Application-Specific Processor for Piecewise Linear Functions Computation

J. Agustín Rodríguez, Omar D. Lifschitz, Víctor Manuel Jiménez-Fernández, Pedro Julián, *Senior Member, IEEE*, and Osvaldo Enrique Agamennoni

***Abstract*—This paper presents an application specific processor architecture for the calculation of simplicial piecewise linear functions of up to six dimensions with 24-bit wide input words. The architecture, in particular registers and bus connections, is specifically designed for the task of simplicial piecewise linear computation. The parameters of the function are stored in an external 16 MB RAM memory. A proof-of-concept integrated circuit (that achieved first silicon success) was fabricated through MOSIS in a $4\ \mathrm{mm} \times 4\ \mathrm{mm}$ $0.5\ \mu\mathrm{m}$ standard CMOS process using an automated design flow based on Synopsys and Cadence tools and the OSU standard cell library.**

***Index Terms*—Application specific, function evaluation, microprocessor architecture, piecewise linear, VLSI.**

I. Introduction

PIECEWISE linear (PWL) functions have been widely used for the last four decades in the areas of circuits, systems, and control, mainly because they allow to represent multivariate functions with reduced complexity. In some cases, they are used to make a large problem numerically tractable [1], while in others, they allow numerically complex algorithms to be run in real time [2] or at high speeds [3].

There are several areas where PWL functions have produced successful implementations. The first and most traditional area has been the computationally efficient resolution of nonlinear circuits (see [4]–[7]). One dimensional PWL functions are used at present for efficient realization of elementary nonlinear functions: multipartite and bipartite methods are one example (see [8] and [2]). These methods make an extremely efficient use of the memory where the function parameters are stored [9], although they not always take advantage of the continuity of the function to further reduce the number of parameters and achieve a "canonical" representation [10]. Direct Look-up Table (LUT) methods and LUT with linear interpolations are both PWL functions. Direct LUT do not perform interpolation and store all data in memory, while LUT with interpolation, reduce the data in the memory and include an interpolation algorithm. Quadratic interpolation methods have been proposed in order to reduce even further the size of the memory [11], although the extension to higher dimensional domains results too expensive from a numerical viewpoint. These implementations are intensively used in Graphic Processor Units (GPU), Digital Signal Processors (DSP), and microprocessors for fast and accurate calculation of one-dimensional functions [11].

In communication systems, baseband digital predistortion [12] allows good linearity and power efficiency, by previoulsy compensating the nonlinearities of the power amplifier. This implies the use of linear filters and nonlinear static operations, most of them based on LUT [13], explicit PWL functions [14]–[17], and piecewise linear networks [18]. Reference [19] presents a dedicated VLSI that implements a predistorter based on a PWL approximated Gaussian function.[1]

The introduction of nonlinear control and estimation strategies in power electronics applications, demands the evaluation of nonlinear functions at speeds of tens and hundreds of Mhz. Digital audio amplifiers and A/D converters require nonlinear operations for predistorsion [14], [20]. Feedback control of dc converters lie in the same category; the dc–dc converter reported in [22] uses a Look-up Table (LUT) approach for the high speed implementation of a nonlinear PID control. Control and estimation of variables in motors require also a considerable volume of static (memoryless) computation [23], [36], [37], and they justify the use of extra memory in the LUT approach to achieve the necessary performance; indeed, the application described in [24] uses a 64 Mb DRAM.

All these applications show the current and future relevance of multidimensional PWL functions. Motivated by this, in this paper we propose an Application Specific Processor (ASP) for PWL function computation.[2] ASP combine high flexibility with hardware resources that optimize the execution of the target application; GPU, DSP, and special purpose micro-controllers are some of the most popular realizations of this design philosophy. In particular, we use the simplicial PWL expression introduced

Manuscript received March, 16 2010; revised August, 23 2010; accepted October 12, 2010. Date of publication December 20, 2010; date of current version April 27, 2011. This work was supported in part by PICT 2006–1864 , PICT 2006–1835, and PAE 37079 of Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT) of the Argentine Ministry of Science and Technology (MINCYT). This paper was recommended by Associate Editor S. Cotofana.

J. A. Rodríguez, O. Lifschitz, and P. Julián are with the Departamento de Ingeniería Eléctrica y Computadoras, and Instituto de Investigaciones en Ingeniería Eléctrica, Universidad Nacional del Sur/CONICET, 8000 Bahía Blanca, Argentina
(e-mail: pjulian@ieee.org).

V. M. Jiménez-Fernández is with the Facultad de Instrumentación Electrónica, Universidad Veracruzana, Xalapa, Veracruz, México.

O. Agamennoni is with the Departamento de Ingeniería Eléctrica y Computadoras, and Instituto de Investigaciones en Ingeniería Eléctrica, Universidad Nacional del Sur/CONICET, 8000 Bahía Blanca, Argentina, and also with the Comisión de Investigaciones Científicas de la Pcia. 1900 Buenos Aires, CIC, Argentina.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2010.2091196

[1] In these applications it is also desirable for the nonlinear block to be able to adapt over time [12]

[2] Preliminary work describing the architecture with simulations was presented in [25]

in [26], [27], which is general with respect to the domain dimension, and can be used as a generic platform for nonlinear dynamic system representations [28], [30], [31]. The processor runs a sorting algorithm to perform simplex identification [6], and retrieves the parameters from an external memory to perform the linear interpolation. A Harvard coprocessor architecture is used together with a master system that loads the program memory with the execution code and also sends different processing requests. The microprocessor is of a RISC type, with a microprogrammed control unit; an on-chip memory stores the 256 (micro) program words. A particular characteristic of this design is that the micro program memory is actually the complete program memory space.

A proof-of-concept 4 mm × 4 mm integrated circuit (IC) was fabricated in the MOSIS 0.5 μm technology, using Synopsys tools and the Oklahoma State University (OSU) standard cell library. The main objective of the IC is to produce a first generic platform, with an optimized architecture (within the limits imposed by the maximum available area of $16\ \mathrm{mm}^2$) in order to perform experiments at the system level. The design sets the foundations for a high performance nonlinear multidimensional PWL-based ASP.

The IC was designed to perform a PWL calculation of a six dimensional input vector ($x \in \mathbf{R}^6$), where every component is represented with 24 bits; in addition, the number of inputs can be reconfigured anywhere between one and six. A friendly and extensible instruction set was developed to program the PWL microprocessor. In addition, a complete software environment was developed to load programs and also to send processing requests to the chip. Thorough debugging tests shows the correct behavior of the IC for the specified operating conditions (55 Mhz at VDD $=$ 3.3 V).

II. Background and Preliminaries

The Simplicial PWL representation proposed in [26] and [27] and the decomposition procedure proposed in [6] for an n-dimensional compact domain $S \in \mathbf{R}^n$ provide the processor computation platform. In a given simplex, any PWL function $F : \mathbf{R}^n \mapsto \mathbf{R}^1$ can be expressed as a weighted sum

$$F(\mathbf{x}) = \sum_{i=0}^{n} \mu_i c_i \tag{1}$$

where $\mathbf{x} = [x_1, \ldots, x_n]$ is the point to be evaluated, μ_i are scaling parameters dependent on $\mathbf{x}$, and c_i are the values of the function at the simplex vertices, $i = 0, \ldots, n$.

For a given $\mathbf{x}$, the values c_i are known—they are the function parameters—but the μ_i-values, associated to the simplicial decomposition, must be computed. This association, hereinafter denoted as vertex addressing, is defined by the hypercube path [32], and can be obtained by the simplicial decomposition of $\mathbf{x}$ as reported in [6]. In fact, a point in a simplex can be decomposed as $\mathbf{x} = \mathbf{x}_{\mathrm{int}} + \mathbf{x}_{\mathrm{frac}}$, with $\mathbf{x}_{\mathrm{frac}}$ defined as follows:

$$\mathbf{x}_{\mathrm{frac}} = \sum_{i=0}^{n} \mu_i \mathbf{v}_i \tag{2}$$

where $\mathbf{v}_i$ are the simplex vertices.

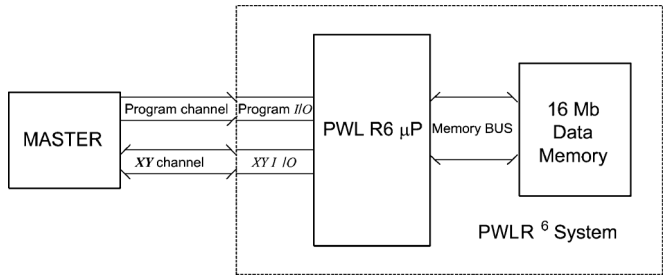

Fig. 1. PWLR6 computation system.

Algorithmic Scheme for Evaluation of $F(\mathbf{x})$

The evaluation scheme of $F(\mathbf{x})$ is based on a sorting procedure to compute the μ_i parameters, and requires a binary-format vertex addressing procedure. The algorithm can be summarized as follows:

1) **Data Input and Decomposition**: Let $\mathbf{x} \in \mathbf{R}^n$ be the input vector. Each x_j-component (for $j = 1, 2, \ldots, n$) is decomposed into integer and fractional part, using the notation: $x_j = x_{\mathrm{int}_j}.x_{\mathrm{frac}_j}$.
2) **Sorting**: Let $\mathbf{x}_{\mathrm{sorted}} = [x_{s_1}, x_{s_2}, \ldots, x_{s_n}]$ be a vector that includes the x_{frac_j} elements sorted by the relation: $x_{s_1} \leq x_{s_2} \leq \cdots \leq x_{s_n}$.
3) $F(\mathbf{x})$ **evaluation**: **μ-Computation and Vertex addressing**: The μ_i-parameters can be computed as follows:

$$\begin{aligned} \mu_0 &= x_{s_1} \; for \; i = 1, 2, \ldots, n-1 \\ \mu_i &= x_{s_{i+1}} - x_{s_i} \\ \mu_n &= 1 - x_{s_n}. \end{aligned} \tag{3}$$

The c_i values are physically stored in a RAM; the following procedure addresses the memory in order to fetch the c_i values that correspond to a specific μ_i parameter:

a) Let $V = \langle x_{\mathrm{int}_n} \cdots x_{\mathrm{int}_2} x_{\mathrm{int}_1} \rangle$ be a binary number formed by the concatenation of the integer part of all x_j input variables, where $j = \{1, 2, \ldots, n\}$.
b) Let $S = \langle S_n \cdots S_2 S_1 \rangle$ be a binary number (with the same word length as V) composed by the concatenation of n S_j-terms (each of them with the same length as x_{int_j}) whose value can be either $\langle 0 \cdots 00 \rangle$ or $\langle 0 \cdots 01 \rangle$.
c) Let $\Lambda = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ be a sequence indicating the order of the components of $\mathbf{x}$ in $\mathbf{x}_{\mathrm{sorted}}$ (i.e., if $\mathbf{x}_{\mathrm{sorted}} = [x_{\mathrm{frac}_3}, x_{\mathrm{frac}_1}, x_{\mathrm{frac}_2}]$ then $\Lambda = \{3, 1, 2\}$). The address memory of a c_i value corresponding to a specific μ_i is obtained by the following procedure:

turn-on S_j, for $j = \{1, 2, \ldots, n\}$
for $k = 0$ to n
  $ADD_k = V + S$
  turn-off $S_{\sigma_{k+1}}$

The notations **turn-on** and **turn-off** are used to indicate the process of setting $S_j = \langle 0 \cdots 01 \rangle$ and $S_j = \langle 0 \cdots 00 \rangle$, respectively. It is important to note that the set of $(n + 1)$ RAM addresses generated to fetch the c_i-terms constitute the path in the

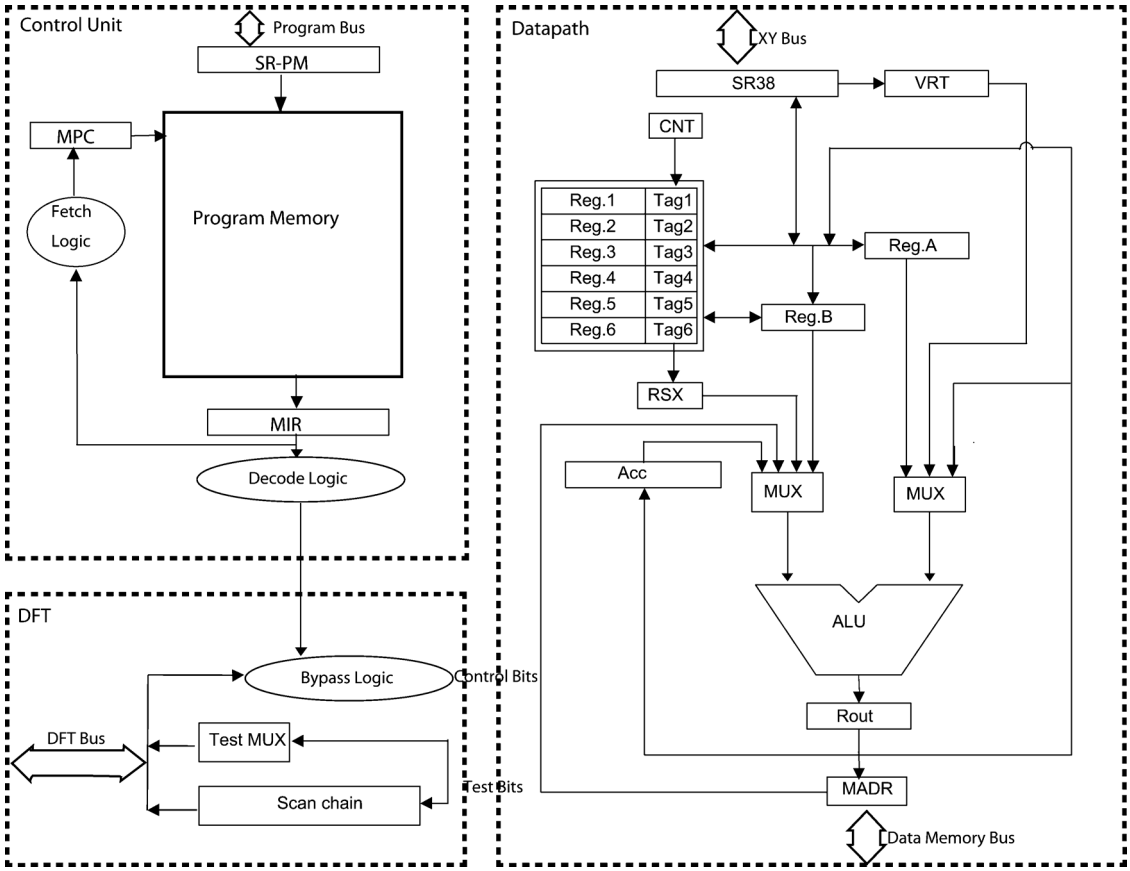

Fig. 2. Microarchitecture of the PWL-μP.

n-dimensional hypercube. Finally, (1) can be evaluated in accordance with the c_i-μ_i correspondence given by the hypercube path.

III. The PWL-μP Architecture

The digital architecture of an ASP strongly depends on its target application. A successful ASP provides the required hardware to solve the target set of computational problems in an optimized way in terms of execution time, power, or chip resources, while maintaining the flexibility and programmability characteristics of a general purpose microprocessor. A trade-off exists among optimization levels and flexibility levels; thus, an ASP can be considered as an intermediate point between a general purpose microprocessor and an ASIC. The three main architectural blocks of the PWL microprocessor, namely, Data Path, I/O, and Control, were designed taking into account the special operations required to perform the PWL calculation. The result is a nearly basic microprocessor with special features that accelerate the PWL computation. This section presents the details of these blocks as well as an overview of the system operation, the memory configuration and an instruction level analysis of the calculation cycle.

Data Path is the most influenced block by the special characteristics and mechanics of the PWL computation. Due to this strong dependence, data path was the first block to be designed. After getting an adequate Data Path design, I/O structures and protocols were defined to supply the required data for computation. Finally, the Control Unit was designed to sequence PWL-μP's operations. The resulting Control and Instruction set are closely related to the attained flexibility and performance.

A. Operation

The PWL-μP is designed as a Harvard coprocessor architecture intended to work with a master system that: 1) loads the PWL-μP's program memory with a data RAM configuration program or an evaluation program for $\mathbf{R}^1 \cdots \mathbf{R}^6$; 2) sends different processing requests (see Fig. 1). Once the PWL function parameters c_i are stored in the data RAM and the evaluation program is loaded, the system is ready to compute the processing requests. The master inputs a value $\mathbf{x}$, the PWL-μP computes the function's value for this point and produces an output.

B. Data Path

The mechanism of the PWL function evaluation algorithm has a direct impact on the Data Path architecture, which is shown in Fig. 2. Although it includes typical blocks like a register file and an ALU, the architecture has been tailored for PWL computations by especially designing word lengths, interconnection buses, and addressing hardware. As was previously described, the input data has to be decomposed into integer and fractional parts. This is implemented using on-line processing with minimum hardware overhead: I/O register SR-38 (3×8-bit shift register) receives a 24-bit value corresponding to one variable x_i; the integer part is concatenated in register SR-VRT (6×4-bit shift register) and the fractional part is stored in the register file. Associated to each fractional part,

an additional 3-bit tag is stored; this tag is generated with CNT (3-bit counter) and is incremented each time an x_i arrives to be used as the index required in the vertex addressing procedure. This digital structure optimizes the execution time for data input compared to a standard microprocessor, where input values have to be first stored into registers before they can be decomposed. In a standard microprocessor, the decomposition has to be done with shift operations while concatenation of the six integer parts can be done with shift-and-add operations. Although CNT tags result in a nonstandard register file, they save operations, and the register file can still be used as a standard register.

Sorting constitutes the second part of the algorithm (it is required to evaluate the μ_i parameters), and was implemented following the comparison sequence of a Bose–Nelson sorting network. In order to maintain the microprocessor structure and to avoid the area overhead of 12 compare-switch blocks, only one comparator (the one included in the ALU) was used. The RF (register file) is composed of 6 registers (20-bit wide plus 3 bits for CNT tags) and has two bidirectional ports. Port A is connected to Register A and port B to Register B; transfers between RF and Registers A and B can be done in parallel. Both Register A and Register B are possible ALU inputs. The compare operation is performed by these registers and depending on the result Register A and Register B values may be written back into the RF, by switching sources and destinations. CNT tags are not part of the fractional part, therefore they are not used in the comparison, but if a register switch occurs, tags need to be switched with their corresponding fractional parts. That is the reason why bidirectional ports Register A and Register B are 23-bit wide but ALU inputs connected to Register A and Register B are only 20-bit wide.

The third and last step is evaluation. For each term of the sum (2), c_i must be addressed and read from the data memory and μ_i must be computed. The result of multiplying these two parameters has to be accumulated into a partial sum result. Again, special features accelerate this computation. The key structures are the connection buses between registers, the ALU inputs and the addressing structures.[3]

With two special purpose registers, the PWL-μP has a simple and efficient addressing method to retrieve the c_i values from memory. SR-VRT stores the concatenated integer parts of x_i values and plays the role of the binary number V (introduced in Section II-A); Register RSX is a 6-bit register, which plays the role of binary number S, and can be operated bit-wise.[4] As shown in Fig. 2, both SR-VRT and RSX can be added directly (they are ALU inputs) to generate the c_i's address; after the memory access operation, the c_i value is stored in Register MADR (Memory Address Data Register). The CNT tags are used to perform the turn-off operations described in Section II-A; this operation is supported by a 3-bit bus from the RF tags to a decoder that selects the RSX bit to be turned off; then, the RSX update depends on the selected RF register and its tag. These features clearly differentiate the PWL-μP from a typical microprocessor architecture providing an efficient address generation. After c_i has been read from data memory, the next step is to compute μ_i by subtracting sequentially x_i components of $\mathbf{x}_{\text{sorted}}$. Sorting has already taken place, so RF stores x_i in ascending order. The μ_i parameters are then computed by subtracting sequentially the RF registers (for example, μ_1 requires the computation of $\mathrm{RF}[2]-\mathrm{RF}[1]$). Finally, the Register MADR (c_i) value is multiplied with the Register ALU-out (μ_i) value (both can be simultaneous ALU inputs). After that, the Register ACC value that stores the partial sum evaluation is added to Register ALU-out (again, both can be simultaneous ALU inputs) and the ith sum term is ready. After repeating the preceding operations $n+1$ times, the value of $F(\mathbf{x})$ is obtained, in accordance with (1).

C. External Memory

The data RAM space depends on the PWL function's dimension n, and on the granularity or number of divisions per domain of the simplicial partition, namely G, so that $RAM space = G^n$. The RAM space is exponentially dependent on the number of dimensions; although the theory behind PWL functions is valid for n-dimensional functions, memory space is obviously restricted. Setting the maximum dimension equal to six gives enough room for a large set of scientific/engineering problems (see Section IV). A minimum number of partitions per domain must be set in order to guarantee enough resolution for function approximation; setting this value equal to 16 resulted in a 16 MB RAM. When the dimension of the PWL function is reduced, a finer granularity is obtained as a result of the flexibility provided by the PWLR6 architecture. If $n = 1, 2, 3$ the six 4-bit registers in SR-VRT are grouped into 1, 2, or 3 groups of 24, 12, or 8 bits, producing $G = 16M$, 4096, and 256 divisions, respectively. For the remaining two cases, $n = 4$ and $n = 5$, register S cannot be matched to a regrouping of SR-VRT and the grid must be fixed at 16 subdivisions. In all cases, the fractional part is 20-bit wide.

D. Input-Output

Communication with the PWL-μP is done by three blocks: 1) XY port; 2) Data memory subsystem; 3) Program port. Each of these blocks communicates by using a set of bits for data transmission and another for synchronization.

1) XY port: This port is used to transmit 8/16/24-bit data to and from the master system. An 8-bit bidirectional bus, a 3×8-bit shift register (SR-38), signals "require-bit" and "acknowledge-bit" define an asynchronous communication channel. In the PWLR6 system, the master sends the 24-bit x_i input values and receives the 28-bit output through the XY port (in both cases, using 3 or 4 send-receive transactions). The PWL parameters c_i are also sent using this port.
2) Data memory subsystem: As was previously mentioned, the data memory subsystem addresses a 16 MBytes RAM, which requires a 24-bit address. Due to pin count limitation, the memory addressing procedure is multiplexed using a similar structure to Intel's 8086. An internal 16-bit Data Address Memory Register (MADR) outputs the 24-bit address in two cycles, the 16-bit lower part is stored

[3]Different configurations provide different advantages; even though the proposed architecture is efficient, it could be further optimized

[4]The binary number S is 24-bit wide, then every bit of RSX is concatenated with a sequence "000" to generate S.

TABLE I
INSTRUCTION SET

Opcode	Mnemonic	Description
0	DIR_CTRL	Registers control (LOAD, STORE, MOV)
1	VSI	Enable/shift for VRT and SR38
2	ALU_OPS	ALU (CMP, ADD, SUB, MUX)
3	LRD	Register direct load
4	MADR_LAML	MADR=Aluout[15:0]
5	MADR_LAMH	MADR[7:0]=Aluout[23:16] & Set ALE
6	SMB	Set ACK, EnMem or ReadMem bits
7	MADR_IN	Data input in MADR higher byte
8	JMP_RELB	Jump relative backward
9	JMP_RELF	Jump relative forward
A	SRA	Register RSX and ACC control
B	NOP	No operation
C	JMP_ABS	Jump absolute
D	MADR_OUT	Data output in MADR higher byte
E	SR48_CTRL	SR48 output register enable
F	SR48_CTRL	SR48 output register shift

in an external register (ALE) and the 8-bit higher part is stored in the MADR's lower byte.

3) Program port: This port is used to load a program into the PWL-μP internal program memory. A bit line and a clock line establish a synchronous communication channel. 20-bit program words are stored in a 20-bit shift register that is used during the programming as input data for the internal program RAM. Each word needs 21 clock cycles to be loaded: 20 cycles for data transfer and an additional cycle to store it. RAM internal control signals are generated during this last cycle. Simplicity and minimum pin count was preferred for this I/O instead of high performance.[5]

E. Control Unit

The control unit evolved form a simple nearly-counter structure to a micro-programmed control unit that implements the PWL-μP instruction set architecture (ISA). Control signals were encoded in 20-bit instructions. The ISA was designed to exploit all the resources inside the PWL-μP; it includes absolute and relative jumps, memory RD/WR instructions, ALU operations, and dedicated registers instructions (see Table I). Opcode "0000" represents all possible register manipulation operations among RF, Register ALU-out and registers A and B. The programmer sets directly the control signals that configure write/read operations on these registers. In other words, each bit of the instruction word is a Read or Write enable of one register. Allowing the programmer to move data between registers in arbitrary ways gives a flexible albeit insecure environment; i.e., simultaneous accesses can be done, but simultaneous writings on the same bus will produce a contention between outputs; logic constraints were included to avoid problematic control configurations.

The remaining instructions are structured and more constrained. The final control unit has a 256×20-bit RAM that stores 256 (micro) instructions. As was mentioned earlier, this memory is actually the program memory and no addressing capabilities are provided for external program memory; although it seems a small program space it is enough for storing the PWL evaluation programs and other micro-programs like the "configure data RAM program"; the compact representation for this computation is consequence of the data path special features that reduce the number of instructions compared to an x86 approach.

The control unit fetches (micro) instructions from on-chip program memory and decodes them to produce the control bits. Program memory is addressed with an MPC (Micro Program Counter) and program words are stored in an MIR (Micro Instruction Register). Decode logic uses MIR data to evaluate the data path's control bits. A final characteristic that is worth mentioning is that the PWL-μP is a pipelined architecture. Fetch/Decode/Execute cycles are performed in parallel; dedicated logic was designed in order to insert wait cycles for jump instructions. Pipelining was used for the fetch, decode, and execution stages, but no pipelining was used for the ALU.

F. Instruction Level Analysis of the Calculation Cycle

In the description of the architecture the characteristics that speed up the execution of the PWL function computation were presented. In this subsection the calculation cycle and its three stages (data input, sorting, and evaluation) are analyzed at the instruction level with the objective of identifying the fundamental gains resulting from the special resources. A comparison between the PWL-μP and a standard RISC (ARM) in terms of number of instructions is presented.[6] Since memory and I/O structures differ substantially (i.e., ARM uses a hierarchical memory cache & main memory and the PWL-μP uses only a main memory), they are not part of this comparison.

Figs. 3(a) and 4(a) show the Data Input code using PWL-μP and ARM's assembler respectively; the convenience of the VSI instruction and its two associated registers can be observed. The statements "Run XY protocol for x" and "Load xi into R10" represent the required steps to input the x_i values.

Figs. 3(b) and 4(b) show a comparison of the code required for sorting using PWL-μP and ARM's assembler. The sorting sequence requires 12 compare and switch operations. Depending on whether the compared inputs have to be switched or not, 2 or 3 instructions are needed for the PWL-μP and 3 or 6 for the ARM. In the worst case, this sequence requires the execution of 36 and 72 instructions, respectively.

Figs. 3(c) and 4(c) compare PWL and ARM assembler codes for the evaluation step. Seven cumulative products are required and each cumulative product requires one memory address generation and access to retrieve the parameter c_i. Register manipulation and arithmetic operations are similar; however, while the standard RISC needs to implement a case statement to actualize the offset (r8) for vertex addressing, the PWL-μP does it in just one instruction. The "Read Value from Data Memory into MADR" and "LDRB" are the memory access operations and are not being considered.

Table II summarizes the instruction count of the PWL microprocessor in comparison with the ARM code. Data input shows a relationship of 4.6 times. In the case of sorting, three values

[5]Even though this mechanism is not efficient in terms of computing time, it was maintained for simplicity, due to the fact that program loading is not part of the PWL function evaluation algorithm. Indeed, the system was conceptually designed to load an evaluation program and execute it repeatedly

[6]Since both architectures are RISCs this comparison can be considered in terms of clock cycles, however given that stall cycles due to conditional jumps are not taken into account, the comparison will be kept at the instruction count.

a - PWLup – data input

```
"Run XY protocol for x1"
VSI RESR38 R1
"Run XY protocol for x2"
VSI RESR38 R2
"Run XY protocol for x3"
VSI RESR38 R3
"Run XY protocol for x4"
VSI RESR38 R4
"Run XY protocol for x5"
VSI RESR38 R5
"Run XY protocol for x6"
VSI RESR38 R6
```

b - PWLuP – 3 data sort

```
LOADAB R1 R2
JMPF NZ 1
STOREAB R2 R1
LOADAB R2 R3
JMPF NZ 1
STOREAB R3 R2
LOADAB R1 R2
JMPF NZ 1
STOREAB R2 R1
```

c - PWLuP – 2 term sum

```
ADD VRT RSX
"Read from Data Memory into MADR"
MOV REGA R1
MUX REGA MADR
SRA AC 1

ADD VRT RSX
"Read from Data Memory" into MADR"
LOADAB R2 R1
SUB REGA REGB
MUX ALUOUT MADR
ADD ALUOUT ACC
SRA AC 2
```

Fig. 3. PWL code for three different operations. (a) Data input. (b) Data sorting (showing only three terms). (c) Evaluation (considering the first two terms of the summation).

are shown corresponding to the best case (data are ordered and no switches are needed), the average case (six switches), and the worst case (twelve switches). The evaluation cycle requires one initial computation, six computation steps and six addressing steps. In the PWL case these three steps require $3+6\times5+6\times1$ cycles. In the ARM case these three steps require $4+6\times7+6\times6$ cycles.

Considering an average case, the PWL instruction count is $(6+30+39)=75$ and the ARM instruction count is $(28+54+82)=164$, so the average relative gain is 2.18. It is important to notice that instruction count cannot be considered directly as a performance measure. However, if the same technology node is used for both our ASP and a standard RISC, the delay in their logic circuits (registers and arithmetic blocks) will be similar resulting in a similar maximum clock frequency. In that case the instruction count can be used as a performance indicator.

IV. Accuracy Analysis

The result $F(\mathbf{x})$ produced by the processor is obtained as a linear combination of memory values (8 bits) times the μ's coefficients (20 bits), so that $F(\mathbf{x})$ is a 28 bits value. As the proposed PWL processor is digital and there are no truncations during the computation, the processor as an isolated digital computing unit does not incur in calculation errors. However, when the processor approximates an analog function, three different types of errors appear [8]: a) PWL approximation errors; b) range quantization errors; and c) domain quantization errors.

If the PWL function is the approximation of a nonlinear function, there is an initial error produced by the approximation. This is an approximation problem which has been treated previously in [27], where the grid size needs to be designed small enough in order to achieve a desired accuracy.[7] However, the PWL function can also be designed from the scratch, without a preexisting analytical nonlinear function. One famous example is the well

[7]If the resulting grid size is too small, the chip will not be able to implement the function. In this case, the domain must be reduced or a chip with more subdivisions must be designed

a - ARM – data input

```
"Load x1 into R10"
MOV  r7 , r10 , LSR #20
SUB  r1 , r10 , r7 , LSL #20
ADD  r1 , r1, #0x00100000
"Load x2 into R10"
MOV  r7 , r7 , LSL #4
MOV  r8 , r10 , LSR #20
ADD  r7 , r7 , r8
SUB  r2 , r10 , r8 , LSL #20
ADD  r2 , r2, #0x00200000
"Load x3 into R10"
MOV  r7 , r7 , LSL #4
MOV  r8 , r10 , LSR #20
ADD  r7 , r7 , r8
SUB  r3 , r10 , r8 , LSL #20
ADD  r3 , r3, #0x00300000
"Load x4 into R10"
MOV  r7 , r7 , LSL #4
MOV  r8 , r10 , LSR #20
ADD  r7 , r7 , r8
SUB  r4 , r10 , r8 , LSL #20
ADD  r4 , r4, #0x00400000
"Load x5 into R10"
MOV  r7 , r7 , LSL #4
MOV  r8 , r10 , LSR #20
ADD  r7 , r7 , r8
SUB  r5 , r10 , r8 , LSL #20
ADD  r5 , r5, #0x00500000
"Load x6 into R10"
MOV  r7 , r7 , LSL #4
MOV  r8 , r10 , LSR #20
ADD  r7 , r7 , r8
SUB  r6 , r10 , r8 , LSL #20
ADD  r6 , r6, #0x00600000
```

b - ARM – 3 data sort

```
    MOV r8 , r2, LSL #12
    CMP r8 , r1, LSL #12
    BHI s1
    MOV r8 ,r1
    MOV r1, r2
    MOV r2 ,r8
s1 MOV r8 , r3, LSL #12
    CMP r8 , r2, LSL #12
    BHI s2
    MOV r8 ,r2
    MOV r2, r3
    MOV r3 ,r8
s2 MOV r8 , r2, LSL #12
    CMP r8 , r1,  LSL #12
    BHI s3
    MOV r8 ,r1
    MOV r1, r2
    MOV r2 ,r8
s3
```

c - ARM – 2 term sum

```
t1 LDRB r9 , [r7 , r8]
   MOV r11 , r1 , LSL #12
   MOV r11 , r11 , LSR #12
   MUL r10 , r9, r11
   MOV r12 , r10 ;acc

   MOV r10, pc
   MOV r11 , r2 , LSR #20
   ADD r10, r10 , r11,  LSL  #3
   BX  r10
a1 SUB r8 , r8 , #0x00100000
   B endcasea
a2 SUB r8 , r8 , #0x00010000
   B endcasea
a3 SUB r8 , r8 , #0x00001000
   B endcasea
a4 SUB r8 , r8 , #0x00000100
   B endcasea
a5 SUB r8 , r8 , #0x00000010
   B endcasea
a6 SUB r8 , r8 , #0x00000001
endcasea

t2 LDRB    r9 , [r7 , r8]
   MOV r14 , r1 , LSL #12
   MOV r10 , r2 , LSL #12
   MOV r10 , r10 , LSR #12
   SUB r11 , r10 , r14, LSR #12
   MUL r10 , r9 , r11
   ADD r12 , r12, r10 ;acc

   MOV r10, pc
   MOV r11 , r2 , LSR #20
   ADD r10, r10 , r11,  LSL  #3
   BX  r10
b1 SUB r8 , r8 , #0x00100000
   B endcaseb
b2 SUB r8 , r8 , #0x00010000
   B endcaseb
b3 SUB r8 , r8 , #0x00001000
   B endcaseb
b4 SUB r8 , r8 , #0x00000100
   B endcaseb
b5 SUB r8 , r8 , #0x00000010
   B endcaseb
b6 SUB r8 , r8 , #0x00000001
endcaseb
```

Fig. 4. ARM code for three different operations. (a) Data input. (b) Data sorting (showing only three terms). (c) Evaluation (considering the first two terms of the summation).

TABLE II
Number of Operations and Relative Gains

Stage	PWL	ARM	Relative Gain
Data input	6	28	4.66
Sorting	24-30-36	36-54-72	1.5-1.8-2
Calculation	39	82	2.1

known Chua's circuit with 3 segments $(G=3)$ [33] and 5 segments $(G=5)$ [34]. Another example is when the PWL function is obtained directly from measured data ([14] implements a correction function in $\mathbf{R}^1$ for an A/D converter calibration with $G=2$ subdivisions per axis; [35] uses a LUT with no interpolation in $\mathbf{R}^3$ with $G=[128,28,256]$ for image halftoning).

The PWL function is usually specified with infinite precision, so that when the function is to be implemented by the processor, the two other sources of errors appear: the coefficient quantization error is due to the fact that the parameters c_i must be stored in a digital memory of finite word length; the input quantization error is due to the fact that the input vectors $\mathbf{x}$ to the function must also be converted into digital numbers of finite

word length. Some reported applications that fall under this category and the grid size used, are the following: [1] implements a magnetic positioning system in $\mathbf{R}^3$ with $G = 30$ subdivisions per axis; [36] implements a nonlinear FIR filter to compensate power amplifiers in $\mathbf{R}^3$ with $G = 128$ subdivisions per axis; [37] implements a real time switched reluctance motor control in $\mathbf{R}^2$ with $G = [115, 14]$ subdivisions (LUT with no interpolation); [22] implements a dc–dc PID controller in $\mathbf{R}^4$ with $G = [9, 9, 9, 1]$ subdivisions; [23] implements an observer-controller pair that can be implemented using several $\mathbf{R}^3$ functions with $G = 16$ subdivisions per axis. [24] presents a similar application in $\mathbf{R}^3$ functions with $G = 120$ subdivisions per axis; [30] is able to reproduce complex nonlinear phenomena in dynamical systems of order three, using an $\mathbf{R}^4$ PWL function with $G = [7, 7, 7, 10]$ subdivisions; [38] uses an approximation in $\mathbf{R}^3$ with $G = [14, 26, 8]$ for an all-region MOS transistor drain current accurate model implemented in SPICE3F5; [39] uses a LUT with no interpolation in $\mathbf{R}^3$ with $G = [16, 16, 4]$ for the analysis of a $\Sigma - \Delta$ ADC. It is interesting to note that the designed chip (even when it was meant to be a proof-of-concept) can fit all cited applications.

Let T denote the word length of every component of input vector $\mathbf{x}$, N the integer word length and M the fractional word length, in such a way that $T = N + M$. As explained in the previous section, $M = 20$ is fixed. In addition, let Q denote the c_i parameter (memory) word length. Accordingly, there are 2^N subdivisions per axis, and every axis subdivision is quantized into 2^M possible locations for linear interpolation. Without loss of generality, range and domain of the function will be normalized to the sets $[0, 1]$ and $[0, 1]^n$ respectively.

Firstly, the effect of coefficient quantization is analyzed. Every coefficient has to be rounded to a number of the form $\tilde{c}_i = k_i \times 2^{-Q}$, with $k_i = 0, 1 \ldots Q$, and as a consequence the quantization error is bounded by

$$e_i = |c_i - \tilde{c}_i| \leq \frac{1}{2} \times 2^{-Q} = 2^{-(Q+1)}. \tag{4}$$

As shown in [26], the quantized PWL function $\tilde{f}$ can be upper and lower bounded by two functions, f_u and f_l respectively, in such a way that $f_l \leq \tilde{f} \leq f_u$ where f_l is the PWL function defined by coefficients $\tilde{c}_i - e_i$, for all i, and f_u is the PWL function defined by coefficients $\tilde{c}_i + e_i$, for all i. In other words,

$$\sum_{i=0}^{n} \mu_i (c_i - e_i) < \sum_{i=0}^{n} \mu_i c_i < \sum_{i=0}^{n} \mu_i (c_i + e_i) \tag{5}$$

which, after considering that $\sum_{i=0}^{n} \mu_i = 1$, and some algebraic manipulation results in

$$f_l = \sum_{i=0}^{n} \mu_i c_i - 2^{-(Q+1)} < f = \sum_{i=0}^{n} \mu_i c_i < f_u = \sum_{i=0}^{n} \mu_i c_i + 2^{-(Q+1)}. \tag{6}$$

As the distance between $\tilde{f}$ and f is bounded by (4), then f satisfies $f_l \leq f \leq f_u$, so that

$$e_C(\mathbf{x}) = |\tilde{f}(\mathbf{x}) - f(\mathbf{x})| \leq 2^{-(Q+1)} \tag{7}$$

for all $\mathbf{x} \in S$.

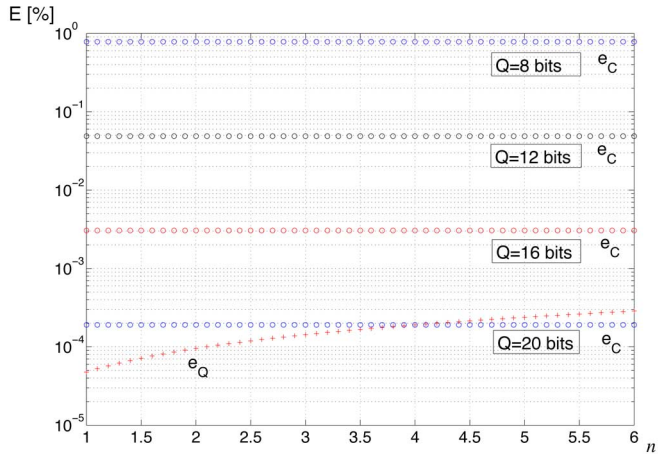

Fig. 5. Relative percentage error due to coefficient quantization e_C and input quantization e_Q.

Secondly, the effect of input quantization is analyzed. Assuming that the function range is $[0, 1]$, every coordinate of the quantized input is of the form $\tilde{x}_i = k_i \times 2^{-(N+20)}$, with $k_i = 0, 1, \ldots, 2^{N+20}$. The ith input coordinate can be written as $x_i = \tilde{x_i} + \Delta x_i$, where $|\Delta x_i| \leq 2^{-(N+1+20)}$. Inside an arbitrary simplex, the PWL function can be written as $f(\mathbf{x}) = f(\tilde{\mathbf{x}}) + (\partial f/\partial \mathbf{x})\Delta \mathbf{x}$, and the error can be bounded as follows: $e = |f(\mathbf{x}) - f(\tilde{\mathbf{x}})| \leq \|(\partial f/\partial \mathbf{x})\| \|\Delta x\|$.

Considering that f is PWL with range bounded to $[0, 1]$, and the width of any simplex is 2^{-N},

$$\left| \frac{\partial f}{\partial x_i} \right| \leq \frac{1}{2^{-N}} \tag{8}$$

and $\|\partial f/\partial \mathbf{x}\|_2 \leq \sqrt{n} \times 2^N$. On the other hand, since $|\Delta x_i| \leq 2^{-(N+21)}$, then $\|\Delta \mathbf{x}\| \leq \sqrt{n} \times 2^{-(N+21)}$. The total input quantization error can be bounded as follows:

$$e_Q(\mathbf{x}) = |f(\mathbf{x}) - f(\tilde{\mathbf{x}})| \leq n \times 2^{-21} \tag{9}$$

where $\mathbf{x}$ is a vector belonging to the simplex under analysis.

Assuming that both errors are independent, the total error is given by

$$e(\mathbf{x}) \leq \sqrt{2^{-2(Q+1)} + n^2 \times 2^{-2\times 21}} \tag{10}$$

for every $x \in S$. For the case where $N = 4$ and $M = 20$, $e_C \leq 0.19\%$, $e_Q \leq 2.86 \times 10^{-4}\%$ and $e \leq 0.19\%$. Fig. 5 shows the relative coefficient quantization error e_C, for different word lengths of the memory ($Q = 8, 12, 16, 20$) and the input quantization error e_Q as a function of the input dimension n. For values of memory word length less than 20 bits, the error is dominated by the memory word length. When the memory word length is 20 bits, for $n = 4$ both errors are approximately equal; for $n > 4$ the input quantization dominates, and for $n < 4$ coefficient quantization dominates. In our implementation $Q = 8$ bits, so the error is dominated by the memory word length, and equals 0.19%.

V. Physical Implementation and Experimental Results

A. IC Synthesis

The PWL-μP logic and physical implementation was produced with a standard cell VLSI design flow, based on the AMI 05 OSU cell library. The flow consisted of logic synthesis with Synopsys Design Compiler; formal verification with Synopsys Formality; Place and route with Cadence's Encounter; and finally, layout with Cadence Virtuoso Layout Editor and the NCSU technology library. A postlayout fast simulation was performed based on a 20 instruction program case that uses buses, ALU, and registers intensively. In all cases, correct operation was predicted up to a frequency of 55 MHz under nominal work conditions with VDD $= 3.3$ V.

B. Testing Environment

A software environment was built to work with the system and debug it. The user interface is written in Matlab to provide a friendly environment to the user; this code communicates serially (RS232) with a Spartan-3 FPGA. The FPGA contains an 8-bit soft-core PicoBlaze controller that handles the serial communication, translates the Matlab messages, and implements a set of state machines that depending on the commands sent from the user interface (UI) perform one the following actions: a) program the PWL-μP internal RAM; b) sets the PWL-μP in execution mode and sends the required input data; c) generates the PWL clock; d) generates the debugging signals of DFT blocks. In addition, a dedicated compiler allows the use of the instruction set using the mnemonic codes (see Table I).

C. DFT Architecture

A set of clock, data, and control signals for DFT provides an adequate environment to verify the PWL-μP's architecture. If an error occurs, these features can be combined to identify it. A set of 16×6 static registers are connected using a transmission gate MUX to 6 output pins for testing, as illustrated in Fig. 6 (see Appendix for an explanation of the signals). Elaborated verification procedures may be set up: for instance, get the state of the PWL-μP clock by clock or generate assertions for different events. The basic DFT routines are the following: a) Stop_Clk: Halts the PWL-μP's clock depending on a configured counter or a flag; b) Do_1_Clk: Triggers a clock pulse allowing the execution of PWL-μP's routines step by step; c) Mux: Multiplexes different internal data signals—this DFT works "on the fly," sending out values while the chip is running; d) Data_Scan: Reads the complete state of the PWL-μP and outputs it serially; e) Control_Bypass: Disables control signals generated by the Control Unit and replaces them by an outside generated control word.

D. Experimental Results

The PWL-μP's IC was fabricated through the MOSIS service, in a 4 mm $\times$ 4 mm 0.5 μm standard CMOS process (see Fig. 7); the IC transistor count is 150 000. The IC was placed on a two layer PCB that includes connectors to a logic analyzer (HP 54620A), an oscilloscope (Agilent DSO 3062A) and multimeter (HP34401A) as shown in Fig. 8. Input data vectors generated in a computer were sent to the chip, and the outputs were analyzed

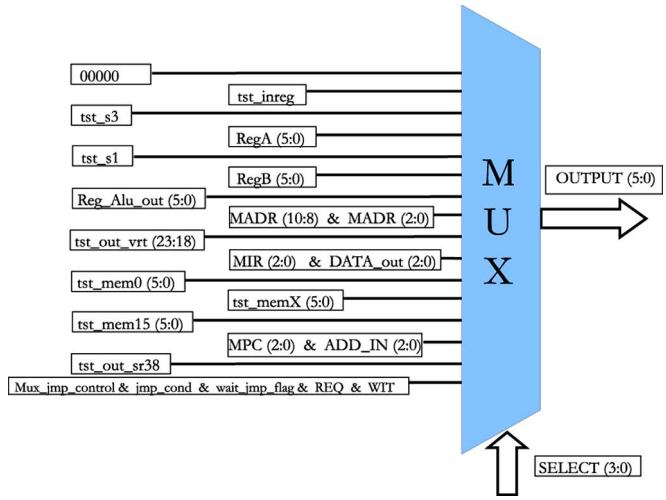

Fig. 6. DFT MUX and corresponding signals.

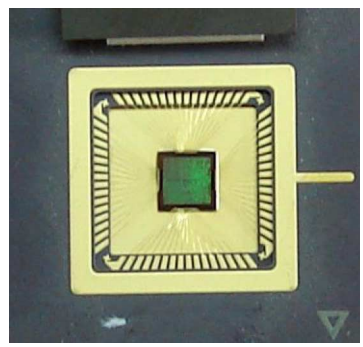

Fig. 7. Photograph of the chip.

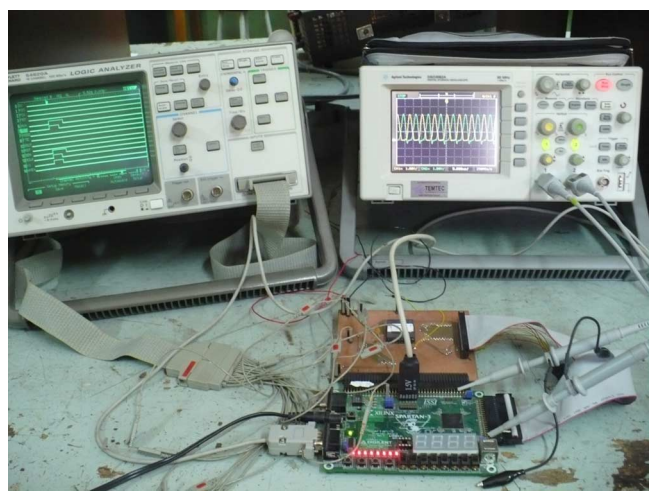

Fig. 8. Experimental setup including the IC dedicated board, the FPGA board, and the instruments during debugging.

in Matlab. In all cases, the behavior of the chip was correct. Internal states were thoroughly tested using the available DFT, and all measurements coincided with the expected results.

In addition, frequency measurements were done for a maximum VDD $= 3.8$ V. Measurements were done by fixing a clock frequency and reducing VDD until a failure occurred. The test program, in this case, included ISA instructions that forced the IC longest path. Operational failure was considered when the calculated output was wrong, despite of the functional state of the chip. Frequency set up was done using the DCM block in

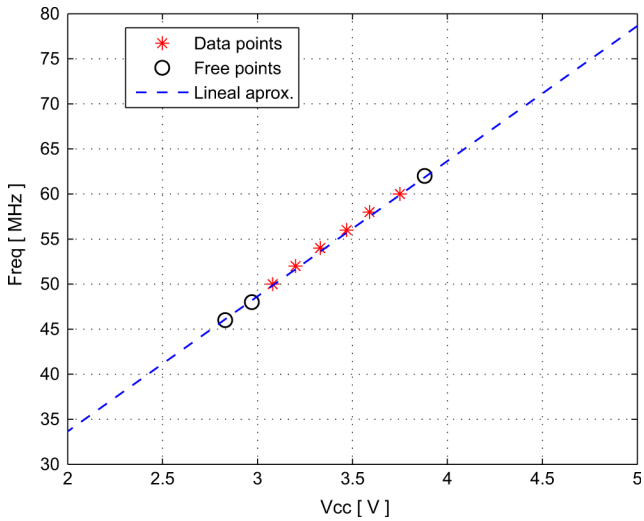

Fig. 9. Freq. versus VDD: The "data points" are the operating points from experimental results; the "free points" are linearly extrapolated points.

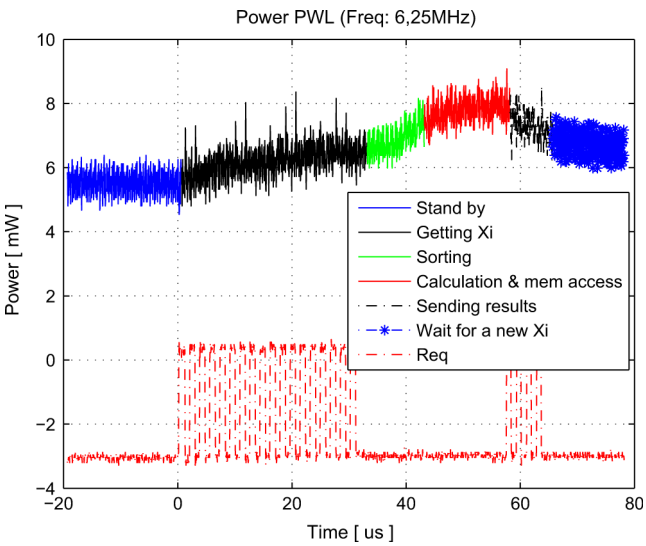

Fig. 10. Power activity (signal REQ—not on scale—is included as a time reference.

the FPGA. Fig. 9 shows the maximum operational frequency at each measured VDD value.[8]

Regarding power consumption, three components were measured: static, clock tree, and dynamic. Static consumption was measured with reset asserted and clock stopped giving a value of 160 nW. Clock tree consumption was measured with reset still asserted and clock running. Dynamic consumption was determined while the PWL-μP was executing a program. A digital scope (Agilent DSO3062A) and a multimeter (Hewlett Packard—HP34401A) were used for this measurement. Dynamic power is shown in the captured scope picture of Fig. 10 while a test is running. The Req (voltage) signal is part of the XY port protocol and is shown just as a timing reference.

[8]Design Compiler predicted a maximum operating frequency of 54 MHz at VDD $= 3.3$ V, which is remarkably close to the experimental result.

TABLE III
POWER @3.3 V.

PWL	Power [mW]		
Freq. [MHz]	Clock Tree	P. Virus	P. Virus + I/O
50	44.64	67.71	107.33
40	33.64	54.57	87.53
25	22.70	34.91	49.68
12.5	11.42	17.60	24.87
6.25	5.69	8.75	12.40

In addition, a "Power Virus" (power-worst-case) test program[9] was developed to measure dynamic power for different clock frequencies. Maximum power consumption occurs in the "calculate and memory access" step; then, the "Power Virus" program loops executing ALU and memory access operations. This program was also used to compare I/O power against internal logic by including or not I/O instructions. Table III summarizes the results of these tests. Clock tree power is also included.

VI. CONCLUSIONS

A microprocessor architecture for the computation of simplicial piecewise linear functions has been presented. A proof-of-concept fully functional 4 mm $\times$ 4 mm IC prototype in a 0.5 μm standard CMOS process, together with a complete environment that allows the user to program the IC and the parameter memory from Matlab, has been developed. The IC can implement six dimensional functions, but it can also be configured to implement lower dimensional functions with more precision. In the six dimensional case, the inputs are 24-bit words, where 4 bits define the partition of the domain and the remaining 20 bits are used for intrapartition definition. In addition, the function parameters can be changed by the user to implement adaptive algorithms. The versatility of the approach comes at the price of memory capacity; in the six dimensional case, 16 partitions per dimension imply a 16 MB memory. Clearly, higher dimensional problems are limited by memory availability.

Area and I/O pin limitations were major factors that impacted on the number of cycles to produce one computation. The maximum area available prevented the use of parallelism, and impacted on several aspects of the architecture: the program memory word length had to be shortened; a more compact encoding had to be defined for the instruction set, increasing the complexity of the decode logic; the data memory addressing had to be multiplexed (requiring extra clock cycles); and some interconnections had to be suppressed. As a result, the processor needs 143 clock cycles to complete one calculation. The availability of more area in an advanced process would lead to a considerable speed up of the evaluation cycle by allowing a higher frequency operation and a decreased cycle count. The optimizations presented for the calculation cycle could be further extended and the use of parallel techniques would dramatically reduce the cycle count; indeed, since the complexity of the algorithm is linear on the number of dimensions, the total number of cycles required is in the order of seven clock cycles. As an example, sorting can be further speed up with the

[9]This program loads two registers of the Register File, moves them to RegA and RegB, adds them up, and writes the result into MADR.

addition of more comparators and dedicated structures [40]; the seven cumulative products can be performed in parallel with additional multipliers and adders; and memory data can be prefetched during sorting time. Ultimately, the bottleneck of the approach is due to memory access, which can be overcome by using on-chip memory (directly, or at least via a cache option). In this direction, the increasing density of DRAM integration and the possibility of vertical integration in 3D technologies provide an ideal platform for the proposed microprocessor in mid-size dimensions.

APPENDIX

The DFT signals in Fig. 6 are the following:

- RegA, RegB, and Reg_Alu_out: 6 LSB of these registers.
- tst_inreg: 3 bits of the "reg_in_control_ram" (command flow) and 3 bits of the "data_in_control_ram" (data path registers).
- MADR: Memory Address Data Read register.
- tst_out_vrt: Register with integer part.
- tst_out_sr38: Serial to parallel register.
- tst_mem0, tst_mem15 and tst_memX: 6 LSB of the internal ROM memory (addresses 0, 15 and 255).
- MIR and DATA_out: 3 LSB of the Micro Instruction Register and 3 LSB of the data from the internal ROM memory.
- MPC and ADD_IN: 3 LSB of the Micro Program Counter and 3 LSB of the address pointing to the internal ROM.
- tst_s1: First 4 bits of the decoder SIN (write_en) for the 1st internal MEM block and first 2 bits of the internal memory address decoder.
- tst_s3: First 4 bits of the decoder SOUT (read_en) for the 1st internal MEM block and the first 2 bits of the internal memory address decoder.
- Mux_jmp_control and jmp_cond and wait_jmp_flag and REQ and WIT: combination of command control flow signals.

ACKNOWLEDGMENT

The authors would like to thank Synopsys University Program, the MOSIS Service for IC fabrication, and also Ariel Arelovich for the development of the software to compile PWL assembler to machine code.

REFERENCES

[1] J. de Boeij, E. Lomonova, and A. Vandenput, "Look-up table based real-time commutation of 6-DOF planar actuators," in *Proc. IEEE Int. Conf. Control Appl. (CCA 2007)*, pp. 1118–1123.

[2] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," in *Proc. 16th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processor (ASAP 2005)*, pp. 328–333.

[3] A. Strollo, D. D. Caro, and N. Petra, "A 630 MHz, 75 mW direct digital frequency synthesizer using enhanced ROM compression technique," *IEEE J. Solid-State Circuits*, vol. 42, no. 2, pp. 350–360, 2007.

[4] T. Fujisawa and E. Kuh, "Piecewise-linear theory of nonlinear networks," *SIAM J. Appl. Math.*, Jan. 1972 [Online]. Available: http://www.jstor.org/stable/2099721

[5] S. Kang and L. Chua, "A global representation of multidimensional piecewise-linear functions with linear partitions," *IEEE Trans. Circuits Syst.*, vol. 25, no. 11, pp. 938–940, 1978.

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

[7] L. Chua and R. Ying, "Canonical piecewise-linear analysis," *IEEE Trans. Circuits Syst.*, vol. 30, no. 3, pp. 125–140, 1983.

[8] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 319–330, 2005.

[9] D. D. Caro, N. Petra, and A. Strollo, "Reducing lookup-table size in direct digital frequency synthesizers using optimized multipartite table method," *IEEE Trans. Circuits Syst. I, Reg. Papers,* , vol. 55, no. 7, pp. 2116–2127, 2008.

[10] L. Chua and S. M. Kang, "Section-wise piecewise-linear functions: Canonical representation, properties, and applications," *Proc. IEEE*, vol. 65, pp. 915–929, 1977.

[11] J.-A. Pineiro, S. Oberman, J.-M. Muller, and J. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 304–318, 2005.

[12] O. Hammi, S. Boumaiza, M. Jaidane-Saidane, and F. Ghannouchi, "Digital subband filtering predistorter architecture for wireless transmitters," *IEEE Trans. Microw. Theory Tech.*, vol. 53, no. 5, pp. 1643–1652, 2005.

[13] L.-C. Chang and J. V. Krogmeier, "A look-up table with amplitude scaling technique for amplifier linearization," in *Proc. 10th IEEE Singapore Int. Conf. Commun. Syst. (ICCS 2006)*, pp. 1–5.

[14] J. Yuan, N. Farhat, and J. V. der Spiegel, "Background calibration with piecewise linearized error model for CMOS pipeline A/D converter," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 1, pp. 311–321, 2008.

[15] Bruno, J. Cousseau, S. Werner, J. Figueroa, M. Cheong, and R. Wichman, "An efficient CS-CPWL based predistorter," *Radioengineering*, vol. 18, no. 2, pp. 170–177, Jun. 2009.

[16] W.-J. Kim, K.-J. Cho, S. Stapleton, and J.-H. Kim, "Piecewise pre-equalized linearization of the wireless transmitter with a Doherty amplifier," *IEEE Trans. Microw. Theory Tech.*, vol. 54, no. 9, pp. 3469–3478, 2006.

[17] M. Zakhama and D. Massicotte, "A systolic architecture for channel equalization based on a piecewise linear fuzzy logic algorithm," in *Proc. IEEE Can. Conf. Elect. Comput. Eng.*, 1999, vol. 2, pp. 1098–1101.

[18] M. Vidal and D. Massicotte, "A VLSI parallel architecture of a piecewise linear neural network for nonlinear channel equalization," in *Proc. 16th IEEE Conf. Instrum. Meas. Technol. (IMTC/99)*, vol. 3, pp. 1629–1634.

[19] A.-O. Dahmane, D. Massicotte, and L. Szczecinski, "A VLSI architecture of a piecewise RBF decision feedback channel equalizer," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS 2001)*, vol. 4, pp. 342–345.

[20] R. Hiorns and M. Sandler, "Power digital to analogue conversion using pulse width modulation and digital signal processing," *IEE Proc. G Circuits, Dev. Syst.*, vol. 140, no. 5, pp. 329–338, 1993.

[21] M. Streitenberger, H. Bresch, and L. Mathis, "Theory and implementation of a new type of digital power amplifier for audio applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS 2000 Geneva)*, vol. 1, pp. 511–514.

[22] A. Prodic and D. Maksimovic, "Design of a digital PID regulator based on look-up tables for control of high-frequency DC-DC converters," in *Proc. IEEE Workshop Comput. Power Electron.*, 2002, pp. 18–22.

[23] M. Sonnaillon, G. Bisheimer, C. D. Angelo, J. Solsona, and G. Garcia, "Mechanical-sensorless induction motor drive based only on DC-Link measurements," *IEE Proc. Elect. Power Appl.*, vol. 153, no. 6, pp. 815–822, 2006.

[24] M. Duran, S. Toral, F. Barrero, and E. Levi, "Real-time implementation of multi-dimensional five-phase space vector PWM using look-up table techniques," in *Proc. 33rd Annu. Conf. IEEE Ind. Electron. Soc. (IECON 2007)*, pp. 1518–1523.

[25] J. Rodriguez, P. Julian, O. Lifschitz, O. Agamennoni, and V. Jimenez-Fernandez, "VLSI microprocessor architecture for a simplicial PWL function evaluation core," in *Proc. Argentine School Micro-Nanoelectron., Technol. Appl. (EAMTA 2008)*, pp. 55–60.

[26] P. Julian, "A high-level canonical piecewise linear representation: Theory and applications," *UMI, Bell & Howell Inf. Learn.*, p. 182, 1999.

[27] P. Julian, A. Desages, and O. Agamennoni, "High-level canonical piecewise linear representation using a simplicial partition," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 46, no. 4, pp. 463–480, 1999.

[28] L. Castro, J. Figueroa, and O. Agamennoni, "An NIIR structure using HL CPWL functions," *Latin Amer. Appl. Res.*, vol. 35, pp. 161–166, 2005.

[29] M. Storace, P. Julian, and M. Parodi, "Synthesis of nonlinear multiport resistors: A PWL approach," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 49, no. 8, pp. 1138–1149, 2002.

[30] M. Storace and O. D. Feo, "Piecewise-linear approximation of nonlinear dynamical systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 4, pp. 830–842, 2004.
[31] T. Poggi, F. Comaschi, and M. Storace, "Digital circuit realization of piecewise-affine functions with nonuniform resolution: Theory and FPGA implementation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 2, pp. 131–135, 2010.
[32] J. P. Bowen, "Hypercubes," *Practical Comput.*, vol. 5, no. 4, pp. 97–99, 1982.
[33] T. Matsumoto, "A chaotic attractor from chua's circuit," *IEEE Trans. Circuits Syst.*, vol. 31, no. 12, pp. 1055–1058, 1984.
[34] G.-Q. Zhong and F. Ayrom, "Periodicity and chaos in chua's circuit," *IEEE Trans. Circuits Syst.*, vol. 32, no. 5, pp. 501–503, 1985.
[35] P. Li and J. Allebach, "Look-up-table based halftoning algorithm," *IEEE Trans. Image Process.*, vol. 9, no. 9, pp. 1593–1603, 2000.
[36] O. Hammi, F. Ghannouchi, S. Boumaiza, and B. Vassilakis, "A data-based nested LUT model for RF power amplifiers exhibiting memory effects," *IEEE Microw. Wireless Compon. Lett.* , vol. 17, no. 10, pp. 712–714, 2007.
[37] P. Chancharoensook and M. Rahman, "Dynamic modeling of a four-phase 8/6 switched reluctance motor using current and torque look-up tables," in *Proc. IEEE 28th Annu. Conf. Ind. Electron. Soc. (IECON 02)*, vol. 1, pp. 491–496.
[38] V. Bourenkov, K. McCarthy, and A. Mathewson, "MOS table models for circuit simulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 3, pp. 352–362, 2005.
[39] G. Yu and P. Li, "Efficient look-up-table-based modeling for robust design of $\Sigma - \Delta$ ADCs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 7, pp. 1513–1528, 2007.
[40] G. Miranker, L. Tang, and C. Wong, "A zero-time VLSI sorter," *IBM J. Res. Develop.*, vol. 27, pp. 140–148, Mar. 1983.

J. Agustín Rodríguez was born in Bahía Blanca, Argentina, in 1981. He received the Computer Systems Engineering degree from Universidad Nacional del Sur (UNS), Bahía Blanca, Argentina, in 2007. He is currently working toward the Ph.D degree in computer science at UNS.

His research interests include: high performance computer architecture, high level synthesis of multicore architectures, static timing analysis, and the place and route problem.

Omar D. Lifschitz recived the Electronic Engineer degree from Instituto Tecnologico Buenos Aires, Argentina (ITBA) in 1998. He is currently working toward the Ph.D. degree in control systems at the Universidad Nacional del Sur (UNS), Bahía Blanca, Argentina

From 1999 to 2007 he worked as Hardware Engineer for Intel Israel with the analog validation group. He specialized in signal integrity.

Víctor Manuel Jiménez-Fernández was born in Puebla, Mexico, in 1974. He received the B.S. degree in electronics engineering from the Instituto Tecnológico de Veracruz in 1998, the M.Sc. degree from the Universidad de las Américas-Puebla in 2000, and the Ph.D. degree from the Instituto Nacional de Astrofísica,Óptica y Electrónica (INAOE) Puebla, Mexico, in 2006.

From 2006 to 2007 he was a postdoctoral student in the Departamento de Ingeniería Eléctrica y de Computadoras at the Universidad Nacional del Sur, Bahía Blanca, Argentina. From 2008 to 2009, he was an Assistant Researcher at the INAOE-Mexico. He is currently an Associate Professor in the Facultad de Instrumentación Electrónica-Universidad Veracruzana, México. His main research interests are integrated circuits design and nonlinear circuit modeling.

Pedro Julián (S'93–M'99–SM'05) received the Electronic Engineer degree in 1994 and the Ph.D. degree in "Control de Sistemas" in 1999, both from Universidad Nacional del Sur (UNS), Bahía Blanca, Argentina.

He was a Visiting Scholar at the University of California Berkeley (2000 to 2002) and Visiting Scholar (2002 to 2003) and Visiting Fulbright Professor (2009) at Johns Hopkins University. He holds positions as an Associate Professor in the Departamento de Ingeniería Eléctrica y Computadoras (DIEC) at UNS, and as an Independent Researcher with the National Research Council of Argentina (CONICET). His research interests include theory and applications of computational circuits and systems, electronic systems, in particular sensory processors (acoustic and vision), with emphasis on low power VLSI systems.

Prof. Julián served as the Region 9 (Latin America) Vice President and on the Board of Governors of the IEEE Circuits and Systems Society (CASS) from 2004–2007, and he is a founding member of the Latin American Consortium for Integrated Services (LACIS) and the Argentine School of Microelectronics (EAMTA). He is the recipient of the Bernardo Houssay 2009 Prize of the Ministerio de Ciencia, Tecnología e Innovación Productiva (MINCyT) and the 2009 Electronic Engineering Prize of Academia Nacional de Ciencias Exactas, Físicas y Naturales (ANCEFN). He also serves as Associate Editor of the *International Journal of Circuit Theory and Applications*, the *CASS Magazine*, and the *CASS Newsletter*.

Osvaldo Enrique Agamennoni was born in Bahía Blanca, Argentina, on December 21, 1953. He received the Electrical Engineering degree and the Doctorate in control systems both from the Universidad Nacional del Sur (UNS), Bahía Blanca, Argentina, in 1979 and 1991, respectively.

From 1980 to 1983 he worked in electronic circuits design. From 1983 to 2001 he was involved in different areas of process control at Planta Piloto de Ingeniería Química (PLAPIQUI). From 1992 to 1994 he was a postdoctoral fellow in the Chemical Engineering Department of the University of Sydney. Since 1986 he has been with the Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC). He is Profesor Titular in the Departamento de Ingeniería Eléctrica y de Computadoras, UNS. He teaches undergraduate courses in control theory and a graduate course in system modeling and identification. His research interests include nonlinear system modeling, identification and control of nonlinear systems.