Circuit Implementation of Piecewise-Affine Functions Based on Lattice Representation

M. C. Martínez-Rodríguez[1,2], I. Baturone[1,2], P. Brox[2]
[1] Department of Electronics and Electromagnetism, University of Seville
[2] Microelectronics Institute of Seville (IMSE-CNM), Spanish National Research Council (CSIC)
Seville, Spain
Email: macarena@imse-cnm.csic.es

***Abstract*—This paper introduces a digital architecture to implement piecewise-affine (PWA) functions based on representation methods from the lattice theory. Given an explicit and continuous PWA function, the parameters required to implement the lattice approach can be obtained by an off-line preprocessing that can be automated. Other advantages of the proposal are that it implements a continuous PWA function with potentially no errors and the minimum number of parameters to store. This has been proven experimentally by implementing the proposal in a Xilinx FPGA and comparing its performance with other implementations, all of them addressing a typical non linear control problem.**

I. Introduction

A piecewise-affine (PWA) function, $f_{PWA} : D \rightarrow \mathbb{R}$, provides a linear (affine) output for each region in which the input domain, D, is partitioned ($D \subset \mathbb{R}^n$):

$$f_{PWA}(x) = [\ x^T \quad 1\ ]^T\ \phi_i, \quad \forall x \in P_i \tag{1}$$

where $\phi_i \in \mathbb{R}^n$, and P_i are P non overlapping regions, called polytopes, that induce a polyhedral partition of the domain.

Each polytope is a closed set of points delimited by E edges:

$$P = \{\ x \in \mathbb{R}^n : h_j^T x + k_j \leq 0\}, \quad j = 1 \dots E \tag{2}$$

where $h_j \in \mathbb{R}^n$, $k_j \in \mathbb{R}$, and the E edges are (n-1)-dimensional hyper-planes in the form $h_j^T x + k_j = 0$.

Since PWA functions can approximate any non linear function, they have been employed in many application domains. In the field of circuit theory, PWA models have been very useful for analyzing non linear circuits (in particular for its low computational cost) [1]. In the field of circuit design, many approaches have been reported to implement PWA functions so as to evaluate non linear functions with small size, low power consumption, and high speed. While the first proposals were analog solutions [2], several digital architectures have been proposed recently [3]-[9]. In particular, digital implementation of PWA functions plays a relevant role in the embedded control area. Since PWA functions can be used to approximate any non linear function, there is an active research in synthesizing PWA controllers using Lyapunov-based methods and, specially, model predictive control (MPC). In MPC, the control action is obtained by solving a finite horizon open-loop optimal control problem at each sample time [10].

Analog implementations of PWA functions follow mainly function expansion models, such as canonical or simplicial forms. Digital ones follow mainly generic and simplicial forms. To the best of our knowledge, this is the first digital implementation following lattice PWA forms. The proposed architecture has been implemented in a Xilinx Spartan 3 FPGA and proven to solve a typical non linear control problem.

The paper is organized as follows. Section II briefly summarizes the digital implementations of PWA functions reported till now. Section III reviews how lattice PWA forms can approximate any continuous function and how they can be derived. The digital architecture proposed to implement them is exposed in Section IV. The case study of designing a PWA controller with the proposed approach is described and compared with other reported approaches in Section V. Finally, some conclusions are given in Section VI.

II. Digital Implementation of PWA functions

A. Generic PWA implementation

To determine the location of the input, a solution is the comparison of the input variable (x) with all the edges of each polytope. However, this implies a high computational cost. As an alternative to avoid this combinatory search, the authors in [11] propose to build off-line a binary search tree, where each non-leaf node represents an edge and each leaf represents the index of a polytope. Exploring the tree from the root to a leaf makes it possible to locate the polytope that contains the input x. Once the polytope is located, the affine function associated with the polytope is evaluated.

A digital architecture to implement this approach is described in [3]. This architecture consists of a serial input acquisition block, a finite state machine block that implements the binary search tree, a memory that stores the coefficients of the edges and the functions, and a multiplier-accumulator block that calculates serially the expression for the edge and the output.

The number of coefficients to store is $(n+1) \times (E+P)$. The time invested to calculate the value of the function is $[n+(n+2)\times treeDepth]T_{CK}$, being T_{CK} the clock period and $treeDepth$ the depth of the search tree. This implementation may be not adequate for certain PWA functions that require a highly deep tree and many parameters to store.

B. Simplicial PWA implementation

A piecewise-affine simplicial (PWAS) architecture was proposed in [12] as a good trade-off between approximation capability and circuit complexity. The domain D is divided into simplexes as follows; every component of D is divided into m subintervals, and each resulting hyper-rectangle is partitioned into $n!$ non overlapping simplexes. The coordinates of the corner of the hyper-rectangle closest to the origin that contains a given point x can be found by extracting the integer part of x. The exact simplex is coded by the decimal part of x.

Several digital architectures have been reported in the literature implementing this approach [4]-[8]. They consist of a memory that stores the values of the function at the simplexes vertices, a block to find the simplex within the hyper-rectangle that the point belongs to, and a multiplier-accumulator or a weighted sum block that calculates the affine expression for the output.

The time to calculate the value depends on the architecture and goes from one T_{CK} to $[log_2(m+n)]T_{CK}$ [7]. The number of coefficients to store is $(m+1)^n$. The main drawback of this implementation is the curse of dimensionality, that is, the complexity grows exponentially with the number n of inputs.

C. Hierarchical PWA implementation

The hierarchical PWA implementation is proposed in [9] to reduce the possible complexity of the above approaches. A complex PWA controller can be obtained by interconnecting simple PWA modules of few inputs or small number of partitions. The modules are directly connected among them or with addition, subtraction, minimum or maximum as the only operators.

The main problem of the hierarchical PWA implementation is that there is not a standard configuration of modules. Each application needs to be studied separately.

III. Lattice PWA Approach

The lattice representation selects properly the affine function of each piece without taking into account the boundaries of the pieces explicitly. According to [14], any continuous and explicit PWA function, $f_{PWA}(x)$, can be represented by a lattice PWA function $L(x|\phi,\psi)$, such that $f_{PWA}(x) = L(x|\phi,\psi)$ in the form:

$$L(x|\phi,\psi) = \min_{1\le i\le P}\left\{ max_{\substack{1\le j\le P\\ \psi_{ij}=1}}\{l(x|\phi_j)\}\right\}, \forall x\in\mathbb{R}^n \quad (3)$$

where $\phi = [\phi_1, ..., \phi_n]^T$ is a $P\times(n+1)$ parameter matrix whose rows are the coefficients of the affine functions, $l_j(x)$, of the P polytopes, and $\psi = [\psi_{ij}]$ is a $P\times P$ zero-one structure matrix defined as follows.

Assume that P_i,P_j are two n-dimensional polytopes where $l(x|\phi_i)$ and $l(x|\phi_j)$ are the values of the local affine functions corresponding to those polytopes, then:

$$\psi_{ij} = \begin{cases} 1 & \text{if } l(v_k|\phi_i) \ge l(v_k|\phi_j), 1\le k\le k_i \\ 0 & \text{otherwise}\end{cases} \quad (4)$$

where v_k are the vertices of P_i with $1\le k\le k_i$, being $k_i\in Z^+$, the number of vertices of P_i.

Given a continuous and explicit PWA function, the authors in [14] provide an algorithm to find the simplest lattice representation with the form in (3). The starting step is to calculate the explicit PWA function, recording the local affine functions, the constrained inequalities, and the vertices of each region, so as to obtain the parameter, ϕ, and structure, ψ, matrixes. The subsequent steps are to simplify the matrixes as follows.

A. Row simplification

Let ψ_i, ψ_j be rows of ψ. If the inequation $\psi_i - \psi_j \le 0$ holds for any $i,j\in\{1,\dots,P\}$, the row ψ_j is redundant in ψ. Hence, there is a simplified structure matrix $\tilde{\psi}\in\mathbb{R}^{(P-1)\times P}$, such that $L(x|\phi,\psi) = L(x|\phi,\tilde{\psi})$ where and $\psi\in\mathbb{R}^{P\times P}$=[$\psi_1$, ..., ψ_P]^T and $\tilde{\psi} = [\psi_1,\dots,\psi_{j-1},\psi_{j+1},\dots,\psi_P]^T$.

B. Column simplification

Given the structure matrix $\psi = [\psi_{ij}]^{P\times P}$, the dual structure matrix, $\hat{\psi} = [\hat{\psi}_{ij}]^{P\times P}$, is defined as follows:

$$\hat{\psi}_{ij} = \begin{cases} 1 & \text{if } l(v_k|\phi_i) \le l(v_k|\phi_j), 1\le k\le k_i \\ 0 & \text{otherwise}\end{cases} \quad (5)$$

a) Given any $i,j,k\in\{1,\dots,P\}$, if $k,j,\in I_i$, and $\hat{\psi}_{i,j} = 1$ then $\psi_{ij} = 0$, where $I_i = \{k|l(x|\phi_k)\le l(x|\phi_i), \forall x\in P_i\}$

b) If $\psi_{ij} = 0, \forall 1\le j\le P$, then there is a simplified structure matrix $\hat{\psi}\in\mathbb{R}^{(P-1)\times P}$ and a simplified parameter matrix $\hat{\phi}\in\mathbb{R}^{(P-1)\times(n+1)}$ such that $L(x|\phi,\psi) = L(x|\phi,\tilde{\psi})$ where $\phi\in\mathbb{R}^{P\times(n+1)}$=[$\phi_1,\dots,\phi_P$]^T and $\tilde{\phi}\in\mathbb{R}^{(P-1)\times(n+1)}$=[$\phi_1,\dots,\phi_{j-1},\phi_{j+1},\dots,\phi_P$]^T.

After such simplifications, the lattice expression is the following:

$$L(x|\tilde{\phi},\tilde{\psi}) = \min_{1\le i\le Q}\left\{ max_{\substack{1\le j\le S\\ \tilde{\psi}_{ij}=1}}\left\{l(x|\tilde{\phi}_j)\right\}\right\}, \forall x\in\mathbb{R}^n \quad (6)$$

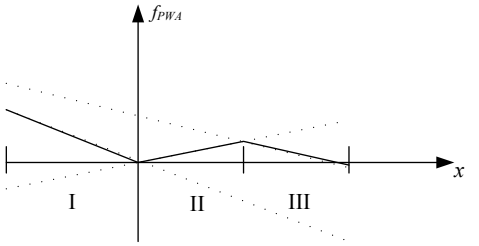
It calculates the minimum of Q maximums (being Q the number of rows in $\tilde{\psi}\in\mathbb{R}^{(Q\times S)}$), where each maximum is applied to as many affine functions as ones are in the corresponding row of $\tilde{\psi}$. The main advantage of this implementation is that, even the off-line pre-processing could be large, it is neither necessary to implement the edges nor to locate the input into a polytope. As an example, the one-dimensional PWA function illustrated in Fig. 1 has the following lattice representation:

$$L(x|\tilde{\phi},\tilde{\psi}) = min\{max\{l(x|\tilde{\phi}_I), l(x|\tilde{\phi}_{II})\}, l(x|\tilde{\phi}_{III})\} \quad (7)$$

IV. Lattice-PWA Implementation

The architecture proposed consists of the following main blocks, shown in Fig. 2:

- Compute: This block contains a multiplier-accumulator (in case of a serial implementation) or multipliers and an adder (in case of a parallel one) that calculates the affine expressions for a given input. It also contains a memory

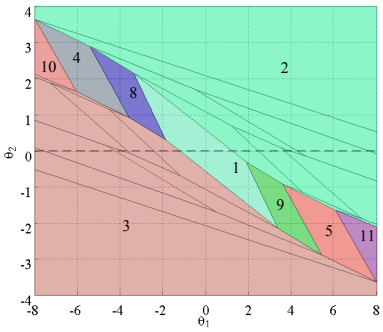


(a)

$$\psi = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \qquad \tilde{\psi} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(b)

Fig. 1. (a) One-dimensional PWA function. (b) Its corresponding structure matrix (ψ) and simplified structure matrix ($\tilde{\psi}$).



(a)

$$\tilde{\psi} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\tilde{\phi} = \begin{pmatrix} -0.4077 & -1.4014 & -0.8271 \\ -0.8166 & -1.7499 & 0 \\ -0.5528 & -1.5364 & -0.4308 \\ -0.3215 & -1.3185 & -1.2127 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

(b)

Fig. 3. (a) The domain D divided into polytopes for the application example. (b) Its corresponding simplified parameter and structure matrixes.

that stores the $S \times (n+1)$ coefficients associated with the simplified parameter matrix.

- Control: This block determines the inputs to the block MAX_MIN, decides the operator to be implemented (maximum or minimum), and addresses the memory of the compute block to calculate the different affine functions. Finally, it indicates (with an enable signal) when the output of the block MAX_MIN is the valid output of the system.
- MAX_MIN: This block calculates the maximum or the minimum of two affine expressions. Its inputs, controlled by the Control block, can be the output of the Compute block, an initialization value (0 or 1), or the output of the MAX_MIN block in a previous state.

The inputs can be loaded in parallel or in serial accordingly to the structure of the Compute block. The designer can select one implementation or another depending mainly on time and area restrictions.

In order to simplify the microelectronic realization, the range of the input and output values is normalized in the interval [0,1], and consequently the coefficients are evaluated for this range. Since the number of MIN operations to carry out is the number of rows (Q) of the simplified structure matrix, $\tilde{\psi}$ and the number of MAX operations is given by the number of 1's (U) in $\tilde{\psi}$, the circuit latency is is proportional to Q+U if the n inputs are processed in parallel or to (Q+U)×(n+1) if they are processed serially.

V. Application Example

The described circuit architecture has been implemented in a Xilinx Spartan 3 FPGA (xc3s200-5ftp256). The developed HDL description is valid for any system with two inputs

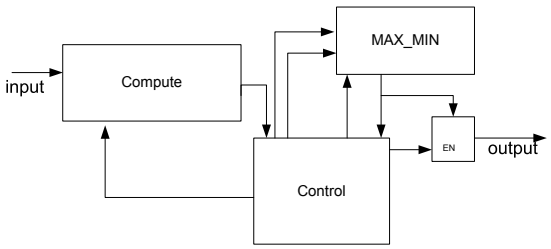


Fig. 2. Proposed architecture.

TABLE I
Approximation Error of the Circuit

Bit Number	8	10	12	14	16
RMSE	0.059	0.013	0.003	5e-4	2e-4

and one output. The methodology used to design and implement the proposed architecture employs CAD tools from two environments: Matlab&Simulink and ISE 12.3 by Xilinx. A design tool for DSPs called Xilinx System Generator, which is integrated into Simulink, is used to develop the circuit implementation. This tool allows the designer to increase the programmability of the circuit since it makes it possible to fit the realization with the parameters stored in the Matlab workspace. For instance, this Matlab configuration file fixes the memory coefficients used in the Compute block and the number of bits to code the inputs and the output.

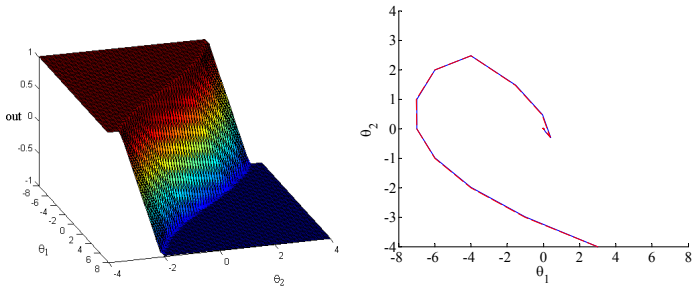
The functionalities of the proposed architecture have been analyzed with the application example of regulating to the origin the double integrator system described in [10]. The design of the optimal PWA control function to implement is performed by the Hybrid Toolbox for Matlab available in [15]. The resulting PWA function is defined over the domain *D=-8.8×-4.4* and has odd symmetry with regards the vertical axis. The implementation considers the advantage given by the symmetry so that the regions that take part in the lattice PWA implementation are 1, 2, 3, 5, 9, and 11, as shown in Fig. 3. The regions indexed as 2 and 3 are the maximum and the minimum of output values, so they are implemented implicitly. Applying the algorithm described in Section III (with the above considerations), the lattice representation obtained to implement is the following:

$$L(x|\tilde{\phi},\tilde{\psi}) = max\{l(x|\phi_1), l(x|\phi_5), l(x|\phi_9), l(x|\phi_{11})\} \quad (8)$$

The lattice approach (as well as the generic PWA approach) can implement a continuous PWA function with ideally no error so that the RMSE tends to zero as the number of bits increases. This is shown in Table I. The root mean square error (RMSE) has been calculated by comparing the desired output given by the Hybrid Toolbox ($\hat{u}$) with the output provided by the lattice PWA circuit ($\tilde{u}$) over 400 points distributed

TABLE II
COMPARISON OF IMPLEMENTATIONS

	Generic PWA [3]	PWAS (serial) [7]	PWAS (parallel) [7]	Hierarchical PWA [9]	Lattice PWA
Slices	12%	11%	16%	7%	7%
Clock Period	9.35 ns	14 ns	48.7 ns	8.16 ns	17.61 ns
Multipliers	1	1	3	2	2
Throughput	22 cycles (206.58 ns)	14 cycles (196 ns)	1 cycle (48.7 ns)	1 cycle (8.16 ns)	5 cycles (88.06 ns)
Latency	22 cycles (206.58 ns)	19 cycles (266 ns)	1 cycle (48.7 ns)	11 cycles (89.76 ns)	11 cycles (193.73 ns)
Parameters to store	21	256	768	15 (exploiting symmetry)	15 (exploiting symmetry)



(a) (b)

Fig. 4. (a) Control surface given by the lattice PWA implementation. (b) Evolution of the plant state.

homogeneously over the application domain, as follows:

$$RMSE = \sqrt{\frac{1}{400}\sum_{1}^{400}(\hat{u}(x_i) - \tilde{u}(x_i))^2} \qquad (9)$$

Considering 12 bits, the control surface provided by the circuit is shown in Fig. 4a. The implementation requires 2 multipliers and employs the 7% of logic slices. The estimated maximum working frequency is 56.78MHz. The circuit latency is 11 cycles and it provides a valid output every 5 cycles (circuit throughput).

Xilinx System Generator allows that the controller implemented in the FPGA can interact with a plant model described in Matlab. This verification is known as hardware-in-the-loop testing. The results obtained with such testing after closed-loop simulations (with the circuit implemented with 12bits) illustrate the evolution of the plant state towards the origin, as desired (Fig. 4b).

Table II allows comparing the features of the proposed lattice implementation with other existing implementations described in Section II (all of them with 12 bits). The proposal based on lattice representation offers a good trade-off between area occupation, throughput, and approximation error.

VI. CONCLUSIONS

The proposed architecture to implement continuous PWA functions based on lattice representation offers small size, high speed, and potentially no error. Its required parameters can be obtained by an off-line preprocessing. The design and FPGA implementation of an application example in the control domain has been automated with Matlab and ISE tools.

ACKNOWLEDGMENT

This work has been partially supported by European Community under the MOBY-DIC Project FP7-IST-248858 (www.mobydic-project.eu), by Ministerio de Ciencia e Innovación under the Project TEC2008-04920 and DPI2008-03847, and by Junta de Andalucía under the Project P08-TIC-03674 (with support from the PO FEDER-FSE).

REFERENCES

[1] C. Kahlert and L. O. Chua, *A generalized canonical piecewise-linear representation*, IEEE Trans. on Circuits and Systems, vol. 37, no. 3, pp. 373-383, 1990.

[2] J. Ramírez-Angulo, E. Sánchez-Sinencio, A. Rodríguez-Vázquez, *A piecewise linear function approximation using current-mode circuits*, IEEE Int. Conf. on Circ. and Systems, pp. 2021-2024, San Diego CA, May 1992.

[3] A. Oliveri, T. Poggi, M. Storace, *Circuit implementation of piecewise-affine functions based on a binary search tree*, European Conference on Circuit Theory and Design, 2009. ECCTD 2009, pp.145-148, Aug. 2009.

[4] R. Rovatti, C. Fantuzzi, S. Simani, *High-speed DSP-based implementation of piecewise-affine and piecewise-quadratic fuzzy systems*, Signal Processing, vol. 80, no. 6, pp. 951-963, June 2000.

[5] J. P. Echevarria, M. Martínez, J. Echanobe, I. del Campo, J. Tarela, *Digital hardware implementation of high dimensional fuzzy systems*, Applications of Fuzzy Sets Theory, Springer, pp. 245252, 2007.

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

[7] M. Storace, T. Poggi, *Digital architectures realizing piecewise-linear multi-variate functions: two FPGA implementations*, Int. J. Circ. Th. Appl., vol. 39, no. 1, pp. 1-15, 2009.

[8] T. Poggi, F. Comaschi, M. Storace, *Digital circuit realization of piecewise affine functions with non-uniform resolution: theory and FPGA implementation*, IEEE Trans. on Circuits and Systems II, vol. 52, no. 2, pp. 131135, 2010.

[9] I. Baturone, M. C. Martínez-Rodríguez, P. Brox, A. Gersnoviez, S. Sánchez-Solano, *Digital implementation of hierarchical Piecewise-Affine Controllers*, 20th IEEE Int. Symp. on Industrial Electronics (ISIE), pp. 1497-1502, June 2011.

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

[11] P. Tondel, T. A. Johansen, A. Bemporad, *Evaluation of piecewise affine control via binary search tree*, Automatica, vol. 39, no. 5, pp. 945-950, May 2003.

[12] P. Julian, A. Desages, O. Agamennoni, *High-level canonical piecewise linear representation using a simplicial partition*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol.46, no.4, pp. 463-480, Apr. 1999.

[13] J. M. Tarela, M. V. Martinez, *Region configurations for realizability of lattice Piecewise-Linear models*, Mathematical and Computer Modelling, vol. 30, no. 11-12, pp. 75-83, Dec. 1999.

[14] C. Wen, X. Ma, B. E. Ydstie, *Analytical expression of explicit MPC solution via lattice piecewise-affine function*, Automatica, vol. 45, no. 4, pp. 910-917, Apr. 2009.

[15] *Hybrid Toolbox*:http://www.ing.unitn.it/ bemporad/hybrid/toolbox, 2004.