2011 International Workshop on ADC Modelling, Testing and Data Converter Analysis and Design and IEEE 2011 ADC Forum
June 30 - July 1, 2011. Orvieto, Italy.

Hardware Implementation of an ADC Error Compensation Using Neural Networks

Hervé Chanal[1]

[1]*Clermont Université, Université Blaise Pascal,CNRS/IN2P3, Laboratoire de Physique Corpusculaire, Pôle Micrhau, BP 10448, F-63000 CLERMONT-FERRAND,France*

***Abstract*-**A compensation technique for Analog-to-Digital Converter (ADC) based on a neural network is proposed. The implementation is done both in software and in a hardware description language. The latter is targeted for a massively parallel ASIC. The training of the neural network is done by learning a Look Up Table generated by processing the output of the ADC for sine waves inputs. Then, the effective number of bits (ENOB) is computed over a large range of frequencies for the raw data of a 100MS/s ADC and for the compensated data. These results are used to compare various neural network architecture and the effects of the approximations made for the hardware implementation.

I. Introduction

Analog-to-digital converters (ADC) are at the core of measurement systems as they interface the analog world with the digital one. They have an ever-growing field of application thanks to storage system, optical or ultra-wideband communication while their running frequency and precision increase. However, their accuracy is greatly affected by the ADC actual behaviour.
They are prone to exhibits two categories of errors: static and dynamic. The first one applies when the input signal is either slowly varying or static. Hence, it includes the quantization error but also the non-ideal spacing of the code transition levels. The dynamic errors are induced by the time variation of the analog signal input. It includes harmonic distortion, dynamic effects on the ADC internal stage, or frequency-dependant code transition level. This type of errors is much more difficult to overcome as the compensation scheme must take into account the history of the converter output.
As a result, the dynamic ADC modelling and compensation techniques have been the subject of intense research in the course of the past decade [1,2]. New methods using enhanced LUT [3] in the state plane or in the space plane have arisen while analytical correction greatly improved [4]. These two examples are post-correction methods which do not modify the converters architecture but do a digital processing on the ADC outputs. Meanwhile, the hardware implementation of these works is barely shown.
In this paper, a neural network approach is used for the compensation. It is aimed at improving the previously reported results [5] and at introducing an actual implementation in an ASIC with the IBM 0.13µm CMOS technology. To be able to cope with high speed application, the presented hardware neural network is massively parallel.
The first part presents the compensation architecture we implemented both in hardware and software. The neural network used is described with an emphasis on the activation function.
The second part of this article is dedicated to the results. The selected neural network architectures are tested with the software model to check their accuracy and with the hardware model to get their resource usage. Both implementations are compared to validate the method and to study the differences.

II. Neural network compensation

A. Compensation architecture

The proposed compensation scheme is a post correction technique. It uses the ADC outputs current sample and a delayed one (see Fig. 1) . The processing is made in two steps. In the first one, the ADC output is sent to the neural network which computes the correction value. Then, the result is added to the ADC current sample.

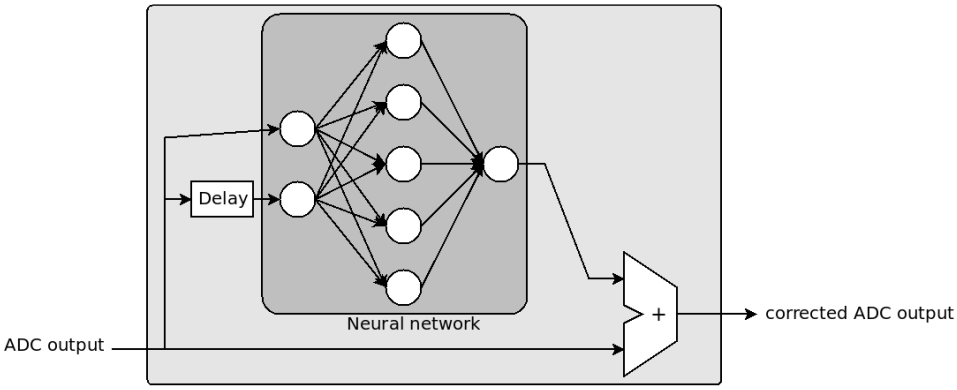

Figure 1. Proposed neural network correction scheme

Choosing the architecture of a neural network is a complex task. Let us recall that it is composed of layers of nodes which are the processing units. The name of a neural network and a part of its performance derive from the connectivity between the nodes.
For the sake of simplicity, a feed-forward neural network is used in this work. It is an architecture where the outputs of the nodes from one layer are only connected to the input of the nodes of the next layer. Hence, its components can be expressed as a set of input, hidden and output nodes. The input nodes are directly the ADC output values, while the output node will give the compensation value. The hidden nodes process the data. Thereby, the output s_j of the j^{th} node on the layer k is given by:

$$s_j = F\left(\sum_i w_{ij} o_i + b_j\right) \tag{1}$$

where F is called an activation function, o_i is the output from the i^{th} node of the k-1 layer, w_{ij} the weight and b_j the bias.
The bias and the weight of each node have to be determined using a learning algorithm. In the proposed method, the neural network is trained to reproduce the output of a Look Up Table (LUT) providing the error code for a given input. The LUT coefficients are computed with the method presented in the in Lundin work [2] with a set of data processed by ADC stimulated by sinusoid at various frequencies. Only a quarter of the LUT is used for the training. Moreover its data is scrambled to prevent effects due to the LUT ordering.
The number of layer and nodes are determined empirically.

B. Activation function

The equation (1) involves an activation function. It can be any function but the best results are obtained using a sigmoid as, for example, a hyperbolic tangent. It is obvious that using a complex function will take too much hardware resources as we intend to have a massively parallel neural network. Hence, two approximated sigmoid are used. The first one has been developed for an FPGA implementation [6] and can be expressed as:

$$sig : x \mapsto \begin{cases} 1 \; if \;\; x \le -2 \\ x(1 + x/4) \; if \;\; -2 < x < 0 \\ x(1 - x/4) \; if \;\; 0 < x < 2 \\ 1 \; if \;\; x \ge 2 \end{cases} \tag{2}$$

This function will be called *sig* in the following sections. Its shape is very similar to the hyperbolic tangent function but it can be generated with only a multiplier. The divide by 4 can be done by a shift of 2 bits of the input data.

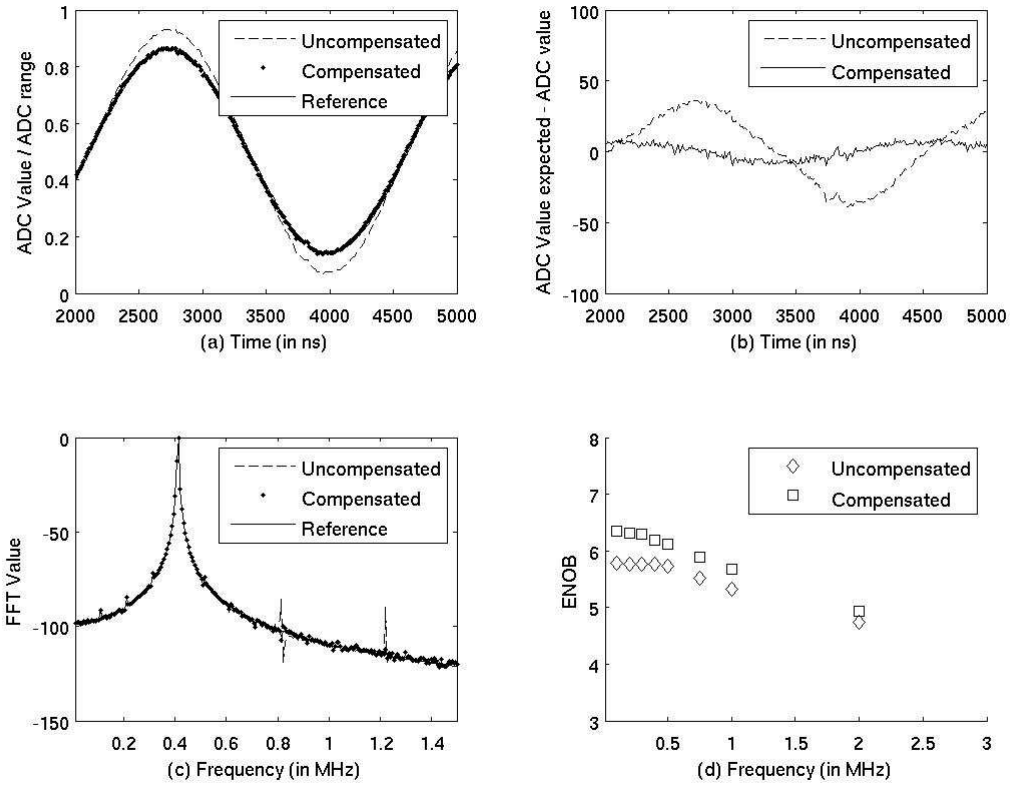

Figure 2. Neural network compensation using the software implementation for 10 hidden nodes and the *sig* activation function: (a) transient result for a 400kHz sinusoid input, (b) difference between the reference ADC transient result and the uncompensated or the compensated ones, (c) Spectrum of the output signal (dBi) and (d) effective number of bits in the 0-2MHz frequency range.

The second function used is even simpler. It takes directly the summation results as output but with saturating bounds. It can be expressed as:

$$lin : x \mapsto \begin{cases} 1 \, if \;\; x \le -1 \\ x \; if \;\; -1 < x < 1 \\ 1 \, if \;\; x \ge 1 \end{cases} \qquad (2)$$

This function will be called *lin* in the following sections. In this case, no arithmetic operations are involved. Hence, its hardware resources should be minimal.

B. Implementation

The neural network is implemented both as a matlab program and as a synthesisable VHDL code targeting an ASIC. It is quite straightforward in both cases. The synthesis of the VHDL code has been done with the Cadence software and the IBM 0.13µm CMOS technology.
In addition to the language difference, the hardware model has to cope with the representation of the floating point weight and bias generated by the software training. Keeping the same resolution in the hardware will lead to large resource usage and power consumption. Hence, fixed point arithmetic is used with 3 bits for the integer part, 1 bit for the sign and 9 bits for the mantissa. The effect of the precision difference will be studied in the next section.
The proposed method is a three steps approach. The first one is to generate the data set from the actual ADC. It has to cover the most of the expected code transition. Then, the results are stored in variables for matlab and in

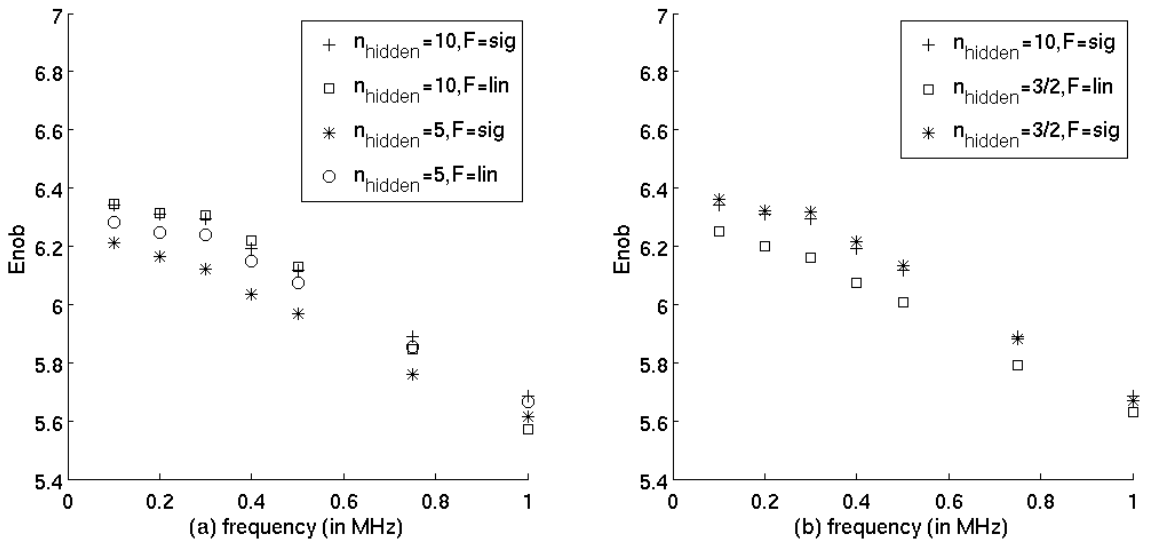

Figure 3. ENOB for the studied neural network architectures (a) using one hidden layer with 5 or 10 nodes and the *lin* or the *sig* activation function (b) comparing between one hidden layer with 10 nodes and two hidden layer respectively 3 and 2 nodes.

the ASIC internal registers for the hardware model. Finally, a larger set of data coming from the actual ADC is used to compute the effective number of bits (ENOB) [7] and to study the algorithms behaviour.

III. Results

A. Data sets

The output of a 100MS/s ADC previously developed in our lab [8] is used to test the compensation approach. Its architecture is pipeline with a 9 bit output. A dedicated board for the dynamic testing has been designed where a sinusoidal signal is sent both on the tested ADC and on a reference 16 bit ADC. The results are stored in the internal memories of a FPGA and read on a control PC. 16000 samples are available for each acquisition.

B. Software simulations

A sample of data set coming from the test bench is then processed by the matlab program to train the neural network and retrieve the weight and the bias. Only results generated from sinusoid with a frequency below 400 kHz are used in this step. In the third and last step, the results presented here are generated from the processing of the full data sets.
The Fig 2. shows the results for a neural network with 10 nodes in one hidden layer. The activation function is the approximated sigmoid *sig*. The transient and spectral results are given for a 400kHz sinusoid. The first set of curves (a) illustrate that the gain error has been significantly compensated. The difference between the matlab program and the reference ADC output are barely visible. To be able to made further analysis, the difference between the expected result and the software outputs is plotted on the set (b). The difference goes from 40 ADC count in the uncompensated case to 8 ADC count in the compensated one. Hence, the method drastically decreases the errors and is able to correct the gain.
The set of curve (c) give a spectral view of the output signal. We can observe that the compensation scheme greatly lowered the distortion as the second and third distortion peak almost vanished.
The last set of curves (d) shows the ENOB for an input frequency varying between 0 and 2MHz. We can cleary see that it is improved by 0.8 bits at the beginning with a performance decreasing with the frequency. It has to be noted that an improvement is seen even in frequency range where the neural network has not been trained.
The figure 3 shows some effects of the neural network architecture on the ENOB. The results presented here have to be mitigated by the fact that difference can be observed between two training even with the same

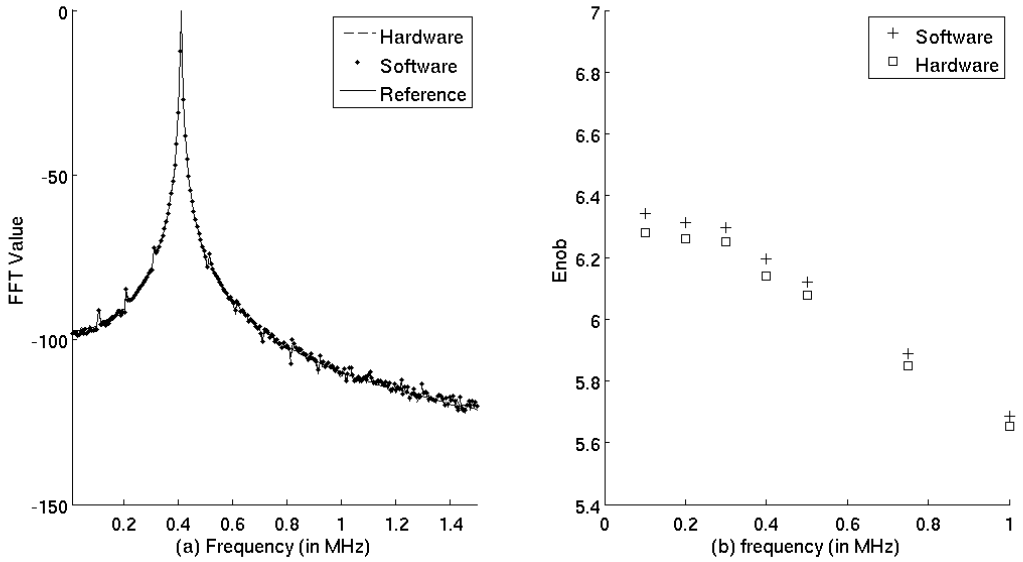

Figure 4. Difference between the software model and the hardware one using a spectrum of the output signal (a) and the ENOB (b)

network architecture. The first set of curve use a neural network with one hidden layer. It shows that there is not so much difference between the results with a sigmoid-like activation function and a linear one for the 10 hidden node cases. The ENOB results with only 5 nodes are between 0.1 and 0.2 bit bellow. On the second set of curve, it can observed that a neural network with two layer (3 nodes followed by 2 nodes) give almost the same results as the 10 nodes on 1 layer case. In this case, the *lin* results are significantly bellow the *sig* activation function.

C. Hardware simulations

All the previous presented architectures have been implemented in synthesisable VHDL. The first step has been to check the consistency of both approach as the accuracy of the floating point weight and bias are degraded in the hardware case. An example of the spectrum of the output signal using the same 400 kHz sinusoid is presented on the Fig. 3.a. The differences between the two cases are barely visible. On the other hand, the Fig. 3.b shows the ENOB in the 0-2MHz frequency range. The difference is about 0.1 bits in the studied results. Finally, the neural networks have been synthesised using Cadence RTL compiler. The results are reported in the following tabular:

Architecture type	Number of standard cell	Estimated Power consumption
1 hidden layer with 10 nodes and *syn* activation function	31047	50mW
1 hidden layer with 5 nodes and *syn* activation function	15638	25mW
1 hidden layer with 5 nodes and *lin* activation function	8479	6mW
2 hidden layer with 2/3 nodes and *syn* activation function	20192	25mW

The reported estimated power consumption goes from 50mW for the largest neural network to 6mW for simpler one. As the number of standard cell remains bellows 30kGates, every of these design fit in a low cost FPGA. These results have to be compared with the expected improvement coming from the proposed post-correction technique.

IV. Conclusion

We have presented a hardware implementation of an ADC error compensation scheme based on a neural network. A third step approach using a software model to compute the weight and the bias has been validated. Significant improvements on the ENOB and the distortion have been demonstrated. Moreover, the neural network as shown its capability of compensating error even where it has not been trained.
Future research will be aimed at improving the hardware implementation to be able either to reduce the power consumption or to include a larger neural network in the same space.

References

[1]Eulalia Balestrieri et al, "A State of the Art on ADC Error Compensation Methods", *IEEE Trans. Intrum. Meas.*, vol. 54, no. 4, August 2005, pp 1388-1395
[2] Xiao Hu et al, State-of-the-Art in Volterra Series Modeling for ADC nonlinearity, *Second Asia International Conference on Modeling & Simulation*, Malaysia 13 – 15 May 2008
Pasquale Daponte, Senior Member, IEEE, and Sergio Rapuano, Member, IEEE
[3] H.F. Lundin, "Characterization and correction of analog-to-digital converters*", Ph.D. Thesis, Royal Institute of Technology (KTH*), TRITA-S3-SB-0575, November 2005
[4] P. Nikaeen, and B. Murmann, "Digital Compensation of Dynamic Acquisition Errors at the Front-End of High-Performance A/D Converters", *IEEE J-STSP*, vol. 3, no. 3, June 2009
[5] A. Baccigalupi, A. Bernieri and C. Liguori, "Error Compensation of A/D Converters Using Neural Networks", *IEEE Trans. Instrum. Meas*., vol.45 no. 2, April 1996
[6] H.K. Kwan, "Simple sigm 01 D-like activation function suitable for digital hardware implementation", *Electronics letters*, Vol. 28, No. 15
[7] IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters, *IEEE Std 1241-2000*
[8] S. Crampon et al., "The 8 bits 100 MS/s Pipeline ADC for the INNOTEP Project", *TWEPP-09*, Paris