. Chang and S. L. Sun, “Efficient Sigmoid Function for Neural Networks Based FPGA Design,” in *International Conference on Intelligent Computing (ICIC), Lecture Notes in Computer Science*, 2006.

[8] D. E. Khodja, S. Simard and R. Beguenan, “Implementation of Optimized Approximate Sigmoid Function on FPGA Circuit to use in ANN for Control and Monitoring,” *Control Engineering and Applied Informatics (CEAI),* vol. 17, no. 2, pp. 64-72, 2015.

[9] H. K. Kwan, “Simple sigmoid-like activation function suitable for digital hardware implementation,” *IET Electronics Letters,* vol. 28, no. 15, pp. 1379-1380, 1992.

[10] K. Leboeuf, A. H. Namin, R. Muscedere, H. Wu and M. Ahmadi, “High Speed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function,” in *Third International Conference on Convergence and Hybrid Information Technology*, 2008.

[11] P. K. Meher, “An Optimized Lookup-Table for the Evaluation of Sigmoid Function for Artificial Neural Networks,” in *18th IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC 2010)*, 2010.

[12] M. C. Miglionico and F. Parillo, “A novel approach for implementing of a log-sigmoid function on a FPGA device using Sfloat24 Math library - An modelling,” in *Proceedings of the International Symposium on the Analytic Hierarchy Process*, 2013.

[13] V. P. Nambiar, M. Khalil, R. Sahnoun and M. N. Marsono, “Hardware implementation of evolvable block-based neural networks utilizing a cost efficient sigmoid-like activation function,” *Neurocomputing,* vol. 140, no. 1, p. 228–241, 2014.

[14] S. Ngah, R. A. Bakar, A. Embong and S. Razali, “Two-steps implementation of sigmoid function for artificial neural network in Field Programmable Gate Array,” *ARPN Journal of Engineering and Applied Sciences,* vol. 11, no. 7, pp. 4882-4888, 2016.

[15] A. Tisan, S. Oniga, D. Mic and A. Buchman, “Digital implementation of the sigmoid function for FPGA circuits,” *Acta Technica Napocensis Electronics and Telecommunications,* vol. 50, no. 2, pp. 15-20, 2009.

[16] M. T. Tommiska, “Efficient digital implementation of the sigmoid function for reprogrammable logic,” *IEE Proceedings-Computers and Digital Techniques,* vol. 150, no. 6, pp. 403-411, 2003.

[17] C.-H. Tsai, Y.-T. Chih, W. H. Wong and C.-Y. Lee, “A Hardware-Efficient Sigmoid Function With Adjustable Precision for a Neural Network System,” *IEEE Transactions on Circuits and Systems - II: Express Briefs,* vol. 62, no. 11, pp. 1073-1077, 2015.

[18] M. Zhang, S. Vassiliadis and J. G. Delgado-Frias, “Sigmoid Generators for Neural Computing Using Piecewise Approximations,” *IEEE Transactions on Computers,* vol. 45, no. 9, pp. 1045-1049, 1996.

[19] H. Faiedh, C. Souani, K. Torki and K. Besbes, “Digital Hardware Implementation of a Neural System Used for Nonlinear Adaptive Prediction,” *Journal of Computer Science,* vol. 2, no. 4, pp. 355-362, 2006.

[20] E. M. Ortigosa, A. Canas, E. Ros and R. R. Carrillo, “FPGA implementation of a perceptron-like neural network for embedded applications,” in *7th International Work Conference on Artificial and Natural Neural Networks, Lecture Notes in Computer Science*, 2003.

[21] R. A. Callejas-Molina, V. M. Jimenez-Fernandez and H. Vazquez-Leal, “Digital Architecture to Implement a Piecewise-Linear Approximation for the Hyperbolic Tangent Function,” in *IEEE International Conference on Computing Systems and Telematics-ICCSS*, 2015.

[22] S. Gomar, M. Mirhassani and M. Ahmadi, “Precise Digital Implementations of Hyperbolic Tanh and Sigmoid Function,” in *50th Asilomar Conference on Signals; Systems and Computers*, 2016.

[23] C. Lin and J. Wang, “A digital circuit design of hyperbolic tangent sigmoid function for neural networks,” in *IEEE International Symposium on Circuits and Systems*, 2008.

[24] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu and M. Ahmadi, “Efficient Hardware Implementation of the Hyperbolic Tangent Sigmoid Function,” in *IEEE International Symposium on Circuits and Systems*, 2009.

[25] M. Panicker and C.Babu, “Efficient FPGA Implementation of Sigmoid and Bipolar Sigmoid Activation Functions for Multilayer Perceptrons,” *IOSR Journal of Engineering (IOSRJEN),* vol. 2, no. 6, pp. 1352-1356, 2012.

[26] B. Zamanlooy and M. Mirhassani, “Efficient VLSI Implementation of Neural Networks with Hyperbolic Tangent Activation Function,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 22, no. 1, pp. 39-48, 2014.

[27] C. M. Bishop, Neural Networks for Pattern Recognition, 13ª ed., Oxford University Press, 2005.

[28] D. Graupe, Principles of Artificial Neural Networks, 3ª ed., World Scientific, 2013.

[29] Y. H. Hu and J. N. Hwang, Handbook of Neural Network Signal Processing, CRC Press, 2002.

[30] A. R. Omondi and J. C. Rajapakse, FPGA Implementations of Neural Networks, Springer, 2006.

[31] D. Tebbe, J. Doner and T. Billhartz, “Neural Network Communications Signal Processing,” Harris Corporation, Government Communications Systems Division, USA, 1994.

[32] S. T. Pérez, C. M. Travieso and J. B. Alonso, “Design Methodology of an Equalizer for Unipolar Non Return to Zero Binary Signals in the Presence of Additive White Gaussian Noise Using a Time Delay Neural Network on a Field Programmable Gate Array,” *Sensor,* vol. 13, no. 12, pp. 16829-16850, 2013.

[33] M. Costa, D. Palmisano and E. Pasero, “A system design methodology for analog feed forward artificial neural networks,” *Analog Integrated Circuit and Signal Processing,* vol. 21, no. 1, pp. 45-55, 1999.

[34] D. Maliuk, H. G. Stratigopoulos and Y. Makris, “An Analog VLSI Multilayer Perceptron and its Application Towards Built-In Self-Test in Analog Circuits,” in *IEEE 16th International On-Line Testing Symposium (IOLTS 2010)*, 2010.

[35] S. Satyanarayana, Y. Tsividis and H. Graf, “A reconfigurable VLSI Neural Network,” *IEEE Journal of Solid-State Circuits,* vol. 27, no. 1, pp. 67-81, 1992.

[36] P. Dong, G. L. Bilbro and M. Y. Chow, “Implementation of artificial neural network for real time applications using field programmable analog arrays,” in *IEEE International Joint Conference on Neural Network (IJCNN)*, 2006.

[37] R. Manjunath and K. Gurumurthy, “Artificial neural networks as building blocks of mixed signal FPGA,” in *2nd International Conference on Field-Programmable Technology (ICFPT 2003)*, 2003.

[38] D. Myers and R. Hutchinson, “Efficient implementation of piecewise linear activation function for digital VLSI neural networks,” *Electronics Letters,* vol. 25, no. 24, pp. 1662-1663, 1989.

[39] X. Wang and Z. Ma, “Discussion on the methodology of neural network hardware design and implementation,” in *6th International Conference on Solid-State and Integrated-Circuit Technology*, 2001.

[40] R. Selow, H. S. Lopes and C. R. Erig, “A comparison of FPGA and FPAA technologies for a signal processing application,” in *International Conference on Field Programmable Logic and Applications*, 2009.

[41] P. Belanovic, *Library of Parameterized Hardware Modules for Floating-Point Arithmetic with An Example Application,* Ph. D. Dissertation, Northeastern University Boston, Massachusetts, 2002.

[42] P. Belanovic and M. Leeser, “A library of parameterized floating-point modules and their use,” in *12th International Conference on Field-Programmable Logic and Applications, Lecture Notes in Computer Science*, 2002.

[43] M. A. Cavuslu, C. Karakuzu, S. Sahin and M. Yakut, “Neural network training based on FPGA with floating point number format and it's performance,” *Neural Computing and Applications,* vol. 20, no. 2, pp. 195-202, 2011.

[44] S. Kawamura and M. S. Yoshida, “FPGA Implementation of Neuron Model Using Piecewise Nonlinear Function on Double-Precision Floating-Point Format,” in *29th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, Lecture Notes in Computer Science*, 2016.

[45] A. W. Savich, M. Moussa and S. Areibi, “The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study,” *IEEE Transactions on Neural Networks,* vol. 18, no. 1, pp. 240-252, 2007.

[46] K. Han, *Automating transformations from floating-point to fixed-point for implementing digital signal processing algorithms,* Ph. D. Dissertation, University of Texas, Austin, 2006.

[47] S. Roy and P. Banerjee, “An algorithm for converting floating-point computations to fixed-point in MATLAB based FPGA design,” in *41st Design Automation Conference*, 2004.

[48] MathWorks, “Fixed-Point Designer,” [Online]. Available: http://www.mathworks.es/products/fixed-point-designer. [Accessed October 2017].

[49] C. Shi, *Floating-point to Fixed-point Conversion,* Ph. D. Dissertation, University of California, Berkeley, 2004.

[50] IEEE, “IEEE Standard for Floating-Point Arithmetic,” [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=4610933, permanent link. [Accessed October 2017].

[51] V. A. Chandrasetty, VLSI Design A Practical Guide for FPGA and ASIC Implementations, Springer, 2011.

[52] K. Van and H. Abdel, “A Reconfigurable Spiking Neural Network Digital ASIC Simulation and Implementation,” in *IEEE National Aerospace and Electronics Conference*, 2009.

[53] T. Ho, P. Lam and C. Leung, “Parallelization of cellular neural networks on GPU,” *Pattern Recognition,* vol. 41, no. 8, pp. 2684-2692, 2008.

[54] K. Oh and J. K., “GPU implementation of neural networks,” *Pattern Recognition,* vol. 37, no. 6, pp. 1311-1314, 2004.

[55] C. Maxfield, The Design Warrior's Guide to FPGAs, Elsevier, 2004.

[56] J. Oldfield and R. Dorf, Field-Programmable Gate Array, John Wiley and Sons, 1995.

[57] A. R. Omondi and J. C. Rajapakse, “Neural Networks in FPGAs,” in *9th International Conference on Neural Information Processing (ICONIP'O2)*, 2002.

[58] R. C. Cofer and B. F. Harding, Rapid System Prototyping with FPGAs, Elsevier, 2006.

[59] M. Gokhale and P. Graham, Reconfigurable Computing Accelerating Computation with Field-Programmable Gate Arrays, Springer, 2005.

[60] S. Himavathi, D. Anitha and A. Muthuramalingam, “Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization,” *IEEE Transactions on Neural Networks,* vol. 18, no. 3, pp. 880-888, 2007.

[61] Z. Lin, Y. Dong, Y. Li and T. Watanabe, “A Hybrid Architecture for Efficient FPGA-based Implementation of Multilayer Neural Network,” in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2010.

[62] A. H. Namin, K. Leboeuf, H. Wu and M. Ahmadi, “Artificial Neural Networks Activation Function HDL Coder,” in *IEEE International Conference on Electro/Information Technology*, 2009.

[63] A. Gomperts, A. Ukil and F. Zurfluh, “Development and Implementation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications,” *IEEE Transactions on Industrial Informatics,* vol. 7, no. 1, pp. 78-89, 2011.

[64] M. Krips, T. Lammert and A. Kummert, “FPGA implementation of a neural network for a real-time hand tracking system,” in *1st IEEE International Workshop on Electronic Design, Test and Applications*, 2002.

[65] T. Orlowska and M. Kaminski, “FPGA Implementation of the Multilayer Neural Network for the Speed Estimation of the Two-Mass Drive System,” *IEEE Transactions on Industrial Informatics,* vol. 7, no. 3, pp. 436-445, 2011.

[66] V. Saichand, D. M. Nirmala, S. Arumugam and N. Mohankumar, “FPGA Realization of Activation Function for Artificial Neural Networks,” in *8th International Conference on Intelligent Systems Design and Applications (ISDA 2008)*, 2008.

[67] M. Bahoura and C. W. Park, “FPGA-Implementation of High-Speed MLP Neural Network,” in *18th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2011), pp. 426-429*, 2011.

[68] A. Mishra, Zaheeruddin and K. Raj, “Implementation of a digital neuron with nonlinear activation function using Piecewise linear approximation technique,” in *International Conference on Microelectronics*, 2007.

[69] D. Larkin, A. Kinane, V. Muresan and N. O'Connor, “An Efficient Hardware Architecture for a Neural Network Activation Function Generator,” in *3rd International Symposium on Neural Networks (ISNN 2006), Lecture Notes in Computer Science*, 2006.

[70] S. Vassiliadis, M. Zhang and J. G. Delgado, “Elementary Function Generators for Neural-Network Emulators,” *IEEE Transactions on Neural Networks,* vol. 11, no. 6, pp. 1438-1449, 2000.

[71] B. K. Bharkhada, J. Hauser and C. Purdy, “Efficient FPGA implementation of a generic function approximator and its application to neural net computation,” in *46th Midwest Symposium on Circuits and Systems*, 2003.

[72] B. Lee and N. Burgess, “Some results on Taylor-series function approximation on FPGA,” in *37th Asilomar Conference on Signals, Systems and Computers*, 2003.

[73] K. Mohamad, M. F. O. Mahmud, F. H. Adnan and W. F. H. Abdullah, “Design of Single Neuron on FPGA,” in *IEEE Symposium on Humanities; Science and Engineering Research*, 2012.

[74] E. Soria-Olivas, J. D. Martín-Guerrero, G. Camps-Valls, A. J. Serrano-López, J. Calpe-Maravilla and L. Gómez-Chova, “A Low-Complexity Fuzzy Activation Function for Artificial Neural Networks,” *IEEE Transactions on Neural Networks,* vol. 14, no. 6, pp. 1576-1579, 2003.

[75] A. Rosado-Muñoz, E. Soria-Olivas, L. Gomez-Chova and J. V. Francés, “An IP Core and GUI for Implementing Multilayer Perceptron with a Fuzzy Activation Function on Configurable Logic Devices,” *Journal of Universal Computer Science,* vol. 14, no. 10, pp. 1678-1694, 2008.

[76] A. Armato, L. Fanucci, E. P. Scilingo and D. De Rossi, “Low-error digital hardware implementation of artificial neuron activation functions and their derivative,” *Microprocessors and Microsystems,* vol. 35, no. 6, pp. 557-567, 2011.

[77] S. Roy and P. Banerjee, “An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design,” *IEEE Transactions on Computers,* vol. 54, no. 7, pp. 886-869, 2005.

[78] L. Deng, K. Sobti, Y. Zhang and C. Chakrabarti, “Accurate Area, Time and Power Models for FPGA-Based Implementation,” *Journal of Signal Processing Systems,* vol. 63, no. 1, pp. 39-50, 2011.

[79] A. Tisan and J. Chin, “An End-User Platform for FPGA-Based Design and Rapid Prototyping of Feedforward Artificial Neural Networks With On-Chip Backpropagation Learning,” *IEEE Transactions on Industrial Informatics,* vol. 12, no. 3, pp. 1124-1133, 2016.

[80] Z. Szadkowski, K. Pytel and P. A. Collaboration, “Artificial Neural Network as a FPGA Trigger for a Detection of Very Inclined Air Showers,” *IEEE Transactions on Nuclear Science,* vol. 62, no. 3, pp. 1002-1009, 2015.

[81] S. Oniga, A. Tisan, D. Mic, A. Buchman and A. Vida, “Optimizing FPGA implementation of Feed-Forward Neural Networks,” in *11th International Conference on Optimization of Electrical and Electronic Equipment*, 2008.

[82] A. S. Ogrenci, “Abstraction in FPGA implementation of neural networks,” in *9th WSEAS International Conference on Neural Networks (NN 08)*, 2008.

[83] C. T. Yen, W. D. Weng and Y. T. Lin, “FPGA realization of a neural-network-based nonlinear channel equalizer,” *IEEE Transactions on Industrial Electronics,* vol. 51, no. 2, pp. 472-479, 2004.

[84] P. Ling, “Taking a battering,” 2005. [Online]. Available: http://www.newelectronics.co.uk/article-images/5488/taking-a-battering.pdf. [Accessed November 2017].

[85] F. Wickersham, “The drive to low power,” 2009. [Online]. Available: http://signal-processing.mil-embedded.com/articles/the-drive-lower-power/. [Accessed November 2017].

[86] W. Meeus, K. Van, T. Goedemé, J. Meel and D. Stroobandt, “An overview of today’s high-level synthesis tools,” *Design Automation of Embedded Systems,* vol. 16, no. 3, pp. 31-51, 2012.

[87] MathWorks, “Simulink,” [Online]. Available: http://www.mathworks.com/products/simulink. [Accessed October 2017].

[88] MathWorks, “MATrix LABoratory de MathWorks (Matlab),” [Online]. Available: http://www.mathworks.com. [Accessed October 2017].

[89] IEEE, “IEEE Standard VHDL Language Reference Manual,” [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=4772738, permanent link. [Accessed October 2017].

[90] V. A. Pedroni, Circuit Design with VHDL, MIT Press, 2004.

[91] IEEE, “IEEE Standard for Verilog Hardware Description Language,” [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=10779, permanent link. [Accessed October 2017].

[92] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, 2ª ed., Prentice Hall, 2003.

[93] Altera, “Altera Corporation,” [Online]. Available: http://www.altera.com/. [Accessed October 2017].

[94] Xilinx, “Xilinx Corporation,” [Online]. Available: http://www.xilinx.com. [Accessed October 2017].

[95] Altera, “DSP Builder,” [Online]. Available: http://www.altera.com/products/software/products/dsp/dsp-builder.html. [Accessed October 2017].

[96] Xilinx, “System Generator for DSP,” [Online]. Available: http://www.x.com/products/design-tools/vivado/integration/sysgen.html. [Accessed October 2017].

[97] Altera, “Quartus II,” [Online]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html. [Accessed October 2017].

[98] Xilinx, “Vivado Design Suite,” [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html. [Accessed October 2017].

[99] S. T. Pérez, C. Osorio, J. L. Vásquez, J. B. Alonso and C. M. Travieso, “Design methodology of a fully parallelized Neural Network on a FPGA,” in *8th WSEAS International Conference on Circuits, Systems, Signal and Telecommunications (CSST '14)*, 2014.

[100] MathWorks, “Matlab Neural Network Toolbox,” [Online]. Available: http://es.mathworks.com/products/neural-network. [Accessed October 2017].

[101] S. T. Pérez, J. L. Vásquez, J. R. Ticay, J. B. Alonso and C. M. Travieso, “Artificial Neural Network in FPGA for Pejibaye Palm Classification Using Molecular Markers,” in *Extended Abstracts of the Thirteen International Conference on COMPUTER AIDED SYSTEMS THEORY ( EUROCAST 2011)*, 2011.

[102] J. L. Vásquez, S. T. Pérez, C. M. Travieso and J. B. Alonso, “Meteorological Prediction Implemented on Field-Programmable Gate Array,” *Cognitive Computation,* vol. 5, no. 4, pp. 551-557, 2013.

[103] A. Vázquez, S. T. Pérez, C. M. Travieso and J. B. Alonso, “Cardiac Pathologies Detection over FPGA using Electrocardiogram,” in *International Conference on Bio-inspired Systems and Signal Processing (BIOSIGNALS 2012)*, 2012.

[104] C. M. Travieso, S. T. Pérez and J. B. Alonso, “Using Fixed Point Arithmetic for Cardiac Pathologies Detection Based on Electrocardiogram,” *Lecture Notes in Computer Science,* vol. 8112, no. 2, p. 242–249, 2013.

[105] A. V. Oppenheim and R. W. Schafer, Discrete-time signal processing, Prentice-Hall, 1989.

[106] L. W. Couch, Digital and Analog Communications Systems, 4ª ed., MacMillan, 1993.

[107] Altera, “Arria II FPGAs,” [Online]. Available: https://www.altera.com/products/fpga/arria-series/arria-ii/support.html. [Accessed October 2017].

[108] Altera, “Intel Quartus Prime Design Software Overview,” [Online]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html. [Accessed October 2017].

[109] Altera, “ModelSim-Altera Edition,” [Online]. Available: https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html. [Accessed October 2017].

[110] MathWorks, “HDL Workflow Advisor,” [Online]. Available: https://www.mathworks.com/examples/matlab-hdl-coder/mw/hdlcoder_product-mlhdlc_tutorial_hdlcodegen-basic-hdl-code-generation-with-the-workflow-advisor. [Accessed October 2017].

[111] Altera, “Cyclone IV FPGAs,” [Online]. Available: https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html. [Accessed November 2017].

[112] Altera, “MAX 10 FPGAs,” [Online]. Available: https://www.altera.com/products/fpga/max-series/max-10/overview.html. [Accessed November 2017].

[113] Altera, “Stratix V FPGAs,” [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-v/overview.html. [Accessed November 2017].

[114] MathWorks, “Fixed-Point Advisor,” [Online]. Available: https://es.mathworks.com/help/fixedpoint/ug/fixed-point-advisor.html. [Accessed October 2017].

[115] MathWorks, “HDL Workflow Advisor,” [Online]. Available: https://www.mathworks.com/examples/matlab-hdl-coder/mw/hdlcoder_product-mlhdlc_tutorial_hdlcodegen-basic-hdl-code-generation-with-the-workflow-advisor. [Accessed October 2017].