Digital Architecture for $\mathbf{R}^6$ PWL Function Computation

Víctor Jiménez-Fernández, Juan A. Rodríguez, Pedro M. Julián, Osvaldo Agamennoni, Martín Di Federico
Universidad Nacional del Sur
Departamento de Ingeniería Eléctrica y Computadoras
Av. Alem 1353, Bahía Blanca, Argentina
E-mail:vjimenez@uns.edu.ar

***Abstract*— In this paper, we present a digital architecture for $\mathbf{R}^6$ piecewise linear (PWL) function computation. It provides an input-output relationship based on the high level canonical piecewise linear (HL-CPWL) model which defines a PWL function over a domain partitioned by simplices. In order to reduce the time to determine the set of parameters needed for the PWL function computation, a sorting algorithmic scheme has been included. Programmability has also been considered in this architecture because the PWL function is programmed in an external RAM memory.**

I. Introduction

PWL functions have proved to be a very powerful tool in the identification of nonlinear dynamical systems. One model structure for the identification of nonlinear dynamical systems is the so called NOE (Nonlinear Output Error). In reference [1] a NOE, based on the High Level Canonical Piecewise Linear (HL-CPWL) model [2], is reported. It includes an algorithm that identifies a nonlinear system starting with a linear estimation and then, iteratively updates the parameters of the HL-CPWL model. This is done by increasing grid division going from a coarse division of the simplicial partition of the domain to finer one. Figure 1 shows the block diagram of the PWL NOE. The $(\mathbf{u}, \mathbf{y})$ vectors correspond to the input and estimated-output system, and (M, N) are the model orders.

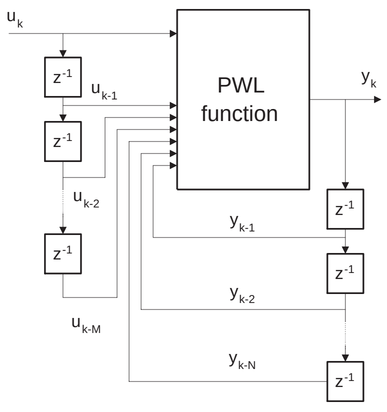

Fig. 1. PWL NOE model.

The PWL NOE is composed of two functional blocks: a PWL function block, and a set of delay blocks (described in Fig.1 as z^{-1}). This feature shows the possibility of implementing the PWL NOE in a digital integrated circuit (IC). In this paper a digital architecture for the PWL function block is presented In a recent paper [3], a IC realization for computing a PWL function has been reported. It is based on a digital architecture with a 8-bits precision. The PWL NOE model structure shown in Fig.1 has been simulated by considering 8-bit precision in the PWL function block. Simulations show the necessity of increasing numerical precision in the input data in order to get a lower error in the identification system. Figures 2 and 3 show the real output system and estimated NOE output for 8-bit and 24-bit precision.

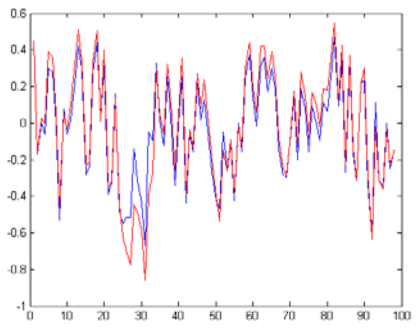

Fig. 2. Real system output(red) and estimated NOE output (blue) for 8-bit precision.

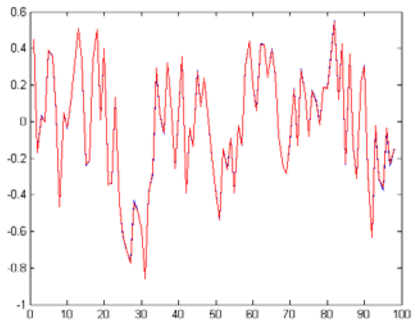

Fig. 3. Real system output(red) and estimated NOE output (blue) for 24-bit precision.

ISBN 978-987-1171-78-1

Figure 3 shows that the estimated signal presents a high error margin. The simulation, for 8-bit precision, was done by selecting the 4 most significant bits to denote the integer part, and the 4 less significant bits to represent the fractional part of every input word data. The simulation, for 24-bit precision, was done by selecting the 4 most significant bits to denote the integer part, and the 20 less significant bits to indicate the fractional part. Figure 2 shows a minimum error between the real output system and the estimated NOE output, in fact both signals look like if they were overlap. In both simulations, 8 and 24-bit, the word length that is assigned to the integer part is the same (4-bit). The difference between these simulations resides in the word length used to represent the fractional part of the input variables.

Due to the simplicial partition domain in which the HL-CPWL function is defined, the integer and fractional components of the input data are also related to the simplicial domain. For instance, the number of simplices in $\mathbf{R}^n$ PWL function is given by $2^{n\times I_N}$, where I_N indicates the number of bits used for describing the integer part of the N-bits input variable. The internal resolution of the simplex is given by 2^{F_N}, where F_N denotes the number of bits used for describing the fractional part of the N-bits input variable. In this sense, it is important to say that although a longer word (24-bits) has been used to achieve a better estimated system output, the domain of the PWL function in both cases, 8 and 24-bits, contains the same number of simplices. The best precision obtained in the NOE output (shown in Fig.3) is directly related to the 20-bits used to represents the fractional part. Although the digital architecture used in the IC of reference [3] could be scaled in bits, for the fractional part, it presents a practical shortcoming. This shortcoming is the time (measured in clock cycles) that the IC requires to compute the value of the PWL function for a given input. This time is given by

$$T_{op} = 2^{(F_N)} \tag{1}$$

Notice that while for $N = 8$, a T_{op} of $2^4 = 16$ clock cycles is required, to compute an output for $N = 24$, $2^{20} = 1048576$ clock cycles are needed for processing an input. A greater dimension (higher than 3) is also preferably in the PWL NOE in order to have better nonlinear system estimation capabilities. In this paper a digital architecture to overcome the above explained shortcomings is presented.

II. A scheme to compute HL-CPWL function

A $\mathbf{R}^n$ HL-CPWL function can be expressed as a weighted sum [5], [6], defined by

$$F(\mathbf{X}) = \sum_{i=0}^{n} c_i \mu_i \tag{2}$$

From eq.(2), we observe that in order to calculate the value of $F(\mathbf{X})$ at the input $\mathbf{X}$, c_i, and μ_i-parameters are required (for $i = 0, \cdots, n$). The set of c_i-parameters represent the value of the function at the vertices. The vertices are the boundaries of every simplex over the PWL function domain. Because the c_i values are physically stored in an external RAM memory, in order to evaluate eq.(2) the μ_i-parameters must be computed. In references [7] and [8] a proposal for computing the μ_i-parameters is presented. Such proposal consists in a set of comparators which compare the input signals ($\mathbf{X}$ vector), with a ramp. This idea was implemented in digital architecture of the IC realization of reference [3]. Due to the number of clock cycles required to compute the μ_i-parameters given by Eq.(1), such architecture works efficiently in low precision applications. However, in accordance with the simulations presented in the previous section, it was shown that a 20-bit precision for the fractional part is needed to obtain accuracy estimation results. In this section a scheme to compute the μ_i-parameters of the HL-CPWL function is presented, that is based on a sorting data procedure.

A. Scheme to compute μ_i-parameters

Let $\mathbf{X} = [x_1, x_2, \cdots, x_n]$ be a n-dimensional input to $\mathbf{R}^n$ HL-CPWL function. Each x_j-variable (for $j = 1, 2, \cdots, n$) is assumed to be composed by integer and fractional part expressed by the notation $x_j = x_{int_j}.x_{frac_j}$.

Now let $\mathbf{X}_{sorted} = [x_{s_1}, x_{s_2}, \cdots, x_{s_n}]$ be a vector which includes the x_{frac_j} elements sorted by the relation: $x_{s_1} \leq x_{s_2} \leq \cdots \leq x_{s_n}$.

The μ_i-parameters can be computed as follows:

$$\mu_n = x_{s_1} \tag{3}$$

$$\mu_{(n-\ell)} = x_{s_{(\ell+1)}} - x_{s_\ell} \tag{4}$$

$$\mu_0 = 1 - x_{s_n} \tag{5}$$

for $\ell = 1, 2, \cdots, n-1$

It is important to point out that the elements of $\mathbf{X}_{sorted}$ vector are simply the sorted elements of $\mathbf{X}$ vector.

B. c_i addressing procedure

Equation (2) also include the c_i-parameters. Although this set of parameters do not need to be computed, it is required a procedure for addressing the external RAM in order to select the c_i which corresponds to any specific μ_i parameter. Data and address in a RAM memory are expressed as binary numbers, therefore the addressing procedure must also be defined in binary format.

Let $V = \langle x_{int_1} x_{int_2} \cdots x_{int_n} \rangle$ be a binary number formed by the concatenation of the integer part of all x_i input variables, where i follows the sequence $i = \{1, 2, \cdots, n\}$.

Consider that $S = \langle S_1 S_2 \cdots S_n \rangle$ is a binary number (with the same word length as x_{int_1}) composed by the concatenation of n S_i-terms which value could be either $\langle 00 \cdots 00 \rangle$ or $\langle 00 \cdots 01 \rangle$.

Also let $\Lambda = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$ be a sequence which indicates the consecutive order in which the i-th element of $\mathbf{X}$ appears in $\mathbf{X}_{sorted}$ (running from the left-most to the right-most element). The address memory, for a c_i-parameter which corresponds to any specific μ_i, is obtained by the following procedure:

```
turn-on $S_j$, for $j = \{1, 2, \cdots, n\}$
for $k$ from 0 to $n$ do
    begin
        $DIR_k = V + S$
        turn-off $S_{\sigma_k}$
    end
```

The notations **turn-on** and **turn-off** are used to indicate the process of setting $S_j = \langle 00 \cdots 01 \rangle$ and $S_j = \langle 00 \cdots 00 \rangle$, respectively.
It is important to mention that the set of $(n+1)$ RAM addressing done to pick up a specific c_i-term constitute a path in a n-dimensional hypercube.
Finally, the addition of the $c_i \mu_i$ terms (for $i = 1, 2, \cdots, n$) constitute the value of the PWL function.

C. Example

In order to illustrate the addressing procedure, let us consider a two-dimensional example. Consider the continuous PWL function, depicted in Fig.4, $F(x_1, x_2)$ which is defined over a simplicial domain partitioned into four simplices with a unitary grid step.

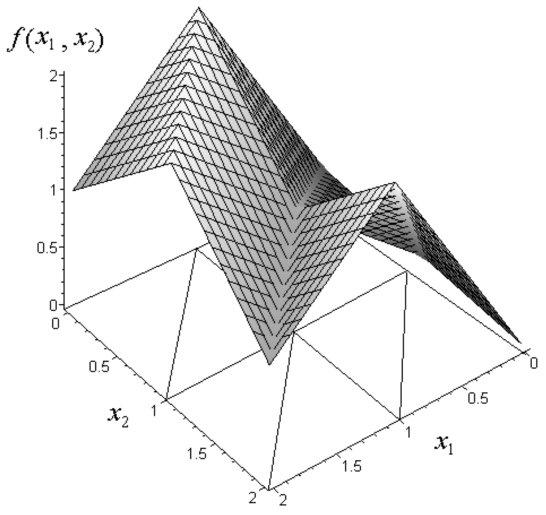

Fig. 4. A two-dimensional PWL function.

The value of the PWL function, $c_i = F(x_1, x_2)$ at the vertex points, is collected and stored into the RAM memory as it is summarized in TABLE I.

TABLE I

i	Vertex	Memory Dir.	$\mathbf{c}_i = F(\mathbf{x_1}, \mathbf{x_2})$
0	$(0,0)$	00000000	0
1	$(0,1)$	00000001	0
2	$(0,2)$	00000010	0
3	$(1,0)$	00010000	2
4	$(1,1)$	00010001	1
5	$(1,2)$	00010010	2
6	$(2,0)$	00100000	1
7	$(2,1)$	00100001	2
8	$(2,2)$	00100010	1

In this example, the evaluation of an arbitrary input point, for example, the point $\mathbf{x} = (1.5, 0.75)$ at the function $F(x_1, x_2)$ is obtained.
Observe that $n = 2$, and the following data are obtained:
$\mathbf{X} = [x_1, x_2] = [1.50, 0.75]$
$\mathbf{X}_{sorted} = [x_{frac_1}, x_{frac_2}] = [0.50, 0.75]$
(notice that $x_{frac_1} \leq x_{frac_2}$ is fullfil)
$\Lambda = \{1, 2\}$
$\mu_2 = x_{frac_1} = 0.5$
$\mu_1 = x_{frac_2} - x_{frac_1} = 0.25$
$\mu_0 = 1 - x_{frac_2} = 0.25$

Consider a 8-bits word length ($N = 8$) with 4-bits for representing integer and fractional parts, respectively ($I_N = F_N = 4$).
$V = \langle 00010000 \rangle$, and in accordance with the addressing procedure, the first value for S is obtained when all the S_j values are set in the turn-on state ($S = \langle 00010001 \rangle$). As $DIR_0 = V + S$, then $DIR_0 = \langle 00100001 \rangle$

The second value for S is obtained when S_1 is set in the turn-off state ($S = \langle 00010000 \rangle$). Because $DIR_1 = V + S$, then $DIR_1 = \langle 00100000 \rangle$.

The third, and last S-value for $n = 2$, is obtained when S_2 is set in the turn-off state ($S = \langle 00000000 \rangle$). It results that $DIR_2 = V + S$ and therefore $DIR_2 = \langle 00010000 \rangle$.

Finally, the c_i values stored in the RAM memory at the addresses: DIR_0, DIR_1, and DIR_2, are multiplied and added in order to compute the value of the HL-CPWL function at the input $\mathbf{X}$, as follows:

$$F(x_1, x_2) = 0.5 \times 2 + 0.25 \times 1 + 0.25 \times 2 = 1.75$$

III. The Proposed Architecture

In this section a digital architecture to implement the procedure for computing $\mathbf{R}^6$ HL-CPWL function is presented. Input data x_i is 24-bit precision ($N = 24$), 4 bits are used to represent the integer part ($I_N = 4$) and 20 bits are assigned to the fractional part ($F_N = 20$). The proposed architecture is based on a microprocessor scheme which includes an ALU (Arithmetic Logic Unit), a register file and a control unit.

A. Description

Figure 5 shows the block scheme of the proposed architecture.

1) *The register file* is composed by a set of $(F_N + 3)$-bit registers. The F_N-bit field in Reg.1-Reg.6 store the fractional part of the input data (x_{frac_i}) and the 3-bit field is used to indicate the i-th position of x_i in the input vector $\mathbf{X}$.
2) *Accumulator register Acc* is a $(F_N + 8)$-bit auxiliar register which accumulates the value of $F(\mathbf{X})$.

3) *Counter CNT* is a 3-bit counter which assigns to each input data (x_i) an index $i = \{1, 2, \cdots, 6\}$. This index is needed to identify the i-th data in the sorting process.
4) *Vertex register VRT* is a register where the interger part of the input data is collected. Because there are 6 x_i-inputs, and 4 bits has been used to represent the integer part of every input, then it is 24-bit register. This register is used to address memory.
5) *Temporal registers Reg.A and Reg.B* are registers which store temporary data for the ALU inputs. Reg.A is a $(F_N + 3)$-bit register and Reg.B a $(F_N + 8)$-bit register.
6) *Output register Rout* is a $(F_N + 8)$-bit register which stores the ALU results.
7) *Label register RSX* is a 24-bit register where S_i addressing terms are stored.
8) *Register SR38* is a (3×8)-bit shift register. The 24-bit inputs x_i (for $i = 1, 2, \cdots, 6$) are sequentially loaded in three 8-bit blocks. While the 4 most significant bits of x_i are stored in VRT, the 20 less significant bits are stored into the register file.
9) *ALU* is an Arithmetic Logic Unit which performs the comparison, additions, subtraction, and multiplication operations.
10) *MUX, Selector*, are multiplexors and data path selector, respectively.

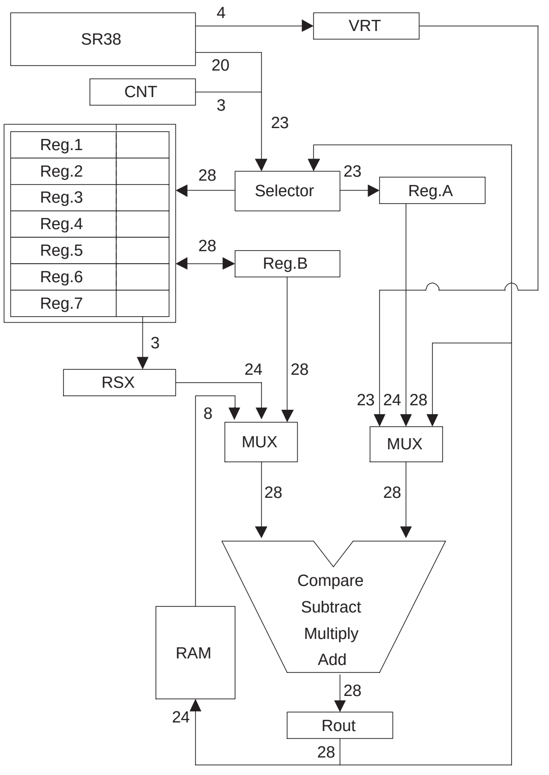

Fig. 5. Block scheme of the proposed architecture.

IV. Architecture

In the proposed PWL architecture, the processing of $F(\mathbf{X})$ is given as follows: firstly, the input data (x_i for $i = 1, 2, \cdots n$) is loaded, then the x_{frac_i} terms are sorted. Next, the μ_i-parameters are computed, c_i-values are read and finally $F(\mathbf{X})$ is computed. This processing-cycle can be divided into the followig stages:

- *Data input* is a sequential process; x_i enter to the system by the following order: $x_1, x_2, x_3, \ldots x_6$. Each x_i is 24-bit wide. Due to limitations in the number of I/O pins, the input process is performed in three cycles of 8 bits for each x_i. After loading SR38, the 4 most significant bits are written into VRT and the less significant 20-bits are written into Reg.i; the value of CNT is written into the 3 less significant bits of Reg.i. CNT is incremented in the first of the 3 cycles.
- *Sorting*. The sorting procedure performs 12 comparisons and switching operations. Each one of this operations is executed in 3 cycles: (1) parallel-loading of Reg.i into Reg.A and Reg.j into Reg.B , (2) compare, (3) if Reg.A>Reg.B then parallel-write Reg.A into Reg.j and Reg.B into Reg.i (switch).
- *Evaluation*. Evaluate the summatory Eq.(2). It is computed as follows:

```
1) Load Reg.1 into Reg.A ($\mu_1$)
2) Add VRT and RSX (address computation for $c_1$)
3) Read memory (address in Rout)
4) Multiply Reg.A and memory[Rout] ($\mu_1 \times c_1$)
5) Write Rout into Reg.7 (accumulation)
6) set = 0 RSX[index(Reg.1)]
   (for next $c_i$'s address computation)
      for i from 2 to 6 do
         begin
            Load Reg.i into Reg.A
            Load Reg.(i-1) into Reg.B
            Subtract Reg.A - Reg.B
            ($\mu_i$, is always $>$ 0 because of
            the previous sorting procedure)
            Load Rout into Reg.A
            Load Reg.7 into Reg.B
            ($\mu_i$ is saved in Reg.A and the partial
            result is loaded in Reg.B )
            Add VRT and RSX
            (address computation for $c_i$)
            Read memory (address in Rout)
            Multiply Reg.A and memory[Rout]
             ($\mu_i \times c_i$)
            Add Rout and Reg.B
             ($\mu_i \times c_i + Acc$)
            Write Rout into Reg.7 (accumulation)
            set = 0 RSX[index(Reg.i)]
            (for next $c_i$'s address computation)
         end
7) Load Reg.6 into Reg.B and set Reg.A = 0 (1-$\mu_6$)
8) Subtract Reg.A - Reg.B
```

(μ_7, always is > 0 because of the previous sorting procedure)

9) Load Rout into Reg.A and Reg.7 into Reg.B
(μ_7 is saved in Reg.A and the partial result is loaded in Reg.B)
10) Add VRT and RSX
(address computation for c_7)
11) Read memory (address in Rout)
12) Multiply Reg.A and memory[Rout]
($\mu_7 \times c_7$)
13) Add Rout and Reg.B
($\mu_7 \times c_7 + Acc$)
14) Write Rout into Reg.7
(accumulation)

Note: In the subtract operation $1 - \mu_6$ of (7), the integer part can be omitted because the result will always be greater than 0; then, only the fractional part is required.

V. Sorting input data procedure

A procedure to compute μ_i-parameters was presented in section II. This procedure requires x_{frac_i} to be sorted. Sorting networks are the most popular method to sort data in VLSI implementations [9]. They are based on compare-switch operations which are performed in parallel. Figure 6 shows the diagram for the Batcher's "odd-even sorter", it is the most used sorting network.

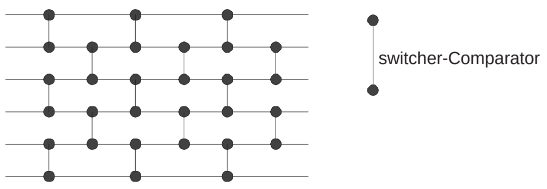

Fig. 6. Batcher's sorting network diagram.

The main advantage of this kind of network is its regular structure; although it is not optimal in the number of comparisons, the execution of this comparisons is done in parallel, so the execution time to sort data is the same as in other optimal irregular strategies like Bose-Nelson's. Figure 7 shows the diagram for the Bose-Nelson sorting network.

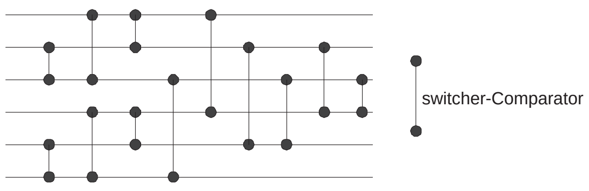

Fig. 7. Bose-Nelson sorting network diagram.

Although a compare-switch strategy is here used, the comparator network is not physically implemented. The sorting procedure is considered as a stage of the HL-CPWL computation process, and it is performed as a sequence of compare-switch operations. In order to justify the sorting method here proposed, a comparative analysis with the Batcher's sorter is presented. Let $\{r_1, r_2, r_3, r_4, r_5, r_6\}$ the set of registers to be sorted.

Batcher's implementation for 6 *variables*: 6 read/write ports, 120-bit bus (20 for each x_{frac_i}) and 15 switcher-comparator blocks are required. Each computation cycle consists of $(2 or 3)$ comparisons and $(2 or 3)$ switch executed in parallel; so the execution time T_1 for 6 stages is given by

$$T_1 = 6 \times (t_{comparator} + t_{switcher})$$

Proposed sorting implementation: 2 read/write ports, only one comparator block and 2 temporal registers are required. An optimal sequence of compare-switch operations is done (12 operations for 6 variables). In this sequence each (x, y) represents a compare-switch operation of x, and y; this sequence is the same as Bose-Nelson's and it is given by

$$(r_2, r_3), (r_5, r_6), (r_1, r_3), (r_4, r_6), (r_1, r_2), (r_4, r_5),$$
$$(r_3, r_6), (r_1, r_4), (r_2, r_5), (r_3, r_5), (r_2, r_4), (r_3, r_4)$$

The switch operation is done by writing back the temporal registers into the file register. Each computation cycle performs two write operations in parallel into temporal registers, one comparison operation and two write operations in parallel into the file register; so execution time T_2 is given by

$$T_2 = 12 \times (t_{write_{in}} + t_{comparator} + t_{write_out})$$

On the one hand, it is important to notice that the odd-even sorter is a combinational circuit; assuming that $t_{comparator}$ depends on the number of bits and that $t_{switcher}$ is constant (equivalent to compare one bit), let t be the time to compare 1 bit of the 20-bit x_{frac_i}-word. Time T_1 is given by

$$T_1 = 6 \times (20 \times t + t) = 126t.$$

On the other hand, the new circuit is synchronous; then the computing time depends to the following restriction: the clock cycle must be as long as the longest operation. Similarly, assuming that $t_{comparator}$ depends on the word length and t_{write} is constant, then it yields

$$T_2 = 12 \times (3 \times max(20t, t, t) = 12 \times (3 \times 20t) = 720t$$

Considering that this circuit performs more than a 6-inputs sorting and that sorting time is about 22-percent of total execution time, the speedup obtained using a sorting network (in accordance with Amdahl's law) is given by

$$SPU = \frac{1}{\left[(1 - 0.22) + \frac{0.22}{(T_2/T_1)}\right]} = 1.22$$

Although the above speedup is not negligible, a minimum area and a most simple design is desirable.

VI. Execution Time Analysis

The objective of defining a new architecture is to overcome the existing execution time limitations in the IC of reference [3] when it is scaled to $F_N = 20$ bits. In this section, a comparative analysis of execution time of both circuit architectures is presented; the execution time depend on the number of bits assigned to F_N and on the number of dimensions of $F(\mathbf{X})$. The circuit of [3] is hereafter referred as circuit A and the new circuit as circuit B.
Let M be the dimension of $F(\mathbf{X})$ and let N be the word length of x_{frac_i}.
Circuit A computes $F(\mathbf{X})$ executing the next cycle of operations:

```
For i from $0$ to $2^N$ do
    begin
        (A1)Increment counter C
        (n bit counter)
        (A2)Compare C with $M$ inputs
         ($N$-bit $x_{frac_i}$ )
        (A3)Compute $c_i$ address
         (12 bits add)
        (A4)Read $c_i$ from memory
        (A5)Add $c_i$ and accumulator
         (8+N bits add)
    end
Divide by $2 \times N$ (shift right accumulator $N$ bits)
```

By extending circuit A for N-bit inputs, it would be necessary to extend the counter (ramp-generator) and the comparator to N-bits, also the adder should be extended to $8 + N$-bit because it will add 2^N 8-bit numbers. It can be seen than execution times for A1, A2, A3, is linearly depend on N, so they are $O(N)$; A2 does not depend on M because comparison operations are executed in parallel. Otherwise, A3 and A4 do not depend on M and N, so their time is constant, so it is $O(1)$. In conclusion, the total execution time for one iteration is $O(3N+2) = O(N)$. Finally, as the cycle executes 2^N times, the total execution time for circuit A is $O(2^N \times N)$.
Circuit B computes $F(\mathbf{X})$ using a completely different strategy; it sorts the x_{frac_i}-inputs and then computes the summatory. It operates as follows:

```
(B1) Sort $M$ $N$-bit numbers using switch-compare
     operations
For i from $1$ to $M + 1$ do
    begin
        (B2) Compute $\mu_i = x_{frac_i} - x_{frac_{i-1}}$
        (if $i = 1$, $x_i - x_{frac_1} = 0$);
        (if $i = M + 1$, $x_{frac_i} = 1$)
        (B3) Compute $c_i$'s address
        (B4) Read $c_i$ from memory
        (B5) Multiply $c_i \times \mu_i = f_i$
        (B6) Add $f_i$ to Acc
    end
```

B1 sorts M inputs of N bits. Instead of using the optimal sequence proposed in section V, this analysis is done by using a standard sorting method which is scalable and does not require any additional structures, for example Bubble Sort. This kind of sorting method will require M^2 operations. Taking into account that data to be compared is 20 bits word lenght, the time to execute B1 will be of $O(M^2 \times N)$. Arithmetic operations in B2, B3 and B6 are $O(N)$, B4 is $O(1)$; in contrast, B5 executes in a time of $O(N^2)$ for a standard implementation. In this way, the execution time for each cycle is $O(N^2 + 3 \times N + 1) = O(N^2)$; so the total time for the iterative cycle is $O(M \times N^2)$. Finally, the total time for circuit B is $O(M^2 \times N + M \times N^2)$.

VII. Conclusion

A new digital architecture for computing HL-CPWL function with 24-bit precision was presented. The proposed architecture overcomes the shortcomings due to long time execution. The asymptotic time analysis has proved the advantages of this new implementation when precision is increased. While the IC implementation presented in [3] exponentially depends on precision, the architecture presented in this paper presents a polynomial dependency.

VIII. Aknowledgment

This work was partially funded by project PICT 2003 No.13468, and PICT No.14628 of ANPCyT.
Juan Agustín Rodríguez has a (Type I) grant from CONICET.
P. Julián is member of *Consejo Nacional de Investigaciones Científicas y Técnicas* CONICET", Av. Rivadavia 1517, Buenos Aires, Argentina.

References

[1] L. Castro, J. Figueroa, O. Agamennoni, *"BIBO stability for NOE model structure using HL CPWL functions"*, IEEE Modelling Identification and Control Meeting. MIC 05. IASTED, Proceedings of MIC 05, Innsbruck IASTED, 2005.
[2] P. Julian, *"A high level canonical piecewise-Linear representation: theory and applications"*, Doctoral Thesis, Universidad Nacional del Sur, Argentina, 1998.
[3] M. Di Federico, P. Julián, T. Poggi, and M. Storace, *"A simplicial PWL integrated circuit realization"*, IEEE International Symposium on Circuits and Systems ISCAS-2007, New Orleans, U.S.A., May 2007.
[4] L. Cervantes, O. Agamennoni, J. Figueroa, *"Robust identification of PWL-Wiener models: use in model predictive control"*, IEEE Latin American Applied Research journal, vol. 33, no.4, pp. 435-442. ISSN 0327-0793 ,2003.
[5] P. Julián, A. Desages, and B. D'Amico, *"Orthonormal high-level canonical PWL functions with applications to model reduction"*, IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications, vol.47, pp. 702-712, May 2000. .
[6] P. Julián and O. Agamennoni, *"High-level canonical piecewise linear representation using a simplicial partition"*, IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications, vol.46, pp. 463-480, April 1999.
[7] P. Julián, R. Dogaru, and L. Chua, *"A piecewise-linear simplicial coupling cell for CNN gray-level image processing"*, IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications, vol.49, pp. 904-913, July 2002.
[8] P. Mandolesi, P. Julián, and A. Andreou, *"A scalable and programmable simplicial CNN digital pixel processor architecture"*, IEEE Transactions on Circuits and Systems-I: Regular papers, vol.51, pp. 988-996, May 2004.
[9] D. Knuth, *The Art of Computer Programming,*, Ed. Addison Wesley, VOL.3, 1998.