A PWL ASIC Design for Maximum Throughput

O. D. Lifschitz, P. Julián and O. Agamennoni

Abstract— **This paper presents the design of a digital architecture for a Simplicial piecewise-linear (PWL) integrated circuit (IC). This design maximizes the IC throughput by using a enhance pipeline architecture and taking advantage of the maximum memory device performance. The system latency is also minimized in order to allow feedback implementations at maximum output speed. Simulations and experimental results on a Field Programmable Gate Array (FPGA) implementation are included to showed the achieved performance.**

Index Terms— **Digital architecture, PWL, NOE**

I. Introduction

The evaluation of non-linear functions in real time is a topic of relevance in various engineering systems, ranging from biological to power networks [1]. Among the known approximations techniques for the realization of non-linear functions, PWL functions are frequently chosen due to its simplicity and approximation capabilities. There are different topologies based on the grid divisions, showed in [2] and they are hardware implementations. In this paper we will consider the simplicial PWL approach [3] because it provides an universal approximation based on a regular grid. This approach subdivides the domain into simplices and inside every simplex performs a linear interpolation [4]. From an algorithmic point of view, it is necessary to produce the sequence of vertices that identify the simplex, obtain the values of the function there (which are stored in a memory), and then perform the interpolation using the convex decomposition of the point into the set of vertices. From an architectural point of view there are two ways to calculate this algorithm. It can be calculated sequentially by ordering the values and then calculating the partial products, or in parallel through the use of more hardware to perform the necessary comparisons and sums of the terms in the minimum amount of time. Examples of the first option are [5], [6], for CNN cell arrangements. Reference [7] presented a dedicated mixed-signal integrated circuit. A dedicated IC microprocessor implementing the algorithm in a sequential way has been presented in [1]. In [8] the authors describe both alternatives and provide an FPGA implementation, although it is not completely optimized for execution speed.

In this paper we propose a 3-input digital PWL architecture that maximizes the output throughput to deal with high speed applications. In order to accomplish this, it is key to take advantage of the integrated memory where the function parameters are stored. The minimization of the system latency was also a design consideration; this is an important factor if the designed PWL function must operate in a feedback system.

O. Lifschitz and P. Julián are with the Departamento de Ingeniería Eléctrica y de Computadoras - Universidad Nacional del Sur - Conicet. Av. Alem 1253 - (B8000CPB) Baha Blanca - Argentina Email: omar.lifschitz@uns.edu.ar.

O. Agamennoni is with Investigador principal CIC (Comisión de Investigaciones Científicas).

The Simplicial PWL function is assumed to be defined over an n-dimensional compact domain S $\in \mathbf{R}^n$. Where each axis domain is partitioned into $2^{(\text{Integer bits})}$ divisions and quantized using $2^{(\text{Integer bits} + \text{Fractional bits})}$ precision. Accordingly, the number of vertices or parameters of the function is equal to:

$$\text{Parameters} = 2^{(\text{n}\times\text{Integer Bits})} = 2^{(3\times 4)} = 4096 \qquad (1)$$

For a given point x in an arbitrary simplex of the domain, the function can be expressed as follows

$$F(\tilde{x}) =\approx \sum_{k=1}^{n+1} c_k \mu_k(x) \qquad (2)$$

where $\tilde{\mathrm{x}} = [x_1, x_2, \ldots, x_n]$ is the point to be evaluated, c_k are the coefficients of the PWL function or simplex vertices, which are stored in the memory device and μ_k are scaling parameters dependent on x that form the elements of the basic function [1]. For a more complete description of the simplicial PWL algorithm, the reader is referred to [4].

The requirements, based on available typical results, can be classified in two groups: configurations and speed.

- Configuration: Dimension, size and digitalization of the PWL function must be defined. These factors are typically dictated by the application. Regarding the dimension, three input variables were chosen, as a proof of concept. The number of divisions per variable is set to 16 so we have 4 bits for the integer part. The memory coefficients are set to 8-bit precision and the fractional part is set to 10 bits [9]. Every input variable has a 14-bit size that could be fed, if needed, with an Analog to Digital commercial device.
- Speed: The design should minimize the PWL calculation time, this implies maximization of the output throughput. Also, the system latency should be minimized to allow feedback implementations, like the Nonlinear Output Error (NOE) structure described in [10]. These two features, throughput and latency are directly linked with the memory device. To maximize the memory performance the device should be accessed constantly so the latency penalty is paid once, at the beginning of the process. In the particular case under consideration, four coefficients must be read from memory, so the use of a two port memory will divide the read process time by two.

In order to meet the speed and latency requirements an internal memory must be used. External memory latency is in the order of tens of clock cycles, which is not acceptable in feedback structures. On the other hand, internal memories or on-die memories have size limitations so a compromise is needed [11]. The total memory size is equal to the number of parameters of the function, multiplied by the Word length. In the present case, this is equivalent to:

$$\text{Mem size} = \text{parameters} \times 1 \text{ Byte} = 4096 \text{ Bytes} \tag{3}$$

The outline of the paper is as follows: Section II describes the architecture; Section III introduces the IC pipeline; Section IV and V show the simulations and FPGA implementation results; and finally, Section VI presents the conclusions.

II. Architecture

In order to calculate the PWL function (2) two parameters are needed: the μ and the memory coefficients (c) of a specific simplex [4]. The μ parameters must be extracted from the fractional part of the sorted input variable. Thus a rank order is implemented using a crossbar block. This combinational crossbar block, shown in Fig.1, compares all the input variables in one clock cycle.

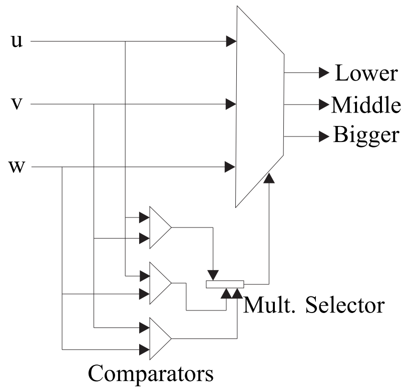

Fig. 1. Crossbar combinational implementation.

Once the inputs are sorted, the μ generator block produces the μ values and saves them in a temporary array waiting until the memory coefficients are available. The following equations show how the μ values are calculated:

$$\begin{aligned}
\mu_0 &= Fraction\ part(x_1)\\
\mu_1 &= Fraction\ part(x_2) - Fraction\ part(x_1)\\
\mu_2 &= Fraction\ part(x_3) - Fraction\ part(x_2)\\
\mu_3 &= 1 - Fraction\ part(x_3)
\end{aligned} \tag{4}$$

where x_1, x_2, x_3 are the sorted input variables. Even though these differences are combinational operations, a synchronization signal, not showed here, forces the μ reading once the crossbar has finished, in order to guarantee correct values.

Regarding the addresses, the first two addresses are instantly known once the inputs reach the PWL front-end. This is a result of the concatenation of the input variable integer part. Thus, the base and the far end addresses of the hypercube are obtained immediately. The intermediate addresses are obtained from the sorting process. The crossbar produces the ordered set of vertices, which is also known as the hypercube path (for a description of the hypercube path method, the reader is referred to [4]). Using this information the 2^{nd} and 3^{rd} addresses are formed.

The four addresses are sent to the memory control unit to activate the memory device. The memory control is a very simple state machine containing three stages. The use of a double-port memory device allows to parallelize the reading phase. The memory access is done with the first two addresses and then with the other two addresses produced by the crossbar. In the case of a single port memory, the latency of the pipeline increases by two clocks because the address is split from two to four addresses in a serial way.

To complete the PWL calculation four multiplications and three additions must be performed. Fig. 2 shows the PWL architecture with the processing blocks. One important assumption here is the one-clock multiplication capability which is enforced by design, setting the limit to the clock frequency.

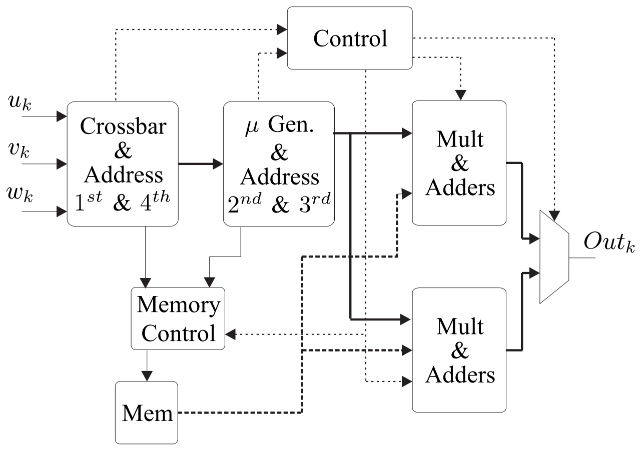
Control
u_k
v_k
w_k
Crossbar & Address 1^{st} & 4^{th}
μ Gen. & Address 2^{nd} & 3^{rd}
Mult & Adders
Mult & Adders
Out_k
Memory Control
Mem

Fig. 2. PWL Architecture showing the data path (solid lines) and control path (dashed lines). Note the two multipliers and adders to process the even and odd data.

We can observe the duplication of the multipliers and adders in order to process the PWL calculation in one clock cycle. This comes from the delay introduced by the memory until the coefficient values are ready. The crossbar and μ Generator are not duplicated, so that they are processing odd and even incoming data and passing their results to the rest of the pipeline.

A. Pipeline Design

The main goal of this work is to produce a faster architecture in terms of execution speed. Note that the approach in [8] presents two alternatives to perform the PWL calculation: a serial option and a parallel option. In the serial approach the output throughput is intentionally sacrificed in order to obtain a reduced hardware architecture. In the parallel approach, the idea is to replicate hardware units in order to speed up the execution. However, the parallel approach presented in [8] cannot operate at the maximum possible frequency because the delay of the combinational circuits limits the clock period.

The strategy we present is based on the optimization of the pipeline to produce the computation result in minimum time. In order to do this, order and access operations are adequately partitioned, while, at the same time, the memory device is read constantly, so the memory latency cost is paid once immediately after the reset sequence.

The Order and Address Generator blocks process the incoming data and produce the addresses to fill the read buffer memory constantly. Fig. 3 shows the pipeline stages and how the incoming data are processed for the different blocks.

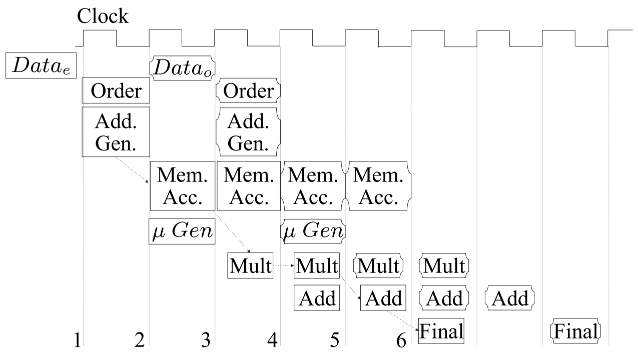

Fig. 3. Pipeline showing the different processes for every stage. The even and odd incoming data are marked with different shapes in the graph. The "final" labels mark the PWL data output which has a throughput which is half the clock system frequency.

The data is captured at the first clock. Between the first and second clock the crossbar produces its ordered output. At the second clock, the memory control fetches the first and fourth addresses. At the third clock the μ parameters are ready. At the fourth clock the following events occur: the ordered odd data input are latched, the two intermediate addresses are fetched and the first two multiplier results from the even data input are obtained. Finally, during the fifth and sixth clock cycles the four additions are done to get the first PWL calculation result, while, at the same time, the process starts for the second input data.

The memory latency is one system clock. In this way we obtain a higher PWL throughput than the one obtained in [8] which was 5Mhz and 20MHz (based on a 75 Mhz system clock) for the serial and parallel implementations. Regarding our design system latency is bigger than the one presented in [2] for the SPWL case this is a direct result of compromise between throughput efficiency and system latency.

If the number of inputs is increased, the throughput can be maintained at the expense of additional hardware. Notice that two addresses are immediately generated at the beginning of the algorithm, and the intermediate addresses must be generated by the crossbar. Therefore, if throughput is to be maintained, a multi-port memory device, multipliers and adders should be added. For an n-dimensional PWL function it is necessary to use an n-1 port device memory, n-1 multipliers and one adder of appropriate size to perform the addition in one clock cycle. The latency parameter is also kept at expense of this extra hardware. As showed in Fig. 3 after the crossbar calculations end, the n-1 memory access could be done. One clock later the n-1 multiplier are ready for activation and a final adder resolved the PWL output results. The n-1 port memory device is achieved by a custom integrated design. External memory device has a high penalty in the latency response. Also memory duplication could be done if the throughput is a main stream of the design target.

Figure 4 are simulations on Modelsim and Matlab using a PWL owner tool [3] [1]. The two models results are equal because

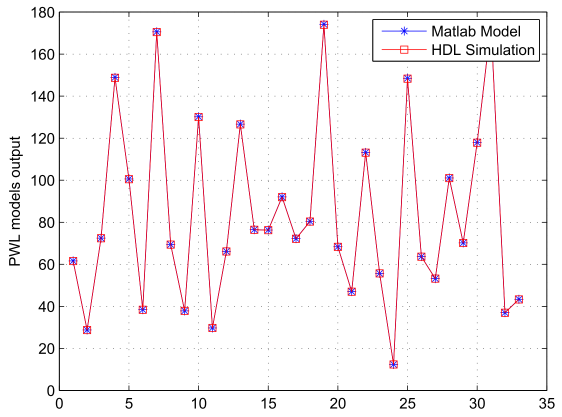

Fig. 4. Results from the HDL simulator and Matlab tool.

the quantization effects were not considered. The PWL inputs as well the memory contents are extracted to Matlab from the HDL simulator, thus the quantization effects are negligible. This is to verify the correct HDL PWL engine design.

III. FPGA IMPLEMENTATION

This section shows the implementation of the PWL algorithm in an FPGA. The FPGA device is a Virtex5 on a Xilinxs evaluation board. The number of PWL inputs are three. A Lecroy logic analyser capture (Fig. 5) was done of the PWL algorithm output running at 50MHz.

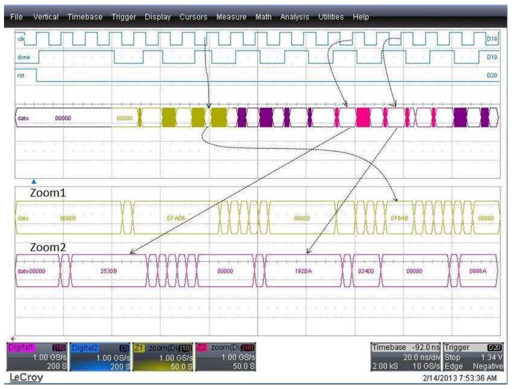

Fig. 5. PWL output capture. The bottom part shows an output zoom. We can see the output throughput is half the clock frequency.

The PWL output is marked with a Done signal which sets the synchronism for the eventual next processing block. The samples in between are the effect of the high sampling frequency of the logic analyser, 1GHz, that captures buffer transactions, due to signal integrity problems, and shows them as output signals. It can be clearly seen that the PWL architecture throughput is the clock frequency divided by two. Fig. 6 shows the laboratory environment.

[1]Site tool: http://uns.academia.edu/PedroJulian

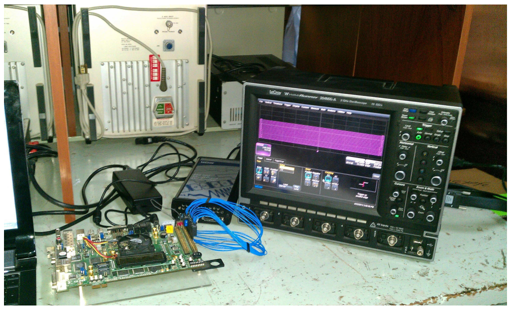

Fig. 6. Laboratory set up. FPGA and Lecroy logic analyzer.

IV. Conclusions

In this paper we have presented a dedicate architecture design to efficiently calculate a PWL function in minimum time. The architecture does not depend on the number of PWL inputs. The maximum output throughput is the system clock divided by two, a situation that can be achieved at the expense of extra hardware.

The system latency was shown to be six system clocks independently of the number of PWL inputs. Thus an improvement of the system speed of feedback implementations, like the Nonlinear Output Error (NOE) is achieved.

References

[1] A. Rodríguez, O. Lifschitz, V. Jiménez, P. Julián, O. Agamennoni, "Application-Specific Processor for Piecewise Linear Functions Computation", *IEEE Trans. Circuits Systems*, vol. 58, no. 5, pp. 971-981, 2011.

[2] F. Comaschi, B. A. G. Genuit, A. Oliveri, W. P. Maurice H. Heemels and M. Storace, "FPGA implementations of piecewise affine functions based on multi-resolution hyperrectangular partitions", *IEEE Transactions on Circuits and Systems - I*, vol. 59, no. 12, pp. 2920-2933, 2012

[3] P. Julián *Ph.D. Thesis: A high level canonical piecewise linear representation: Theory and applications*, Universidad Nacional del Sur, Bahía Blanca, Argentina, 1999

[4] M.J. Chien, E. Kuh, "Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision", *IEEE Trans. Circuits Systems*, vol. 24, no. 6, pp. 305-317, 1977.

[5] P. Julián, R. Dogaru, L.O. Chua "A piecewise-linear simplicial coupling cell for CNN gray-level image processing", *IEEE Trans. Circuits Systems*, vol. 49, no. 7, pp. 904-913, 2002.

[6] R. Dogaru, P. Julián, L.O. Chua, M. Glesner "The simplicial neural cell and its mixed-signal circuit implementation: an efficient neural-network architecture for intelligent signal processing in portable multimedia applications", *IEEE Trans. Neural Networks*, vol. 13 , no. 4 , pp. 995-1008, 2002.

[7] M. Di Federico, P. Julian T. Poggi, M. Storace, "A Simplicial PWL Integrated Circuit Realization ", *IEEE Trans. Circuits Systems*, pp. 685-688, 2007.

[8] M. Storace, T. Poggi "Digital architectures realizing piecewise-linear multivariate functions: Two FPGA implementations", *Journal of Circuit Theory and Applications*, vol. 39, no. 1, pp. 1-15, 2011.

[9] O. Lifschitz, P. Julián, O. Agamennoni "Accuracy Analysis for on-chip digital PWL realization" *in Proc. RPIC Universidad Nacional de Entre Rios, pp. 429-434, 2011*

[10] O. Lifschitz, P. Julián, O. Agamennoni, "A Generic Nonlinear Output Error Structure Implemented on a PWL ASIC", *in Proc. Arg. School of Micro Nanoelectronics, pp. 11-16, 2012.*

[11] O. Lifschitz, J. A. Rodríguez, P. Julián, O. Agamennoni, "Post-silicon Validation Procedure for a PWL ASIC Microprocessor Architecture", *IEEE Latin America Transactions*, vol. 9, no. 4, pp. 492-497, 2011.