Abstract—This paper presents a small, fast, low power consumption solution for piecewise-affine (PWA) controllers. To achieve this goal, a digital architecture for Very Large Scale Integration (VLSI) circuits is proposed. The implementation is based on the simplest lattice form, which eliminates the point location problem of other PWA representations and is able to provide continuous PWA controllers defined over generic partitions of the input domain. The architecture is parameterized in terms of number of inputs, outputs, signal resolution, and features of the controller to be generated. The design flows for Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) are detailed. Several application examples of explicit model predictive controllers (such as an adaptive cruise control and the control of a buck-boost DC-DC converter) are included to illustrate the performance of the VLSI solution obtained with the proposed lattice-based architecture.

Index Terms—Piecewise-affine controllers, PWA, Model Predictive Control, Lattice, digital VLSI, FPGAs, ASICs

Digital VLSI Implementation of Piecewise-Affine Controllers Based on Lattice Approach

M.C. Martínez-Rodríguez, P. Brox, and I. Baturone

1 Introduction

The attraction of piecewise-affine (PWA) functions lies in their capability to approximate any nonlinear behavior within any specified error (including linear threshold events and mode switching). They are also the simplest extension to linear functions with which engineers are familiar. The use of PWA functions in the control community started with the seminal work in [1]. Some years later, the development of PWA analysis and synthesis techniques was favored by the advent of new computational techniques and environments, such as [2] and [3]. More recently, PWA functions have arisen naturally as a solution to many engineering problems such as those of model predictive control (MPC).

Model predictive controllers have become popular as a way of approaching non-linear controller design for linear systems with constraints, piecewise-affine systems and both non-linear constrained and hybrid systems [4]-[7]. For the problem of regulating to the origin the linear time invariant system

$$\begin{array}{c} x(t+1)=Ax(t)+Bu(t) \\ y(t)=Cx(t) \end{array} \tag{1}$$

with the constraints

$$y_{min} \leq y(t) \leq y_{max},\ u_{min} \leq u(t) \leq u_{max},\ \text{for}\ t \geqslant 0 \tag{2}$$

where $x(t) \epsilon \mathbb{R}^n$, $u(t) \epsilon \mathbb{R}^m$ and $y(t) \epsilon \mathbb{R}^p$ are the state, input, and output vectors, respectively, and the pair $(A,\ B)$ is stabilizable, the optimization problem that arises in MPC and is solved at each time instant, t, is as follows

$$\min_{U} \left\{ \begin{array}{l} J(U, x(t)) = x_{t+N_y|_t}^T \cdot P \cdot x_{t+N_y|_t} \\ + \sum_{k=0}^{N_y-1} \left[x_{t+k|_t}^T \cdot Q \cdot x_{t+k|_t} + u_{t+k}^T \cdot R \cdot u_{t+k} \right] \end{array} \right\} \tag{3}$$

where

$$U \triangleq \{u_t, ..., u_{t+N_u-1}\} \tag{4}$$

is subject to

$$\begin{array}{c} x_{t|_t} = x(t) \\ y_{min} \leq y_{t+k|_t} \leq y_{max} \quad k = 1, ..., N_c \\ u_{min} \leq u_{t+k} \leq u_{max} \quad k = 0, ..., N_c \\ x_{t+k+1|_t} = Ax_{t+k|_t} + Bu_{t+k} \quad k \geq 0 \\ y_{t+k|_t} = Cx_{t+k|_t} \quad k \geq 0 \\ u_{t+k} = Kx_{t+k|_t} \quad N_u \leq k \leq N_y \end{array} \tag{5}$$

where the predicted state vector at time $t+k$, $x_{t+k|_t}$ is obtained by applying the input sequence $u_t, ..., u_{t+N_u-1}$ to the system model (1) starting from the state $x(t)$; N_y, N_u, N_c are the output, input, and constraint horizons, respectively; K is some feedback gain; and it is assumed that $Q = Q^T \succeq 0$, $R = R^T \succ 0$, and $P \succeq 0$ [8].

The solution of the optimization problem in (3) to (5) is a PWA controller $u(x) :\ M \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ as follows [4], [8]

$$u(x) = F_i x + G_i \quad \text{if}\ x \epsilon O_i,\ i = 1, ..., D \tag{6}$$

where $x \epsilon \mathbb{R}^n$ is the input to the controller function, which is usually the state vector fully measured at time t, $x(t)$; $F_i \epsilon \mathbb{R}^{m \times n}$ are gain matrices and $G_i \epsilon \mathbb{R}^m$ are offset vectors. The input domain, M, is partitioned into D non overlapping regions, called polytopes , $M = O_1 \cup ... \cup O_D$. Each polytope, O_i, is defined as a closed set of points delimited by E_i edges in the form of $(n-1)$-dimensional hyper-planes, $h_j^T x + k_j = 0$, as follows

$$O_i = \{x \epsilon \mathbb{R}^n \,|\, h_j^T x + k_j \leqslant 0,\ j = 1, ..., E_i\} \tag{7}$$

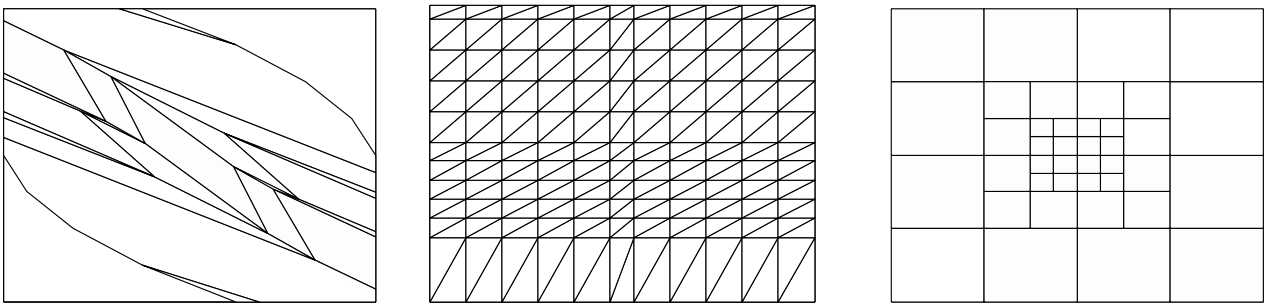
where $h_j \epsilon \mathbb{R}^n$, $k_j \epsilon \mathbb{R}$, and h^T denotes the transpose of h.

Evaluation of the explicit MPC approach in (6) is much simpler than solving the optimization problem in (3) to (5) on line.

Hence, the issue of implementing the explicit MPC approach in (6) in hardware has recently been given much attention [7]-[11]. Implementation of PWA functions in hardware has also been contemplated for fuzzy control and virtual sensors [12]-[17].

Several digital approaches have recently been proposed for hardware realizations of PWA functions [7], [13]-[26]. The implementation of PWA functions is solved using a Digital Signal Processor (DSP) in [13] and [16], while the generation of PWA functions using FPGAs is reported in [14]-[15], [17]-[20], [22], [25]-[26]. The proposal described in [14] implements a nonlinear controller

M.C. Martínez-Rodríguez, P. Brox, and I. Baturone are with the Instituto de Microelectrónica de Sevilla, Spanish National Research Council (IMSE-CNM-CSIC) and University of Seville, Seville, Spain (e-mail: macarena@imse-cnm.csic.es; brox@imse-cnm.csic.es; lumi@imse-cnm.csic.es).



(a) Generic partition (b) Simplicial partition (c) Rectangular partition

Figure 1. Different partitions of a bi-dimensional input domain into polytopes.

(including PWA) as an intellectual property (IP) module that accelerates the inferring task of a general-purpose processor. Solutions using digital ASICs are proposed in [7], [21], [23] and mixed-signal ASICs are described in [24]. In general, digital VLSI implementations of PWA controllers have many practical applications when very small size, low power consumption and/or short response times are required [9], [17], [27].

Several approaches have been adopted for implementing PWA functions. They differ in how the affine function corresponding to the input is found (the point location problem) and how the input domain is partitioned into polytopes. Polytopes can be of any shape if the partition is generic (Fig. 1a) [7], [18], [21]; they are simplexes if the partition is simplicial (Fig. 1b) [19], [23], [24]; and they are hyper-rectangles if the partition is hyper-rectangular (Fig. 1c) [22]. Hardware solutions that implement simplicial and hyper-rectangular approaches usually provide faster responses than solutions that implement a generic approach. As a drawback, explicit MPC approaches in (6) usually employ generic partition, so simplicial and hyper-rectangular approaches have to approximate the generic polytopes with simplexes or hyper-rectangles. This type of approximation may negatively impact the controller performance, in particular, its closed-loop stability.

Another solution is the realization of PWA controllers with a hierarchical architecture, which decomposes the multivariate PWA controller into modules, preferably uni-dimensional PWA modules connected in cascade [20].

Lattice representations of PWA controllers eliminate the point location problem [28], reducing memory requirements because the edges in (7) do not have to be computed. Another advantage of lattice PWA controller representations is that they can implement without approximation any continuous explicit MPC that employs a generic partition. Moreover, there exists a systematic way to find the simplest lattice form of a given PWA function [29]-[31]. A preliminary (non-parameterized) digital architecture for implementing PWA functions with two inputs and one output in lattice form is presented in [25]. The architecture is addressed to FPGAs from Xilinx and uses the Xilinx DSP blockset for Simulink in System Generator. A design methodology for implementing the architecture shown in [25] in FPGAs is briefly described in [26]. The methodology uses The Mathworks model-based design environment Simulink for FPGA design. All of the FPGA implementation steps are automatically performed by System Generator tool from Xilinx, so RTL design methodologies are not employed.

This paper proposes a digital architecture for implementing PWA controllers based on lattice representation. The architecture is parameterized in terms of the number of inputs and outputs (to handle Multi-Input Multi-Output or MIMO PWA controllers), the input, output and the parameters involved resolution and the complexity of the PWA controller (more or less affine control laws). The implementation of the architecture into FPGAs and ASICs is detailed together with the computer-aided design flow employed in both cases. The starting point for the design flow is the PWA function to be implemented and the end point is the set of design files for the FPGA realization or the set of configuration/programming files for the ASIC realization. In particular, the paper illustrates how the design flow can be linked to tools from the control domain, so that the digital implementation of a given explicit MPC is obtained automatically. The architecture does not use the Xilinx DSP blockset for Simulink in System Generator, as in [25], but has been described in Verilog. Unlike the methodology in [26], this proposal does employ RTL design methodologies.

The paper is organized as follows. Section 2 summarizes the lattice representation of a PWA function. Section 3 describes the digital architecture proposed and compares it with other solutions. The design flows for implementing the architecture in FPGAs and ASICs are described in Section 4. Application examples in the control domain are shown in Section 5 and, finally, conclusions are given in Section 6.

2 PWA FUNCTIONS BASED ON THE LATTICE APPROACH

2.1 The lattice representation

The work done in [28] shows how a continuous PWA function can be represented by a lattice form. In the lattice form it is not necessary to locate the input in a polytope as a first step when computing the PWA function and consequently, the edges that define the polytopes do not have to be computed explicitly.

For the sake of simplicity, let us consider the PWA function in (6) but with only one output, that is, $f_{PWA}(x): M \subset \mathbb{R}^n \rightarrow \mathbb{R}$, as follows

$$f_{PWA}(x) = f_i^T x + g_i \quad \text{if } x \epsilon O_i \quad i = 1, ..., D \tag{8}$$

with $f_i \epsilon \mathbb{R}^n$ and $g_i \epsilon \mathbb{R}$.

According to the notation in [29], this can be expressed as

$$f_{PWA}(x) = \phi_i^T \begin{bmatrix} x^T & 1 \end{bmatrix}^T = l(x|\phi_i) = l_i(x) \quad \text{if } x \epsilon O_i \tag{9}$$

where $\phi_i \epsilon \mathbb{R}^{n+1}$ is the parameter vector that defines the affine function at polytope O_i ($\phi_i = \begin{bmatrix} f_i^T & g_i \end{bmatrix}^T$). The parameter matrix, ϕ, is a $D \times (n+1)$ matrix containing the D parameter vectors.

The PWA function is continuous if

$$f_{PWA}(x) = \phi_j^T \begin{bmatrix} x^T & 1 \end{bmatrix}^T = \phi_k^T \begin{bmatrix} x^T & 1 \end{bmatrix}^T \ \forall x \epsilon O_j \cap O_k$$

with $j, k \epsilon \{1, ..., D\}$.

Any continuous PWA function can be fully specified by a parameter matrix, ϕ, and a structure matrix, ψ, in the following lattice representation $L(x|\phi, \psi) = f_{PWA}(x)$

$$L(x|\phi, \psi) = \min_{1 \leq i \leq D} \left\{ \max_{1 \leq j \leq D, \psi_{ij}=1} \{l_j(x)\} \right\}, \ \forall x \epsilon \mathbb{R}^n \tag{10}$$

The matrix, ψ, is defined as a structure matrix if its elements are calculated as

$$\psi_{ij} = \begin{cases} 1 & \text{if } l_i(x) \geq l_j(x), \ \forall x \epsilon O_i \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

for $i, j = \{1, ..., D\}$.

The lattice representation in (10) is not unique because there are several ways of combining the local affine functions with minimum and maximum operators. To obtain an efficient VLSI realization, it is best to implement the simplest lattice form, as addressed in the following subsection. More details about lattice PWA representation can be found in [29].

As example, let us consider the PWA function shown in Fig. 2 and described as follows

$$f_{PWA}(x) = \begin{cases} l_1(x) = 2x + 1 & \text{if } 0 \leq x \leq 1 \\ l_2(x) = -3x + 6 & \text{if } 1 \leq x \leq 2 \\ l_3(x) = 0.75x - 1.5 & \text{if } 2 \leq x \leq 6 \end{cases} \tag{12}$$

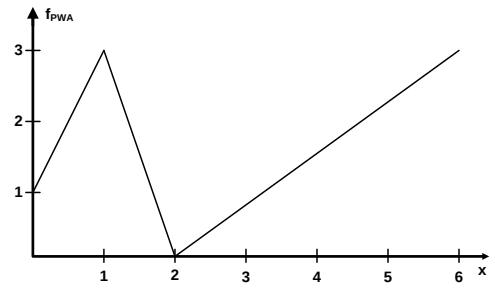


Figure 2. Example of a one-dimensional PWA function

One of its lattice representations is the following

$$f_{PWA}(x) = min \left\{ \begin{array}{c} max\{l_1(x), l_3(x)\}, \\ max\{l_2(x), l_3(x)\}, \\ max\{l_2(x), l_3(x)\} \end{array} \right\} \tag{13}$$

where the structure matrix, ψ, and the parameter matrix, ϕ, are given by

$$\psi = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \phi = \begin{bmatrix} 2 & 1 \\ -3 & 6 \\ 0.75 & -1.5 \end{bmatrix} \tag{14}$$

2.2 The simplest lattice representation

The way to obtain the simplest lattice representation of a continuous PWA function was addressed in [29]-[33]. Herein, the algorithm used to find the simplest lattice representation of a PWA function with the form described in (10) is the one presented in [29] and updated in [30] and [31]. The steps of the algorithm can be summarized as follows

- Given an explicit PWA function, record the local affine functions ($l_i(x), 1 \leq i \leq D$), the constrained inequalities ($h_j^T x + g_j \leq 0, \ j = 1, ..., E_i$) that define the polytopes, and the k_i vertexes v_k of each polytope O_i.
- Calculate the values of each affine function at each set of vertexes, $l_i(v_k)$.
- Calculate the structure matrix using

$$\psi_{ij} = \begin{cases} 1 & \text{if } l_i(v_k) \geq l_j(v_k), \ 1 \leq k \leq k_i \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

- Simplify the rows in the structure matrix as described in *Lemma 3* in [29]. After such simplification, the rows of the new simplified structure matrix correspond to super-regions made up of several polytopes. Super-regions can be concave or even disconnected.
- Simplify the columns in the structure matrix and the consequent rows in the parameter matrix as described in *Lemma 1* in [31]. After this last step, super-regions that are not involved in the PWA function are removed.

The new simplified structure matrix is $\tilde{\psi} \in \mathbb{R}^{X \times W}$ and the new simplified parameter matrix is $\tilde{\phi} \in \mathbb{R}^{W \times (n+1)}$,

with $X \leq W \leq D$. The simplest lattice representation, $L(x|\tilde{\phi},\tilde{\psi})$, of the PWA function, $f_{PWA}(x)$, is the following

$$L(x|\tilde{\phi},\tilde{\psi}) = \min_{1\leq i\leq X}\left\{ \max_{1\leq j\leq W,\tilde{\psi}_{ij}=1}\left\{\tilde{l}_j(x)\right\}\right\}, \forall x\in\mathbb{R}^n \tag{16}$$

This calculates the minimum of X maxima, where each maximum is applied to as many affine functions as there are logic 1's in the corresponding row of $\tilde{\psi}$.

Applying the simplification procedure to the example shown in Fig.2, the simplified structure matrix, $\tilde{\psi}$, and the simplified parameter matrix, $\tilde{\phi}$, are given by

$$\tilde{\psi} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \tilde{\phi} = \begin{bmatrix} 2 & 1 \\ -3 & 6 \\ 0.75 & -1.5 \end{bmatrix} \tag{17}$$

Hence, the simplest lattice representation of the function in Fig. 2 is given by (in this case $\tilde{\phi} = \phi$)

$$f_{PWA}(x) = min\left\{ \begin{array}{c} max\left\{\tilde{l}_1(x),\tilde{l}_3(x)\right\}, \\ max\left\{\tilde{l}_2(x),\tilde{l}_3(x)\right\} \end{array} \right\} \tag{18}$$

3 DIGITAL ARCHITECTURE

This section presents a configurable, programmable digital architecture that implements the simplest lattice representation in (16). First, the main building blocks are introduced, then the timing performance is described secondly and, finally, the proposed architecture is compared with existing architectures used to implement other forms of PWA controllers. The architecture is described in terms of the following parameters:

- n: number of inputs.
- X: number of rows in the simplified structure matrix.
- W: number of columns in the simplified structure matrix or rows in the simplified parameter matrix.
- U: defined as $U = 2\sum_{i=1}^{X}\left\lceil \frac{U_i}{2}\right\rceil$, where U_i is the number of logic 1's in the i-th row of the simplified structure matrix $\tilde{\psi}$.
- no: number of outputs.

3.1 Building blocks

In order to implement the simplest lattice representation in (16), the following building blocks are required.

ArithUnit

This block calculates the local affine expressions, $\tilde{l}_j(x)$ in (16), for a given input, x, with one multiplier per x_k and one adder per element in the parameter vector (for a parallel implementation) or with a multiplier-accumulator (for a serial one)

$$\tilde{l}_j(x) = \sum_{k=1}^{n} f_{jk}x_k + g_j \tag{19}$$

Max

This block calculates the maximum among several local affine control laws ($\tilde{l}_j$)

$$max_i = \max_{1\leq j\leq W,\tilde{\psi}_{ij}=1}\left\{\tilde{l}_j(x)\right\} \tag{20}$$

Min

This block calculates the minimum among the maximum local affine control laws of each row in the structure matrix (max_i)

$$min = \min_{1\leq i\leq X}\{max_i\} \tag{21}$$

Acquisition

This block can acquire the input serially or in parallel according to the external communication protocol selected.

MemPar

This element is a memory that stores the simplified parameter matrix, $\tilde{\phi}$. The j-th word of the memory stores the $n+1$ parameters, $f_{j1}, ..., f_{jn}, g_j$, associated to the local affine control law $\tilde{l}_j(x)$. The depth of this memory is at least W, so that the memory is addressed by a bus of $\lceil log_2 W\rceil$ bits.

MemLat

This block is another memory whose words are related to the pair of indices (i, j) of those elements of the simplified structure matrix that have a logic value '1' ($\tilde{\psi}_{ij} = 1$). The index j refers to the address of the memory *MemPar* where the parameters associated to the local affine function $\tilde{l}_j(x)$ are stored. Each word of this memory has $\lceil log_2 W\rceil + 1$ bits. The first $\lceil log_2 W\rceil$ bits are used to codify that address. The last bit is related to the index i. Since the memory *MemLat* is read from the beginning to the end, if the last bit is '0', it means that i is unchanged and the same row of matrix $\tilde{\psi}$ is being processed. Otherwise, if the last bit takes the logic value '1', it means that the row is finished, the index i has to be increased, and the next row has to be processed. The depth of the memory *MemLat* is at least U, so this memory is addressed by a bus of $\lceil log_2 U\rceil$ bits.

Let us illustrate the structure of the memories *MemLat* and *MemPar* with the example in Fig. 2. Given the simplified structure matrix, $\tilde{\psi}$, and the simplified parameter

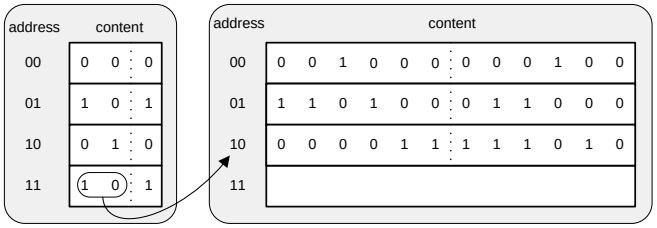


Figure 3. *MemLat* (left) and *MemPar* (right) memories for the example described in Fig. 2.

matrix, $\tilde{\phi}$, in (17), the words stored in these memories are shown in Fig. 3. In this example, the parameters stored in *MemPar* are coded with 6 bits in two's complement format, using the 4 most significant bits for the integer part.

It can be seen how the number of super-regions and different affine functions is correlated with the memory resources.

Control

This element sets the addresses of the memory *MemLat* and handles the enable signals of the blocks in the architecture according to the timing schedule described below.

3.2 Description of the architecture

When choosing between parallel and serial architectural solutions there is always a trade-off, mainly in terms of area consumption and time processing. The solutions selected here are as follows. The inputs, $x_1, ..., x_n$, are acquired serially, one after the other, and each input is acquired in parallel, so that the input bus has the same width as the input bit resolution. The memories selected are dual-port memories, so the parameters of two local affine functions can be read simultaneously from *MemPar*. The Control block therefore has to simultaneously address two words of *MemLat*, which in turn addresses two words of the memory *MemPar*. The *ArithUnit* block is fully parallel and computes two local affine functions in one clock cycle. The *Max* block computes the maximum among both local functions and the maximum value previously computed, $maxold$, provided by the *Control* block. The first value of $maxold$ is set to the minimum value of the PWA function. The output provided by the *Max* block is as follows

$$\begin{aligned} l_i(x) &= f_{i1}x_1 + \cdots + f_{in}x_n + g_i \\ l_j(x) &= f_{j1}x_1 + \cdots + f_{jn}x_n + g_j \\ max_{new} &= max\ (l_i(x),\ l_j(x),\ max_{old}) \end{aligned} \tag{22}$$

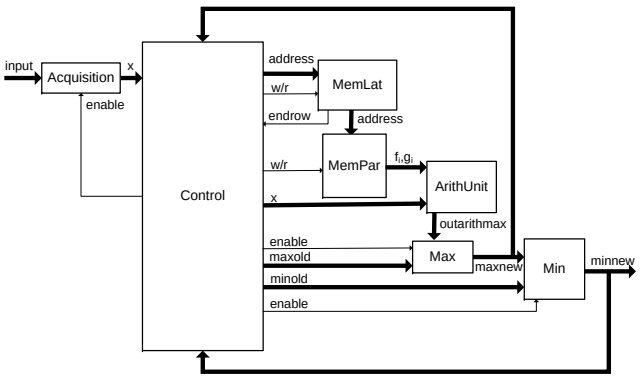


Figure 4. Block diagram of the architecture for computing lattice-based PWA controllers

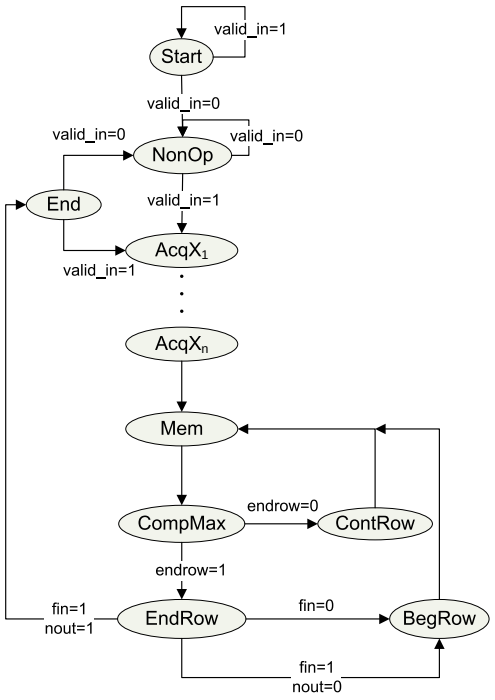


Figure 5. State diagram of the operation

The block diagram of the proposed architecture is shown in Fig. 4. Fig. 5 shows the state diagram that explains the operation. Once the system exits the reset state (*Start*), it remains in a non-operating state (*NonOp*). The output signal *ready* takes a high logic value to indicate that the system can acquire new inputs. When the input signal *valid_in* takes a high logic value, it indicates that there is an input available and ready to be processed. The first input is acquired in parallel during state *AcqX*1. The state changes from *AcqX*1 to *AcqX*n until, one after the other, all the n inputs are acquired. Thereafter, the *Control* block provides the first two *MemLat* addresses. The data stored in *MemLat* provides the two addresses where the first two sets of parameters are located in the memory *MemPar*. This is done at state *Mem*. Once the parameters have been retrieved, the two local affine functions and the maximum between them and the previously calculated maximum (the $maxold$ value) are computed. This is done at state *CompMax*. The *endrow* signal, which comes from the last bit of the words provided by *MemLat*, indicates whether the row in the simplified structure matrix, $\tilde{\psi}$, that is being processed has already finished (*endrow*='1') or not (*endrow*='0'). If it has not finished, the maximum value of the local affine functions associated to the row is still being calculated. In this case, the current maximum value is stored (at state *ContRow*), new parameters are retrieved from *MemPar* (at state *Mem*), and a new maximum is calculated (at state *CompMax*). If the row is finished (*endrow*='1') then the minimum value is calculated between the maximum value of the row and the previously calculated minimum value (the

Table 1
Comparison between architectures

	Latency (Number of clock cycles)	Bits to store	Multipliers	Exact optimal MPC
PWAG (A)	$n+2d+nd+2$	$n_{bit}(n+1)(D+E)$	1	Yes
PWAG (B)	$n+2d$	$n_{bit}(n+1)(D+E)+N\lceil log_2 T\rceil$	n	Yes
PWAS (A)	$n+4$	$n_{bit}\prod_{i=1}^{i=n}(m_i+1)$	1	No (approx.)
PWAS (B)	3	$n_{bit}(n+1)\prod_{i=1}^{i=n}(m_i+1)$	$n+1$	No (approx.)
PWAR (A)	$n+2$	$n_{bit}(n+1)\prod_{i=1}^{i=n} m_i$	1	No (approx.)
PWAR (B)	2	$n_{bit}(n+1)\prod_{i=1}^{i=n} m_i$	n	No (approx.)
MultiTree	h_M+2	–	nM	Yes
PWAL	$n+U+1$	$n_{bit}(n+1)W+U(\lceil log_2 X\rceil+1)$	$2n$	Yes

$minold$ value). This is done at state *EndRow*. While there are rows in the simplified structure matrix, $\tilde{\psi}$, still to be processed, the current minimum value is stored and the processing of a new row is started (state named *BegRow*), by repeating the steps described above. If there are no more rows (*fin*='1'), the minimum calculated is the system's output (the value of the PWA function) and *valid_out* signal is set to '1'. The *Control* block sets the value of *fin* signal to '1' when the address of the memory *MemLat* reaches the value X, which is the number of rows in the simplified structure matrix.

In the case of a Multiple Input Single Output (MISO) PWA function, if *fin* signal is '1' then the system has finished processing the input and the *nout* signal is '1'. The circuit is ready to receive a new input, so the *ready* signal is set to '1'. In the case of a Multiple Input Multiple Output (MIMO) PWA function, the *nout* signal is set to '0', because the other outputs have to be computed. The same process is then repeated except for the steps of acquiring the inputs, because the outputs are calculated for the same inputs. When all the outputs have been calculated, the *nout* signal is set to '1' and the *ready* signal is set to '1', the latter indicating that the system is ready to receive a new input. The next state is the non operation state whenever a new valid input is not provided.

The latency of the architecture to provide each output depends on several operations. One cycle is needed to acquire each input (at states *AcqXi*). Two cycles are needed to compute the maximum of two affine functions (one to extract the parameters, at state *Mem*, and other to compute it, at state *CompMax*). In a row of the simplified structure matrix, the maximum of U_i affine functions has to be computed, and so U_i clock cycles are needed. The minimum operation at the end of each row is computed in one clock cycle. However, this cycle is not necessary if there are more rows to valuate, since the minimum is calculated at the same time as the parameters are extracted (at state BegRow, the minimum is computed and the parameters are extracted). Hence, only one more cycle is needed, in addition to the U cycles, to compute all the rows. The latency of the architecture to provide each output is therefore $n+U+1$, U being as defined at the beginning of this section. The throughput for each output is $U+1$.

3.3 Comparison with other architectures

Different ways of implementing multivariate PWA controllers have been reported in literature. In the method known as the Piecewise-Affine Generic (PWAG) method, three steps are taken to compute the PWA function. The first, called the point location step, determines the polytope where the input is located. The second step retrieves the parameters of the affine control law associated to the polytope and, finally, the PWA controller is computed [34]. A digital architecture for FPGA realizations is proposed in [18]. The point location step is performed by sweeping on line a binary search tree that has been constructed off line, prior to programming the FPGA. The work in [7] uses the hardware design program PICOExpress to translate the C source code of the PWAG controller evaluation algorithm into a hardware description language definition that is implemented in an ASIC. The digital architecture presented in [21] sweeps on line the binary search tree that is configured and programmed into the ASIC. PWAG forms can implement PWA controllers defined over polytopes of any shape, but they are costly for certain controllers that require a very deep and/or non-balanced search tree. A PWAG representation based on multiway trees to accelerate the control law evaluation is described in [10].

To cut down the time needed to solve the point location problem, the method known as the Piecewise-Affine Simplicial (PWAS) method employs a partition of regular polytopes called simplexes [35]. Architectures for the realization of mixed-signal integrated circuits (ASICs) that implement PWAS controllers have been described in [23], [24], with several architectures being presented for the realization of PWAS controllers with FPGAs. The architecture proposed in [19] allows for two versions (one parallel and one serial). PWAS implementations often provide faster solutions than PWAG, but they have two important limitations. The first, known as 'the curse of dimensionality', means that the number of parameters needed to define the PWAS controllers grows exponentially with the number of dimensions of the input domain, requiring high memory resources. The

second limitation is that PWAS representation has to approximate a PWA controller defined over a generic partition of the input domain, as commented in Section 1.

To further reduce the time needed to solve the point location problem and to allow the realization of very simple digital architectures, the method known as the Piecewise-Affine Rectangular (PWAR) method performs a hyper-rectangular partition of the input domain [22]. A multi-resolution PWAR architecture is also proposed in [22] to mitigate the curse of dimensionality. It allows memory requirements to be reduced at the cost of a slightly higher latency and greater circuit complexity. The PWAR representation also has to approximate a PWA controller defined over a generic partition.

Another solution to reduce the curse of dimensionality is to implement PWA controllers with a hierarchical architecture, thus decomposing the multivariate PWA controller into modules, preferably unidimensional cascade-connected PWA modules. This type of architecture implemented in FPGAs has been explored in different case studies in [20]. The problem is that there is no systematic way to find the best hierarchical PWA form to approximate a given controller.

In Table 1, the proposed architecture is compared with the above-mentioned architectures. The architecture labeled PWAG(A) is an architecture that calculates a PWA function based on a binary search tree constructed off-line, with an arithmetic unit implemented as a multiplier-accumulator [18]. The architecture reported in [21] for calculating a PWA function based on a binary search tree constructed on line is labeled PWAG(B). The mostly serial and mostly parallel architectures based on simplicial partition, as presented in [19]], are labeled PWAS(A) and PWAS (B) respectively . The architecture based on the hyper-rectangular partition proposed in [22] is labeled PWAR(A) for the serial version and PWAR(B) for the parallel version. The architecture based on multiway trees is labeled MultiTree [10].

The symbols employed in the comparison are: the input dimension of the PWA function (n), the depth of the binary search tree (d), the number of nodes in the search tree (N), the number of different local affine functions (W), the number of edges defining the polytopes (E), the number of bits representing the input (n_{bit}), the number of vertices in the simplicial partition, $\prod_{i=1}^{i=n}(m_i+1)$, m_i being the number of partitions for each input, the order of the trees in the Multitree (M), the internal height of the Multitree (h_M) and U, which is related to the number of logic 1's in the simplified structure matrix ($\tilde{\psi}$).

Compared to PWAS and PWAR architectures, the proposed PWAL architecture can generate any continuous PWA function defined over a generic partition without approximations and usually requiring less memory. Compared to PWAG architectures, the proposed PWAL architecture offers the advantage of requiring less memory while maintaining competitive performance in terms of latency. The performance of PWAL architecture improves as the value of U gets smaller, since U is related to the stored parameters and the response time. The worst case scenario for a PWAL implementation is when the structure matrix has not been simplified and $U = D \times (D-1)$. These conclusions are illustrated quantitatively in the examples shown in Section V.

4 VLSI IMPLEMENTATION

The architecture described above was implemented in FPGAs and ASICs, with the following specifications:

- Input resolution: 12 bits.
- Parameter resolution: 12 bits.
- Fixed-point arithmetic.

The steps of both design flows are summarized below.

4.1 FPGA design flow

Since the FPGAs considered were from Xilinx, the Xilinx ISE Design Suite was employed as the design environment. The RTL description of the architecture was developed in the HDL language *Verilog*. The description was parameterized to allow a different number of inputs, n, outputs, no, and simplified structure and parameter matrices, $\tilde{\psi}$, $\tilde{\phi}$.

Depending on the specifications of the simplified structure and parameter matrices of the PWAL controller to be implemented, the Xilinx tool Core Generator was used to generate the memories *MemLat* and *MemPar*. Core Generator provides Intellectual Property (IP) cores for Xilinx FPGAs. More specifically, two dual-port memories were generated using the block RAMs available in the selected FPGA family. *MemLat* has an address bus width of $log_2 U$ and a data bus width of $log_2 W + 1$. *MemPar* has an address bus width of $log_2 W$ and a data bus width of $12(n+1)$.

The arithmetic unit comprised of n 2-input signed multipliers with 12 bits of input resolution. These multipliers were implemented with dedicated hardware generated with the Core Generator tool. The arithmetic unit also contained $n+1$ 2-input signed adders implemented with dedicated hardware. Since all these resources were generated by the Core Generator tool, they were optimized in terms of area, speed, and power.

The Xilinx ISE environment contains all the tools needed to translate the design described in *Verilog* into a bitstream configuration file for the selected FPGA device.

4.2 ASIC design flow

An ASIC is not a programmable device like an FPGA. Hence, the RTL description developed in Verilog corresponded to a programmable, configurable core that implemented a MIMO PWA function with the following versatility:

- Number of inputs: up to 4 (configurable from 1 to 4).
- Number of outputs: from 1 to 2.

- Maximum of logic 1's in the simplified structure matrix: 128 for one output and 64 for two outputs.
- Maximum number of affine functions in the simplified parameter matrix: 32.

The RTL specification was written in *Verilog*, using high level constructs. The synthesis tool (*Design Analyzer* from *Synopsis*) transformed this RTL specification into a set of logic gates, taking into account technology information. The arithmetic unit comprised 4 signed 2-input multipliers with 12 bits input resolution and 5 signed 2-input adders with 24 bits input resolution. The place and route tool employed was *SoC Encounter* from *Cadence*. The selected technology was *TSMC* (Taiwan Semiconductor Manufacturing Company) 90-nm, 9-layer metal, since this offered a good trade-off between performance and cost.

The *MemLat* and *MemPar* memory blocks were implemented with IP modules provided by Europractice. These IP modules are high-performance synchronous dual-port memory designs that take full advantage of TSMC's N90LP-LK CMOS Process. The size of the memory *MemLat* was 128 words coded with 6 bits (5 bits to address MemPar). Hence, the width of the address bus was 7 and the width of the data bus was 6. The depth of this memory indicates that the lattice representation can have a maximum of 128 logic 1's in the structure matrix ($U_{max} = 128$) for PWA functions with one output, or 64 logic 1's in each structure matrix for functions with two outputs ($U_{max} = 64$ for each output). The width of the words means that this memory can address a memory with a depth of 32 words, since the 6th bit was used to indicate the end of a row in the structure matrix. The size of the memory *MemPar* was 32 words coded with 60 bits, and so the width of the address bus was 5 and the width of the data bus was 60. The depth of this memory meant that 32 different affine functions could be stored. The width allows storage of 5 parameters with a resolution of 12 bits.

The layout is shown in Fig. 6. The active area was $0.29\,mm^2$.

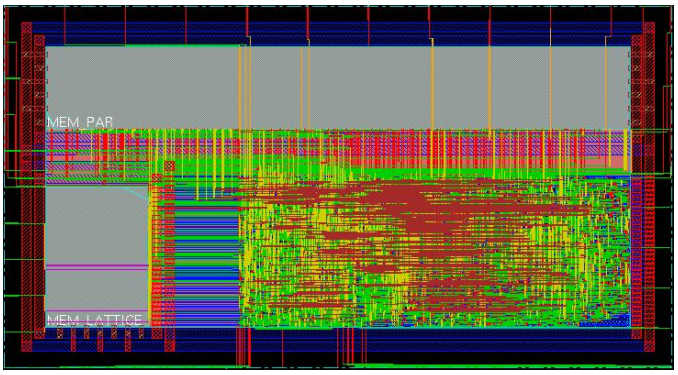


Figure 6. Layout after place&routing in *SoC Encounter* from *Cadence*

4.3 Programming and verification flow

Given a PWA controller of the type described in (6), a design flow was developed that made it possible to obtain either the bitstream to program a selected FPGA or the configuration files to configure and program the ASIC realization described above. Some fundamental steps of this design flow are devoted to extracting the contents to be stored in the memories *MemLat* and *MemPar*. These steps were carried out exclusively in MATLAB as shown in Fig. 7. First, a mathematical formulation of the PWA controller was described in MATLAB. After that, the parameter and structure matrices (ψ, ϕ) were generated. Then, the rows and columns were simplified to obtain a simplified structure matrix ($\tilde{\psi}$) and a simplified parameter matrix ($\tilde{\phi}$) and finally a MATLAB function was used to generate the parameters required for the hardware realization (n, X, W, U, W, no). This function set the number of bits in the signals, the parameters that the memories were to store and the control circuitry specifications.

The design flow developed also facilitates the verification of the VLSI implementation. A MATLAB function generates a Verilog file called <Testbench.v> that provides input values for the VLSI implementation. A simulator, such as ModelSim from Mentor Graphics or ISim from Xilinx, takes this <Testbench.v> file, together with the file describing the VLSI design (<Lattice_FPGA.v> or <Lattice_ASIC.v>) and computes the output values. The MATLAB function takes the output values and can either compare them with the output values given by the implemented PWA controller (open-loop verification) or employ them in the application domain where the VLSI design is going to be embedded (for example, closed-loop verification, as will be shown in Section V). The block diagram of this verification flow is shown in Fig. 8.

5 Application Examples in Model Predictive Control

The design flow described above was used to design digital VLSI implementations of PWA controllers. PWA controllers of the type shown in (6) were obtained with the Moby-dic Toolbox [36] from MATLAB&Simulink and its interfaces to Hybrid Toolbox [37] and to MPT Toolbox

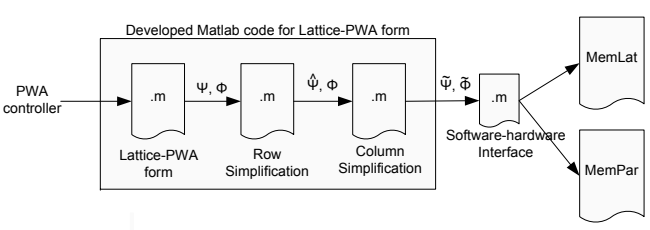


Figure 7. Design flow for extracting the contents of the *MemLat* and *MemPar* memories

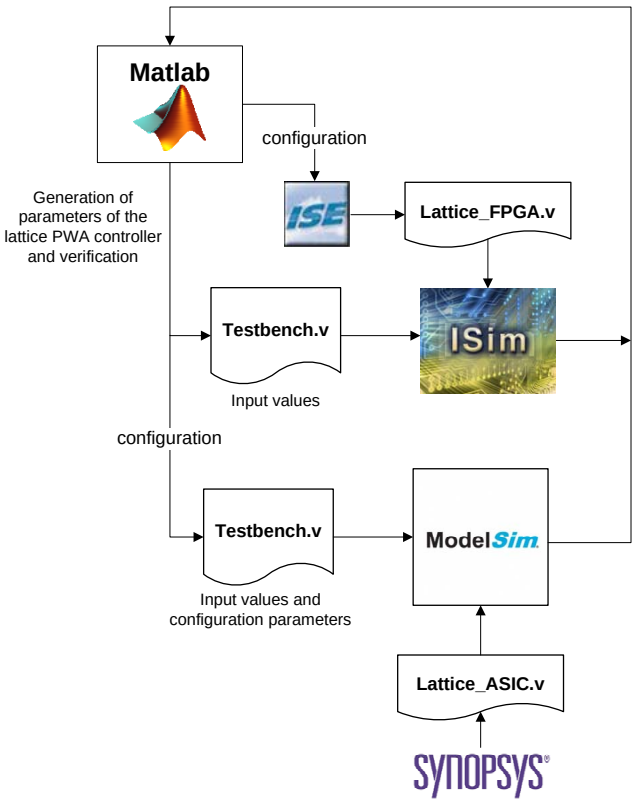


Figure 8. Block diagram of the verification scheme

[38], which can solve control problems using explicit model predictive control approaches.

To illustrate the whole design and verification process of PWA controllers based on lattice representation and implemented as digital circuits with the proposed architecture, three application examples are summarized. The examples considered are the control of a double integrator system, a typical benchmark in the control domain, and two real world applications: an adaptive cruise control (ACC) system and a controller for a buck-boost DC-DC converter. The last two examples are well suited to the inclusion of the proposed VLSI solution: the adaptive cruise control because digital circuits are embedded in current automotive solutions and the DC-DC converter control because a sampling time of only 5 microseconds is employed (which requires fast response from the implementation).

Firstly, the design flow in Fig. 7 was employed to extract the contents of the memories for each particular problem. The verification flow in Fig. 8 was then carried out. Controller performance was verified in an open-loop configuration, by comparing the desired outputs with the current outputs provided by the circuit for a given set of inputs. Controller performance was also verified in a closed-loop configuration, by connecting the simulated circuit with the system to be controlled (the double integrator, the adaptive cruise control and the buck-boost DC-DC converter) modeled in MATLAB&Simulink.

5.1 Double Integrator

Let us consider the double integrator

$$u(t) = \ddot{y}(t) \tag{23}$$

and its equivalent discrete-time state-space representation with a sampling time $T_s = 1$ s.

$$\begin{aligned} x(t+1) &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) \end{aligned} \tag{24}$$

The optimal explicit PWA controller is obtained as the first element of U from (3)-(5) with the weight matrices (Q, R) and the horizons given as follows

$$\begin{aligned} Q &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 0.01 \\ N_y &= N_u = 4 \end{aligned} \tag{25}$$

The state domain is given by $M = [-8, 8] \times [-4, 4]$ and $u_{min=} -1 \leq u(t) \leq 1 = u_{max}$.

The simplest lattice PWA representation for the obtained optimal PWA controller is given by the expression

$$\begin{aligned} f_{PWA}(x) = min\{max\{l_1(x), l_3(x), l_5(x), l_7(x)\}, \\ l_6(x), l_2(x), l_4(x)\} \end{aligned} \tag{26}$$

where the structure and parameter matrices are

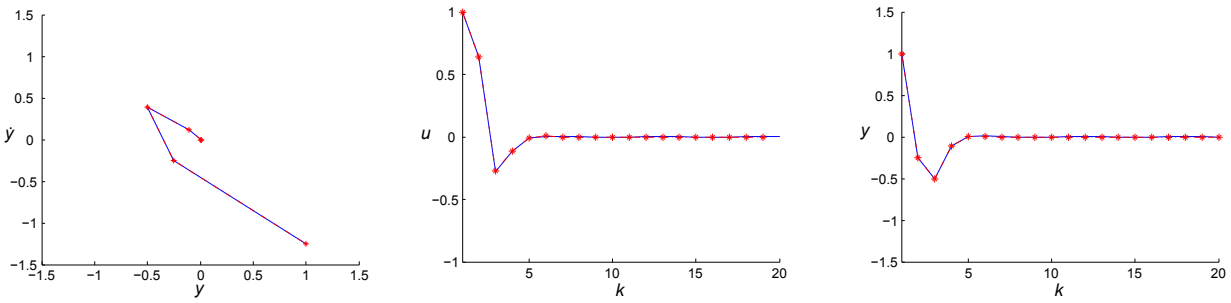
$$\tilde{\psi} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{27}$$

$$\tilde{\phi} = \begin{bmatrix} -0.8166 & -1.7499 & 0 \\ -0.4077 & -1.4014 & 0.8271 \\ -0.4077 & -1.4014 & -0.8271 \\ -0.5528 & -1.5364 & 0.4308 \\ -0.5528 & -1.5364 & -0.4308 \\ -0.32125 & -1.3185 & 1.2127 \\ -0.32125 & -1.3185 & -1.2127 \end{bmatrix} \tag{28}$$

The optimal explicit controller employs 41 affine functions whereas only 7 different affine functions are employed by the lattice PWA representation.

The stars in Fig. 9 show the evolution of the plant state ($\dot{y}$ versus y), the control variable (u) and the plant output (y) at each state (k) when the plant is controlled by the VLSI realization, as described in the verification flow in Section IV.C, with the specifications shown at the beginning of Section IV. The continuous lines show the evolution when the controller is the optimal PWA function obtained with the Moby-dic Toolbox (software results in MATLAB&Simulink with 64 bits of resolution).

Table 2 shows the performance of different PWA controllers for the double integrator plant. The implementations PWAG(A) [18], PWAS(A) and PWAS(B)

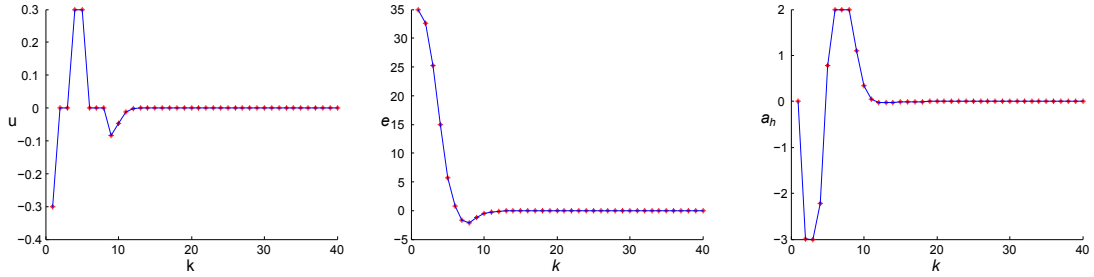


(a) Plant state (b) Control variable (c) System output

Figure 9. Performance of the designed lattice-based PWA VLSI controller for a double integrator system (stars) compared with the optimal controller in software (continuous line)

Table 2
Comparison of PWA controllers for the double integrator system.

	Device	Latency (μs)	Freq. (MHz)	N_{clk}	Memory (KB)	Mult.	Error (MRE)
PWAG (A)	FPGA	1.6	20	32	3.35	1	0.0013
PWAG (B)	ASIC	1.4	20	18	0.99	2	0.016
PWAS (A)	FPGA	0.3	20	6	3.07	1	0.45
PWAS (B)	FPGA	0.15	20	3	9.22	3	0.45
PWAR (A)	FPGA	0.2	20	4	9.22	1	0.29
PWAR (B)	FPGA	0.1	20	2	9.22	2	0.29
PWAH	FPGA	0.55	20	11	0.07	2	0.29
MultiTree*	FPGA	0.06	50	3	-	54	0.38%**
LOTBST*	AVR-XMEGA	5.1-26.9	165-861	-	-	-	-
PWAL	FPGA/ASIC	0.6	20	13	0.03	4	0.00077



(a) Control variable (b) Tracking error (c) Host acceleration

Figure 10. Performance of the designed lattice-based PWA VLSI controller for an adaptive cruise control (stars) compared with the optimal MPC controller in software (continuous line)

[19], PWAR(A) and PWAR(B) [22], and PWAH [20] correspond to realizations on a *Xilinx Spartan 3 FPGA (xc3s200)*. The implementation PWAG(B) [21] corresponds to an ASIC implemented in a 90-nm technology. The realization labeled LOTBST is an algorithm executed on a low-cost AVR-XMEGA processor [39]. The implementation labeled MultiTree was implemented in a *Xilinx Spartan 3-E FPGA (xc3s500e)*. The results of the PWAL realization correspond to the proposed architecture which can be implemented in a FPGA or an ASIC. If the architecture is implemented on ASIC, a maximum frequency of 144MHz can be achieved. All the implementations were coded with 12 bits for the inputs and the parameters except for the one marked with *, where the data was coded with 18 bits. Error (MRE) was calculated as the maximum absolute difference between the optimal MPC control law and the control law in hardware, except in ** where the error was the absolute relative error. The PWAL realization provided the best performance in terms of memory requirements and approximation error (the optimal controller was not approximated), with competitive latency time.

5.2 Adaptive Cruise Control

The second application example is the adaptive cruise control, described in [40]. The goal of this application problem was for a car, called the host vehicle, to follow a preceding vehicle, called the target vehicle, at a desired

Table 3
Comparison between PWA controllers for ACC system.

	Throughput μs	Memory KB	Occ. Slices %	Mult.
PWAG (A)	5.2	3.3	87	1
PWAS (A)	0.2	11.5	31	1
PWAS (B)	0.15	57.6	95	5
PWAL	1.45	1.06	5	8

distance, with full control of the throttle and the brakes of the host vehicle. The host vehicle was defined by its host speed, v_h, and its host acceleration, a_h. The target vehicle was defined by its target speed, v_t, and its target acceleration, a_t, both unknown. The inter-vehicle relative distance, x_r, and relative speed, $v_r = v_t - v_h$ were defined and measured by a radar installed on the host vehicle. Thus, the desired relative distance, $x_{r,d}$, was defined as $x_{r,d} = x_{r,o} + v_{h,d}t_{hw,d}$,$x_{r,o}$ being the desired distance at a standstill and $t_{hw,d}$, the desired headway time.

Taking into account the considerations expressed in [11] the model of the system to be controlled was as follows

$$x_{k+1} = \begin{bmatrix} 1 & -T_s & 0 & T_s t_{hw,d} + \frac{1}{2}T_s^2 \\ 0 & 1 & 0 & -T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_k \tag{29}$$

with the state variable $x_k = \begin{bmatrix} e_k & v_{r,k} & v_{t,k} & a_{h,k} \end{bmatrix}^T$, with $e_k = x_{r,d,k} - x_{r,k}$, $u_k = a_{h,k+1} - a_{h,k}$ and with the following constraints

$$\begin{gathered} x_{r,o} + (v_t - v_r)t_{hw,d} - x_{r,r} \leq e \leq x_{r,o} + (v_t - v_r)t_{hw,d} \\ 0 \leq v_t \leq v_{t,max} \\ v_t - v_{h,max} \leq a_h \leq a_{h,max} \\ j_{h,min} \leq u \leq j_{h,max} \end{gathered} \tag{30}$$

where $j_{h,min}$ and $j_{h,max}$ are the constraints of the host jerk (derivative of the acceleration) and $x_{r,r}$ is the radar range. The values of the constants were: $x_{r,o} = 3.5$, $x_{r,r} = 200$, $t_{hw,d} = 1.5$, $v_{t,max} = 50$, $v_{h,max} = 50$, $a_{h,min} = -3$, $a_{h,max} = 2$, $j_{h,min} = -0.3$, $j_{h,max} = 0.3$. The sampling time was $T_s = 0.1$ s.

Last included element to obtain the optimal explicit PWA controller are the control specifications. The weight matrices and the horizons in are the following

$$\begin{gathered} Q = \begin{bmatrix} 2.5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 1 \\ N_y = N_u = 4 \end{gathered} \tag{31}$$

and the state domain is $\begin{bmatrix} -196 & -50 & 0 & -3 \end{bmatrix}^T \times \begin{bmatrix} 56 & 50 & 50 & 2 \end{bmatrix}^T$ and $-0.3 \leq u \leq 0.3$.

Fig. 10 shows the evolution of the control variable and the host acceleration (a_h). The stars refer to the designed lattice PWA controller implemented as a VLSI digital circuit, as described in the verification flow in Section 4.3, with the specifications shown at the beginning of Section 4. The continuous lines show the evolution when the controller is the optimal PWA controller obtained with the Moby-dic Toolbox [36] (software results in MATLAB&Simulink with 64 bits of resolution). The NRMSE calculated over $N_{pts} = 400$ points in the control variable is 0.037, where NRMSE is calculated as follows

$$NRMSE = \frac{\sum_{i=1}^{N_{pts}} \sqrt{\frac{1}{N_{pts}} \left(\hat{u}(i) - u(i)\right)^2}}{u_{max} - u_{min}} \tag{32}$$

$\hat{u}$ being the control law of the optimal MPC, u the control law of the implemented controller and u_{max} and u_{min} the saturation values of the control law.

Table 3 shows the features of the controller as implemented in different architectures: PWAG (A), PWAS(A), PWAS (B), and the implementation proposed here in a *Xilinx Spartan 3AN (xc3s700AN)* as in [11]. Throughput values are provided for a frequency of 20MHz in all cases.

5.3 Buck-boost DC-DC converter

A lattice controller for a buck-boost DC-DC converter (Fig. 12), as described in [27], was obtained. The control signals were the duty-cycle ratios of switches, (s_1 and s_2), that determined the stages in the buck-boost DC-DC converter: d_1 for the buck stage and d_2 for the boost stage.

The continuous-time model was as follows

$$\begin{gathered} \frac{dv_C}{dt} = \frac{d_2 i_L - i_{load}}{C} \\ \frac{di_L}{dt} = \frac{d_1 v_S - i_L R_L - d_2 R_L}{C} \end{gathered} \tag{33}$$

where $d_i \epsilon [d_{min}, d_{max}]$, v_S is the supply voltage, i_{load} is the load current, i_L is the inductance current and v_C is the capacitor voltage.

The goal of the converter was to maintain the output voltage (v_C) at the reference level V_{ref} while keeping the inputs and states within the limits.

Several considerations were taken into account when employing the model in (33) to compute the MPC control

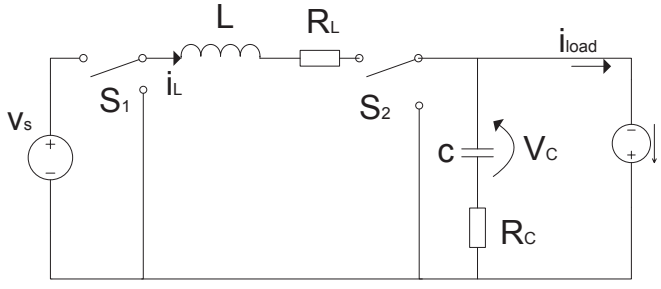


Figure 11. Schematic of buck-boost DC-DC converter

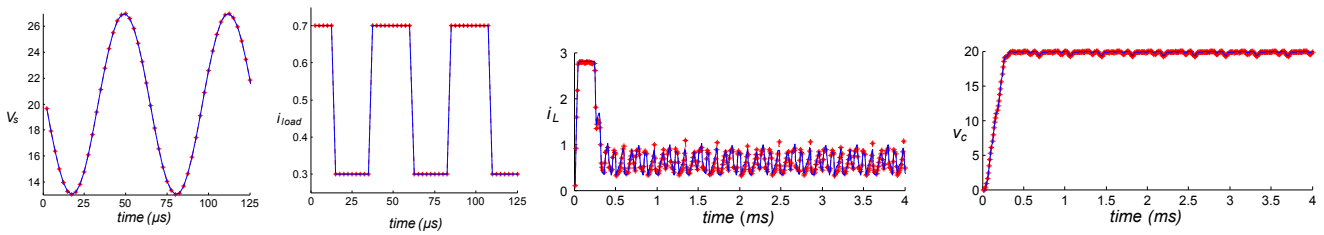


Figure 12. Performance of the designed lattice-based PWA VLSI controller for the buck-boost DC-DC converter (stars) compared with the optimal MPC in software (continuous line)

law. The model was discretized using the Euler forward method, and d_2 was considered constant, $d_2 = ci$, for a given range of $v_S \epsilon \mathbb{V}_i$. The resulting model was affine for the entire sampling period, as follows

$$\begin{array}{c} x_{k+1} = A_i x_k + B u_k + f v_{S,k} \\ v_{S,k} \epsilon \mathbb{V}_i \end{array} \quad (34)$$

with:

$$A_i = \begin{bmatrix} 1 & \frac{T_s c_i}{C} \\ \frac{-T_s c_i}{L} & \frac{-T_s R_L + L}{L} \end{bmatrix}$$
$$B = \begin{bmatrix} 0 \\ \frac{T_s}{L} \end{bmatrix} \quad f = \begin{bmatrix} \frac{-T_s}{C} \\ 0 \end{bmatrix} \quad (35)$$

where $u_k = d_{1,k} \cdot v_{S,k}$, $v_{S,k}$ and $i_{load,k}$ remain constant during each sampling period, T_S and $x = \begin{bmatrix} v_C & i_L \end{bmatrix}^T$.

The optimization problem in (3)-(5) was addressed and solved as a multiparametric mixed-integer linear problem with 4 parameters $\begin{bmatrix} v_{C,k} & i_{L,k} & v_{S,k} & i_{load,k} \end{bmatrix}$ with $N_y = N_c = 2$ and $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ obtaining a PWA function $u(v_C, i_L, v_s, i_{load})$.

The following converter parameters and constraints were used: $R_L = 0.2\Omega$, $L = 220\mu H$, $C = 22\mu F$, $v_{C,min} = 0V$, $v_{C,max} = 22V$, $v_{S,min} = 10V$, $v_{S,max} = 30V$, $i_{load,min} = 0.02A$, $i_{load,max} = 1A$, $i_{L,min} = 0A$, $i_{L,max} = 3A$, $d_{min} = 0$, $d_{max} = 1$, and $V_{ref} = 20V$. The sampling time was $T_s = 5\mu s$.

The lattice PWA representation obtained for the optimal controller is given by

$$\begin{array}{l} f(x) = min\{\, max\{l_1(x), l_3(x)\}, max\{l_2(x), l_3(x)\}, \\ \quad max\{l_3(x), l_4(x)\}, max\,\{l_3(x), l_7(x), l_8(x)\}\,, \\ \quad max\{l_3(x), l_6(x)\}, max\,\{l_3(x), l_7(x), l_9(x)\}\,, \\ \quad max\{l_{14}(x), l_{16}(x)\}, max\{l_8(x), l_{10}(x)\}, l_4(x) \\ \quad max\{l_3(x), l_5(x)\}, max\{l_3(x), l_7(x), l_{10}(x), l_{11}(x)\}, \\ \quad max\{l_5(x), l_7(x)\}, max\{l_3(x), l_7(x), l_{10}(x), l_{12}(x)\}, \\ \quad max\{l_3(x), l_7(x), l_{10}(x), l_{13}(x), l_{14}(x)\}, \\ \quad max\{l_3(x), l_7(x), l_{10}(x), l_{13}(x), l_{15}(x)\}, \\ \quad max\{l_7(x), l_{10}(x), l_{13}(x)\}, l_{14}(x)\}, \\ \quad max\{l_3(x), l_7(x), l_{10}(x), l_{13}(x), l_{18}(x)\}, \\ \quad max\{l_7(x), l_{19}(x)\}, max\{l_{11}(x), l_{13}(x)\}\} \end{array} \quad (36)$$

In this case, the optimal explicit controller employed 144 affine control laws and regions, while the PWAL controller only employed 19 different affine functions in 21 super-regions.

Table 4
Comparison of PWA controllers for buck-boost DC-DC converter.

	Latency μs	Memory KB	Occ. Slices %	Mult.
PWAG_ser	3.15	6.875	14	4
PWAG_par	1.2	6.875	10	4
PWAS_ ser	0.5	12	17	2
PWAS_ par	0.25	60	38	6
PWAL	3.55	1.45	5	8

The stars in Fig. 12 show the evolution of the input voltage source v_S, the load current i_L, the output voltage v_C and output current i_L obtained when the plant was controlled with the lattice PWA implemented as a VLSI digital circuit with 12-bit precision, as described in the verification flow in Section IV.C. The continuous lines show the evolution when the controller was the optimal PWA controller obtained with the Moby-dic Toolbox and implemented in MATLAB&Simulink with 64-bit resolution. The NRMSE was 0.055 calculated as in (32) over $N_{pts} = 10000$ points.

Table 4 compares different PWA controllers for the buck-boost DC-DC converter, PWAG_ser and PWAG_par correspond to the serial and parallel PWAG realizations described in [27]. PWAS_ser and PWAS_par are the serial and parallel versions, respectively, of the PWAS controllers described in [27], and PWAL is the digital VLSI implementation proposed here. All the controllers were implemented in a *Spartan 3AN (xc3s700AN)*. Latency values were provided for a frequency of 20MHz. The PWAL implementation was advantageous in terms of memory and occupation resources.

6 Conclusions

A novel digital implementation for piecewise-affine (PWA) controllers has been presented. The solution is based on the simplest lattice representation of a given PWA controller. It has been implemented in Xilinx FPGAs using the block RAMs required for the particular function, configured as dual-port memories, and the necessary multipliers from those available in the FPGA.

The architecture has also been implemented in a configurable, programmable core for ASIC realizations in TSMC 90-nm CMOS technology. This realization used dual-port intellectual property (IP) memories provided by TSMC. A design flow was developed that made it possible to obtain the bitstream for programming a selected FPGA or the configuration files for configuring and programming the ASIC realization. Some fundamental steps of this design flow were devoted to extracting the contents to be stored in the memories of the proposed architecture. These steps were carried out exclusively in MATLAB. A MATLAB function was also developed to facilitate verification of the VLSI implementation. The proposed solution was evaluated in three application examples (a double integrator, an adaptive cruise control, and the control of a buck-boost DC-DC converter). The generated PWA functions corresponded to explicit model predictive controllers (MPC) obtained with the Moby-dic Toolbox. Compared to other solutions reported in literature for the same examples, the implementation proposed here provides the optimal MPC with very low approximation error, very low memory resources, a very low number of slices (in the case of FPGA realizations), and competitive latency.

ACKNOWLEDGMENTS

This work was partially supported by MOBY-DIC project FP7-INFSO-ICT- 248858 (www.Moby-dic-project.eu) from European Community, and TEC2011-24319 projects from the Spanish Government (with support from FEDER). M.C. Martínez-Rodríguez is supported by FPI fellowship program for Ph.D. Students from Spanish Government. P. Brox is supported by 'V Plan Propio de Investigación' from the University of Seville. The authors acknowledge helpful discussions with A. Oliveri regarding Moby-dic Toolbox.

REFERENCES

[1] E. D. Sontag, "Nonlinear regulation: The piecewise linear approach," *IEEE Trans. on Automatic Control*, vol. 26, pp. 346 – 358, 1981.

[2] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan, "Linear matrix inequalities in control theory," *Studies in Applied Mathematics, SIAM*, vol. 15, 1994.

[3] P. Gahinet, A. Nemirovski, A. J. Laub, and M. Chilali, "The LMI control toolbox," in *IEEE Conf. on Decision and Control*, 1994, pp. 2038 – 2041.

[4] A. Bemporad, F. Borelli, and M. Morari, "Model predictive control based on linear programming: the explicit solution," *IEEE Trans. on Automatic Control*, vol. 47, no. 12, pp. 1974 – 1985, 2002.

[5] M. Lazar and W. Heemels, "Predictive control of hybrid systems: Input-to-state stability results for sub-optimal solutions," *Automatica*, vol. 45, no. 1, pp. 180 – 185, 2009.

[6] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear Model Predictive Control*, ser. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2009, vol. 384, pp. 345 – 369.

[7] T. Johansen, W. Jackson, R. Schreiber, and P. Tϕndel, "Hardware synthesis of explicit model predictive controllers," *IEEE Trans. on Control Systems Technology*, vol. 15, no. 1, 2007.

[8] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, pp. 3 – 20, 2002.

[9] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations," *IEEE Trans. on Automatic Control*, vol. 56, no. 12, pp. 2883 – 2897, Dec. 2011.

[10] M. Mönnigmann and M. Kastsian, "Fast explicit model predictive control with multiway trees," in *18th IFAC world congress 2011 : Milan, Italy*, Sept. 2011, vol. 2, pp. 1356 – 1361.

[11] A. Oliveri, G. Naus, M. Storace, and W. P. M. Heemels, "Low-complexity approximations of pwa functions: A case study on adaptive cruise control," in *20th European Conference on Circuit Theory and Design (ECCTD)*, Aug 2011, pp. 669 – 672.

[12] R. Rovatti, M. Borgatti, and 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, 1998.

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

[14] S. Sánchez-Solano, A. Cabrera, I. Baturone, F. J. Moreno-Velo, and M. Brox, "FPGA implementation of embedded fuzzy controllers for robotic applications," *IEEE Trans. on Industrial Electronics*, vol. 54, no. 4, pp. 1937 – 1945, 2007.

[15] P. Echevarria, M. V. Martínez, J. Echanobe, I. del Campo, and J. M. Tarela, "Digital hardware implementation of high dimensional fuzzy systems," *Applications of Fuzzy Sets Theory, Springer*, pp. 245 – 252, 2007.

[16] I. Baturone, F. J. Moreno-Velo, V. Blanco, and J. Ferruz, "Design of embedded DSP-based fuzzy controllers for autonomous robots," *IEEE Trans. on Industrial Electronics*, vol. 55, pp. 928 – 936, 2008.

[17] T. Poggi, M. Rubagotti, A. Bemporad, and M. Storace, "High-speed piecewise affine virtual sensors," *IEEE Trans. on Industrial Electronics*, vol. 59, no. 2, pp. 1228 – 1237, 2012.

[18] A. Oliveri, T. Poggi, and M. Storace, "Circuit implementation of piecewise-affine functions based on a binary search tree," in *IEEE European Conf. on Circuit Theory and Design*, 2009, pp. 145 – 148.

[19] M. Storace and T. Poggi, "Digital architectures realizing piecewise-linear multi-variate functions: two fpga implementations," *Int. Journal of Circuit Theory and Applications*, vol. 39, no. 1, pp. 1 – 15, 2009.

[20] I. Baturone, M. C. Martínez-Rodríguez, P. Brox, A. Gersnoviez, and S. Sánchez-Solano, "Digital implementation of hierarchical piecewise-affine controllers," in *IEEE Int. Symposium on Industrial Electronics*, Jun. 2011, pp. 1497 – 1502.

[21] P. Brox, J. Castro, M. C. Martínez-Rodríguez, E.Tena, C. Jiménez, I. Baturone, and A. J. Acosta, "A programmable and configurable ASIC to generate piecewise-affine functions defined over general partitions," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 60, no. 12, pp. 3182–3194, Dec. 2013.

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

[23] J. Rodríguez, O. D. Lifschitz, V. M. Jiménez-Fernández, P. Julián, and O. E. Agamennoni, "Application-specific processor for piecewise linear functions computation," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 58, no. 5, pp. 971–981, 2011.

[24] M. Di Federico, T. Poggi, P. Julián, and M. Storace, "Integrated circuit implementation of multi-dimensional piecewise-linear functions," *Digital Signal Processing*, vol. 20, no. 6, pp. 1723 – 1732, 2010.

[25] M. C. Martínez-Rodriguez, I. Baturone, and P. Brox, "Circuit implementation of piecewise-affine functions based on lattice representation," in *20th European Conf. on Circuit Theory and Design*, Aug. 2011, pp. 644 – 647.

[26] M. C. Martínez-Rodríguez, I. Baturone, and P. Brox, "Design methodology for FPGA implementation of lattice piecewise-affine functions," in *IEEE Int. Conf. on Field-Programmable Technology*, Dec. 2011, pp. 1 – 4.

[27] V.Spinu, A.Oliveri, M.Lazar, and M.Storace, "Fpga implementation of optimal and approximate model predictive control for a buck-boost dc-dc converter," in *IEEE International Conference on Control Applications (CCA), 2012*, Oct 2012, pp. 1417–1423.

[28] J. M. Tarela and M. V. Martínez, "Region configurations for realizability of lattice piecewise-linear models," *Mathematical and Computer Modelling*, vol. 30, no. 11 - 12, pp. 75 – 83, 1999.

[29] C. Wen, X. Ma, and B. E. Ydstie, "Analytical expression of explicit MPC solution via lattice piecewise-affine function," *Automatica*, vol. 45, no. 4, pp. 910 – 917, 2009.
[30] F. Bayat, "Comments on "analytical expression of explicit MPC solution via lattice piecewise-affine function " Automatica 45 (2009) 910 - 917," *Automatica*, vol. 48, no. 11, pp. 2993 – 2994, 2012.
[31] "Reply to "comments on "analytical expression of explicit MPC solution via lattice piecewise-affine function" Automatica 45 (2009) 910 - 917"," *Automatica*.
[32] J. Tarela, J. Perez, and V. Aleixandre, "Minimization of lattice polynomials on piecewise linear functions (part I)," *Mathematics and Computers in Simulation*, vol. 17, no. 2, pp. 79 – 85, 1975.
[33] J. Tarela, L. Bailon, and E. Sanz, "Minimization of lattice polynomials on piecewise linear functions (part II)," *Mathematics and Computers in Simulation*, vol. 17, no. 2, pp. 121 – 127, 1975.
[34] P. Tϕndel, T. A. Johansen, and A. Bemporad, "Computation and approximation of piecewise affine control laws via binary search trees," in *IEEE Conf. on Decision and Control*, vol. 3, 2002, pp. 3144 – 3149.
[35] P. Julián, A. Desages, and O. Agamennoni, "High-level canonical piecewise linear representation using a simplicial partition," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 46, no. 4, pp. 463 – 480, 1999.
[36] "MOBY-DIC toolbox." [Online]. Available: http://ncas.dibe.unige.it/software/MOBY-DICToolbox/
[37] A. Bemporad, "Hybrid Toolbox - User's Guide," 2004, http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox.
[38] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004. [Online]. Available: http://control.ee.ethz.ch/~mpt/
[39] F. Bayat, T. A. Johansen, and A. A. Jabli, "Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC," *IEEE Trans. on Control Systems Technology*, 2011.
[40] G. Naus, J. Ploeg, M. V. de Molengraft, W. Heemels, and M. Steinbuch, "Design and implementation of parameterized adaptive cruise control: An explicit model predictive control approach," *Control Engineering Practice*, vol. 18, no. 8, pp. 882 – 892, 2010.