FPGA Implementations of Piecewise Affine Functions Based on Multi-Resolution Hyperrectangular Partitions

Francesco Comaschi, Bart A. G. Genuit, Alberto Oliveri, W. P. Maurice H. Heemels, *Senior Member, IEEE*, and Marco Storace, *Member, IEEE*

***Abstract*—In this paper we propose a digital architecture suited for fast, low-power and small-size electronic implementation of PieceWise Affine (PWA) functions defined over *n*-dimensional domains partitioned into multi-resolution hyperrectangles. The point location problem, which requires most of the computational effort, is solved through an orthogonal search tree, which is easily and efficiently implementable. In the case of domains partitioned into single-resolution hyperrectangles, a simpler and even faster architecture is proposed. After introducing the new architectures, their key features are discussed and compared to previous architectures implementing PWA functions with domains partitioned into different types of polytopes. Case studies concerning the FPGA implementation of so-called explicit Model Predictive Control (MPC) laws for constrained linear systems are used as benchmarks to compare the different architectures.**

***Index Terms*—Digital circuits, digital control, field programmable gate arrays, piecewise linear techniques, predictive control.**

I. Introduction

THE need for circuits evaluating nonlinear functions in real time arises in many engineering problems where it is not possible (usually due to size limitations) to resort to personal computers or other general-purpose devices, such as DSP boards. Applications that may benefit of the availability of electronic circuits implementing multivariate nonlinear functions can be found in many different fields, ranging from biology (neural networks) [1], to computer simulation (Internet models) [2], economics (heterogeneous agents models) [3], electrical power grids [4], and so forth. Another appealing application of circuits implementing nonlinear functions is the synthesis of embedded control systems where the control law has a nonlinear dependence on the state variables [5], [6].

Among possible techniques for the approximation and circuit synthesis of nonlinear functions, piecewise affine (PWA) techniques present the advantages of being versatile and having the ability of approximating any nonlinear system (or nonlinear control law) arbitrarily well [7]–[10].

PWA functions, which already have a long history in circuit theory [11]–[16] and control theory [17], [18] are characterized by expressing the output values in terms of a collection of affine functions and corresponding domains. In particular, given an input point, the value of the function is expressed by an affine map whose coefficients depend on the region containing the input. Recently, the attention in the control community for PWA functions has increased due to the discovery that popular optimization-based control laws, called Model Predictive Control (MPC), can be written explicitly as a PWA state feedback law defined using a polytopic partition of the feasible set [19], [20]. However, the resulting partitioning is typically irregular and often consists of a huge number of regions. As a consequence, the problem of determining the region the state belongs to, called the point location problem [21], requires a large amount of computational effort. To solve this problem, in [5] a binary search tree is built offline to minimize the resulting depth. By exploring the tree online, the polytope containing the input can be located in a smaller time, if compared to a combinatorial approach. Digital architectures for the implementation of these PWA Generic (henceforth denoted as PWAG) functions have been recently proposed, solving the point location problem through a VLSI [6] or a FPGA [22] implementation of the binary search tree proposed in [5]. However, the irregular shapes of the polyhedral regions can still lead to an enormous computation time for large-scale problems. In [23], a lattice approach is introduced to allow a reduction of online computation and storage when dealing with explicit MPC solutions that have many polyhedral regions with equal affine control laws. In [24], algorithms for online evaluation of PWAG functions, based on a truncated binary search tree and/or lattice representation are presented, while in [25] the implementation of PWAG functions is simplified by a two-stage algorithm utilizing the concept of hash tables together with the usual direct search. The latter two methods enable the designer to tradeoff between (off-line and on-line) time and storage complexity. A digital architecture implementing PWA functions based on a lattice representation has been proposed in [26], [27].

Manuscript received December 30, 2011; revised April 16, 2012; accepted May 21, 2012. Date of publication August 07, 2012; date of current version November 21, 2012. This work was supported in part by the European Commission through the project MOBY-DIC "Model-based synthesis of digital electronic circuits for embedded control" (FP7-INFSO-ICT-248858), http://www.mobydic-project.eu. This paper was recommended by Associate Editor Xinmiao Zhang.

F. Comaschi is with the Electronic Systems Group, Department of Electrical Engineering, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands, and also with the Biophysical and Electronic Engineering Department, University of Genoa, I-16145 Genova, Italy(e-mail: f.comaschi@tue.nl).

A. Oliveri and M. Storace are with the Biophysical and Electronic Engineering Department, University of Genoa, I-16145 Genova, Italy (e-mail: alberto.oliveri@unige.it; marco.storace@unige.it).

B. A. G. Genuit and W. P. M. H. Heemels are with the Hybrid and Networked Systems Group, Department of Mechanical Engineering, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands (e-mail: m.heemels@tue.nl; contact@bartgenuit.nl).

Digital Object Identifier 10.1109/TCSI.2012.2206490

Another way to improve the computation efficiency in the online evaluation of PWAG functions is to approximate the optimal PWAG control law through suboptimal PWA control laws defined over more regularly-shaped partitions [25], [28]–[32]. For instance, the point location problem can be highly simplified by using a simplicial [33] or (hyper)rectangular [30]–[32] partition of the domain. PWA Simplicial (PWAS) functions can be easily implemented on specific digital or mixed-signal architectures [8], [34], [35]. Two completely digital solutions suitable for FPGA implementation, one mainly serial and one fully parallel (henceforth referred to as architectures A and B, respectively), were proposed in [10]. Generalizations of them in [36] allow to represent PWAS functions defined over non-uniform partitions, but the number of coefficients required still grows exponentially with the number of dimensions, thus making this approach affected by the "curse of dimensionality".

The shape of a PWA function is coded by a set of coefficients that are embedded in the circuit, and one of the main problems the circuit designers are concerned with is to minimize the number of coefficients, i.e., to compress the information since this heavily affects the overall circuit size. This can be fruitfully done by resorting to multi-resolution methods (see, e.g., [37], [38] in the field of wavelets), where the domains over which the function is affine are characterized by regularity and self-similarity in scale. The circuit architectures proposed in this paper rely on a multiresolution-based compression algorithm for the approximation of PWAG (control) functions by PWA functions defined over a multi-resolution hyperrectangular partition of the domain (which we will denote as PWAR functions) [30]–[32]. We remark that the proposed method has been adopted not owing to its compression performances (other algorithms are more efficient from this point of view), but since it is suitable for circuit implementation. As stated above, from a circuit implementation perspective, a multi-resolution partition of the domain [29]–[32], [39] is of particular interest since it provides a mitigation to the curse of dimensionality.

Since suitable digital implementations of PWAR functions are still missing, the focus of this paper is on introducing suitable architectures to evaluate (possibly discontinuous) PWAR functions, in particular those with a multi-resolution partition. The point location problem is solved through an orthogonal tree [30], [31],[1] which provides fast on-line computation and a low circuit complexity for large-scale problems. In the case the domain is partitioned into single-resolution hyperrectangles, a simpler and even faster architecture is proposed.

The outline of the paper is as follows. Some notations together with the necessary preliminaries are introduced in Section II. In Section III the procedure adopted for the evaluation of PWAR functions is presented, with a particular focus on the point location strategy. Two different implementations are proposed in Section IV for each of the new architectures and they are compared with other existing circuit solutions in Section V. The results obtained by the FPGA implementations for two case studies of MPC are used to point out the main features of the different digital realizations in Section VI. Section VII contains the concluding remarks.

[1]Note that the type of search tree used here (specifically, a generalized octree or 2^n-tree [21]) is different from the binary search tree used e.g., in [5].

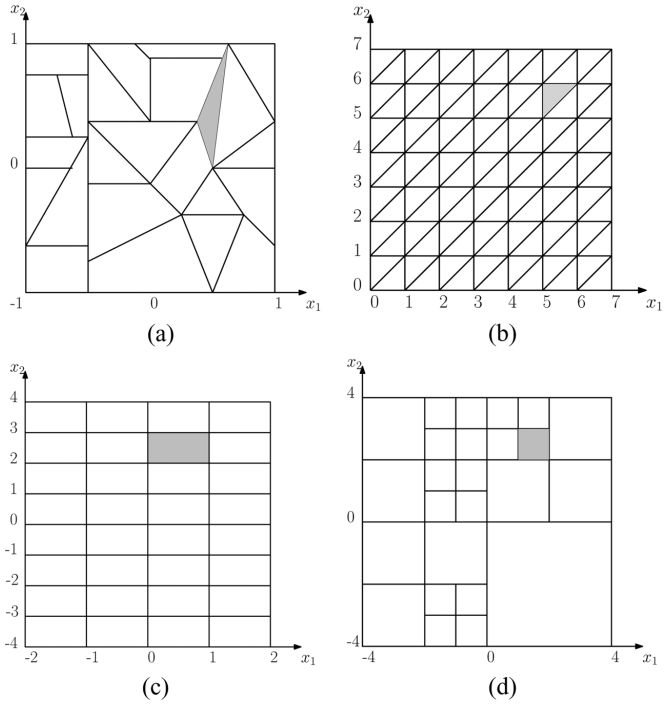

Fig. 1. Examples of two-dimensional domain partitions: polytopic (a), simplicial single-resolution (b), hyperrectangular single- (c) and multi-resolution (d). The corresponding classes of PWA functions are: PWAG (a), PWAS (b), sPWAR (c), and mPWAR (d).

II. Preliminaries

We call a collection of regions $\{\chi_1, \chi_2, \ldots, \chi_{L_G}\}$, $\chi_j \subseteq \mathbb{R}^n$, $j = 1, \ldots, L_G$, a partition of a domain $S = \{\mathbf{z} \in \mathbb{R}^n : a_i \leq z_i \leq b_i, i = 1, \ldots, n\}$, if $S = \cup_j^{L_G} \chi_j$ and $\chi_j \cap \chi_k = \oslash$ for $j, k = 1, \ldots, L_G, j \neq k$. If in addition all sets χ_j are polytopes, we call it a polytopic partition. A scalar function f on a domain $S \in \mathbb{R}^n$ is called a PWAG scalar function if there is a polytopic partition of S and coefficients $\mathbf{f}_j \in \mathbb{R}^n$ and $g_j \in \mathbb{R}$, $j = 1, \ldots, L_G$ such that $f : S \to \mathbb{R}$ satisfies

$$f(\mathbf{z}) = \mathbf{f}_j'\mathbf{z} + g_j, \quad \mathbf{z} \in \chi_j. \tag{1}$$

where A' denotes the transpose of matrix A. Fig. 1(a) shows an example of a polytopic partition of a two-dimensional domain. While PWAG functions are a rich class of non-linear functions, from an implementation point of view it is convenient (as we will show in Section IV) to consider subclasses of PWA functions defined over domains partitioned into more regularly shaped regions. In the next subsections we define three particular partitions of compact domains: simplicial, single-resolution hyperrectangular and multi-resolution hyperrectangular.

A. Simplicial Partition

The hyperrectangular compact domain $S = \{\mathbf{z} \in \mathbb{R}^n : a_i \leq z_i \leq b_i, i = 1, \ldots, n\}$ can be subdivided by using a simplicial boundary configuration [40] such that the i-th dimensional component z_i of S contains m_{Si} subintervals of ampli-

tude $\alpha_i = (b_i - a_i)/(m_{Si})$. Thus, S is partitioned in a particular manner (see [40] for details) into $\prod_{i=1}^{n} m_{Si}$ hyperrectangles, each hyperrectangle containing $n!$ nonoverlapping hypertriangular simplices, whose $N_v = \prod_{i=1}^{n}(m_{Si}+1)$ vertices are collected in the set $V_z = \{\mathbf{v}_j \in \mathbb{R}^n : j = 1, \ldots, N_v\}$. For circuit implementation reasons, we choose $m_{Si} = 2^{p_i} - 1$, with $p_i \in \mathbb{N}$, so that the resulting number of vertices for each dimension is an exact power of two.

We call the obtained partition a *simplicial partition* [33], [41], and we can use it to define a subclass of continuous PWAG functions henceforth referred to as PWAS. In fact, PWAS functions are all PWAG functions, as in (1), in which the underlying partition $\{\chi_1, \ldots, \chi_{L_G}\}$ is a simplicial partition and the function is continuous. The interested reader is referred to [28] for further information.

As we shall see in the following, from a circuit-synthesis point of view, it is convenient to handle variables ranging over scaled intervals. For this reason we introduce an affine transformation $\mathbf{x} = T(\mathbf{z})$ of the original input variable $\mathbf{z}$ to map the aforementioned hyperrectangular domain S to a domain $S_S = \{\mathbf{x} \in \mathbb{R}^n : 0 \le x_i \le m_{Si},\ i = 1, \ldots, n\}$, $m_{Si} \in \mathbb{N}$, where the hyperrectangles become hypercubes and the amplitude of the m_{Si} subintervals is $\alpha_i = 1$ [22], [34].

Fig. 1(b) shows a scaled two-dimensional domain partitioned into simplices with $m_{S1} = m_{S2} = 2^3 - 1 = 7$.

B. Single-Resolution Hyperrectangular Partition

Also in this case we subdivide each axis of the hyperrectangular domain S into m_{Ri} subintervals of amplitude $\alpha_i = (b_i - a_i)/(m_{Ri})$, but we choose $m_{Ri} = 2^{t_i}$, with $t_i \in \mathbb{N}$, so that the resulting partition contains $\prod_{i=1}^{n} m_{Ri} = 2^{\sum_{i=1}^{n} t_i} = L_R$ hyperrectangles χ_j of identical shape.[2] We call the obtained partition a *single-resolution hyperrectangular partition*; the term single-resolution refers to the fact that we choose a single resolution for each axis, thus leading to a domain partitioned into equally-sized hyperrectangles. All PWAG functions defined over such a regular partition form a subclass of PWAG functions, referred to as sPWAR (single-resolution PWA hyperRectangular). For circuit implementation reasons, we restrict our attention to PWAR functions defined over an n-dimensional compact domain $S_R = \{\mathbf{x} \in \mathbb{R}^n : (-m_{Ri})/(2) \le x_i < (m_{Ri})/(2),\ i = 1, \ldots, n\}$. Each coordinate axis x_i is then divided into m_{Ri} subintervals of unitary length $\alpha_i = 1$. Fig. 1(c) shows a two-dimensional example of a scaled domain S_R partitioned into single-resolution hyperrectangles with $m_{R1} = 2^2 = 4$ and $m_{R2} = 2^3 = 8$.

C. Multi-Resolution Hyperrectangular Partition

Consider a similar procedure as above, where each state axis of the domain S is divided not just into intervals with a single resolution given by m_{Ri}, but instead divided subsequently into higher and higher levels of resolution by halving the subintervals each time (also called dyadic or binary division), up to some chosen maximum resolution. That is, $\alpha_{l,i} = (b_i - a_i)/(m_{Rl})$, where $m_{Rl} = 2^l$, with levels of resolution $l = 1, \ldots, r$. This procedure of refinement results in a collection of overlapping sets corresponding to different levels of resolution l, also referred to as levels of refinement or simply levels. Clearly their union is the whole domain. By selecting now L_M regions χ_j, $j = 1, \ldots, L_M$, out of this collection, such that they still cover S, but are non-overlapping, i.e., $S = \cup_j \chi_j$ and $\chi_j \cap \chi_k = \oslash$ for $j, k = 1, \ldots, L_M, j \ne k$, a partition of S is obtained. The resulting partition contains at most $\prod_{i=1}^{n} m_{Rr} = 2^{nr}$ hyperrectangles χ_j, thus $L_M \le 2^{nr}$. We call such a partition a *multi-resolution hyperrectangular* partition.

All (possibly discontinuous) PWAG functions of the form (1) based on a multi-resolution hyperrectangular partition $\{\chi_1, \chi_2, \ldots, \chi_{L_M}\}$, $\chi_j \subseteq \mathbb{R}^n$, $j = 1, \ldots, L_M$ where L_M is the total number of multi-resolution hyperrectangular regions, form a subclass of PWAG functions referred to as mPWAR (multi-resolution PWA hyperRectangular) functions.

For circuit implementation reasons we introduce again an affine mapping of the original domain, passing from S to the scaled domain $S_M = \{\mathbf{x} \in \mathbb{R}^n : (-m_{Rr})/(2) \le x_i < (m_{Rr})/(2), i = 1, \ldots, n\}$, where the multi-resolution hyperrectangles become hypercubes. Fig. 1(d) displays a two-dimensional example of a such a partition with $r = 3$ levels and $L_M = 25$ regions.

III. PWAR Function Evaluation

On-line evaluation of PWAG functions as in (1) for an input $\mathbf{x}$ is usually performed in three steps:

1) solve the point location problem to obtain the index j of the region χ_j that contains the current input, i.e., $\mathbf{x} \in \chi_j$;
2) obtain the function parameters $\mathbf{f}_j, g_j$ in (1) from a lookup table;
3) construct the output by evaluating the affine expression $\mathbf{f}_j'\mathbf{x} + g_j$.

From a circuit point of view, the second and third steps can be carried out in a simple way by a few functional blocks. Indeed, what we need is a memory to store the values of the coefficients and a multiplier and an adder to evaluate the affine expression. In particular, once index j is known, the corresponding $\mathbf{f}_j$ and g_j are supposed to be already available, as a result of off-line computation. The first step, known as the point location problem, can become computationally very expensive, especially in case of irregular partitions and/or a large input dimension.

To reduce the computational complexity of the point location step, approximation methods have been proposed in recent years to obtain functions on partitions with more regular shapes such as simplices [28], [29] or hyperrectangles [30]–[32]. For PWAG and PWAS forms, several circuit implementations have already been proposed [6], [8], [10], [22], [42]. In this paper, we will focus on circuit implementations of the PWAR family of functions, for both the single- and multi-resolution case.

[2]The reason why it is preferable to choose a different number of subdivisions in the simplicial and in the hyperrectangular cases is due to the way in which the PWA expressions are evaluated in the two cases. Indeed, the PWAS function evaluation is based on the values of the function at the vertices of the partition, as detailed in Section V.B, whereas in the hyperrectangular case the function evaluation requires the values of the coefficients $\mathbf{f}_j, g_j$ for each hyperrectangle. As we will see, this difference in the evaluation procedures justifies the choice of the number of subintervals for each case, as well as the different affine mappings introduced.

A. Point Location for sPWAR Partitions

When having a sPWAR function, the point location problem can be solved through a simple strategy resulting in short computation times.

Once we have fixed the resolution t_i for each input variable, $i = 1, \ldots, n$, in the single-resolution case, each of the domain axes is partitioned into $m_{Ri} = 2^{t_i}$ subintervals, thus leading to a total of $2^{\sum_{i=1}^{n} t_i}$ single-resolution hyperrectangular regions. This means that if we use $b = t_i + s_i$ bits to code the components of the input variable in the domain S_R, each hyperrectangle χ_j can be univocally determined by the t_i most significant bits (MSBs), coding the signed integer part of each input component, $i = 1, \ldots, n$. Let us define the operator $\lfloor \cdot \rfloor$ as the operator extracting the signed integer part of the input vector $\mathbf{x} = [x_1\, x_2\, \ldots\, x_n]'$. Then, we define $\beta_{\mathbf{t}}$ as the binarising operator that, given a vector of n signed integer values and a vector $\mathbf{t}$ of precisions t_i for $i = 1, \ldots, n$, returns a $\sum_{i=1}^{n} t_i$-long string of bits concatenating the 2's complement binary values of the elements of the vector starting from x_n down to x_1. For instance, suppose that the current input belongs to the gray hyperrectangle in Fig. 1(c) for a two-dimensional case, with $t_1 = 2$ and $t_2 = 3$, then $\beta_{\mathbf{t}}(\lfloor \mathbf{x} \rfloor) = 010\ 00$. Therefore, $\beta_{\mathbf{t}}(\lfloor \mathbf{x} \rfloor)$ allows to compute an unambiguous binary label χ_j^b for the single-resolution hyperrectangle χ_j the current input belongs to, and we can map each of the single-resolution hyperrectangles making up the partition to the corresponding binary label simply by joining the t_i MSBs from each of the input components:

$$\chi_j^b = \beta_{\mathbf{t}}(\lfloor \mathbf{x} \rfloor) \tag{2}$$

This simple operation can be performed in only one clock cycle in a parallel fashion, as detailed in Section IV.B.

Remark 1: We point out that the binary label χ_j^b does not correspond to the mere binary conversion of j since this would not allow the efficient digital implementation (detailed in Section IV.B) of the point location strategy described in this section.

B. Point Location for mPWAR Partitions

The main idea at the base of our multi-resolution approach is to build offline an orthogonal search tree made up of a number of nodes ν_q, $q = 0, \ldots, N_M - 1$, such that the real-time search complexity is logarithmic with respect to the number of regions. In the following, we often refer to the multi-resolution regions as hypercubes instead of hyperrectangles, since the hyperrectangles of the original domain S become hypercubes in the scaled domain S_M. The root of the tree ν_0 corresponds to the initial partition $S^{(0)}$ $(= S_M)$, the non-leaf nodes correspond to hypercubic regions, which are in turn partitioned into higher-resolution hypercubes, whereas the leaf nodes represent the hypercubic multi-resolution sets χ_j, $j = 1, \ldots, L_M$. The level of depth l where the leaf nodes are located can vary from 1 to r. When exploring the tree online from the root to a leaf node, one ends up with a region χ_j containing the input and leading to the corresponding control law, characterized by the coefficients $\mathbf{f}_j$ and g_j.

The orthogonal tree depth r influences the maximum time required by the on-line tree exploration, whereas the total number of nodes N_M influences the circuit dimensions. Each non-leaf node has 2^n children representing the refined hypercubes within the current region, thus leading to an upper bound of $\sum_{i=0}^{r} 2^{in}$ nodes.

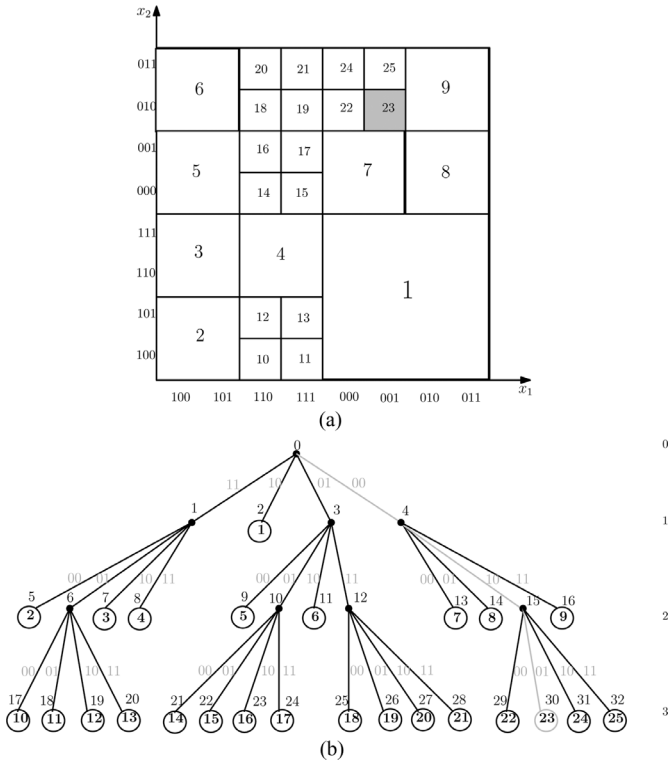

Fig. 2. Example of a two-dimensional domain partitioned with refinement depth $r = 3$ (a) and corresponding orthogonal search tree (b). The number above each circle indicates each node ν_q, $q = 0, \ldots, N_M - 1$ while the numbers inside the big circles (the leaf nodes of the tree) indicate the hypercubic regions χ_j, $j = 1, \ldots, L_M$. On the right of the tree the level of depth l is shown.

Fig. 2(a) shows an example, where a two-dimensional domain is partitioned at a maximum of $r = 3$ levels of refinement with $L_M = 25$ regions. The corresponding orthogonal search tree, illustrated in Fig. 2(b), is characterized by $N_M = 33$ nodes.

Remark 2: We choose to label both the N_M nodes and the L_M regions in increasing order, primarily with respect to the refinement level, and secondarily with the dimensional component x_i, $i = 1, 2, \ldots, n$ within each hypercube (see Fig. 2).

The orthogonal tree structure is particularly suited for an FPGA implementation with fixed point arithmetic, since each node can be easily related to a state of a Finite State Machine (FSM). Exploiting the regularity of the hypercubic partition, we can indeed associate each node of the tree (and thus each state of the related FSM) to an unambiguous binary label that can be easily computed from the r Most Significant Bits (MSB) of the input variable $\mathbf{x}$.

Let us define $\hat{\beta}_l$ as the binarising operator that, given a scalar precision l, returns a nl-long string of bits concatenating the 2's complement binary values of the elements of the vector $\lfloor \mathbf{x} \rfloor$ starting from x_n down to x_1. Therefore, $\hat{\beta}_l(\lfloor \mathbf{x} \rfloor)$ allows to compute an unambiguous binary label ν_q^b for each node making up the tree:

$$\nu_q^b = \hat{\beta}_l(\lfloor \mathbf{x} \rfloor) \tag{3}$$

This simple operation can be performed in a maximum of r clock cycles in a parallel fashion, regardless of the total number of inputs. The on-line exploration of the tree, and therefore the computation of the label ν_q^b, is described below.

Remark 3: The binary label ν_q^b does not correspond to the mere binary conversion of q, since this would not allow the efficient digital implementation (detailed in Section IV.A) of the point location algorithm described below.

At the first step, i.e., at $l = 0$, the MSB is checked in parallel for all the input variables. If the input belongs to a level-1 hypercube, the corresponding leaf node is uniquely determined by the first digit taken from each input component and the point location procedure ends. E.g., in the example of Fig. 2(a), region 1 is identified by $\hat{\beta}_1(\lfloor \mathbf{x} \rfloor) = 10$. Otherwise, at the second step, i.e., at $l = 1$, we must check the $(\mathrm{MSB} - 1)$-th bit from each input component. If the input belongs to a level-2 hypercube, the point location procedure ends and we obtain a $2n$-long label $\hat{\beta}_2(\lfloor \mathbf{x} \rfloor)$ that identifies the current leaf-node (e.g., in the example of Fig. 2(a), region 2 is identified by $\hat{\beta}_2(\lfloor \mathbf{x} \rfloor) = 10\,10$). In general, at each step, i.e., at each clock cycle, the $(\mathrm{MSB} - l)$-th bit is checked in parallel for all the input variables. This point is stressed by the couple of gray digits beside each branch of the tree in Fig. 2(b), where the left digit represents the $(\mathrm{MSB} - l)$-th bit of x_2, whereas the right one is the $(\mathrm{MSB} - l)$-th bit of x_1. In an n-dimensional case we would have n digits where the leftmost digit comes from x_n while the rightmost one comes from x_1. This search is repeated up to a maximum of r clock cycles, unless a leaf node has already been reached before. Once a leaf node is reached, the address from which to retrieve the coefficients in the circuit memory can be outputted as a simple binary conversion of the number j of the current region. Indeed, if we define $a = \lceil \log_2(L_M) \rceil$, each region χ_j making up the multi-resolution partition can be labeled by a string of a bits representing the binary conversion of the region number j. We now define $\tilde{\beta}_a$ as the binarising operator that, given an integer scalar value, returns a string of a bits corresponding to the binary unsigned representation of the provided scalar value. Therefore each multi-resolution hypercube χ_j making up the partition can be labeled by:

$$\chi_j^b = \tilde{\beta}_a(j) \tag{4}$$

A simple example of application of the point location algorithm is given below.

Remark 4: The main drawback exhibited by the proposed approach is then represented by the total number of FSM nodes, which for large-scale applications with a high level of refinement may become too large to fit on a (particular) FPGA. In order to reduce this problem, in the next section we describe an enhancement in the FSM implementation, which allows to avoid the instantiation of the states belonging to the last level of refinement r.

Example: In the following, we provide a simple two-dimensional example to show how the search strategy works online. On the domain axes of the example shown in Fig. 2(a), the first $r = 3$ binary digits coding the signed integer part of x_1 and x_2 normalized in the range S_M are shown. To see how the search strategy works, suppose that the current input belongs to region 23 (gray hypercube in Fig. 2(a)), i.e., $\mathbf{x} \in \chi_{23}$. Starting from the root of the tree, we first check the MSB from each input component, thus obtaining the string "00" and the binary label $\hat{\beta}_1(\lfloor \mathbf{x} \rfloor) = 0\,0$. Taking the corresponding branch of the tree, we end up in a node (node 4 in Fig. 2(b)) which is a non-leaf node, thus we go on checking the $(\mathrm{MSB} - 1)$-th bit from each input component, which provides us with the string "10". Again, we take the corresponding branch of the tree and we end up in node 15, associated with the binary label $\hat{\beta}_2(\lfloor \mathbf{x} \rfloor) = 01\,00$ which is a non-leaf node. Finally, we check the $(\mathrm{MSB} - 2)$-th from each input component, thus obtaining the string "01". This final step brings us to node 30, associated with the binary label $\hat{\beta}_3(\lfloor \mathbf{x} \rfloor) = 010\,001$ which is the leaf node corresponding to region 23. Once the leaf node is reached, the address from which to retrieve the coefficients in the circuit memory can be outputted as a string of $a = \lceil \log_2(25) \rceil = 5$ bits: $\chi_{23}^b = \tilde{\beta}_5(23) = 10111$.

IV. Circuit Implementations of mPWAR and sPWAR Functions

In this section we give a description of the main blocks composing the digital architecture for the implementation of mPWAR functions using the solution to the point location problem proposed in Section III.B. A different architecture is proposed for the implementation of sPWAR functions. Moreover, we present an alternative implementation of the proposed architectures, which allows a lower latency at the cost of a higher number of multipliers. The first proposed architectures, mainly serial, will be denoted by letter A, whereas the alternative architectures, mainly parallel, will be denoted by letter B. The proposed architectures have been realized through a VHDL description which is fully parametric, i.e., all the parameters (n, b, r, t_i) can be tuned at the user's demand. In this way it is possible to reprogram the digital architectures in order to represent different mPWAR and sPWAR functions by simply tuning a few parameters.

A. The mPWAR(A) Architecture

The computation of the mPWAR function is carried out through the tree steps described at the beginning of Section III. A simplified scheme of the mPWAR architecture is shown in Fig. 3. For ease of presentation, we show the particular case of a two-dimensional mPWAR function implementation. The *CK* and *RESET* signals as well as the control signals are identified by dashed lines. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

The architecture is made up of six main blocks:

- *Input* for a synchronous and parallel acquisition of the input vector $\mathbf{x}$, represented with b bits; this means that all the b bits of the n inputs are supposed to be available at the same time;
- *FSM*, implementing the orthogonal search tree described in Section III.B to find the j-th region the input $\mathbf{x}$ belongs to;
- $n + 1$ *Memory_i* blocks, storing the coefficients f_j^i, $i = 1, \ldots, n$ and g_j;
- A Multiply-and-Accumulate (*MAC*) block performing the n products $f_j^i x_i$ sequentially and adding them in the internal accumulator;

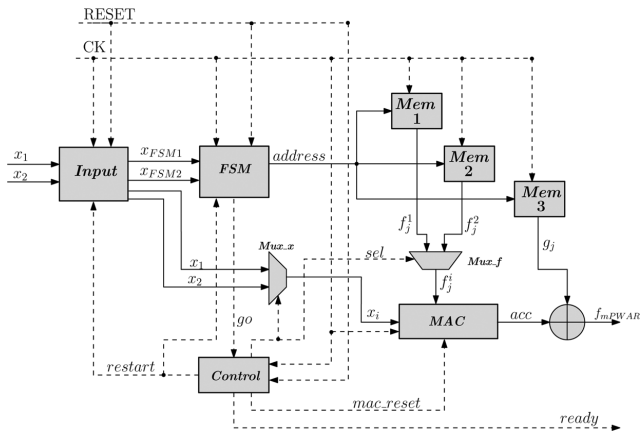

Fig. 3. A functional block scheme of the mPWAR(A) architecture.

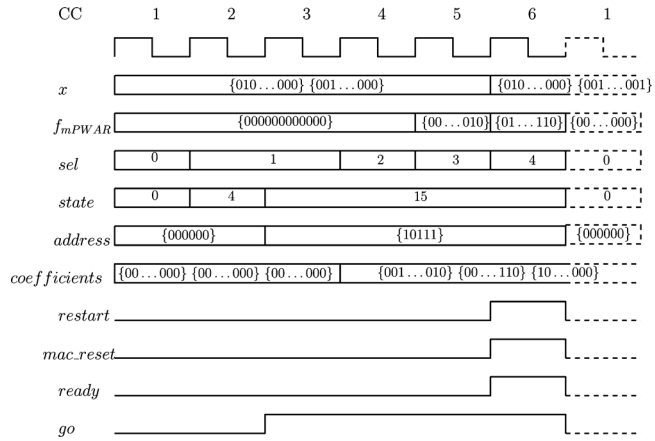

Fig. 4. Timings example for architecture mPWAR(A). In this case we assume $\mathbf{x} \in \chi_{23}$ from Fig. 2(a), i.e., $\nu_q^b = \hat{\beta}_l(\lfloor \mathbf{x} \rfloor) = 010001$ and $\chi_j^b = \tilde{\beta}_a(j) = 10111$.

- A two-input *Adder*, adding together the value of the accumulator output from the *MAC* block and the constant term g_j from *Memory_*$(n+1)$ output;
- *Control*, a block managing the control signals.

The other inputs to the circuit are the clock signal *CK*, responsible for the synchronization of the whole circuit, and the *RESET* signal. The overall timings of the circuit are managed by the *Control* block, which is a FSM whose internal state, assigned to the signal *sel* in Fig. 3, can assume the integer values in the range $[0,\, n+2]$. The control signals guarantee the synchronization between the operations performed by the digital architecture. The *restart* signal states that the computation of the previous output f_{mPWAR} has completed and the circuit is ready for the acquisition of a new input. When *restart* is asserted, the *FSM* block is reset to state 0, corresponding to the root of the orthogonal tree described in Section III.B. The *go* signal is sent from the *FSM* block to the *Control* block and it is asserted when the point location has concluded and the on-line exploration of the orthogonal tree has led to a leaf-node, thus allowing the architecture to retrieve from memory the coefficients for the evaluation of the mPWAR expression. The *ready* control signal states if the output value from the circuit is valid or not, so the output is discarded when $\mathrm{ready} = 0$.

The timings of the architecture are shown in Fig. 4.

Once the *RESET* signal is deasserted, the circuit starts processing. In the first Clock Cycle $(\mathrm{CC} = 1)$, the input variable $\mathbf{x}$ is acquired by the *Input* block and the r MSBs of each input component are sent to the *FSM* through the signal $\mathbf{x}_{\mathrm{FSM}}$, which is a bus of $n \times r$ bits. In the same clock cycle, the whole input variable is passed to the *Mux_x* block so that it is already available to perform the n multiplications $f_j^i x_i$ once the coefficients are retrieved. In $\mathrm{CC} = 2$, the internal state of the control *sel*, changes to 1 and the *FSM* block starts the point location for the actual input. The maximum level of refinement r states the maximum number of clock cycles required to perform the point location. In particular, since we have to consider also the state 0, corresponding to the root of the search tree, i.e., to the initial domain S_M, a maximum of $r+1$ clock cycles will be required. Actually, a simple enhancement (described later on) allows to reduce the required clock cycles from $r+1$ to r. For this reason, from clock cycle 1 to r the circuit will be involved in the point location. In clock cycle r, the point location procedure has concluded and a leaf node of the search tree has been reached,[3] so the *go* signal is asserted and the *address* of the coefficients to be retrieved from the *Memory_i* blocks is ready in output from the *FSM* block. In the next clock cycle $(\mathrm{CC} = r+1)$ the n coefficients f_j^i are retrieved from the *Memory_i* blocks and sent to *Mux_f*. From $\mathrm{CC} = r+2$ to $\mathrm{CC} = r+n+1$, the n products $f_j^i x_i$ are performed sequentially and added in the accumulator within the *MAC* block. In the last clock cycle $(\mathrm{CC} = r+n+1)$ the constant term g_j is added to the *acc* output signal from the *MAC* by a two-input combinatorial *Adder*, so that the value of the mPWAR function (f_{mPWAR}) is ready in output. The *ready* and *restart* signals are asserted in the last clock cycle, the former stating that the evaluation of f_{mPWAR} has concluded and the value in output is valid, the latter resetting the *FSM* block to state 0 so that a new point location can start in the next clock cycle. From the timings analysis we get that the overall circuit latency is given by $N_{\mathrm{CK}} = r+n+1$, where N_{CK} indicates the number of clock cycles.

The main drawback of the mPWAR architecture is represented by the total number of nodes of the FSM, whose upper bound is expressed by $\sum_{i=0}^{r} 2^{in}$. As an enhancement, we can avoid the instantiation of the states belonging to the last level of refinement r. When the exploration of the tree leads to a leaf node located at a level of depth from 1 to $r-1$, the FSM behaves as a Moore machine [43], i.e., the address provided in output depends only on the current state, regardless of the input value. On the contrary, when the online exploration leads to a non-leaf node belonging to level $r-1$, it necessarily means that its children are leaf nodes of the tree, so instead of waiting for the next clock cycle to provide the output, we can directly get the address of the first component of the $\mathbf{f}_j$ coefficient vector in memory making a combinatorial check on the $(MSB - r + 1)$-th bits of the input, thus adopting a Mealy machine approach [44]. An example of this behavior is shown in the piece of pseudo-code reported in Algorithm 1, which describes a section of the FSM output process.

[3]This is the worst case. If the current multi-resolution region is located in a higher level of the tree, the point location ends in an earlier clock cycle.

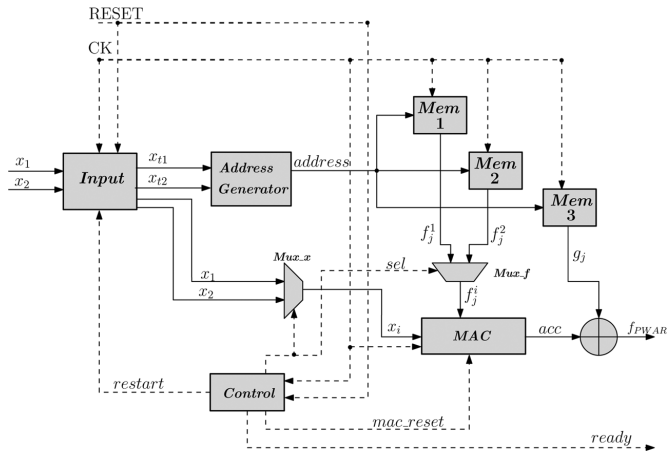

Fig. 5. A functional block scheme of the sPWAR(A) architecture.

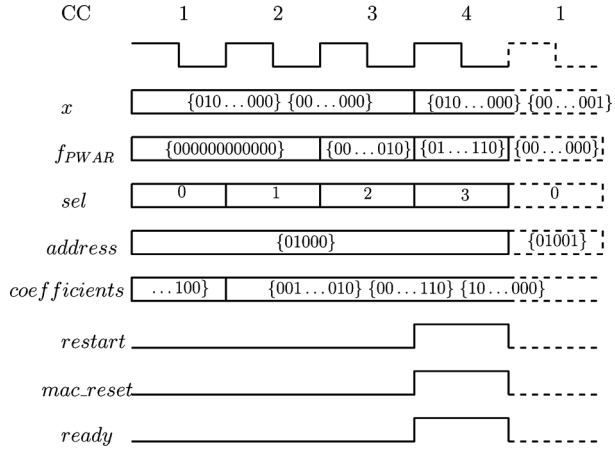

Fig. 6. Timings example for architecture sPWAR(A). In this case we assume $\beta_{\mathbf{t}}(\lfloor \mathbf{x} \rfloor) = 010\ 00$ (gray hyperrectangle in Fig. 1(c)).

Algorithm 1 FSM output process

```
case state is
  when 2: address ← 1
    ⋮
  when 13: address ← 7
  when 14: address ← 8
  when 15:
    if x_FSM(MSB − 2) = ``00" then
      address ← 22
    else if x_FSM(MSB − 2) = ``01" then
      address ← 23
    else if x_FSM(MSB − 2) = ``10" then
      address ← 24
    else
      address ← 25
    end if
  when 16: address ← 9
end case
```

This enhancement reduces the maximum number of clock cycles required by the point location task from $r+1$ to r and, most importantly, allows a circuit complexity reduction avoiding the instantiation of the states belonging to the last level of refinement.

B. The sPWAR(A) Architecture

In the previous subsection the digital architecture for the multi-resolution case was described. Interestingly, when having a single-resolution PWA hyperrectangular (sPWAR) function, one can even obtain a simpler digital architecture resulting in a shorter evaluation time, as described in this section.

Fig. 5 shows a simplified scheme of the sPWAR architecture in the case of a two-dimensional sPWAR function implementation. More in general, for a n-dimensional case, the circuit would contain $n+1$ *Memory* blocks. The *CK* and *RESET* signals as well as the control signals are identified by dashed lines.

Most of the remarks about the mPWAR architecture still hold for this digital circuit. The main difference lies in the presence of a combinatorial *Address Generator* block, which allows to solve the point location problem in one clock cycle, regardless of the dimensionality of the problem. Fig. 1(c) shows an example of a two-dimensional domain partitioned into single-resolution hyperrectangles. As we have already stated in Section III.A, the hyperrectangular region the actual input variable belongs to can be obtained simply by joining the t_i MSBs from each of the n input variables.

The simple operation of joining the t_i MSBs from the input variables into a single string identifying the current hyperrectangular partition is performed by the *Address Generator* block.

The timings for the sPWAR architecture are shown in Fig. 6.

The circuit inputs are the state variable $\mathbf{x}$ and the signals *CK* and *RESET*. In this case, the internal state of the *Control* block assumes integer values in the range $[0, n+1]$. In the first clock cycle $(\mathrm{CC}=1)$, the n components of the input are acquired in parallel by the *Input* block and the t_i, $i=1,\ldots,n$ most significant bits of each state variable are sent to the *Address Generator* (signals x_{t1} and x_{t2} in Fig. 5). In one clock cycle, the *Address Generator* joins together the t_i MSBs coming from the *Input* block and sends the *address* signal in parallel to the $n+1$ memories. In $\mathrm{CC}=2$ the n coefficients f_j^i are retrieved from the memories and sent to *Mux_f*. From $\mathrm{CC}=3$ to $\mathrm{CC}=n+2$ the n products $f_j^i x_i$ are performed sequentially and added in the accumulator within the *MAC* block. In the last clock cycle $(\mathrm{CC}=n+2)$ the constant term g_j is added to the *acc* output signal from the *MAC* block by a two-input combinatorial *Adder*, so that the value of the sPWAR function (f_{sPWAR}) is ready in output after a latency of $N_{\mathrm{CK}}=n+2$.

C. The mPWAR(B) and sPWAR(B) Architectures

In architecture mPWAR(B) the point location problem is solved exactly in the same way as in architecture mPWAR(A), the only difference being the way in which the mPWAR expression is evaluated. Indeed, in the mPWAR(B) architecture the instantiation of n multipliers allows to perform the n products $f_j^i x_i$ in parallel. From $\mathrm{CC}=1$ to $\mathrm{CC}=r$ the architecture performs the point location and produces the *address* signal in output from the *FSM* block. In $\mathrm{CC}=r+1$, the coefficients are retrieved from the *Memory_i* blocks and the value of the f_{mPWAR} expression is obtained immediately owing to the

presence of n multipliers and a combinatorial $n+1$-input adder.

The gain in latency, from $N_{\mathrm{CK}} = r+n+1$ to $N_{\mathrm{CK}} = r+1$, is obtained at the cost of a greater use of hardware resources, since n multipliers are required.

Also in the case of the sPWAR(B) architecture, the only difference with respect to architecture sPWAR(A) lies in the presence of n multipliers. Once the point location problem has been solved in $\mathrm{CC} = 1$, the sPWAR function evaluation can be performed in just one clock cycle, thus leading to an overall latency of $N_{\mathrm{CK}} = 2$.

V. Comparisons With Other PWA Implementations

Many different forms have been proposed in literature for the representation of PWA functions, but most of them are not tailored to digital circuit implementation. For comparison reasons, we focus on the PWAG and PWAS forms, as digital implementations are available for such forms.

In particular, in this section we provide a direct comparison between the new architectures for mPWAR and sPWAR functions introduced in the previous section and three digital architectures implementing PWAG and PWAS functions described in [22] for the PWAG case and in [10] (A and B) for the PWAS case.

We point out that in many cases prior to the implementation of PWAS, sPWAR and mPWAR functions, an approximation step is required as a general PWAG function is available (e.g., through explicit model predictive control), which can not be written exactly in the form of PWAS and/or PWAR functions. As a consequence, this approximation step provides PWAS or PWAR functions that are not the same as the original PWAG function and an approximation error intrinsic in the (mathematical) approximation problem is introduced. Note that this approximation error is not due to the circuit implementations. Clearly, there will be a tradeoff between the approximation error and the complexity of the resulting PWAS or PWAR functions obtained using the approximation method. For this reason, when comparing different implementations of PWA functions we must keep in mind that the overall effectiveness of the implemented function is determined both by the approximation method and the digital architecture adopted. For an overview of methods to approximate PWAG functions through canonical PWA functions defined on more regularly shaped partitions, the reader is referred to [14], [28], [29], [39], [45], [46] (simplicial partitions), and [30]–[32] (hyperrectangular partitions).

The considered PWA forms (PWAG, PWAS, sPWAR and mPWAR) differ in representation capability, expressed in terms of the size of the PWA function class the form belongs to, and complexity, related to the number of parameters and coefficients required to represent a given PWA function. In particular, the circuit implementations of PWAG forms can obviously represent any PWAG function, whereas the realizations based on a regular partitioning (simplicial or hyperrectangular) can represent only a subclass of PWAG functions that are affine over each simplex/hyperrectangle of the simplicial/hyperrectangular partition. Regarding complexity, the generic structure of PWAG typically results in more complex/less efficient implementations, while the particular structure of the partitions in PWAS and PWAR functions can be exploited for fast and efficient on-line implementations. Moreover, the proposed hyperrectangular implementations offer the possibility to realize discontinuous PWA functions, which is not the case of the PWAS architectures realized until now, and the multi-resolution version provides a partial solution to the "curse of dimensionality", because it requires fewer parameters to store.

These implementation benefits of mPWAR, sPWAR and PWAS functions over PWAG functions were exactly the motivations for the control community to investigate how the PWAG control laws (e.g., obtained by optimization-based control methodologies such as MPC) can be approximated by mPWAR, sPWAR and PWAS functions without sacrificing much of the control performance specification [28]–[32].

A. The PWAG Architecture

The architecture we have considered for the implementation of PWAG functions is mainly based on the digital circuit described in [22]. The main difference with respect to the mPWAR architecture is represented by the digital block responsible for the point location task. Indeed, the PWAG architecture solves this problem by a FSM implementation of a binary search tree whose depth is in general much larger than the depth of the orthogonal tree implemented by the mPWAR architecture. In the following we provide a brief description of the elements that determine the complexity of the PWAG architecture. For a more detailed description of the blocks making up the circuit, the reader is referred to [22].

Each edge of the polytopic partition in the PWAG case (see Fig. 1(a)) is an $(n-1)$-dimensional hyperplane in the form $\mathbf{h}_k\mathbf{x} + k_k = 0$, $k = 1, \ldots, E$, where $\mathbf{h}_k \in \mathbb{R}^n$, $k_k \in \mathbb{R}$ and E is the total number of edges of the polytopic partition. The next-step decision at each non-leaf node requires $n+2$ clock cycles for the evaluation of the inequality $\mathbf{h}_k\mathbf{x} + k_k \leq 0$, which determines the branch to be taken [22]. The total time required by the circuit to provide the output $f_{\mathrm{PWAG}}(\mathbf{x})$, i.e., the latency of the circuit, is $N_{\mathrm{CK}} = (d+1)(n+2)$, where d is the maximum depth of the binary search tree. An upper bound to the number of nodes of the tree and, consequently, to the number of states of the FSM is given by $\sum_{i=0}^{d} 2^i$ [22]. In the mPWAR architecture the time required by the next-step decision is independent of the problem dimensionality thus leading to a faster on-line search. Moreover, in the PWAG architecture the $n+1$ coefficients $h_k^1, h_k^2, \ldots, h_k^n, k_k$ must be stored for each of the edges determining the polytopic partition of the domain, thus increasing the memory requirements up to $(E+L_G)(n+1)$ cells.

B. The PWAS Architectures

For a simplicial partition, the algorithm usually adopted to locate the simplex χ_j a given input vector belongs to is based on Kuhn's lemmas [33], [41], [47] and is optimal with respect to the number of inputs [48]. The value of any PWAS function f_{PWAS} can be calculated as a linear interpolation of the values at the vertices of the simplex containing the input vector $\mathbf{x}$, i.e., as a linear interpolation of a subset of $n+1$ coefficients w_j:

$$f_{\mathrm{PWAS}}(\mathbf{x}) = \sum_{j \in J_x} \mu_j w_j \tag{5}$$

where $\mathbf{x} = \sum_{j \in J_x} \mu_j \mathbf{v}_j$ and $J_x \subset \{1, \ldots, N_v\}$ (N_v being the total number of vertices of the partition) is the set of indices

corresponding to the vertices $\mathbf{v}_j$ of the simplex $\mathbf{x}$ in the scaled domain S_S. The μ_j values are scalar interpolation weights and $w_j = f_{\mathrm{PWAS}}(\mathbf{v}_j)$, $j = 1, \ldots, N_v$.

The evaluation of a PWAS function through this algorithm requires three main elements:

1) a memory where the N_v coefficients w_j are stored;
2) a block that finds, for every given input $\mathbf{x}$, the set of indices J_x and the interpolation weights μ_j (this step requires a sorting algorithm [34]);
3) a block performing the weighted sum (5).

We assume that the f_{PWAS} function to be implemented is already scaled and defined over a n-dimensional compact domain S_S.

Each component of $\mathbf{x}$ is represented by a digital word of $b = p_i + s_i$ bits: p_i codes the integer part of x_i, s_i bits are used for the decimal part. We note here that in the point location strategy the role played by the first p_i bits in the PWAS case is similar to that played by the first t_i bits in the sPWAR case; they allow to find the hyperrectangle containing the current input variable.

However, the simplicial architecture requires also the sorting of the decimal parts of the input to find the correct simplex within the actual hyperrectangle. In summary, p_i determines the number of sub-intervals along each dimension ($m_{Si} = 2^{p_i} - 1$ for any i) as well as t_i determines the number of subdivisions along each dimensional component ($m_{Ri} = 2^{t_i}$).

Concerning the three elements listed above, the N_v weights are stored in an internal memory; the decreasing ordering of the decimal parts of $\mathbf{x}$ is obtained by using a sorter suitable for digital implementations [49], the weighted sum $\sum_{j \in J_x} \mu_j w_j$ is performed by the high-speed multipliers and accumulators internal to the FPGA. For a detailed description of the blocks making up the circuit, see [10].

We note here that the difference between architectures A and B proposed in [10] is that the former is serial, while the latter is completely parallel, thus requiring the instantiation of $n + 1$ *Multiplier* as well as $n + 1$ *Memory* blocks. For architecture A, the on-line computation time grows with $n + 4$ and the memory size increases according to the exponential law $\prod_{i=1}^{n}(m_{Si}+1)$. For architecture B, the latency of the circuit is 3 clock cycles while the memory size is $(n + 1)\prod_{i=1}^{n}(m_{Si} + 1)$. The serial architecture results in a slower but smaller circuit, while the parallel one achieves extremely small computation times but has larger memory requirements.

C. Comparisons

Summing up, the PWAS and sPWAR architectures are suitable for small-scale problems with high real-time requirements owing to their higher speed. The sPWAR architecture presents an extremely simple structure, since all the operations can be carried out with very simple functional blocks, and offers the further capability to implement discontinuous functions. On the other hand, both architectures are affected by the "curse of dimensionality", since the memory size grows exponentially with the input dimension n. The PWAG architecture has the great advantage of implementing every generic PWA function, but the latency of the circuit grows very fast for problems with a highly irregular domain partitioning or a large number of regions. The mPWAR architecture allows much shorter computation times as well as considerable memory savings with respect to the to the single-resolution approach. The main drawback in this case is due to the dimensions of the FSM, which can become too large for high-dimensional problems requiring a high level of refinement.

TABLE I
CIRCUIT PARAMETERS

n	number of input variables, i.e. dim($\mathbf{x}$)=n
L_G	number of polytopic regions for the PWAG architecture
E	number of edges defining the domain partition in the PWAG case
d	maximum depth of the binary search tree for the PWAG architecture.
L_M	number of multi-resolution hypercubic regions for the mPWAR architecture
r	refinement level for the mPWAR architecture, corresponding to the maximum depth of the orthogonal search tree
m_{Ri}	number of single-resolution subintervals along each dimension for the sPWAR architectures: $m_{Ri} = 2^{t_i}$
m_{Si}	number of single-resolution subintervals along each dimension for the PWAS architectures: $m_{Si} = 2^{p_i} - 1$

TABLE II
CIRCUIT PERFORMANCES

Architecture	N_{CK}	Memory cells	Max # of nodes	Multipliers
PWAG	$(d+1)(n+2)$	$(E + L_G)(n+1)$	$\sum_{i=0}^{d} 2^i$	1
PWAS(A)	$n+4$	$\prod_{i=1}^{n}(m_{Si}+1)$	NO FSM	1
PWAS(B)	3	$(n+1)\prod_{i=1}^{n}(m_{Si}+1)$	NO FSM	$n+1$
mPWAR(A)	$r+n+1$	$L_M(n+1)$	$\sum_{i=0}^{r-1} 2^{in}$	1
mPWAR(B)	$r+1$	$L_M(n+1)$	$\sum_{i=0}^{r-1} 2^{in}$	n
sPWAR(A)	$n+2$	$(n+1)\prod_{i=1}^{n} m_{Ri}$	NO FSM	1
sPWAR(B)	2	$(n+1)\prod_{i=1}^{n} m_{Ri}$	NO FSM	n

All the presented architectures have been obtained through fully parametric VHDL descriptions, thus allowing to represent different PWA functions simply by tuning few parameters and to automatically write the VHDL descriptions.

The most significant parameters for the implemented architectures are reported in Table I. Table II summarizes how the circuit performances are influenced by the most significant parameters.[4] These tables are a key result, as they provide a detailed comparison of the complexities of the considered digital implementations. The column N_{CK} in Table II shows the number of clock cycles required by each architecture to complete the computation of the output function. The column *Memory Cells* reports the number of elements that must be stored for each architecture. In the fourth column, we list the maximum number of nodes for the architectures implementing a FSM (this is a worst-case measure since not all the states would be instantiated in a typical implementation). The last column shows the number of multipliers instantiated for each implementation.

From the results reported in Table II it is possible to infer the complexity and performance of a PWAG, PWAS or PWAR implementation as the parameters reported in Table I change. In

[4]Timings and results for architectures PWAG, PWAS(A) and PWAS(B) slightly differ from those presented in [10], [22] due to changes in the VHDL code. In particular, we made a different choice for the implementation of the memory block of architecture PWAS(B), in order to allow fair comparisons. As a consequence of this choice, architectures PWAS(A) and PWAS(B) have now different memory sizes with respect to [10], [22].

TABLE III
DOUBLE INTEGRATOR

Architecture	ϵ	e	Slices (%)	Latency (μs)	Memory ($bits$)	Multipliers	Power (mW)
PWAG	0.00	0.000	6	1.6	3,348	1	52
PWAS(A)	0.45	0.022	7	0.3	3,072	1	50
PWAS(B)	0.45	0.022	7	0.15	9,216	3	50
mPWAR(A)	0.31	0.061	5	0.35	2,952	1	52
mPWAR(B)	0.31	0.061	5	0.25	2,952	2	52
sPWAR(A)	0.29	0.027	2	0.2	9,216	1	51
sPWAR(B)	0.29	0.027	1	0.1	9,216	2	51

the next section we provide an experimental verification of these results by the FPGA implementation of two MPC controllers.

VI. EXAMPLES

We consider two examples of PWA functions to quantitatively illustrate some of the aspects discussed in the previous sections. The first example is related to a two-dimensional system, which is a benchmark problem in the context of MPC. The second example is related to MPC of a three-dimensional PWA system [50]. In both examples, all the different PWA functions are implemented on a Xilinx Spartan III FPGA (xc3s200). The developed VHDL description has been realized with the software tool Xilinx ISE 12.3 and ModelSim has been used for simulation and debugging.

The original PWA control functions are obtained by explicit MPC (EMPC) [19], [20] algorithms and implemented using the PWAG architecture based on a binary search tree proposed in [22], as discussed shortly in Section V.A. The PWAS approximations have been obtained through the techniques described in [28] whereas the implementations rely on the architectures proposed in [10] and shortly described in Section V.B. The mPWAR and sPWAR approximations are based on the procedures described in [31] and implemented using the architectures proposed in this paper.

In order to provide a comparison between the approximation capabilities of the different architectures, we use two different quantitative measures of the approximation errors, the first one being

$$\epsilon = \max_{\mathbf{x}\in S} |u^*(\mathbf{x}) - \hat{u}(\mathbf{x})|, \tag{6}$$

where u^* is the PWAG optimal control law, $\hat{u}$ is the PWAS, mPWAR or sPWAR approximation. Hence ϵ is the maximum approximation error for each case.

The second quantitative measure is the relative error e evaluated over a uniform grid of 100 points for each domain axis, given by

$$e = \frac{\sum_{j=1}^{100^n} |u^*(\mathbf{x}_j) - \hat{u}(\mathbf{x}_j)|}{\sum_{j=1}^{100^n} |u^*(\mathbf{x}_j)|}, \tag{7}$$

where $\mathbf{x}_j$ are the sample points.

A. Double Integrator

1) Model and Control Laws: The discrete-time double integrator with sampling time of one time unit can be represented as follows [20]:

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k$$
$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}_k,$$

where $\mathbf{x}_k \in \mathbb{R}^2$ denotes the state variable and u_k the control input at discrete time $k \in \mathbb{N}$. The control objective is to regulate the system to the origin under the hard constraint $-1 \leq u_k \leq 1$, $k \in \mathbb{N}$. The state domain is given by[5] $S = \{\mathbf{x} \in \mathbb{R}^n : -8 \leq x_1 \leq 8, -4 \leq x_2 \leq 4\}$. An optimal explicit control law (PWAG) $u_k = u^*(\mathbf{x}_k)$, $k \in \mathbb{N}$ has been obtained with EMPC as in [51] using the Matlab Hybrid Toolbox (v1.2.8) [52].[6] The PWAG control law is characterized by $L_G = 33$ regions and $E = 60$ edges while the off-line search tree results in 87 nodes with a maximum depth of $d = 7$.

In order to provide a fair comparison between the approximations $\hat{u}_{\mathrm{PWAS}}$, $\hat{u}_{\mathrm{sPWAR}}$ and $\hat{u}_{\mathrm{mPWAR}}$, we have selected coherent values for the architectures parameters. The PWAS approximation is obtained with $m_{Si} = 15$ subintervals along each dimension, while for the sPWAR case we choose $t_i = 4$, $i = 1, 2$, i.e., $m_{Ri} = 16$. The mPWAR approximation, characterized by a maximum refinement level of $r = 4$, is defined over $L_M = 82$ multi-resolution hyperrectangles, and the orthogonal search tree is made up of 53 nodes. The fact that the number of regions is larger than the number of nodes is due to the enhancement described in Section IV.A which allows to avoid the instantiation of the nodes belonging to the last level of refinement.

The obtained approximation errors are shown in Table III. The time evolution of the system controlled with the optimal (PWAG) and the approximated (PWAS, sPWAR, mPWAR) control laws is shown in Fig. 7. Panels (a) and (b) show the evolution of the state variables, whereas panel (c) shows the optimal and the approximated control laws. The black solid lines are related to the optimal MPC control, the gray solid lines correspond to the PWAS approximations while the dashed lines are related

[5]For simplicity we use the notation $\mathbf{x}$ to refer to the state variable in the original non-scaled domain.

[6]Using cost function $J = x'_N P x_N + \sum_{k=0}^{N-1} x'_k Q x_k + u'_k R u_k$, where $Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $R = 0.1$, P is chosen as the solution of the LQR problem with weights Q, R, and the horizon is taken $N = 4$.

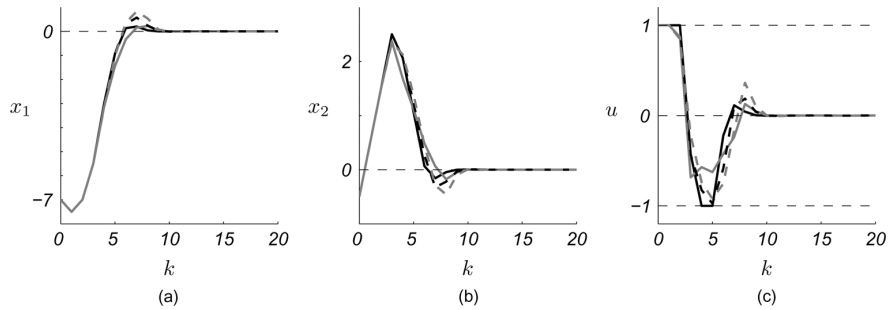

Fig. 7. Time evolution of the double integrator system subject to the optimal MPC PWAG control (black solid lines), or to its PWAS (gray solid lines), sPWAR (black dashed lines), and mPWAR (gray dashed lines) approximations. The horizontal dashed lines in panel (c) are constraints to be fulfilled. The discrete time-step k is reported on the horizontal axis.

TABLE IV
3D EXAMPLE

Architecture	ϵ	e	Slices (%)	Latency (μs)	Memory ($bits$)	Multipliers	Power (mW)
PWAG	0.00	0.000	53	3.25	24,624	1	58
PWAS(A)	0.75	0.065	12	0.35	49,152	1	52
PWAS(B)	0.75	0.065	12	0.15	196,608	4	58
mPWAR(A)	0.30	0.042	79	0.45	74,304	1	64
mPWAR(B)	0.30	0.042	78	0.3	74,304	3	65
sPWAR(A)	0.45	0.074	2	0.25	196,608	1	57
sPWAR(B)	0.45	0.074	2	0.1	196,608	3	57

to the sPWAR (black) and mPWAR (gray) approximations. The constraints to be fulfilled are represented by horizontal dashed lines where appropriate.

2) Circuit Implementations: In all cases we have used $b = 12$ bits to represent the data. The performance of the architectures on the chosen FPGA are summarized in Table III. We remark here that the error due to the 12-bit representation is negligible in all cases with respect to the approximation errors shown in Tables III and IV [53].

The column *Slices* reports the percentage of space occupied on the device after the place-and-route process performed in Xilinx ISE. The *Latency* values are obtained by multiplying the number of clock cycles needed to perform the PWA computation, reported in Table II, by a clock period of 50 ns. The chosen clock frequency, i.e., 20 MHz, allows to obtain correct results on each of the digital architectures after the post-route simulation on the FPGA. The values expressed in the column *Memory* are obtained by multiplying the number of required cells, reported in Table II, by the number of bits used to represent the coefficients in memory, i.e., $b = 12$. The last column shows the total power consumption estimated by Xilinx XPower Estimator (XPE) 11.1 after the place-and-route process performed with a clock frequency of 20 MHz and an external temperature of 25°C.

In this simple example all the architectures use a limited amount of the resources available on the FPGA, but the digital architectures implementing PWAS and PWAR functions already offer a considerable gain in terms of computation time with respect to the PWAG architecture. Moreover, the mPWAR architectures require smaller memories if compared to the single-resolution approaches.

B. 3D PWA System

1) Model and Control Laws: The second example is the control of a third-order PWA system from [50], defined by

$$\mathbf{x}_{k+1} = \begin{cases} \begin{bmatrix} 1 & 0.5 & 0.3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_k + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, & \text{if } x_2 \le 1 \\ \begin{bmatrix} 1 & 0.2 & 0.3 \\ 0 & 0.5 & 1 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_k + \begin{bmatrix} 0.3 \\ 0.5 \\ 0 \end{bmatrix}, & \text{if } x_2 \ge 1 \end{cases}$$

The state domain is given by $S = \{\mathbf{x} \in \mathbb{R}^n : -10 \le x_1 \le 10, -5 \le x_2 \le 5, -10 \le x_3 \le 10\}$. The control constraint is $-1 \le u_k \le 1$, $k \in \mathbb{N}$. The explicit solution $u^* = u^*(\mathbf{x}_k)$ has been obtained as in [54] using the Matlab Multi-Parametric toolbox (v2.6.3) [55].[7] The resulting control law $u^*(\mathbf{x}_k) = \mathbf{f}_j'\mathbf{x}_k + g_j, \mathbf{x}_k \in \chi_j$ is a PWAG function of the state $\mathbf{x}_k$, defined over the domain S partitioned into $L_G = 264$ polytopes $\chi_j, j = 1, \ldots, L_G$ (after simplification of the original 719) by $E = 249$ edges. The binary search tree computed offline results in 949 nodes with a maximum depth of $d = 12$. The approximated PWAS control function is defined over a domain partitioned into $m_{S1} = 31, m_{S2} = 7$ and $m_{S3} = 15$ subintervals, where the subscript stands for the input component, thus implying 19530 simplices and 4096 vertices.

The sPWAR approximation is characterized by $m_{R1} = 32$, $m_{R2} = 8$, $m_{R3} = 16$, thus leading to a total of 4096 hy-

[7]Using cost function $J = \sum_{k=0}^{N-1} \|Qx_k\|_\infty + \|Ru_k\|_\infty$, with $Q = I$, $R = 0.1$ (no stability enforced). The horizon length is chosen $N = 3$.

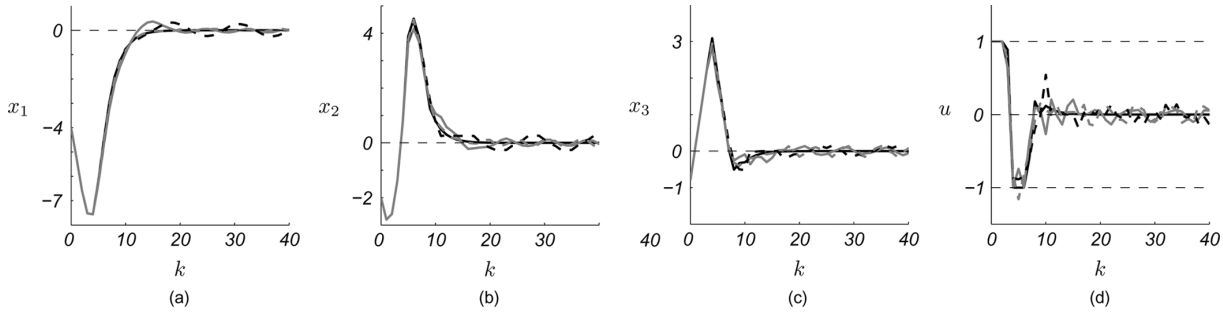

Fig. 8. Time evolution of the third-order PWA system subject to the optimal MPC PWAG control (black solid lines), or to its PWAS (gray solid lines), sPWAR (black dashed lines), and mPWAR (gray dashed lines) approximations. The horizontal dashed lines in panel (d) are constraints to be fulfilled. The discrete time-step k is reported on the horizontal axis.

perrectangular regions and 16384 coefficients. In the mPWAR case, the off-line tree has a maximum depth of $r = 5$ and is made up of 1337 nodes, and the input domain is partitioned into $L_M = 1548$ regions.

The approximation errors, computed according to (6) and (7), are summarized in Table IV. From Fig. 8 we can see how in this case the approximated control laws result in slight oscillations around the origin, but they are still effective in stabilizing the state to a region close to the origin. Because no (asymptotic) stability requirements were put on the approximations, the approximation errors around the origin are allowed to remain relatively high, resulting in these oscillations. Note that in certain cases it is possible to impose stability on the PWA approximations [29], [31], [32], thereby removing these oscillations.

2) Circuit Implementations: Also in this case we choose $b = 12$ bits to represent the data.

The PWAG architecture occupies around half of the space available on the FPGA board and the memory requirements are still quite small since the number of polytopic regions of the domain is limited. On the other hand, the latency of the circuit is high if compared to the other architectures, since the maximum depth of the binary search tree associated with the domain partition is $d = 12$ and the evaluation of an affine expression requiring $n + 2$ clock cycles is performed at each step within the search tree.

The mPWAR architectures offer the advantage of a much faster computation at the cost of a slightly larger memory and a larger space occupation on the FPGA. The sPWAR architectures are much simpler than the other implementations and the space occupied on the circuit is extremely small. In the sPWAR(B) implementation the latency is more than one order of magnitude less than the PWAG case but the great advantages offered by the sPWAR architectures are balanced by a significant increase in the memory requirements.

The PWAS architectures still offer the advantage of a lower latency if compared to the PWAG case and the memory requirements are smaller than for the sPWAR case, especially in the PWAS(A) implementation, at the cost of a larger maximum approximation error.

In conclusion, the considered examples provide a validation of Table II and demonstrate the advantages and disadvantages offered by the different implementations.

VII. Conclusion

In this paper we have introduced two new digital architectures for the implementation of PWA functions defined over both single-resolution and multi-resolution hyperrectangular partitions. The proposed architectures allow a significant reduction of the computation time of the PWA function with respect to previous solutions implementing generic PWA functions. Moreover, the chosen hyperrectangular partition allows the realization of very simple digital architectures in the single-resolution case, which makes our solution more appealing for large-scale problems with respect to previous digital architectures implementing PWA functions defined over domain partitioned into different types of polytopes. In order to mitigate the curse of dimensionality, which affects the architectures for the evaluation of PWA functions defined over regularly-shaped partitions, we have introduced a multi-resolution architecture, which allows a better control over the memory requirements at the cost of a slightly higher latency and a larger circuit complexity. These conclusions were illustrated by two numerical case studies.

The proposed architectures are completely reconfigurable and the chosen FPGA implementation allows customization of the hardware at an attractive price even in low quantities and this makes them effective for cost and time-to-market savings when making individual configurations of standard products [56]. These advantages make the proposed architectures of particular interest for the implementation of optimization-based control methodologies such as EMPC for fast, small-size and low-power applications.

Future research will involve a further integration between the algorithms for the control approximation and the circuit implementation, as this will lead to a highly automated procedure for the creation of embedded controllers, making the quantitative properties of the designed controllers predictable without having to realize the circuit implementations. This integration should lead to the development of a toolbox to support the whole design chain.

References

[1] T. Mestl, C. Lemay, and L. Glass, "Chaos in high dimensional neural and gene networks," *Physica D*, vol. 98, no. 1, pp. 33–52, Nov. 1996.

[2] J. Eckmann and E. Moses, "Curvature of co-links uncovers hidden thematic layers in the world-wide web," *P. Nat. Acad. Sci. USA*, vol. 99, no. 9, pp. 5825–5829, Apr. 2002.

[3] W. B. Arthur, "Complexity and the economy," *Science*, vol. 284, no. 5411, pp. 107–109, Apr. 1999.

[4] B. Carreras, V. Lynch, I. Dobson, and D. Newman, "Critical points and transitions in an electric power transmission model for cascading failure blackouts," *Chaos*, vol. 12, no. 4, pp. 985–994, Dec. 2002.

[5] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003.

[6] T. A. Johansen, W. Jackson, R. Schreiber, and P. Tøndel, "Hardware synthesis of explicit model predictive controllers," *IEEE Trans. Contr. Syst. Technol.*, vol. 15, no. 1, pp. 191–197, Jan. 2007.

[7] R. Rovatti, A. Ferrari, and M. Borgatti, , A. Kandel and G. Langholz, Eds., "Automatic implementation of piecewise-linear fuzzy systems addressing memory-performance trade-off," in *Fuzzy Hardware: Architectures and Applications*. Norwell, MA: Kluwer Academic, 1998, pp. 159–179.

[8] P. Echevarria, M. Martínez, J. Echanobe, I. del Campo, and J. Tarela, "Digital hardware implementation of high dimensional fuzzy systems," in *Applications of Fuzzy Sets Theory*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2007, pp. 245–252.

[9] M. Storace and M. Parodi, "Towards analog implementations of PWL two-dimensional non-linear functions," *Int. J. Circuit Theory and Applications*, vol. 33, no. 2, pp. 147–160, Mar.–Apr. 2005.

[10] M. Storace and T. Poggi, "Digital architectures realizing piecewise-linear multi-variate functions: Two FPGA implementations," *Int. J. Circuit Theory and Applications*, vol. 39, pp. 1–15, 2011.

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

[12] D. M. W. Leenaerts and W. M. G. van Bokhoven, *Piecewise Linear Modelling and Analysis*. Boston, MA: Kluwer Academic, 1998.

[13] P. Julián, A. Desages, and B. D'Amico, "Orthonormal high level canonical PWL functions with applications to model reduction," *IEEE Trans. Circuits Syst. I: Fund. Theory Applicat.*, vol. 47, no. 5, pp. 702–712, May 2000.

[14] M. Storace, P. Julián, and M. Parodi, "Synthesis of nonlinear multiport resistors: A PWL approach," *IEEE Trans. Circuits Syst. I: Fund. Theory Applicat.*, vol. 49, no. 8, pp. 1138–1149, Aug. 2002.

[15] P. Julián, "The complete canonical piecewise-linear representation: Functional form for minimal degenerate intersections," *IEEE Trans. Circuits Syst. I: Fund. Theory Applicat.*, vol. 50, no. 3, pp. 387–396, Mar. 2003.

[16] C. Wen and X. Ma, "A canonical piecewise-linear representation theorem: Geometrical structures determine representation capability," *IEEE Trans. Circuits Syst. II: Expr. Briefs*, vol. 58, no. 12, pp. 936–940, Dec. 2011.

[17] J. W. Schwartz, "Piecewise linear servomechanisms," *Trans. AIEE*, vol. 72, pp. 401–405, 1953.

[18] E. D. Sontag, "Nonlinear regulation: The piecewise linear approach," *IEEE Trans. Automat. Contr.*, vol. 26, pp. 346–358, 1981.

[19] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming: The explicit solution," *IEEE Trans. Automat. Contr.*, vol. 47, no. 12, pp. 1974–1985, 2002.

[20] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[21] M. de Berg, M. van Kreveld, M. Overmars, and O. Cheong, *Computational Geometry*, 3rd ed. Berlin, Germany: Springer-Verlag, 2008.

[22] A. Oliveri, A. Oliveri, T. Poggi, and M. Storace, "Circuit implementation of piecewise-affine functions based on a binary search tree," in *Proc. 18th Eur. Conf. Circuit Theory and Design (ECCTD'09)*, Antalya, Turkey, Aug. 23–27, 2009, pp. 145–148.

[23] C. Wen, X. Ma, and B. E. Ydstie, "Analytical expression of explicit MPC solution via lattice piecewise-affine function," *Automatica*, vol. 45, no. 12, pp. 910–917, 2009.

[24] F. Bayat, T. A. Johansen, and A. A. Jalali, "Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC," *IEEE Trans. Contr. Syst. Technol.*, vol. 20, no. 3, pp. 632–640, May 2012.

[25] F. Bayat, T. A. Johansen, and A. A. Jalali, "Using hash tables to manage time-storage complexity in point location problem: Application to explicit MPC," *Automatica*, vol. 47, pp. 571–577, 2011.

[26] M. C. Martinez-Rodriguez, I. Baturone, and P. Brox, "Circuit implementation of piecewise-affine functions based on lattice representation," in *Proc. 20th Eur. Conf. Circuit Theory and Design (ECCTD)*, Linkoping, Sweden, Aug. 29–31, 2011, pp. 644–647.

[27] M. C. Martinez-Rodriguez, I. Baturone, and P. Brox, "Design methodology for FPGA implementation of lattice piecewise-affine functions," in *Proc. 2011 Int. Conf. Field-Programmable Technology (FPT)*, Dec. 12–14, 2011, pp. 1–4.

[28] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations," *IEEE Trans. Automat. Contr.*, vol. 56, pp. 2883–2937, Dec. 2011.

[29] S. Summers, C. N. Jones, J. Lygeros, and M. Morari, "A multiresolution approximation method for fast explicit model predictive control," *IEEE Trans. Automat. Contr.*, vol. 56, no. 11, pp. 2530–2541, 2011.

[30] T. A. Johansen and A. Grancharova, "Approximate explicit constrained linear model predictive control via orthogonal search tree," *IEEE Trans. Automat. Contr.*, vol. 48, no. 5, pp. 810–815, 2003.

[31] B. A. G. Genuit, L. Lu, and W. P. M. H. Heemels, "Approximation of explicit model predictive control using regular piecewise affine functions: An input-to-state stability approach," *IET Control Theory and Applications*, vol. 6, no. 8, pp. 1015–1028, May 2012.

[32] L. Lu, W. P. M. H. Heemels, and A. Bemporad, "Synthesis of low-complexity stabilizing piecewise affine controllers: A control-Lyapunov function approach," in *Proc. 50th IEEE Conf. Decision and Control and Eur. Control Conf.*, Orlando, FL, 2011, pp. 1227–1232.

[33] H. Kuhn, "Some combinatorial lemmas in topology," *IBM J. Research and Development*, vol. 4, pp. 518–524, 1960.

[34] M. Parodi, M. Storace, and P. Julián, "Synthesis of multiport resistors with piecewise-linear characteristics: A mixed-signal architecture," *Int. J. Circuit Theory and Applications*, vol. 33, no. 4, pp. 307–319, Jul.–Aug. 2005.

[35] M. Di Federico, T. Poggi, P. Julián, and M. Storace, "Integrated circuit implementation of multi-dimensional piecewise-linear functions," *Digital Signal Process.*, vol. 20, pp. 1723–1732, Dec. 2010.

[36] T. Poggi, F. Comaschi, and M. Storace, "Digital circuit realization of piecewise affine functions with non-uniform resolution: Theory and FPGA implementation," *IEEE Trans. Circuits Syst. II: Expr. Briefs*, vol. 57, no. 2, pp. 131–135, Feb. 2010.

[37] M. Floater, E. Quak, and M. Reimers, "Filter bank algorithms for piecewise linear prewavelets on arbitrary triangulations," *J. Comput. Appl. Math.*, vol. 119, pp. 185–207, 2000.

[38] E. Suter, "A strategy for the construction of piecewise linear prewavelets over type-1 triangulations in any space dimension," in *Mathematical Methods for Curves and Surfaces: Oslo 2000*. Nashville, TN: Vanderbilt University, 2001.

[39] M. Parodi, M. Gaggero, and M. Storace, "Piecewise linear approximations of multivariate functions: A multiresolution-based compression algorithm suitable for circuit implementation," *Appl. Numer. Math.*, vol. 60, pp. 924–933, 2010.

[40] P. Julián, A. Desages, and O. Agamennoni, "High-level canonical piecewise linear representation using a simplicial partition," *IEEE Trans. Circuits Syst. I: Fund. Theory Applicat.*, vol. 46, no. 4, pp. 463–480, Apr. 1999.

[41] H. Kuhn, "Simplicial approximation of fixed points," in *Proc. Nat. Acad. Sci. USA*, 1968, vol. 61, pp. 1238–1242.

[42] R. Rovatti, C. Fantuzzi, and S. Simani, "High-speed DSP-based implementation of piecewise-affine and piecewise-quadratic fuzzy systems," *Signal Process.*, vol. 80, pp. 951–963, 2000.

[43] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies, Annals of Mathematical Studies*, vol. 34, pp. 129–153, 1956.

[44] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Syst. Tech. J.*, vol. 34, pp. 1045–1079, 1955.

[45] M. Storace, L. Repetto, and M. Parodi, "A method for the approximate synthesis of cellular non-linear networks. Part 1: Circuit definition," *Int. J. Circuit Theory and Applications*, vol. 31, pp. 277–297, 2003.

[46] L. Repetto, M. Storace, and M. Parodi, "A method for the approximate synthesis of cellular non-linear networks. Part 2: Circuit reduction," *Int. J. Circuit Theory and Applications*, vol. 31, pp. 299–313, 2003.

[47] M. Chien and E. Kuh, "Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision," *IEEE Trans. Circuits Syst.*, vol. 24, pp. 305–317, 1977.

[48] R. Rovatti, M. Borgatti, and R. Guerrieri, "A geometric approach to maximum-speed n-dimensional continuous linear interpolation in rectangular grids," *IEEE Trans. Computers*, vol. 47, no. 8, pp. 894–899, 1998.

[49] V. Pedroni, "Compact hamming-comparator-based rank order filter for digital VLSI and FPGA implementations," in *Proc. 2004 IEEE Int. Symp. Circuits and Systems (ISCAS'2004)*, Vancouver, Canada, May 23–26, 2004, pp. 585–588.

[50] D. Q. Mayne and S. Rakovic, "Model predictive control of discrete piecewise affine discrete-time systems," *Int. J. Robust and Nonlinear Control*, vol. 13, no. 3–4, pp. 261–279, Mar.–Apr. 2003.

[51] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit mpc solutions," *Automatica*, vol. 39, no. 5, pp. 489–497, 2003.

[52] A. Bemporad, *Hybrid Toolbox—User's Guide*, 2004 [Online]. Available: http://www.dii.unisi.it/hybrid/toolbox

[53] T. Poggi, I. Baturone, and M. Storace, "Selection of architectures for PWA functions implementation," Moby-Dic, 7th Framework Programme, Deliverable D4.1, Nov. 2010 [Online]. Available: http://www.mobydic-project.eu/communication/deliverables

[54] F. Borrelli, M. Baotic, A. Bemporad, and M. Morari, "An efficient algorithm for computing the state feedback optimal control law for discrete time hybrid systems," in *Proc. 2003 Amer. Control Conf. (ACC'03)*, Denver, Colorado, Jun. 2003, pp. 4717–4722.

[55] M. Kvasnica, P. Grieder, M. Baotic, and M. Morari, "Multi-parametric toolbox (MPT)," *Hybrid Systems: Computation and Control*, vol. 2993, pp. 121–124, 2004.

[56] M. Thompson, "FPGAs accelerate time to market for industrial designs," *EE Times* 2004 [Online]. Available: http://www.us.design-reuse.com/articles/8190/fpgas-accelerate-time-to-market-for-industrial-designs.html

Francesco Comaschi was born in Novi Ligure, Italy, on April 23, 1987. He received the M.Sc. degree (*summa cum laude*) in electronic engineering in December 2011 from the University of Genoa, Italy. He was a visiting researcher at the Department of Mechanical Engineering at the Eindhoven University of Technology (TU/e) in 2011. Currently, he is pursuing the Ph.D. in the Department of Electrical Engineering at the Eindhoven University of Technology, The Netherlands.

His research interests include digital circuits design, real-time embedded systems, analysis and synthesis of predictable hardware/software systems.

Bart A. G. Genuit was born in Alkmaar, The Netherlands, in 1985. He received the M.Sc. degree in mechanical engineering from the Eindhoven University of Technology, The Netherlands, in 2011.

Currently, he is a researcher in the Hybrid and Networked Systems group of the Department of Mechanical Engineering of the Eindhoven University of Technology. His research interests include model predictive control, optimization and adaptive control.

Alberto Oliveri received the M.Sc. degree in electronic engineering from the University of Genoa, Genova, Italy, in September 2009. He is currently pursuing the Ph.D. degree in electrical engineering in the Department of Naval, Electric, Electronic and Telecommunications Engineering, University of Genoa.

His main research interests include piecewise-affine approximation of nonlinear functions, circuit implementation of explicit model predictive control, design and circuit implementation of piecewise-affine virtual sensors. He is also interested in bifurcation analysis and detection of invariant manifolds in nonlinear dynamical systems.

W. P. Maurice H. Heemels received the M.Sc. degree in mathematics (with highest distinction) and the Ph.D. degree (*summa cum laude*) from the Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands, in 1995 and 1999, respectively.

From 2000 to 2004, he was with the Electrical Engineering Department, TU/e, as an Assistant Professor and from 2004 to 2006 with the Embedded Systems Institute (ESI) as a Research Fellow. Since 2006, he has been with the Department of Mechanical Engineering, TU/e, where he is currently a Full Professor in the Hybrid and Networked Systems Group. He has held visiting research positions at the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (2001) and at the University of California at Santa Barbara (2008). In 2004, he was also at the Research and Development Laboratory, Océ, Venlo, The Netherlands. His current research interests include hybrid and nonsmooth dynamical systems, networked control systems and constrained systems including model predictive control.

Dr. Heemels is an Associate Editor for the journal *Automatica* and was an Associate Editor for *Nonlinear Analysis: Hybrid Systems*.

Marco Storace (M'01) was born in Genoa, Italy, in 1969. He received the "Laurea" (M.Sc.) five-year degree (*summa cum laude*) in electronic engineering in March 1994 and the Ph.D. degree in electrical engineering in April 1998, both from the University of Genoa, Italy.

Since 1999 he has been with the Engineering School, University of Genoa, where he is currently a Full Professor. He held visiting research positions at the EPFL, Lausanne, Switzerland, in 1998 and 2002. His current research interests include modelling and analysis of nonlinear dynamical systems, methods for the circuit synthesis of nonlinear systems (e.g., embedded control systems, neurons), bifurcation analysis. He is the author or coauthor of about 100 scientific papers, more than half of which have been published in international journals.

Dr. Storace served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II (2008–2009) and is a member of the IEEE Technical Committee on Nonlinear Circuits and Systems (TC-NCAS). He is coordinator of the Moby-Dic European project (FP7, CNECT-ICT-248858, www.mobydic-project.eu).