

UNIVERSIDAD DE SEVILLA

DOCTORAL THESIS

---

**Design of Trusted Piecewise Affine  
Controllers and Virtual Sensors into  
CMOS Integrated Circuits**

---

*Author:*

Macarena C. MARTÍNEZ RODRÍGUEZ

*Supervisors:*

Dr. Iluminada BATURONE

Dr. Piedad BROX

*A thesis submitted for the degree of  
Doctor of Philosophy*





## *Acknowledgements*

This research work has been developed at the Instituto de Microelectrónica de Sevilla (IMSE, CNM), CSIC, Universidad de Sevilla. This work was supported in part by TEC2014-57971-R, IPT-2012-0695-390000, TEC2011-24319, TEC2008-04920 projects from the Spanish Government, P08-TIC-03674 project from the Andalusian Regional Government (with support from the PO FEDER-FSE), and FP7-INFISO-ICT-248858 project from European Community. It was supported by FPI fellowship program for Students from Spanish Government (October 2013- January 2018). During 2015 a stay of 3 months has been accomplished at the “Computer Security and Industrial Cryptography group (COSIC)” of the KU Leuven, Belgium supported also by FPI fellowship program.

I would like to thank my thesis directors for their patient guidance, encouragement and advice during this stage. Firstly, I would like to thank Lumi for contributing with her enormous capacity for global and critical vision, as well as her meticulousness and perseverance. I would also like to thank Piedad for her encouragement and guidance on a daily basis achieving a relationship beyond the professional perspective.

I’d like to thank anyone who invest his time on helping me and make my work more fruitful, particularly Adrián, Lola and Jose Miguel, all the colleges that have worked in PB-12 to make our office a good environment to work each day, and all the members of the TIC-180 (¡ánimo equipo!).

Por último, dar las gracias a mis padres por dejarme soñar, equivocarme y crecer. También a mis hermanas por las risas y a Jorge por todo lo compartido. Y pedir perdón a Leo por no sentirme culpable de quitarle su tiempo. Estoy convencida que en el futuro lo comprenderá.



# *Preface*

I started my research activities collaborating in the tasks of the European Project MOBY-DIC (“Model-based synthesis of digital electronic circuits for embedded control”, FP7-ICT: 2009-2012). This project was focused on developing a unique paradigm and supporting tool chain for the design of embedded control systems, with the aim of reducing both design costs and time to market. Particularly, the project was centred on applying PieceWise Affine (PWA) controllers into automotive products. My participation in this project was the germ of this Ph.D. study: the design of PWA functions into Complementary Metal-Oxide-Semiconductor (CMOS) Integrated Circuits (ICs).

Nowadays, there is a growing interest in smart devices with not only embedded controllers (to act in response to measures) but also embedded sensors (to measure variables of interest either direct or virtually), thus providing autonomous intelligence in the physical system they are included. Consumer products in our homes, professional equipment, health care, and automotive and transportation systems all heavily rely on embedded control and sensing technologies. The Internet of Things (IoT) ecosystem is a clear deployment scenario for these devices.

The hypothesis in this Ph.D. work was that trusted hardware realizations of PWA functions into configurable and programmable CMOS ICs could increase widely the horizons of application of embedded controllers and virtual sensors. To validate this hypothesis, the first challenge was the development of architectures that overcome the proposals in the state-of-the-art. To implement and verify them, a hardware design flow was developed and employed to generate PWA functions with Application Specific Integrated Circuits (ASICs) and programmable devices as Field-Programmable Gate Arrays (FPGAs).

In order to reduce time-to-market of these designs, an automated and integrated design flow was used to apply them as embedded controllers and virtual sensors. In the case of embedded control, the design flow is based on a model-based methodology, which applies model predictive control (MPC) and uses a Matlab toolbox to extract the PWA controllers from mathematical models of the physical system to control, obtaining the parameters needed to program and configure the proposed ASIC and FPGA hardware realizations of the PWA controllers. Several cases of study were addressed to illustrate the high-performance in terms of area, power,

and speed of the resulting embedded controllers into nanometer CMOS ICs.

In the case of embedded sensors, PWA virtual sensors are employed to model non-linear relations between input variables which are easy to measure and output variables which are costly to evaluate directly. The parameters needed to program and configure the proposed ASIC and FPGA hardware realizations of the PWA virtual sensors were obtained with an identification algorithm that uses numerical input-output data obtained from experiments or simulations. Since PWA systems are a subset of fuzzy systems, a design methodology based on Xfuzzy environment is presented. A case of study from the automotive domain was selected to illustrate the performance of the resulting nanometer CMOS ICs acting as virtual sensors.

In spite of the importance of embedded control and sensing systems in daily life, there is a surprising lack of security primitives to protect them. The inclusion of security is crucial to guarantee not only data privacy and authenticity but also device authenticity, thus increasing the horizons of application of embedded controllers and virtual sensors. Hence, another challenge addressed in this Ph.D. Dissertation was the design of trusted PWA hardware. For that purpose, lightweight cryptographic primitives were analysed and the most suitable were selected to be included in the proposed ASIC realizations. A prototype of trusted virtual sensor that offers privacy, authenticity and integrity of the virtually sensed measurement and the circuit itself was designed into a 90-nm CMOS ASIC.

In summary, the goals addressed with this Ph.D. work are the following:

- Review the state-of-the-art in the microelectronic design of PWA functions for model predictive control (MPC) and virtual sensing.
- Review the state-of-the-art in cryptographic primitives suitable to be included in embedded PWA controllers and virtual sensors.
- Design of efficient hardware architectures to generate PWA functions.
- Design of trusted hardware for embedded control and sensing using PWA functions and cryptographic primitives.
- Develop a high-level design flow to implement and verify PWA solutions in FPGAs and ASICs for control and sensing applications.

- Validate the architectures and design flow with ASIC and FPGA prototypes using a CMOS nanometric technology.
- Evaluation of the proposed PWA solutions in hardware with cases of study of embedded control and virtual sensing domains.

The Thesis is organized in eight chapters. The first chapter introduces the main issues addressed with this research work. The following six chapters gather the main results. Dissertation culminates with the discussion and conclusions of the achievements.





# Contents

<b>Acknowledgements</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Piecewise affine (PWA) functions . . . . .	5
1.1.1 Generic PWA form based on binary search tree (PWAG)	6
1.1.2 Generic PWA form based on lattice (PWAL) . . . . .	7
1.1.3 Simplicial PWA functions (PWAS) . . . . .	10
1.1.4 Hyperrectangular PWA functions (PWAR) . . . . .	11
1.2 PWA forms for Model Predictive Control (MPC) . . . . .	12
1.3 PWA forms for virtual sensors . . . . .	15
1.4 Trusted controllers and sensors . . . . .	16
1.5 Conclusions . . . . .	17
<b>2 Hardware architectures to implement PWA functions</b>	<b>19</b>
2.1 Architecture for PWAG form . . . . .	19
2.2 Architecture for PWAL form . . . . .	19
2.3 Architecture for multiple PWA forms . . . . .	19
2.4 Comparison between architectures . . . . .	20
2.5 Conclusions . . . . .	22
<b>3 Hardware design flow to implement PWA functions</b>	<b>23</b>
3.1 Hardware design flow for ASICs . . . . .	23
3.1.1 Manufactured ASICs . . . . .	23
3.2 Hardware design flow for FPGAs . . . . .	24
3.3 Hardware verification . . . . .	24
3.4 Conclusions . . . . .	24
<b>4 PWA solutions in hardware for Model Predictive Control</b>	<b>25</b>
4.1 The Moby-dic Toolbox . . . . .	25

- 4.2 Methodology to configure and program PWA ASICs with the Moby-dic Toolbox . . . . . 26
- 4.3 Methodology to extract PWAL form from Moby-dic Toolbox 27
- 4.4 Application examples . . . . . 27
- 4.5 Conclusions . . . . . 28
- 5 PWA solutions in hardware for virtual sensors 29**
  - 5.1 Identification algorithm for PWAR virtual sensors . . . . . 29
  - 5.2 Methodology using Xfuzzy environment . . . . . 32
  - 5.3 Application example . . . . . 33
  - 5.4 Conclusions . . . . . 35
- 6 Cryptographic modules for PWA solutions in hardware 37**
  - 6.1 Description of candidate modules . . . . . 38
    - 6.1.1 Hash-based Message Authentication Code (HMAC) based on the PHOTON hash function . . . . . 38
    - 6.1.2 AEGIS Authenticated Cipher . . . . . 38
    - 6.1.3 ASCON Authenticated Cipher . . . . . 38
  - 6.2 Comparative analysis of the cryptographic modules . . . . . 38
  - 6.3 Conclusions . . . . . 38
- 7 ASIC design of a trusted PWA virtual sensor 39**
  - 7.1 Functional and architectural description . . . . . 39
    - 7.1.1 PWAR Unit . . . . . 40
    - 7.1.2 Cryptographic Unit . . . . . 42
    - 7.1.3 Control Unit . . . . . 43
  - 7.2 Features of the trusted virtual sensor . . . . . 44
  - 7.3 Implementation results . . . . . 45
  - 7.4 Conclusions . . . . . 47
- 8 Conclusions 49**
- A Brief CV 53**
  - A.1 Journal Papers . . . . . 53
  - A.2 Conference Papers . . . . . 54
  - A.3 Other Merits . . . . . 55
  - A.4 Projects . . . . . 56
- Bibliography 57**

# List of Figures

1.1	Example of a PieceWise Affine function of a two-dimensional domain. . . . .	2
1.2	Different partitions of a bi-dimensional input domain into polytopes. . . . .	3
1.3	(a) A binary search tree (BST). (b) Polytopes. . . . .	6
1.4	Example of a one-dimensional PWA function. . . . .	9
1.5	Example of a hyperrectangular single-resolution partition of a two-dimensional domain. . . . .	12
4.1	Design flows for circuits implementing controllers: (a) Traditional (b) Moby-dic proposal. . . . .	26
4.2	Scheme of the Moby-dic Toolbox. The functionalities added with our research are depicted in gray. . . . .	27
5.1	Vehicle yaw rate estimation by a PWAR virtual sensor. . . . .	33
5.2	HIL simulation of the PWAR virtual sensor. . . . .	34
7.1	Architectural scheme of the proposed CMOS sensor. . . . .	41
7.2	State diagram of the Control Unit. . . . .	43
7.3	Timing of configuration mode. . . . .	45
7.4	Timing of trusted sensing mode. . . . .	46
7.5	Layout of the trusted virtual sensor using AEGIS. . . . .	47



# List of Tables

2.1	Comparison between architectures. . . . .	20
5.1	Error comparison between virtual sensors. . . . .	34
7.1	Information in the NVM used by the AEGIS module. . . .	45
7.2	Area and power consumption during trusted sensing mode.	46



# List of Algorithms

1	Pseudo-code of PWAR virtual sensor algorithm. . . . .	31
---	---	----





*A mis padres.*

*A Leo.*



## Chapter 1

# Introduction

From long time ago, PieceWise Affine (PWA) functions have been widely employed to approximate nonlinear systems in many theoretical domains, such as circuit or control theory, due to their low computational cost and low analytical complexity [1–7]. Many engineering applications in the embedded control area demand hardware implementations of PWA functions to meet the size, power and/or response time constraints imposed not only by the need of generating a control action but also to estimate the variables involved in the control problem [8–12]. PWA functions are of immediate application in Model Predictive Control (MPC) [7, 9–15] or problems solved by Fuzzy Logic-based systems [16–20]. PWA functions are also employed to implement virtual sensors [8, 21–23]. They are usually implemented as software installed in the electronics units but faster responses, smaller sizes and lower power consumption are obtained with hardware implementations. Other application of the PWA functions are approximation of dynamical systems [5, 24], and neural networks [25].

A PWA function provides a linear (affine) output for each region in which the input domain is partitioned. An example of a PWA function with two inputs and one output is shown in Fig. 1.1. The number of regions of this example is 12.

PWA functions have been studied in circuit theory for a long time [26, 27]. Many contributions on the electronic implementation of multi-input or multi-variate PWA functions can be found. Among them, analog implementations [28–32] and mixed-signal [32, 33] may not be robust enough for certain applications, so digital implementations are usually proposed, either on Digital Signal Processor (DSP) boards [16, 17], or reconfigurable hardware, as for instance Field Programmable Gate Arrays (FPGAs) [10,

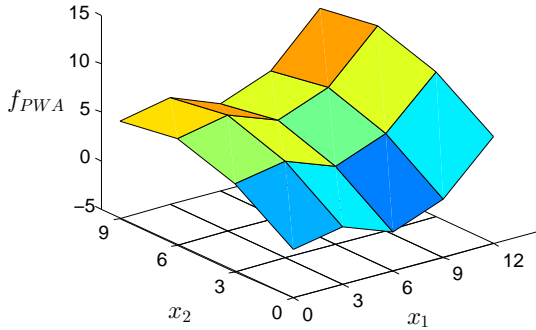


FIGURE 1.1: Example of a Piecewise Affine function of a two-dimensional domain.

18, 34–38], or on Application Specific Integrated Circuits (ASICs) [39, 40]. Other digital circuit architectures for MPC are conceived to be implemented on FPGAs or ASICs [9, 12]. Concerning ASIC implementations, the two latter works include a feasibility study in 130-nm Complementary Metal-Oxide-Semiconductor (CMOS) technology. DSP and FPGA implementations are easy to design and to adjust to the application. The software executed by a DSP-based realization can be changed depending on the application and the hardware employed by a FPGA-based solution can be configured after manufacturing (since it is field programmable). Very Large Scale Integration (VLSI) implementations on ASICs provide the highest performance in terms of size, power and speed, and are the cheapest solution when the number of units required is high. Increasingly, the tendency is to handle applications that require embedded control solutions in the digital domain (automotive, robotics, aviation, automation systems, etc.), where ASIC solutions are of immediate applications [41–43]. As drawback, they are not so versatile as DSPs or FPGAs. The advantages and drawbacks of several platforms for embedded systems and fuzzy logic-based systems are described in [44] and [45], respectively.

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. The most widespread form of generating a PWA function is to allow any kind of polytopic partition (Fig. 1.2(a)). A way to evaluate generic PWA functions requires the exploration of a binary search tree (BST) in order to find the polytope that the input belongs to, but this

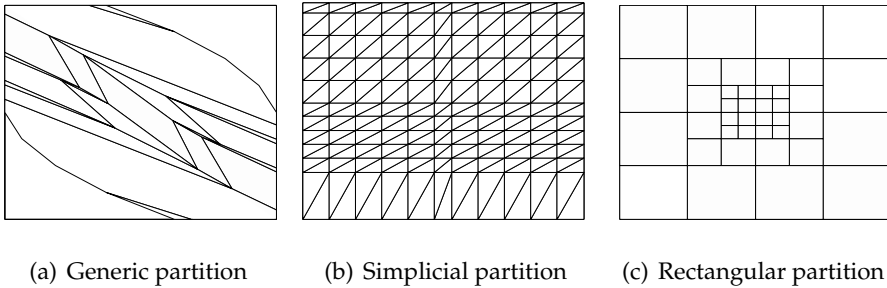


FIGURE 1.2: Different partitions of a bi-dimensional input domain into polytopes.

can be a slow solution if the tree depth is high [46]. Such generic PWA functions evaluation is referred to as PWAG form. This realization strategy has been implemented digitally in both FPGAs and ASICs [9, 34].

A way to accelerate the exploration of the BST is described in [14]. It is based on multiway trees that compute in parallel several branches of the BST.

A solution to reduce the BST complexity is proposed in [15]. It is based on the concept of hash tables and the associated hash functions. Hence, it is called PWAG-hash. The algorithm is based on two stages. During the first stage, the domain is divided into hyper-rectangular partitions (see Fig. 1.2(c)) and the polytopes into a determined hyper-rectangle are identified. In the second stage, a simpler BST is applied for the subset of polytopes identified in the previous stage. A hardware implementation of this algorithm is presented in [35].

Another way to evaluate generic PWA functions is the lattice form (PWAL), which does not need to find the polytope where the input is located and therefore, requires many fewer parameters to define the PWA function over any kind of polytopic partition [47, 48]. PWAL is mainly used to address continuous PWA functions.

Other implementations generate PWA functions defined over particular partitions of the input domain [4, 11, 16, 32, 36, 37, 39]. Simplicial PWA (PWAS) functions are defined over non-overlapping hypertriangular regions called simplices (1.2(b)). One great advantage of PWAS functions is that, given the simplicial partition, they can be designed easily to approximate any non-linear function by applying least-square minimization.

Several digital architectures for implementing PWAS functions have been reported in literature [16, 19, 32, 37, 40, 49, 50]. A solution based on DSPs is reported in [16], and a mixed-signal ASIC is described in [32]. More recently, an architecture has been developed to implement PWAS functions with FPGAs. Two versions of this architecture, one parallel and one serial, are described in [37]. However, PWAS functions are affected by two limitations. Firstly, the number of parameters to define a PWAS function grows exponentially with the number of input variables, which is known as the “curse of dimensionality”. This is why reported ASIC implementations for PWAS functions use a low number of partitions per dimension, thus generating a reduced subset of PWA functions [19, 39, 40]. Secondly, not all PWA functions can be expressed by a PWAS function, but only a subset of them.

A simple partition of the input domain is a hyperrectangular partition (1.2(c)). PWA functions defined over hyperrectangular partitions (PWAR) have been employed to approximate explicit model predictive controllers in [51]. The input domain can be divided into single-resolution partitions, thus leading to a domain partitioned into equally-sized hyperrectangles. Other option is the division of each state axis of the input domain not just into intervals with a single resolution, but instead divided subsequently into higher and higher levels of resolution by halving the subintervals each time, up to some chosen maximum resolution. This partition of the input domain is known as multi-resolution hyperrectangular. Digital architectures for the implementation of PWAR functions in FPGAs have been introduced in [36]. From a hardware point of view, PWAR functions are even simpler than PWAS ones. As drawback, their approximation capability is inferior.

Hardware solutions that implement simplicial and hyper-rectangular approaches usually provide faster responses than solutions that implement a generic approach, PWAG or PWAL.

Interconnected PWA functions working in a hierarchical way called hierarchical PWA (PWAH) have also been reported together with solutions implemented on FPGAs [CV14] and [38]. PWAH consists of single-input single-output PWA modules connected in cascade so that the memory required increases linearly instead of exponentially with the input dimension. Their problem is that they are not suitable to implement any PWA function but a subset of them.

Following the mathematical formalism associated to the PWA functions, a review of the state-of-the-art of the PWA functions for MPC and for virtual sensors, and their hardware security concerns is presented in the rest of the chapter.

## 1.1 Piecewise affine (PWA) functions

A generic PWA function with multiple inputs and multiple outputs,  $F_{PWA}(x): \mathbb{R}^n \rightarrow \mathbb{R}^m$ , can be represented by:

$$F_{PWA}(\mathbf{x}) = \begin{cases} \sum_{j=1}^n f_{ij}^1 \cdot x_j + f_{i0}^1, & \text{if } \mathbf{x} \in \Omega_i \\ \dots \\ \sum_{j=1}^n f_{ij}^m \cdot x_j + f_{i0}^m, & \text{if } \mathbf{x} \in \Omega_i \end{cases} \quad (1.1)$$

where  $\mathbf{x} \in S \subset \mathbb{R}^n$ ,  $F = \{f^1 \dots f^m\}$  is a set of matrices of dimensions  $(n+1) \times \Theta$ , and the  $S$  domain is divided into  $\Theta$  polytopes:

$$S = \bigcup_{i=1}^{\Theta} \Omega_i \quad (1.2)$$

Figure 1.2(a) shows a two-dimensional input domain that is divided into 21 polytopes.

The PWA function is defined over a  $n$ -dimensional compact  $S$  domain as follows:

$$S = \{\mathbf{x} \in \mathbb{R}^n : a_j \leq x_j \leq b_j, \quad j = 1, \dots, n\} \quad (1.3)$$

For a single output, the expression in (1.1) is as follows:

$$f_{PWA}(\mathbf{x}) = \sum_{j=1}^n f_{ij} \cdot x_j + f_{i0}, \quad \text{if } \mathbf{x} \in \Omega_i \quad (1.4)$$

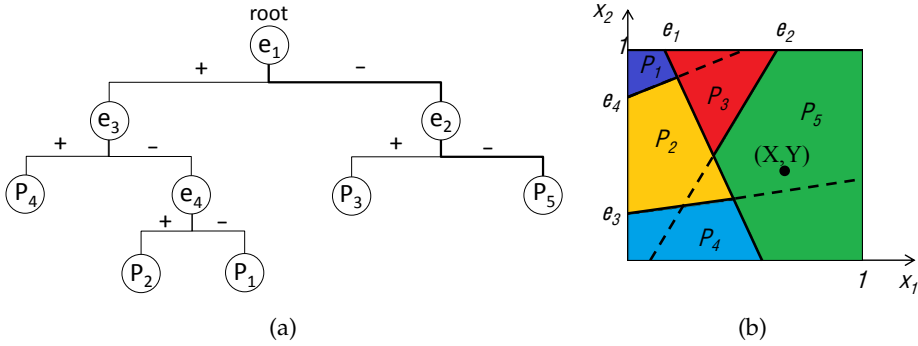


FIGURE 1.3: (a) A binary search tree (BST). (b) Polytopes.

For the sake of simplicity, different strategies to implement PWA functions with a single output are summarized in the following.

### 1.1.1 Generic PWA form based on binary search tree (PWAG)

In order to evaluate  $f_{PWA}(x)$  (eq. 1.4), the first required step is to find the index  $i$  such that  $x \in \Omega_i$ . Something that is similar to the point location problem in the computational geometry field. This problem can be solved by constructing a BST, as presented in [46]. By exploring the tree on line, from the root to a leaf, it is possible to locate the polytope containing the input vector. A relatively small number of boundary conditions ( $\sum_{j=1}^n h_{kj} \cdot x_j + h_{k0} < 0$  with  $k = 1, \dots, E$ ) have to be computed. The tree is usually constructed to minimize its depth (maximum distance between the root and the leaves) by a proper node-boundary assignment. In the same way, a maximum symmetry of the tree is desired in order to optimize timing performances. The result is that the time to evaluate the PWA function can be logarithmic in the number of polytopes [52].

As example, a binary search tree is illustrated in Fig. 1.3. The way this tree is explored for the point  $(X, Y)$  depicted in Fig. 1.3(b) is highlighted in Fig. 1.3(a). Given an input  $x$ , the tree is explored starting from the root by checking if  $\sum_{j=1}^n h_{kj} \cdot x_j + h_{k0} < 0$  (to see if the input point is at one side of the domain split by the first edge,  $e_1$ , or at the other). If this condition holds true, the right branch is selected, otherwise the choice is the left branch. This procedure is repeated until a leaf node is reached. The path followed is  $e_1 \leq 0 \rightarrow e_2 \leq 0 \rightarrow P_5$ .



The point location problem requires the iterative evaluation at each tree node of affine functions whose parameters ( $h_k$ ) are real values. Once the search is finished, the generation of the PWA function requires the evaluation of another affine function whose parameters ( $f_i$ ) are also real values.

### 1.1.2 Generic PWA form based on lattice (PWAL)

According to the notation in [48], the expression in eq. (1.4) can be expressed as:

$$f_{PWA}(\mathbf{x}) = \sum_{j=1}^n f_{ij} \cdot x_j + f_{i0} = l(\mathbf{x}|f_i) = l_i(\mathbf{x}) \text{ if } \mathbf{x} \in \Omega_i \quad (1.5)$$

PWAL functions are able to implement any continuous PWA function by using its parameter,  $F$  and structure,  $\psi$ , matrices [47], as follows:

$$f_{PWAL}(\mathbf{x}) = L(\mathbf{x}|F, \psi) = \min_{1 \leq i \leq \Theta} \left\{ \begin{array}{l} \max_{1 \leq j \leq W} \{l_j(\mathbf{x})\} \\ \psi_{ij} = 1 \end{array} \right\} \quad (1.6)$$

The structure matrix,  $\psi=[\psi_{ij}]$  is a  $\Theta \times W$  zero-one matrix obtained by applying the methodology described in [48], which indicates (with  $\psi_{ij} = \begin{cases} 1 \\ 0 \end{cases}$ ) which affine functions are considered. Its elements are calculated as:

$$\psi_{ij} = \begin{cases} 1 & \text{if } l_i(\mathbf{x}) \geq l_j(\mathbf{x}), \forall \mathbf{x} \in \Omega_i \\ 0 & \text{otherwise} \end{cases} \quad (1.7)$$

The parameter matrix,  $F=[f_1, \dots, f_W]$ , is a  $W \times (n + 1)$  matrix whose rows,  $f_i$ , contain the coefficients and the offset of the affine functions used by the PWA function ( $W \leq \Theta$ ).

For a multiple-output PWAL function, different structure and parameter matrices are evaluated for each output.

The value of the  $f_{PWAL}(\mathbf{x})$  is evaluated by the following the steps: (i) read successively the elements  $(\psi_{ij})$  of the structure matrix, starting from the first row and the first column, and, for each row,  $i$ , retrieve  $f_j$  if  $\psi_{ij} = '1'$ , (ii) evaluate the maximum between all the affine expressions for each row, and (iii) calculate the minimum between the maximum results.

### 1.1.2.1 The simplest PWAL form

The way to obtain the simplest PWAL form of a continuous PWA function was addressed in [48, 53–56]. Herein, the algorithm used to find the simplest PWAL form of a PWA function with the form described in (1.6) is the one presented in [48] and updated in [53] and [54]. The steps of the algorithm can be summarized as follows:

- Given an explicit PWA function, record the local affine functions  $(l_i(x), 1 \leq i \leq \Theta)$ , the constrained inequalities  $(\sum_k^n h_{kj} \cdot x_j + h_{k0} \leq 0, k = 1, \dots, E)$  that define the polytopes, and the  $v_k$  vertexes of each polytope  $\Omega_i$ .
- Calculate the values of each affine function at each set of vertexes,  $l_i(v_k)$ .
- Calculate the structure matrix using

$$\psi_{ij} = \begin{cases} 1 & \text{if } l_i(v_k) \geq l_j(v_k), 1 \leq k \leq k_i \\ 0 & \text{otherwise} \end{cases} \quad (1.8)$$

for  $1 \leq i, j \leq \Theta$ .

- Simplify the rows in the structure matrix as described in *Lemma3* in [48]. 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 [54]. After this last step, super-regions that are not involved in the PWA function are removed.

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

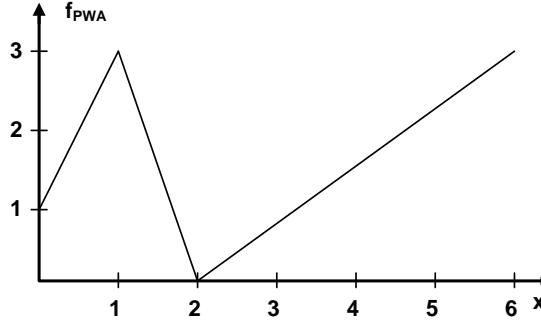


FIGURE 1.4: Example of a one-dimensional PWA function.

The simplest PWAL form,  $L(x|\tilde{F}, \tilde{\psi})$ , of the PWA function,  $f_{PWA}(x)$ , is the following:

$$L(x|\tilde{F}, \tilde{\psi}) = \min_{1 \leq i \leq X} \left\{ \max_{1 \leq j \leq V, \tilde{\psi}_{ij}=1} \left\{ \tilde{l}_j(x) \right\} \right\}, \forall x \in \mathbb{R}^n \quad (1.9)$$

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  $\tilde{\psi}$ .

As example, let us consider the PWA function shown in Fig. 1.4 and described as follows:

$$f_{PWA}(x) = \begin{cases} l_1(x) = 2x + 1 & \text{if } 0 \leq x \leq 1 \\ l_2(x) = -3x + 6 & \text{if } 1 \leq x \leq 2 \\ l_3(x) = 0.75x - 1.5 & \text{if } 2 \leq x \leq 6 \end{cases} \quad (1.10)$$

The structure matrix,  $\psi$ , calculated as in 1.7 and the parameter matrix,  $F$ , are given by:

$$\psi = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad F = \begin{bmatrix} 2 & 1 \\ -3 & 6 \\ 0.75 & -1.5 \end{bmatrix} \quad (1.11)$$

The PWAL form calculated as in 1.6 is the following:

$$f_{PWA}(x) = \min \left\{ \begin{array}{l} \max \{l_1(x), l_3(x)\}, \\ \max \{l_2(x), l_3(x)\}, \\ \max \{l_2(x), l_3(x)\} \end{array} \right\} \quad (1.12)$$

which can be simplified.

The simplification procedure explained above allows obtaining the simplified structure matrix,  $\tilde{\psi}$ , and the simplified parameter matrix,  $\tilde{F}$ . In this case, only the structure matrix is simplified:

$$\tilde{\psi} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \tilde{F} = \begin{bmatrix} 2 & 1 \\ -3 & 6 \\ 0.75 & -1.5 \end{bmatrix} = F \quad (1.13)$$

Hence, the simplest PWAL form of the function in Fig. 1.4 is given by:

$$f_{PWA}(x) = \min \{ \max \{l_1(x), l_3(x)\}, \max \{l_2(x), l_3(x)\} \} \quad (1.14)$$

### 1.1.3 Simplicial PWA functions (PWAS)

PWAS functions are characterized by a regular partition.

The generic instance in (1.3) can always be reported to this particular instance:

$$S = \{z \in \mathbb{R}^n : 0 \leq z_i \leq m_i, \quad i = 1, \dots, n\} \quad (1.15)$$

The equation in (1.15) is obtained by applying a proper affine transformation [57]:

$$z = T(x) \quad (1.16)$$

We also assume that every dimensional component  $z_i$  of the  $S$  domain is partitioned into  $m_i$  subintervals of unitary length. Consequently, the  $S$  domain is partitioned into  $\prod_{i=1}^n m_i$  regions and contains  $N = \prod_{i=1}^n (m_i + 1)$  vertices [32]. Each one of these regions can be further partitioned into  $n!$

non-overlapping regions called simplices. The PWAS function is affine over each simplex of the partitioned  $S$  domain.

The value of the PWAS function can be evaluated as a linear interpolation of a subset of  $n + 1$  coefficients  $c_j$  (for an  $n$ -dimensional function):

$$f_{PWAS}(z) = \sum_{j=0}^n \mu_j c_j \quad (1.17)$$

where the interpolation weights  $\mu_j$  depend on  $z$ , the vertices surrounding  $z$  are  $n + 1$  [32], and  $c_j$  are the values of the function at the vertices  $v_j$  of the simplex containing  $z$  are  $c_j$ .

The value of  $f_{PWAS}(z)$  is evaluated following the steps: (i) find the vertices of the simplex that the input  $z$  belongs to, (ii) retrieve  $c_j$  from a memory, (iii) calculate the coefficients  $\mu_j$  for the given input  $z$ , and (iv) evaluate the affine expression  $\sum_{j=0}^n \mu_j c_j$ . For a multiple-output PWAS function, this process is repeated for each output variable using different coefficients ( $c_j$ ) for each one.

#### 1.1.4 Hyperrectangular PWA functions (PWAR)

PWAR functions are also characterized by a regular partition of the  $S$  domain. Each axis of the  $S$  domain can be subdivided into  $m_{R_i}$  subintervals of the same amplitude:

$$\alpha_i = \frac{b_i - a_i}{m_{R_i}} \quad (1.18)$$

and

$$m_{R_i} = 2^{t_i}, \text{ with } t_i \in \mathbb{N} \quad (1.19)$$

yielding many possible partitions.

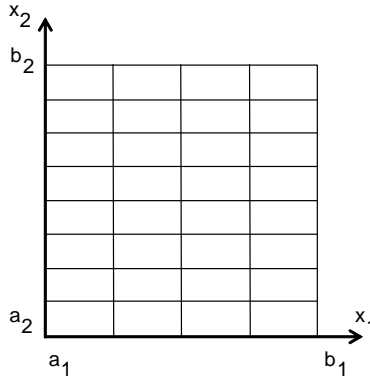


FIGURE 1.5: Example of a hyperrectangular single-resolution partition of a two-dimensional domain.

Consequently, the domain is divided into  $\prod_{i=1}^n m_{R_i} = 2^{\sum_{i=1}^n t_i}$  identical hyperrectangles. This kind of partition is called a ‘single-resolution hyperrectangular partition’ [36]. Figure 1.5 shows a two-dimensional  $S$  domain partitioned into single-resolution hyperrectangles with  $m_{R_1} = 2^2 = 4$ , and  $m_{R_2} = 2^3 = 8$ .

The value of  $f_{PWAR}(\mathbf{x})$  is evaluated following the steps: (i) find the hyperrectangle that the input  $(\mathbf{x})$  belongs to, (ii) retrieve  $f_i = [f_{i1} \dots f_{i0}]$  from a memory, and (iii) evaluate the affine expression  $\sum_j f_{ij} \cdot x_j + f_{i0}$ . For a multiple-output PWAR function, this process is repeated for each output variable using different parameters  $(f_i)$  for each one.

## 1.2 PWA forms for Model Predictive Control (MPC)

A great advantage of PWA functions is their capability to approximate any nonlinear behavior within any specified error. 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 [6]. Some years later, the development of PWA analysis and synthesis techniques was favored by the advent of new computational techniques and environments, such as [58] and [59]. More recently, PWA

functions have arisen naturally as a solution to many engineering problems such as those of MPC.

Model predictive controllers have become popular as a way of approaching non-linear controllers for linear systems with constraints, PWA systems and both non-linear constrained and hybrid systems [7, 9, 60, 61]. Let us consider the problem of regulating to the origin the linear discrete-time system:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \quad (1.20)$$

with the constraints:

$$y_{min} \leq y(t) \leq y_{max}, \quad u_{min} \leq u(t) \leq u_{max}, \quad \text{for } t \geq 0 \quad (1.21)$$

where  $x(t) \in \mathbb{R}^n$ ,  $u(t) \in \mathbb{R}^m$  and  $y(t) \in \mathbb{R}^p$  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:

$$\min_U \left\{ \begin{aligned} J(U, x(t)) &= x_{t+N_y|t}^T \cdot P \cdot x_{t+N_y|t} \\ &+ \sum_{k=0}^{N_y-1} [x_{t+k|t}^T \cdot Q \cdot x_{t+k|t} + u_{t+k}^T \cdot R \cdot u_{t+k}] \end{aligned} \right\} \quad (1.22)$$

where

$$U \triangleq \{u_t, \dots, u_{t+N_u-1}\} \quad (1.23)$$

is subject to

$$\begin{aligned} x_{t|t} &= x(t) \\ y_{min} \leq y_{t+k|t} &\leq y_{max} & k = 1, \dots, N_c \\ u_{min} \leq u_{t+k} &\leq u_{max} & k = 0, \dots, N_c \\ x_{t+k+1|t} &= Ax_{t+k|t} + Bu_{t+k} & k \geq 0 \\ y_{t+k|t} &= Cx_{t+k|t} & k \geq 0 \\ u_{t+k} &= Ku_{t+k|t} & N_u \leq k \leq N_y \end{aligned} \quad (1.24)$$

where the predicted state vector at time  $t + k$ ,  $x_{t+k|t}$  is obtained by applying the input sequence  $u_t, \dots, u_{t+N_u-1}$  to the system model (1.20) 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  $P \succeq 0$ , where  $Q^T$  denotes the transpose of  $Q$  [62].

The solution of the optimization problem in (1.22) to (1.24) is a PWA controller  $u(x) : M \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  as follows [7, 62]:

$$\begin{cases} u^1(x) = \sum_{j=1}^n f_{ij}^1 \cdot x_j + f_{i0}^1, \\ \dots \\ u^m(x) = \sum_{j=1}^n f_{ij}^m \cdot x_j + f_{i0}^m, \end{cases} \quad \mathbf{x} \in \Omega_i \quad (1.25)$$

where  $x \in \mathbb{R}^n$  is the input to the controller function, which is usually the state vector fully measured at time  $t$ ,  $x(t)$ ; and  $f_{ij}^i \in \mathbb{R}^{m \times n+1}$  are gain and offset matrices. The input domain,  $M$ , is partitioned into  $\theta$  non overlapping regions, called polytopes,  $M = \Omega_1 \cup \dots \cup \Omega_D$ . Each polytope,  $\Omega_i$ , is defined as a closed set of points delimited by  $E_i$  edges in the form of  $(n - 1)$ -dimensional hyper-planes,  $\sum_{j=1}^n h_{kj} \cdot x_j + h_{k0} = 0$ , as follows:

$$\Omega_i = \{x \in \mathbb{R}^n \mid \sum_{j=1}^n h_{kj} \cdot x_j + h_{k0} \leq 0, k = 1, \dots, E_i\} \quad (1.26)$$

where  $h_k \in \mathbb{R}^{n+1}$ .

Evaluation of the explicit MPC approach in (1.25) is much simpler than solving the optimization problem in (1.22) to (1.24) on line.

Explicit MPC approaches in (1.25) 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.



### 1.3 PWA forms for virtual sensors

A virtual sensor estimates the value of a variable that is very difficult or costly to measure physically by modeling the relation between that variable and others that can be measured easily with low-cost commercial sensors. The use of virtual sensors has increased continuously since the early 1980s in a wide number of industrial applications, such as building monitoring [63], robotics [64], process control [65, 66], or automotive engineering [67]. In the latter case, for example, virtual sensors are employed to monitor vehicle and driving status as well as road conditions and even communication between vehicles [67–69]. The model that relates input and output variables can be derived from fundamental physical laws using adjustable parameters, from empirical data of the input and output variables without any knowledge of the physical process (usually referred to as black-box models), or a combination of physical and empirical knowledge (a gray-box model). Neural networks and fuzzy logic techniques are used widely to obtain black- and gray-box models [68, 69]. PWA virtual sensors are also employed to provide black-box models [8, 22, 23].

Virtual sensors are usually implemented as software installed in the electronic control units [22, 68, 69]. However, faster responses, smaller sizes and lower power consumption are achieved if virtual sensors are implemented in hardware. Solutions that employ PWAS models implemented in FPGAs were proposed in [8, 23]. PWAS-based virtual sensors are simpler to implement in FPGAs than neural-network-based sensors and provide higher computation speed, as shown in [23]. FPGA implementations of PWA functions based on hyper-rectangular partitions (PWAR-based models) are further simpler than PWAS-based models, as shown in [36]. Taking into account that the sensor market size is growing (in the automotive industry, for example, approximately 22 billion sensors are estimated to be used per year by 2020 [70]), a CMOS Integrated Circuit (IC) solution for virtual sensors is very interesting since it can provide a minimum unitary cost with high performance in terms of area, power and speed.

## 1.4 Trusted controllers and sensors

People will be surrounded in the Internet of Things (IoT) ecosystem by a growing number of smart devices with sensors (for example PWA virtual sensors) and actuators (for example PWA controllers), which capture information about our environments and act upon them autonomously (our cities, homes, cars or bicycles and even our body). As a matter of fact, people already interact more with or through these devices instead of interacting directly. The IoT infrastructure is aimed at improving our quality of life. Hence, security is becoming very important in the IoT environment [71, 72].

Nowadays, virtual sensors and controllers are usually part of ubiquitous and distributed networks such as critical infrastructures. The receiver of the data must trust in the data originated from the device. The way is to confirm that data have not been tampered with, altered or changed. Not only data integrity but also device integrity must be ensured. The reliability of the data should be ensured by authentication of their origin. Message authentication codes (MACs) are the usual constructions employed to ensure data integrity [73]. The MAC construction that has become more popular over the last decade is the HMAC. HMAC, standardized in [74], is an approach to implement MAC using a two-pass hash-based MAC that avoids the weakness of previous constructions. The problem with many existent HMACs is that their primary targets are high transmission rates of data with long key spaces, but do not pay so much attention to area or power consumption. This makes them not suitable for resource constrained scenarios like IoT devices, which have limited computational and memory capabilities. Lightweight cryptographic primitives should be employed. Specific lightweight hash functions based on sponge constructions have been proposed, as is the case of SPONGENT [75], Quark [76], and PHOTON families [77, 78].

There is also a growing interest in the use of lightweight cryptographic protocols for IoT devices so as to ensure that data transmitted are confidential, which is required by applications using sensitive information [79, 80]. In these cases, not only message authentication but also message encryption is required. CAESAR competition is evaluating different authenticated encryption schemes that are fast in software and efficient in hardware [81]. The idea is to combine encryption and authentication into a single algorithm, called Authenticated Encryption, so as to obtain

smaller area and power consumption compared to two separate algorithms. Among the CAESAR candidates, AEGIS [82] and ASCON [83, 84] have been selected for the trusted PWA controllers and virtual sensors analyzed herein. AEGIS is based on the AES encryption round function while ASCON is based on sponge constructions like PHOTON.

The usual way to ensure the integrity of a device is to check that it knows a cryptographic key. However if the key is stored in the device memory, it can be attacked [85] and a counterfeit device can be fabricated. The problem of counterfeit chips is so significant that the Semiconductor Industry Association (SIA) maintains an anti-counterfeiting task force. A solution is the use of Physical Unclonable Functions (PUFs) inside the device hardware, which ensures that the device has a unique inherent identity given, in general, by the variability of its manufacturing process [86]. Among the wide number of PUFs proposed in the literature, PUFs based on Static Random Access Memories (SRAMs) are a good alternative since both CMOS PWA controllers and virtual sensors require SRAMs for its functionality. The start-up values of the SRAM cells are employed to generate the random seeds required by the cryptographic primitives as well as to recover the cryptographic key [87]. The key is not stored anywhere but recovered whenever needed by using a Helper Data Algorithm that employs non-sensitive data (known as Helper Data) and the response of the PUF (herein the start-up values of SRAM cells are considered) [86]. An impostor device is not able to provide a similar PUF response and, hence, is unable to recover the key while the genuine device is able to prove its authenticity against aging and variations of the operation conditions, namely, temperature and power supply voltage.

## 1.5 Conclusions

Microelectronic designs to implement PWA functions in different forms are required in many application domains, particularly for model predictive controllers and for virtual sensors. The inclusion of cryptographic modules is crucial in hardware PWA realizations that are part of the IoT infrastructure.

In the following, Chapter 2 presents several architectures for different PWA forms and compares them. In Chapter 3, a hardware design flow for ASICs and for FPGAs is presented. Chapter 4 summarizes how the design

flow is integrated into a CAD environment to design model predictive controllers and presents several application examples. In Chapter 5, an algorithm to design virtual sensors using the PWAR form is described. The algorithm is applied using the XFuzzy environment and an application example is detailed. In Chapter 6, the hardware design of cryptographic modules is exposed, and a trusted virtual sensor using AEGIS is described in Chapter 7. Finally, the global conclusions of this Ph.D. Dissertation are given in Chapter 8.

## Chapter 2

# Hardware architectures to implement PWA functions

In this Chapter several architectures for different PWA forms are presented and compared.

### 2.1 Architecture for PWAG form

The architecture proposed in this section is explained in detail in [CV5] and in the patent [CV16]. The method to explore the BST is detailed in the patent of addition [CV15].

### 2.2 Architecture for PWAL form

A preliminary version of this digital architecture for implementing PWAL functions with two inputs and one output was presented in [CV13]. That architecture is addressed to FPGAs from Xilinx and uses the Xilinx DSP blockset for Simulink in System Generator. The final version of the architecture exposed in this section is described in detail in [CV4].

### 2.3 Architecture for multiple PWA forms

The architecture is described in [CV3].

## 2.4 Comparison between architectures

Let us compare several ways of implementing multivariate PWA functions reported in literature with the architectures proposed in this Dissertation.

TABLE 2.1: Comparison between architectures.

	Latency ( $N_{cc}$ )	Bits to store	Multi- pliers	Generic PWA
PWAG_s [34] in FPGA	$n + 2d + nd + 2$	$n_{bit}(n + 1)(\Theta + E)$	1	Yes
MultiTree [14] in FPGA	$h_M + 2$	–	$nM$	Yes
PWAG-hash [35] in FPGA	$12 + \Theta + 6 \cdot m_o + \max(\sum_{k \in \mathcal{J}} E_k)$	$n_{bit}(n + 1)(\Theta + E) + T \log_2 E + 1$	$n$	Yes
PWAH [38] in FPGA	$n$	$n_{bit} \sum_{i=1}^n 3m_i - 1$	1	No
PWAR_s [36] in FPGA	$n + 2$	$n_{bit}(n + 1) \prod_{i=1}^n m_i$	1	No
PWAR_p [36] in FPGA	2	$n_{bit}(n + 1) \prod_{i=1}^n m_i$	$n$	No
PWAS_s [37] in FPGA	$n + 4$	$n_{bit}(n + 1) \prod_{i=1}^n (m_i + 1)$	1	No
PWAS_p [37] in FPGA	3	$n_{bit}(n + 1) \prod_{i=1}^n (m_i + 1)$	$n + 1$	No
Proposed PWAG	$n + 2d$	$n_{bit}(n + 1)(\Theta + E) + N \log_2 T$	$n$	Yes
Proposed PWAL	$n + U + 1$	$n_{bit}(n + 1)V + U(\log_2 X + 1)$	$2n$	Yes

Table 2.1 summarizes their features in terms of latency (number of clock cycles to provide the output), memory (bits to store), computing resources (multipliers), and capability of implementing or not PWA functions with generic partitions of the input domain. The architecture labeled PWAG\_s is an architecture that calculates a PWA function based on a BST constructed off-line, with an arithmetic unit implemented as a multiplier-accumulator reported in [34]. The architecture based on multiway trees reported in

[14] is labeled MultiTree. The architecture labeled PWAG-hash is the proposal in [35]. PWAH is the architecture reported in [38] and consists of single-input single-output PWA modules connected in cascade. The architecture based on the hyper-rectangular partition proposed in [36] is labeled PWAR\_s for the serial version and PWAR\_p for the parallel version. The mostly serial and mostly parallel architectures based on simplicial partition, as presented in [37], are labeled PWAS\_s and PWAS\_p respectively.

The symbols employed in the comparison are: the input dimension of the PWA function ( $n$ ), the depth of the BST ( $d$ ), the number of nodes in the BST ( $N$ ), the number of hyper-rectangular partitions of the domain in the PWAG-hash implementation ( $m_0$ ), the subset of polytopes that are directly searched ( $\mathcal{J}$ ),  $E_k$  are the edges of the polytopes in the subset  $k$  in PWAG-hash form, the number of different local affine functions ( $V$ ), 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 ( $\tilde{\psi}$ ).

The proposed PWAG architecture offers better time responses since the computation of the edges and affine function is carried out in parallel. In addition, it allows on-line programmability of the BST.

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. Compared to the proposed PWAG architecture, the proposed PWAL architecture offers the advantage of requiring less memory while maintaining competitive performance in terms of latency if the value of  $U$  is small. However, if the PWAL form does not simplify enough and the depth  $d$  of the BST in the PWAG form is small, then PWAG architecture offers better performance. Therefore, depending on the PWA function, PWAG or PWAL forms offer better performance.

Compared to PWAS, PWAR and PWAH architectures, the proposed PWAG and PWAL architectures are more versatile since they can generate any continuous PWA function defined over a generic partition without approximations. The continuity property is not necessary in the PWAG form.

## 2.5 Conclusions

Several architectures have been proposed for the hardware realization of PWA functions: an architecture for the PWAG form, an architecture for the PWAL form, and an architecture for multiple PWA (PWAL, PWAS, and PWAR) forms. The architecture for PWAG is the only generic form available in the literature that allows on-line programmability. The PWAL form provides smaller time responses than the PWAG form for many continuous PWA functions. The architecture for multiple PWA forms allows selecting the best form among PWAL, PWAS, and PWAR depending on the application, without increasing significantly the cost in resources and power consumption.



## Chapter 3

# Hardware design flow to implement PWA functions

A hardware design flow for ASICs and for FPGAs is presented in the following. It has been applied to implement the PWAL form in ASIC and FPGAs, and to implement, fabricate and verify the PWAG form and the PWAR, PWAS, and PWAL forms into two different ASICs.

### 3.1 Hardware design flow for ASICs

This implementation is described in [CV5].

#### 3.1.0.1 PWAL implementation

This implementation is described in [CV4].

#### 3.1.0.2 PWAX implementation

This implementation is described in [CV3].

#### 3.1.1 Manufactured ASICs

Two ASICs were fabricated. A first ASIC that implements the PWAG form explained in the section 2.1, so called PWAG ASIC [CV5]. , and a

second ASIC, called PWAX ASIC, that implements multiple PWA forms (PWAS, PWAR, PWAL) [CV3], thus, covering all the PWA forms addressed in this Dissertation.

## **3.2 Hardware design flow for FPGAs**

The PWAL form was described in Xilinx ISE Generator and implemented in a Xilinx Spartan 3 FPGA (xc3s200-5ftp256) [CV13].

## **3.3 Hardware verification**

The ASIC-in-the-loop design flow was presented in [CV20].

In the case of the FPGA implementation, the design flow was presented in [CV12].

## **3.4 Conclusions**

A hardware design flow for ASIC realizations of PWA functions has been presented. It was used to implement the three architectures presented in Chapter 2. Two ASICs were fabricated in a CMOS nanometer technology and tested following the proposed design flow. The open and closed-loop verification developed was applied for both ASICs when they were configured as PWA controllers.

A hardware design flow for FPGAs has been also presented. This flow includes a HIL verification and a closed-loop verification flow for PWA controllers.

## Chapter 4

# PWA solutions in hardware for Model Predictive Control

This Chapter summarizes how the design flow of the hardware solutions presented in this research work has been integrated into a complete automatic CAD environment to design model predictive controllers. This is illustrated with several application examples.

### 4.1 The Moby-dic Toolbox

There are several tools that design controllers as well as several methods that embed the resultant controller into hardware, as shown in Fig. 4.1(a). However, to the best of our knowledge, only Moby-dic Toolbox [88] is able to provide automated and integrated design flows from the mathematical model of the process to the synthesis of the electronic circuits, as shown in Fig. 4.1(b).

The Moby-dic Toolbox for Matlab provides an automatic tool chain for the circuit design of embedded control systems, as illustrated in Fig. 4.2. Given a linear discrete-time model of the physical process as well as the constraints on states and input variables, the optimal MPC controller is generated by the Multiparametric Toolbox [89] or the Hybrid Toolbox [90] included in the Moby-dic Toolbox.

From the optimal MPC controller, the PWAG form is directly extracted taking into account the specifications of the digital realization. In the case of the PWAR and PWAS forms, the approximation of each case is

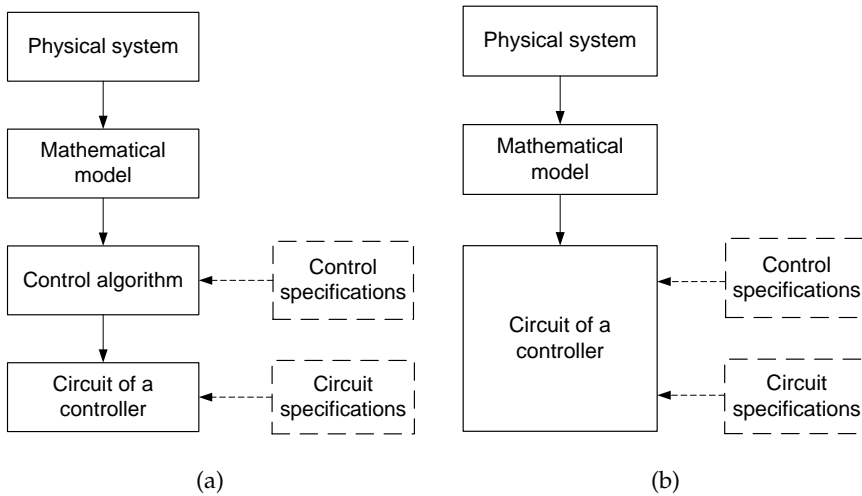


FIGURE 4.1: Design flows for circuits implementing controllers: (a) Traditional (b) Moby-dic proposal.

performed. Circuit implementations (with automatic generation of VHDL code) in Xilinx FPGAs are provided for PWAG, PWAR and PWAS forms.

Since PWAR and PWAS forms approximate the optimal control law, the stability of the resultant controller is affected. Hence, a posteriori stability analysis and evaluation of the domain of attraction [92] should be carried out. To aid in this issue, the Toolbox offers methods for the simulation of the closed-loop system, using Xilinx System Generator for the controller and a Simulink block for the plant (system to be controlled).

## 4.2 Methodology to configure and program PWA ASICs with the Moby-dic Toolbox

A new functionality was included in the Moby-dic Toolbox in order to configure and program the designed ASICs [CV19].

This new functionality added to the Moby-dic Toolbox is shown in gray in Fig. 4.2.

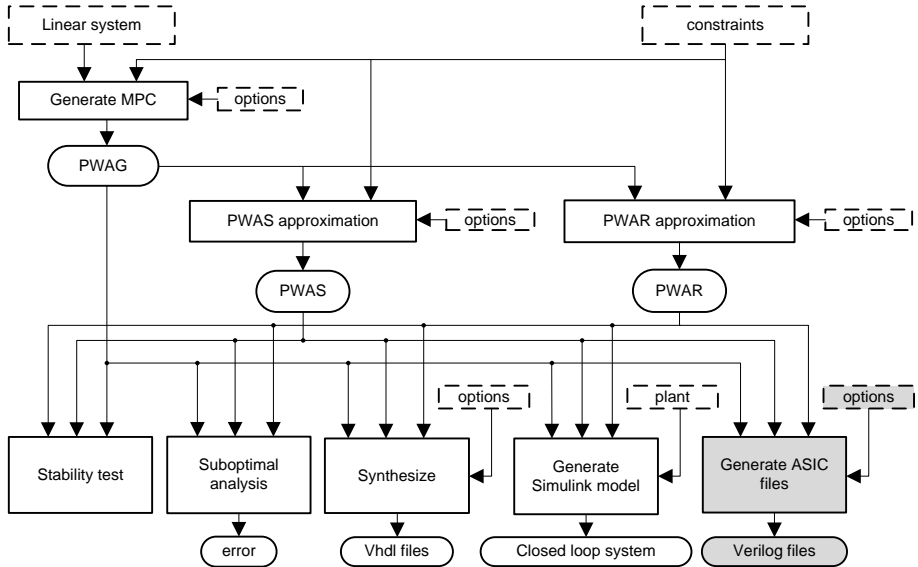


FIGURE 4.2: Scheme of the Moby-dic Toolbox. The functionalities added with our research are depicted in gray.

### 4.3 Methodology to extract PWAL form from Moby-dic Toolbox

In addition, as Moby-dic Toolbox extracts PWAG, PWAS and PWAR forms, a design flow for the PWAL form was developed [CV12]. This methodology also extracts the Verilog files to configure and program the PWAL form in the PWAX ASIC, as commented in the previous section.

### 4.4 Application examples

Several application examples of PWA functions for MPC were studied with the hardware solutions designed. Double integrator [CV5], [CV15], [CV13], [CV4], [CV3], [CV20], [CV12] and MIMO [CV3] systems are two typical testbench examples in the control domain. The adaptive cruise control [CV4] and the buck-boost DC-DC converter [CV4] are two application examples from the industrial field.

## 4.5 Conclusions

The hardware realizations of PWA functions presented in the previous chapters have been applied for Model Predictive Control (MPC) were exposed. A new functionality to program and configure PWA controllers in ASICs was added to a Matlab Toolbox called Moby-dic so that these ASICs can be easily applied for MPC problem described at mathematical level. A methodology to extract the PWAL form from Moby-dic Toolbox was described. Finally, several application examples were presented to illustrate the hardware performance of the implementation of PWA controllers.

The latency achieved by the proposed ASIC solutions surpasses the requirements of many embedded control systems and opens the way to control more challenging problems which require not only high-speed but also low-power and small-size controllers.

## Chapter 5

# PWA solutions in hardware for virtual sensors

As commented in previous chapters, PWAG and PWAL forms are able to implement generic PWA functions and, hence, they are able to implement optimal model predictive controllers that are expressed in explicit form. However, for applications like virtual sensing, where explicit PWA forms are not available, but the information provided by numerical input and output data, PWAS and PWAR forms are easier to design, particularly PWAR forms, which are simpler than PWAS forms.

Virtual sensors estimate the value of variables which are difficult or costly to measure physically by modeling the relation between them and others that can be measured easily. This chapter describes how to implement in hardware a virtual sensor by using a black-box identification algorithm which assumes that the virtually measured output,  $y$ , is set as a PWAR function of the input variables,  $x = \{x_1, \dots, x_n\}$ . Xfuzzy environment was used to carry out the proposed algorithm. An application example in the automotive domain illustrates the use of the PWAX ASIC acting as PWAR virtual sensor.

### 5.1 Identification algorithm for PWAR virtual sensors

A PWAR function can be adjusted conveniently to approximate the relation between input variables and sensing variable to estimate. An algorithm has been developed to find the partition and affine functions of the PWAR form that minimize the mean square error (MSE) between the

PWAR function output (variable to sense,  $y$ ) and the empirical or simulated output data,  $y_\mu$ . This algorithm has been presented in [CV2]. The algorithm takes into account the constraints imposed by the architecture for multiple PWA forms presented in Chapter 2 and integrated in the PWAX ASIC presented in Chapter 3, which are the following:

- Maximum number of input variables:  $n$ .
- Maximum number of hyper-rectangles:  $P = 2^p$ .
- Maximum number of intervals per input:  $Q = 2^q$ , being  $L_k = 2^{p_k}$  the number of intervals per input, so that,  $q \geq p_k \forall k$ .

The algorithm determines the most adequate hyper-rectangular partition of the input domain,  $L = \{p_1, \dots, p_n\}$ , and the value of the affine functions,  $F = \{f_1, \dots, f_P\}$ .

First, the partition is established, and then an off-line optimization algorithm extracts the  $F$  parameters using a set of simulated or empirical values of the output and its corresponding input data. All input data should belong to the domain  $D$ . At least, a given percentage,  $C$ , of the hyper-rectangles in a partition should be covered by the data in order to proceed with optimization. Otherwise, such high resolution is not required and the partition is not considered. The result of the approximation algorithm for a given partition,  $L$ , is not only the value of the  $F$  parameters but also the MSE between the simulated or empirical output data and the output of the PWAR function (the MSE taking into account training and testing data). Then, the MSE of the partitions that can be implemented in the ASIC are compared, and the partition with the lowest MSE (with the MSE that allows the best trade-off taking into account training and testing data) is selected.

The algorithm steps are detailed in the pseudo-code of Algorithm 1. Its inputs are a set,  $B$ , of output data,  $y_\mu$ , with  $\mu = 1, \dots, M$ , corresponding to the input data  $\{x_{1\mu}, \dots, x_{n\mu}\}$ ; the value of  $p$  that fixes the maximum number of hyper-rectangles; the value  $q$  that fixes the maximum number of intervals per input; and the minimum percentage,  $C$ , of hyper-rectangles that should be covered by input data. Data set  $B$  is divided into two subsets, one used as training file and the other as test (validation) file.

The partitions with high resolution, that is, which verify that  $p = \sum_{k=1}^m p_k$ , are firstly explored with the function *TryPartition* (line 1.17) because they have higher capability of providing a lower approximation



**Algorithm 1** Pseudo-code of PWAR virtual sensor algorithm.

---

**Require:**  $B = [x_{11}, \dots, x_{m1}, y_1, \dots, x_{1M}, \dots, x_{mM}, y_M], p, q, C$   
 $p_1 = q; p_2 = \dots = p_m = 0;$   
 $part = \neg SUCCESS; Pool = []; Emax = 1;$   
**while**  $p_1 \geq 0$  **do**  
 $p_2 = p - \sum_{j=1}^m \& j \neq 2 p_j;$   
5: **if**  $p_2 > q$  **then**  
 $p_2 = q;$   
**end if**  
**while**  $p_2 \geq 0$  **do**  
 $\dots$   
10: **while**  $p_{m-1} \geq 0$  **do**  
 $p_m = p - \sum_{j=1}^m \& j \neq m p_j;$   
**if**  $p_m > q$  **then**  
 $p_m = q;$   
**end if**  
15: **while**  $p_m \geq 0 \& part = \neg SUCCESS$  **do**  
 $A = \{p_1, \dots, p_m\};$   
 $part = \text{TryPartition}(A, B, C);$   
**if**  $part == SUCCESS$  **then**  
 $Pool = [Pool \parallel A];$   
20: **end if**  
 $p_m = p_m - 1;$   
**end while**  
 $p_m = 0; p_{m-1} = p_{m-1} - 1;$   
**end while**  
25:  $\dots$   
 $p_3 = 0; p_2 = p_2 - 1;$   
**end while**  
 $p_2 = 0; p_1 = p_1 - 1;$   
**end while**  
30: **for**  $i \in Pool$  **do**  
 $[RMSE, F] = \text{Optimization}(i, B);$   
**if**  $RMSE < Emax$  **then**  
 $Emax = RMSE;$   
 $L = i;$   
35: **end if**  
**end for**  
**return**  $L, F$

---

error (and they can be implemented in the ASIC as well as partitions with lower resolution). The function  $TryPartition(A, B, C)$  evaluates if the partition,  $A$ , meets or not the condition to be included in the partition list,  $Pool$ . The condition is that the data set,  $B$ , should belong to no less than  $C$  in percentage of the hyper-rectangles of the partition,  $A$ . Otherwise, such high resolution model has too many undefined affine functions. Finally, the partition list is explored (with the function  $Optimization$ ) to determine the partition that provides the lowest approximation error. The function  $Optimization(i,B)$  (line 1.31) applies Levenberg-Marquardt algorithm to find the parameters  $F$  that achieve the minimum MSE (and, hence, the Root Mean Square Error, RMSE) for each partition candidate,  $i$ , considering the data set  $B$  [91]. Since the partition candidate (hyper-rectangle) is fixed, this optimization finds the best linear (affine) regression for the data. The  $F$  parameters of hyper-rectangles not covered by data are fixed as the arithmetic mean of the  $F$  in the neighbourhood. Once the algorithm finishes, if all the coefficients,  $f_{ij}$ , associated with the input  $x_j$  ( $i = 1, \dots, P$ ), are zero, that input can be removed, thus resulting  $x = \{x_1, \dots, x_m\}$  with  $m \leq n$ .

The proposed sensor can be configured to provide several PWAR functions depending on the application. Therefore, it can be used to sense different variables. The configuration data are the number of input variables, and the number of intervals in which each dimension is divided that is, the hyper-rectangular partition is configurable. The sensor is also configurable in the affine functions of each hyper-rectangle by changing the value of the parameters associated with each hyper-rectangle. The input variables can be a same variable measured at several sampling times as well as the output measured at previous instants, for example,  $x = \{\alpha[k] \ \alpha[k-1] \ \beta[k] \ y[k-1]\}$ .

## 5.2 Methodology using Xfuzzy environment

Moby-dic Toolbox is able to provide virtual sensors in the PWAR form, however as far as we know no other function extracts virtual sensors in the PWAR form. One of the results of this Dissertation has been to develop a methodology using Xfuzzy environment to extract the PWAR form from an input/output dataset. This section is pending of be published.

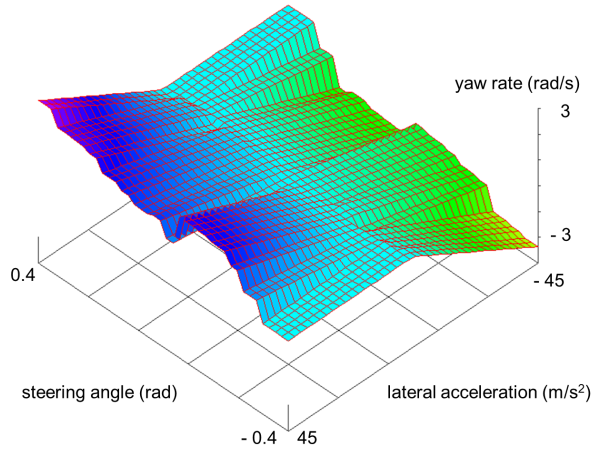


FIGURE 5.1: Vehicle yaw rate estimation by a PWAR virtual sensor.

### 5.3 Application example

As illustrative example, a problem from the automotive domain, the estimation of vehicle yaw rate, was selected to be solved. In literature, this problem was firstly addressed with a silicon micromachining (MEMS)-based sensor in [92]. Since MEMS-based sensors have inaccuracy problems including severe DC-offset, an additional module based on fuzzy logic was proposed in [93]. Lately, neural-network-based direct virtual sensors implemented in software were proposed in [68].

A PWAR-based model was found by using Algorithm 1 taking into account the features of the PWAR form in the PWAX ASIC. The set of input-output data needed to apply the algorithm (training and test subsets) was obtained from simulations of the vehicle modelled by the differential equations described in [68], discretized with a zero-order-hold approach with a sampling time of 10 ms, and including noise in the simulated outputs (lateral acceleration and yaw rate) as in [68]. The input variables are the longitudinal vehicle speed,  $v_x(t)$ , the steering angle,  $\alpha_s(t)$ , and the lateral acceleration,  $a_y(t)$ . After applying PWAR virtual sensor algorithm, Algorithm 1, (with  $C = 80\%$ ), the best partition found divided the longitudinal speed, the steering angle, and the lateral acceleration into 4, 16 and 2 intervals, respectively. Figure 5.1 shows the yaw rate provided by the

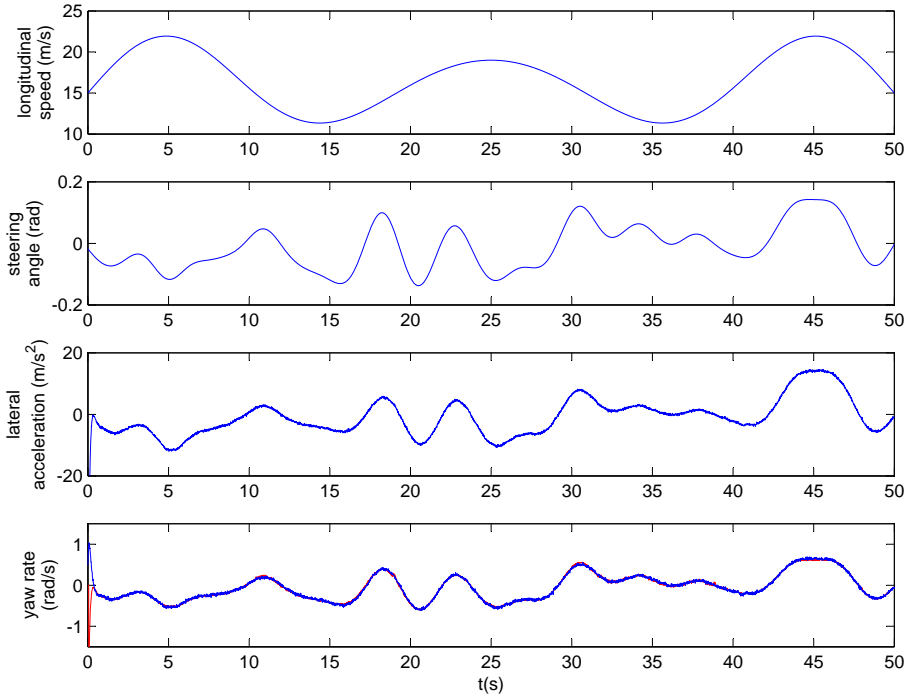


FIGURE 5.2: HIL simulation of the PWAR virtual sensor.

PWAR-based model for a fixed value of the longitudinal speed, obtained with the tool `xfplot` of `Xfuzzy` environment.

Therefore, the sensor estimates the yaw rate as  $\dot{\Psi} = f_{i0} + f_{i1} \cdot v_x + f_{i2} \cdot \alpha_s + f_{i3} \cdot a_y$  if  $(v_x, \alpha_s, a_y) \in P_i$ , with  $i = 1, \dots, 128$ . The RMSE values of the PWAR-based sensor and the direct virtual sensor based on neural networks, DVS1 in [68], using simulated data, are shown in Table 5.1. The RMSE is lower for the designed PWAR virtual sensor.

TABLE 5.1: Error comparison between virtual sensors.

	Direct Virtual Sensor in [68]		PWAR Virtual Sensor	
	Training Data	Testing Data	Training Data	Testing Data
RMSE	3.9%	6.5%	0.25%	1.08%

A HIL simulation was carried out in ModelSim. The evolution of the longitudinal vehicle speed, the steering angle, and the lateral acceleration

scaled to their real values, are shown at the upper part of Figure 5.2. The yaw rate estimated by the sensor (also scaled to the real value) is shown in blue at the bottom of Figure 5.2. The yaw rate obtained by the model is shown in red. Noise can be observed in the lateral acceleration and the yaw rate of the model that makes it more realistic. To achieve a response time lower than the 10 ms sampling time, the sensor should work at more than 2.1 KHz, which is quite affordable for the hardware designs presented in this Dissertation.

## 5.4 Conclusions

The use of PWAR functions for virtual sensors has been described. An identification algorithm for virtual sensors based on the PWAR form has been proposed. This methodology uses the Xfuzzy environment. The methodology has been validated with the configuration and programming of the PWAX ASIC presented in Chapter 3 to perform as a PWAR virtual sensor in an application example of the automotive domain where small and low-power sensors are required.



## Chapter 6

# Cryptographic modules for PWA solutions in hardware

The inclusion of cryptographic primitives in a controller or a virtual sensor allows ensuring the integrity, authenticity, and confidentiality of the control law or the virtual measurement. Several options have been evaluated to address this goal.

In order to detect possible manipulations of the control law or the virtual measurement, a first approach includes a module to solve the problem of data integrity and authenticity based on Message Authentication Codes (MACs). The second approach to increase the level of security is the use of authenticated ciphers to ensure not only the integrity of the provided control law or the virtual measurement but also to encrypt it at the same time. Two authenticated encryption algorithms, which are third-round candidates of the CAESAR competition, have been considered. Their features are briefly summarized in the following, the content of the following subsections are described in [CV1].

## **6.1 Description of candidate modules**

### **6.1.1 Hash-based Message Authentication Code (HMAC) based on the PHOTON hash function**

### **6.1.2 AEGIS Authenticated Cipher**

### **6.1.3 ASCON Authenticated Cipher**

## **6.2 Comparative analysis of the cryptographic modules**

The three alternatives implemented in a 90-nm CMOS technology are compared and described in [CV1].

## **6.3 Conclusions**

Three alternatives of cryptographic modules has been analyzed. Two of them implement authenticated ciphers (AEGIS and ASCON), that are very suitable to ensure authenticity and confidentiality. The third option based on the standard HMAC uses the lightweight hash function PHOTON 20/80/16, which ensures authenticity but not confidentiality. Considering a 90-nm CMOS technology, ASCON-128 is the smallest module and AEGIS-128 allows the fastest response.



## Chapter 7

# ASIC design of a trusted PWA virtual sensor

In this Chapter an architecture of a trusted virtual sensor using AEGIS is presented [CV2]. The implementations results of the trusted virtual sensor in a 90-nm technology are exposed.

### 7.1 Functional and architectural description

A virtual sensor provides one output variable that is virtually measured. The inclusion of cryptographic primitives allows ensuring the integrity, authenticity, and confidentiality of the virtual measurement achieving a trusted virtual sensor. The AEGIS algorithm was implemented in a virtual sensor integrated circuit. The output provided by the sensor is the encrypted value ( $C$ ) that is attached to an authentication code that ensures the integrity, and authenticity of the sensor data:

$$\text{sensor}_{\text{out\_AEGIS}} = \text{nonce} \parallel C \parallel \text{tag} \quad (7.1)$$

The proposed sensor estimates the virtual measurement ( $y$ ) from a PWA-based model. Both PWAR and PWAS forms are able to approximate any function and extract any black-box model. The PWAS form has been widely explored for virtual sensors [8, 23]. However, the PWAR form is selected herein since its implementation is simpler than PWAS implementation as was commented in Chapter 5.

The integrity of the virtual sensor itself is ensured if the *key* employed by the cryptographic module is not stored but recovered whenever needed by using PUFs. The trusted sensor is able to recover the cryptographic key shared with the receiver of the sensing data, while any impostor is unable due to the uniqueness provided by the start-up values of the SRAM in the sensor, which is exploited as a PUF. Non-sensitive Helper Data, *H*, are stored to recover the key with a Helper Data Algorithm (HDA) based on an Error Correcting Code (ECC) [86]. Helper Data do not reveal anything about the cryptographic *key* because the start-up values of SRAM cells obfuscate it. Similarly, Helper Data do not reveal anything about the intrinsic nature of the sensor because the cryptographic key obfuscates it. A repetition ECC was selected in the design of the sensor.

The sensor designed has two main behavioral modes. The configuration mode is carried out whenever the sensor is powered up. It recovers the *key*, generates a seed for a *nonce*, and establishes the PWAR relation between the input data and the data to estimate. The trusted sensing mode is set once the configuration mode is finished. It generates a PWAR virtual measurement, authenticates it, and encrypts it.

The building blocks of the CMOS sensor designed are exposed in the following. It is composed of three main units: the PWAR, the cryptographic, and the control units. Figure 7.1 illustrates the block diagram of the sensor architecture, including the main signals and buses that interconnect the blocks. Note that the SRAM is shared by two units, thus saving resources since it is not used simultaneously by both units.

### 7.1.1 PWAR Unit

The architecture of the PWAR Unit corresponds to the architecture exposes previously in Section 2.3 for multiple inputs and one output. The partition and affine functions of the PWAR function are adjusted conveniently to provide a black-box model that approximates the relation between input variables and sensing variable to estimate as described in Chapter 5. Therefore, the virtually measured output, *y*, is a PWA function of the inputs  $x = \{x_1, \dots, x_n\}$ .

The sensor can be configured for different PWAR functions. Therefore, it can be used to sense different variables. The configuration data are the number of inputs and the number of intervals in which each input

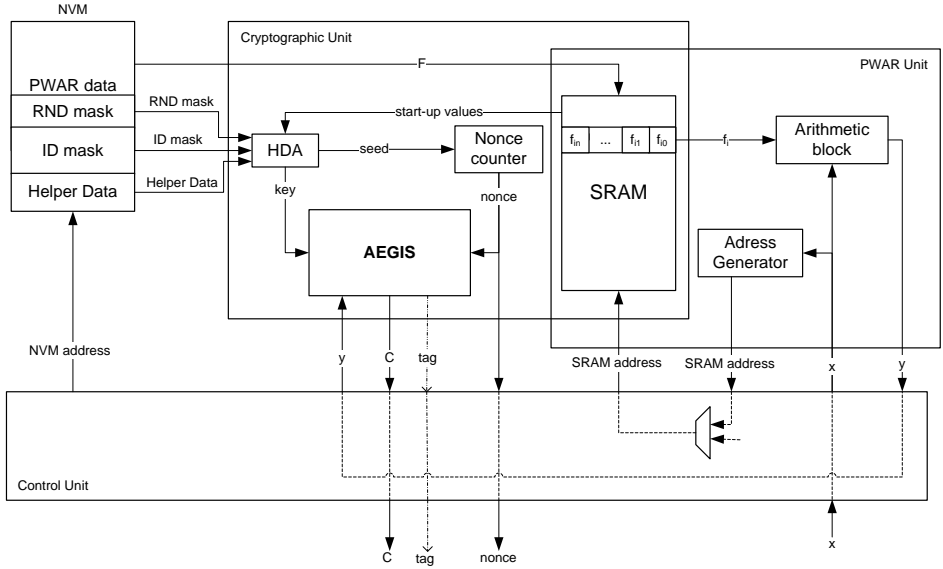


FIGURE 7.1: Architectural scheme of the proposed CMOS sensor.

is divided, that is, the hyper-rectangular partition is configurable. The sensor is also configurable in the affine functions by changing the value of the parameters  $f_i = \{f_{i0}, f_{i1}, \dots, f_{in}\}$  associated to each hyper-rectangle,  $P_i$ . Prior to use the sensor, the sensor manufacturers or their authorized distribution channels program in a non-volatile memory (NVM) these configuration data and parameters.

In the configuration mode, the PWAR parameters are read from the NVM and stored in the SRAM. Each word of the SRAM contains a parameter,  $f_i$ , associated to an hyper-rectangle,  $P_i$ , of the input domain. The SRAM depth is the number of hyper-rectangles. In the trusted sensing mode, the PWAR unit receives the input data. The address generator block determines the address of the SRAM where the parameters are stored. For that purpose, it concatenates the  $p_k$  most significant bits (MSBs) of each input  $x_k$ . The MSBs determine which of the  $2^{p_k}$  intervals the input belongs to, and, hence, determine the hyper-rectangle. Then, the arithmetic block computes in parallel the operation in (1.4) for the input,  $x$ , and the parameters,  $f_i$ .

### 7.1.2 Cryptographic Unit

The SRAM employed in the PWAR Unit is also employed in the Cryptographic Unit as a PUF. As discussed in [87], there are SRAM cells whose intrinsic conditions dominate over the external conditions. Hence, although the external condition change (such as temperature or power supply voltage), their start-up values are mostly the same. They will be named as ID cells. There are also SRAM cells whose external conditions dominate over the intrinsic conditions so that they are able to extract the noise of the external conditions as a source of entropy. They will be named as RND cells.

The NVM stores which cells of the SRAM are ID cells (an 'ID\_mask' is in charge of such information) and which ones are RND\_cells (an 'RND\_mask' stores such information). These masks have to be stored in the NVM by the sensor manufacturers or their authorized distribution channels prior to use the sensor. These manufacturers or distributors follow a simple procedure to classify the SRAM cells: the SRAM is powered-up and down several times under different operating conditions. Each time, the start-up values are compared with the values at the previous time. If the start-up value never changes, the cell is classified as ID (it has not shown bit flipping). If the start-up value changes, the cell is classified as RND. The ID cells are used to recover the *key* while the RND cells are used to generate a seed for the nonces.

A Helper Data Algorithm (HDA) is used in order to recover the *key*. The *key* is not stored in the NVM, and therefore, the attacks to obtain the *key* are forced to be done with the sensor powered. Instead of the *key*, Helper Data, *H*, are stored in the NVM prior to use the sensor. Helper Data do not reveal information about the cryptographic *key*, so that the NVM does not require any special security feature. The way to generate the Helper Data is as follows. Each bit of the cryptographic key  $key = [k_1, \dots, k_a]$  is repeated  $r$  times, so that  $key_r = [k_{11}, \dots, k_{1r}, \dots, k_{a1}, \dots, k_{ar}]$ . A response,  $R$ , is obtained by concatenating  $a \cdot r$  values resulting from the start-up values of  $a \cdot r$  SRAM ID cells. Helper Data are obtained by XORing  $key_r$  and  $R$ ,  $H = key_r \oplus R$ .

In the configuration mode, the *key* is recovered using the ID\_mask and the Helper Data, *H*, as follows. The start-up values are extracted. An ID is generated with the start-up values of those cells that the ID\_mask indicates as ID cells. A response,  $R'$  (slightly different to  $R$ , since even

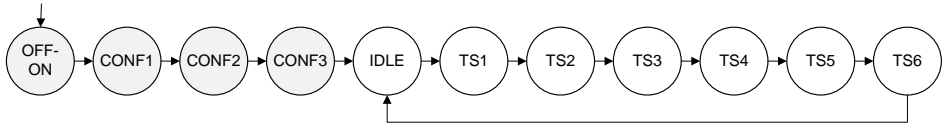


FIGURE 7.2: State diagram of the Control Unit.

ID cells may provide some bit flipping) is obtained by taking  $a \cdot r$  values of ID cells. Helper Data are XORed with  $R'$ :  $key' = H \oplus R' = key_r \oplus R \oplus R'$ .  $key$  is recovered by using the decoder of the error correcting code:  $key = ECC(key')$ .

Besides, the seed of the nonces is generated with the start-up values of those cells indicated by the RND\_mask as RND cells. Although the ID\_mask, RND\_mask and the Helper Data,  $H$  are stored in the NVM, this fact does not compromise security since the key cannot be recovered without the SRAM start-up values. During the configuration mode, they are loaded from the NVM into the lower part of the SRAM. The upper part of the SRAM is used in the Cryptographic Unit to extract the start-up values.

The nonce counter is initialized with the seed during the configuration mode. During the trusted sensing mode, it increases the count to generate a new nonce.

The core of the Cryptographic Unit is the AEGIS module described in the previous Chapter.

### 7.1.3 Control Unit

The Control Unit copes with the two possible behavioural modes. It is implemented as a finite state machine (FSM). Figure 7.2 illustrates the states and the main operations. When the FSM is set in one state, only the required blocks are enabled, thus saving power consumption. Two parts are differentiated, the left part corresponding to the configuration mode (gray states) and the right part corresponding to the trusted sensing mode (white states). Whenever the sensor is powered up, the configuration mode starts.

The main operations in the configuration mode are as follows. The mask\_ID, mask\_RND, and Helper Data are read from the NVM (CONF1).

The ID and the seed are generated (CONF2). The *key* is recovered, the nonce is initialized with the seed, configuration data of the PWAR are read from the NVM, and the PWAR Unit is configured (CONF3). Additionally, during the configuration mode, the AEGIS module reads the *key*.

Once the configuration mode is finished, the FSM is set to state IDLE, where it remains until the sensor receives an input data to generate a new trusted PWAR virtual measurement.

In the trusted sensing mode, the main operations are the following. Input data ( $x$ ) are read and a new nonce is generated (TS1). The Address Generator block generates the address and the AEGIS block processes the nonce (TS2). The SRAM provides the parameters stored in the address previously generated (TS3). The PWAR Unit generates the virtual measurement (TS4). The AEGIS block processes the virtual measurement and provides the encrypted virtual measurement,  $C$  (TS5), and finally the AEGIS block generates the authentication *tag* (TS6). Once the trusted virtual sensor output is provided, the FSM goes to the state IDLE and waits for a new input data.

If the sensor is powered down and then it is powered up, the FSM starts from the state OFF-ON and all the configuration states are carried out again. Consequently, a correct performance is ensured after (expected or unexpected) suspension of power supply since the sensor is always configured before entering the trusted sensing mode.

## 7.2 Features of the trusted virtual sensor

The main features of the proposed trusted virtual sensor are the following:

- PWAR unit. The maximum number of input variables,  $n$ , is 4. The bits of the input variables  $\{x_1, \dots, x_4\}$  and the parameters that define affine functions are 12. The bits of the output variable,  $y$ , are 26. The maximum number of hyper-rectangles,  $P = 2^p$ , is 4,096 ( $p = 12$ ), and the maximum number of intervals per input,  $L_k = 2^{p_k}$ , is 128 ( $p_k = 7$ ). Hence, the SRAM has  $4,096 \times 60$  bits.
- Cryptographic Unit. Second row of the Table ?? shows the size of the *key* and the *nonce* used by AEGIS module. A binary repetition code

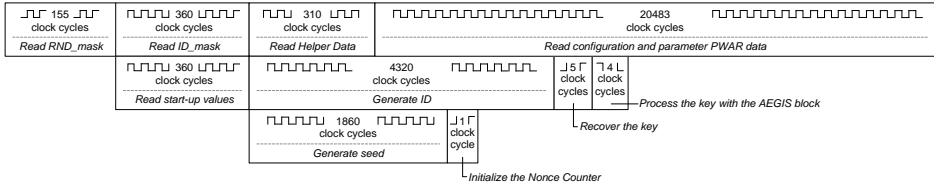


FIGURE 7.3: Timing of configuration mode.

with codeword length  $r = 29$  is employed to recover the *key* in the HDA block.

- NVM. This memory has an output bus with 12 bits. The size of the Helper Data depends on the key size and the length of the error correcting code. A length of 29 was proven enough to cope with the maximum bit flipping measured experimentally in the SRAM employed. The sizes of the masks were proven enough according to the average ID and RND cells found experimentally in the SRAM employed. The bits required by the PWAR Unit are 245,796.

The sizes of the parameters stored in this memory for the c are shown in Table 7.1.

TABLE 7.1: Information in the NVM used by the AEGIS module.

	RND_mask (bits)	ID_mask (bits)	Helper Data (bits)
Cryptographic Unit	1860	4320	3720

## 7.3 Implementation results

The trusted virtual sensor exposed in the previous section was implemented in TSMC 90 nm using the design flow explained in Chapter 3.

Figure 7.3 shows how parallelization is carried during the configuration mode. The total number of clock cycles is  $155 + 360 + 310 + 20,483 = 21,308$  clock cycles.

Figure 7.4 shows the timing of the trusted sensing mode. Again the operations have been parallelized to accelerate the trusted virtual sensing

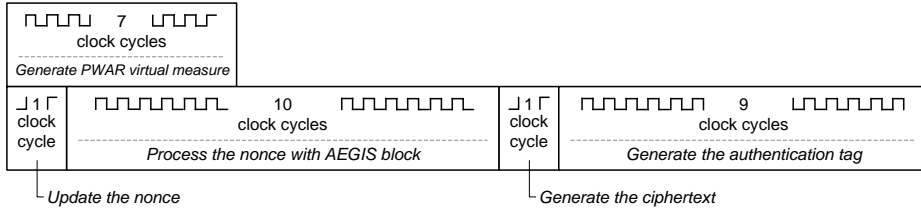


FIGURE 7.4: Timing of trusted sensing mode.

mode as much as possible. The output is provided in  $1+10+1+9=21$  clock cycles.

Table 7.2 shows the power consumption during trusted sensing mode and the area of the main blocks provided by Design Vision tool from Synopsys.

TABLE 7.2: Area and power consumption during trusted sensing mode.

	Area (mm <sup>2</sup> )	Power (mW) @50MHz
HDA block	0.0028	0.13
Nonce Counter	0.0041	0.15
AEGIS	0.14	0.77
Arithmetic block	0.015	0.24
Address Generator block	0.0007	0.15
SRAM	0.67	4.49
Control Unit	0.025	1.19

The trusted virtual sensor occupies 0.86 mm<sup>2</sup> and consumes 7.12 mW at 50 MHz in normal operation mode (trusted sensing mode). The area occupied by the Cryptographic Unit is 17.1% and its power consumption is 14.8% (considering that the SRAM and the Control are needed by virtual sensing and, hence, re-used by Cryptographic Unit).

Figure 7.5 shows the layout of the trusted virtual sensor extracted with the SoC Encounter tool from Cadence. The largest gray rectangular area in the upper part of the layout is the SRAM.



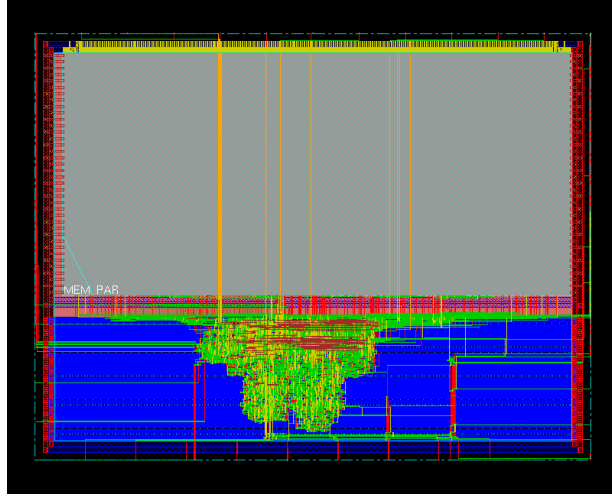


FIGURE 7.5: Layout of the trusted virtual sensor using AEGIS.

## 7.4 Conclusions

The VLSI design of trusted virtual sensors is very suitable for industrial applications where security is becoming increasingly important since it offers privacy, authenticity and integrity of the virtually sensed measurement and the circuit itself. The implementation of the design into a 90-nm CMOS technology occupies  $0.86 \text{ mm}^2$  and consumes  $7.12 \text{ mW}$  when trusted sensing at  $50 \text{ MHz}$ . Working at maximum frequency, the trusted virtual sensor allows sampling times lower than  $0.25 \mu\text{s}$ . The inclusion of security to the virtual sensor needs  $17.1\%$  of active area and  $14.8\%$  of power consumption.



## Chapter 8

# Conclusions

The main conclusion of this Ph.D. Dissertation is that the hardware proposed to generate PWA functions is easy to configure and program and allows including embedded controllers and sensors into challenging applications requiring very small size, low power consumption, rapid response, short time-to-market, and security in the data provided by the hardware as well as integrity of the hardware itself.

- The hardware architecture proposed to generate generic PWA functions (PWAG) allows the on-line programming of the binary search tree to explore, so that the same circuit can be used to generate different PWAG functions. The parameters that fix the edges of the polyhedral input partition and those that fix the output affine functions are downloaded into SRAMs previous to the normal operation mode when the PWA output corresponding to the input is computed. Hence, the circuit can be reprogrammed without being unplugged to perform different functionalities.
- Since continuous PWA functions with completely irregular polyhedral partitions can also be represented by using a scheme based on the lattice theory, a novel digital architecture has been proposed to implement the simplest lattice representation of a given PWA function (PWAL). The proposed architecture is advantageous compared to the PWAG architecture whenever the PWA functions have a large number of polytopes and complex binary search trees, because these functions usually apply the same affine function to many polytopes that can be clustered into super regions. The proposed PWAL architecture combines properly the affine functions according to the information stored in a binary matrix, without taking into account the

edges of the polytopes explicitly. Hence, it requires lower memory resources than PWAG architecture to implement many continuous PWA functions.

- The hardware architecture proposed to generate multiple PWA functions allows implementing very versatile circuits which can be configured and programmed to provide a PWAL, PWAS or PWAR function depending on the application. For applications where the generic description of a PWA function is available, as the case of model predictive control (MPC), the PWAL form can be generated by the circuit. For applications like virtual sensors, where the nonlinear relation between inputs and outputs can be approximated by a PWA function, the simplicial (PWAS) and hyper-rectangular (PWAR) PWA functions are more suitable because they can be obtained from numerical data after applying supervised learning algorithms. Hence, the same circuit can be used in many different scenarios. The proposed architecture reuses as much as possible the building blocks and it only enables the blocks required by the PWA form to generate, thus obtaining area efficient and low-power circuits.
- The ASIC-in-the-loop methodology developed in this Ph.D. has allowed the rapid verification of the manufactured prototypes working in open-loop as well as in closed-loop configurations. The two ASICs manufactured in TSMC 90-nm technology (named as PWAG and PWAX ASICs) were successfully tested in this way.
- The functionality added to Moby-dic Toolbox, which allows generation of the Verilog files needed to configure and verify the PWA ASICs in open-loop configuration, starting from the mathematical description of the problem to solve, increases the ease-of-use and development of the proposed designs in embedded control applications, which are relevant features demanded by the IC electronic market.
- The methodology developed to extract the PWAL form from Moby-dic Toolbox allows exploiting the advantages of this PWA form in hardware, which had not been used before this Ph.D. by hardware designers.
- The two ASICs manufactured, which were configured and programmed to solve several MPC cases of study, provided lower response times and much lower power consumption than the FPGA


solutions reported in the literature for the same problems. The versatility of the PWAX ASIC was exploited because, depending on the case of study, one form was proven to provide better performance than the others.

- The methodology based on Xfuzzy, which allows extracting the parameters needed to configure the PWAX ASIC to approximate the non-linear input-output behavior given by numerical information, increases the ease-of-use and development of the proposed design in virtual sensing applications, and, hence, also increases the versatility of the proposed PWAX ASIC.
- The PWAX ASIC, configured and programmed to act as a virtual sensor in an application example of the automotive domain, provided lower RMSE than a direct virtual sensor based on neural networks reported in the literature for the same problem.
- The authenticated ciphers AEGIS and ASCON are very suitable modules to ensure authenticity and confidentiality of the data provided by the proposed PWA hardware. Considering a 90-nm CMOS technology, ASCON-128 is very competitive in terms of area and AEGIS-128 in terms of speed.
- According to the implementation results of a trusted virtual sensor into a 90-nm CMOS technology, the inclusion of security needs 17.1% of active area and 14.8% of power consumption. Such cost can be affordable for many applications where security (privacy, authenticity, integrity, and anti-counterfeiting) is becoming increasingly important.



## Appendix A

### Brief CV

M. C. Martínez-Rodríguez  received the B.Sc. degree in computer engineering, the B.Sc. (Hons.) degree in electronic engineering, and the M.Sc. degree in microelectronics from the University of Seville, Seville, Spain, in 2007, 2010, and 2012, respectively. She has been with the Instituto de Microelectrónica de Sevilla IMSE-CNM, CSIC, Universidad de Sevilla, since 2010. She has been funded by an FPI grant from the Spanish Government. Her main research interest is the development of efficient digital circuits to implement piece-wise affine functions, microelectronic design of crypto-biometric systems, and trusted virtual sensors and controllers.

#### A.1 Journal Papers

- CV1** M. C. Martínez-Rodríguez, P. Brox, and I. Baturone. “A Comparative Analysis of VLSI Trusted Virtual Sensors”. Submitted to Special Issue: Selected Papers from NORCHIP-2017 of Elsevier’s Microprocessors and Microsystems, 2018.
- CV2** M. C. Martínez-Rodríguez, M.A. Prada, P. Brox, and I. Baturone. “VLSI Design of Trusted Virtual Sensor”. *Sensors* 2018, 18(2), 347.
- CV3** P. Brox, M. C. Martínez-Rodríguez, E. Tena-Sánchez, I. Baturone, and A. J. Acosta. “Application specific integrated circuit solution for multi-input multi-output piecewise-affine functions”. *Int. J. Circ. Theor. Appl.*, 44: 4–20, (2016).
- CV4** M. C. Martínez-Rodríguez, P. Brox and I. Baturone, “Digital VLSI Implementation of Piecewise-Affine Controllers Based on Lattice

Approach”, in *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 842-854, (May 2015).

**CV5** P. Brox, J. Castro-Ramirez, M. C. Martínez-Rodríguez, E. Tena, C.J. Jiménez, I. Baturone, and A.J. Acosta. “A programmable and configurable ASIC to generate piecewise-affine functions defined over general partitions”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 60.12 (Dec. 2013), pp. 3182–3194.

## A.2 Conference Papers

**CV6** M.C. Martínez-Rodríguez, M.A. Prada-Delgado, P. Brox, and I. Baturone. “CMOS Digital Design of a Trusted Virtual Sensor”. *IEEE Nordic Circuits and Systems Conference (NORCAS)*, October 2017, Linköping (Sweden).

**CV7** M.C. Martínez-Rodríguez, R. Arjona, P. Brox and I. Baturone, “Dedicated hardware IP module for fingerprint recognition”, *International Symposium on Consumer Electronics (ISCE)*, June 2015, Madrid (Spain).

**CV8** M.C. Martínez-Rodríguez, P. Brox, E. Tena, A. J. Acosta and I. Baturone, “Programmable ASICs for model predictive control”, *IEEE International Conference on Industrial Technology (ICIT)*, pp. 1593-1598, March 2015, Seville (Spain).

**CV9** M. C. Martínez-Rodríguez, R. Arjona, P. Brox and I. Baturone, “Dedicated hardware IP module for extracting singular points from fingerprints”, *21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 534-537, December 2014, Marseille (France).

**CV10** M.C. Martínez-Rodríguez, P. Brox, J. Castro-Ramirez, E. Tena, A. J. Acosta, I. Baturone, “ASIC-in-the-loop methodology for verification of piecewise affine controllers”, *19th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, December 2012, Seville (Spain).

**CV11** S. Eiroa, J. Castro, M. C. Martínez-Rodríguez, E. Tena, P. Brox, I. Baturone, “Reducing bit flipping problems in SRAM physical unclonable functions”, *19th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, December 2012, Seville (Spain).



- CV12** M.C. Martínez-Rodríguez, I. Baturone, P. Brox, “Design methodology for FPGA implementation of lattice piecewise-affine functions”, International Conference on Field-Programmable Technology (FPT), December 2011, pp.1-4, New-Delhi (India).
- CV13** M.C. Martínez-Rodríguez, I. Baturone, P. Brox, “Circuit implementation of piecewise-affine functions based on lattice representation”, 20th European Conference on Circuit Theory and Design (ECCTD), pp.644-647, August 2011, Linköping (Sweden).
- CV14** I. Baturone, M.C. Martínez-Rodríguez, P. Brox, A. Gersnoviez, S. Sánchez-Solano, “Digital implementation of hierarchical piecewise-affine controller”, IEEE International Symposium on Industrial Electronics (ISIE), pp.1497-1502, June 2011, Gdansk (Poland).

### A.3 Other Merits

- CV15** I. Baturone, P. Brox, M.C. Martínez-Rodríguez, A. J. Acosta, Patente “Método para acelerar la generación de funciones multivariantes afines a tramos con computación on-line del árbol de búsqueda y dispositivo para implementación del método”, Universidad de Sevilla (62%), CSIC (38%). Patent P201400091 in the Spanish territory on 05/02/2014.
- CV16** A.J. Acosta, I. Baturone, J. Castro, C.J. Jiménez, P. Brox, M.C. Martínez-Rodríguez, Patente “Método para generar funciones multivariantes afines a tramos con computación on-line del árbol de búsqueda y dispositivo para implementación del método”, Universidad de Sevilla (62%), CSIC (38%). Patent P201200608 in the Spanish territory on 18/05/2012. International extension PCT/ES2013/000134 on 04/06/2013. Licensed by “Canaan Research & Investment S.L.” on 01/11/2014. Current state: US Patent 14/405,552 on 04/12/2014 and EU Patent 13801227.3 on 18/12/2014.
- CV17** Pre-doctoral stay at “Computer Security and Industrial Cryptography group (COSIC)” of the KU Leuven, Belgium. 1 September 2015 - 3 December 2015. Leuven. Belgium.
- CV18** Summer school on real-world crypto and privacy. 31 May - 5 June 2015. Šibenik. Croatia.

- CV19** A. Oliveri, T. Poggi, B.A.G. Genuit, D. Barcelli, C. Alberto, F. Comaschi, M. Rubagotti, M.C. Martínez-Rodríguez, P. Brox, “Developers of Moby-dic toolbox for Matlab”.
- CV20** P. Brox, J. Castro, M. C. Martínez-Rodríguez, E.Tena, C. J. Jiménez, I. Baturone, and A. J. Acosta. “ASIC solution for piece-wise-affine forms”, Europractice Activity Report 2012.

## A.4 Projects

- ID-EO** Diseño de hardware cripto-biométrico para cifrado y autenticación de vídeo (TEC2014-57971-R); PI: Piedad Brox-Jiménez and Iluminada Baturone-Castillo; Supported by Proyecto I+D+i Retos Investigación - Ministerio de Economía, Industria y Competitividad; 2015 - 2018.
- CB-DOC** Gestión documental con autenticación segura mediante técnicas cripto-biométricas vía hardware (IPT-2012-0695-390000); PI: Iluminada Baturone-Castillo; Supported by Proyecto INNPACTO - Ministerio de Economía y Competitividad; 2012 - 2015.
- SEIs** Diseño hardware para sistemas empotrados en entornos inteligentes (TEC2011-24319); PI: Santiago Sánchez-Solano; Supported by Ministerio de Ciencia e Innovación; 2012 - 2014.
- CRIPTO-BIO** Diseño microelectrónico para autenticación cripto-biométrica (P08-TIC-03674); PI: Iluminada Baturone-Castillo; Supported by Proyecto de Excelencia Junta de Andalucía; 2009 - 2013.
- MOBY-DIC** Model-based synthesis of digital electronic circuits for embedded control (EC-IST-VIIPM No.-248858); PI: Antonio J. Acosta-Jiménez; Supported by the 7th Framework Programme for Research and Technological Development (7PM), European Commission; 2009 – 2012.
- DIMISION** Diseño microelectrónico de sistemas de visión para redes de sensores inteligentes (TEC2008-04920); PI: Santiago Sánchez-Solano; Supported by Ministerio de Ciencia e Innovación; 2009 - 2011.

# Bibliography

- [1] L. O. Chua and S.M. Kang. "Section-wise piecewise-linear functions: Canonical representation, properties, and applications". In: *Proceedings of the IEEE* 65.6 (June 1977), pp. 915–929.
- [2] D. M. W. Leenaerts and W. M. G. van Bokhoven. *Piecewise linear Modelling and Analysis*. Boston: Kluwer Academic Publisher, 1998.
- [3] P. Julian, M. Jordan, and A. Desages. "Canonical piecewise-linear approximation of smooth functions". In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 45.5 (May 1998), pp. 567–571.
- [4] P. Julian, A. Desages, and O. Agamennoni. "High-level canonical piecewise linear representation using a simplicial partition". In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 46.4 (Apr. 1999), pp. 463–480.
- [5] M. Storage and O. De Feo. "Piecewise-linear approximation of nonlinear dynamical systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 51.4 (Apr. 2004), pp. 830–842.
- [6] E. Sontag. "Nonlinear regulation: The piecewise linear approach". In: *IEEE Transactions on Automatic Control* 26.2 (Apr. 1981), pp. 346–358.
- [7] A. Bemporad, F. Borelli, and M. Morari. "Model predictive control based on linear programming: the explicit solution". In: *IEEE Trans. on Automatic Control* 47.12 (2002), pp. 1974–1985.
- [8] T. Poggi, M. Rubagotti, A. Bemporad, and M. Storage. "High-speed piecewise affine virtual sensors". In: *IEEE Trans. on Industrial Electronics* 59.2 (2012), pp. 1228–1237.
- [9] T. A. Johansen, W. Jackson, R. Schreiber, and P. Tondel. "Hardware Synthesis of Explicit Model Predictive Controllers". In: *IEEE Transactions on Control Systems Technology* 15.1 (Jan. 2007), pp. 191–197.
- [10] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare. "A System-on-a-Chip Implementation for Embedded Real-Time Model Predictive Control". In: *IEEE Transactions on Control Systems Technology* 17.5 (Sept. 2009), pp. 1006–1017.

- [11] A. Bemporad, A. Oliveri, T. Poggi, and M. Storage. "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations". In: *IEEE Transaction on Automatic Control* 56.12 (Dec. 2011), pp. 2883–2897.
- [12] A. G. Wills, G. Knagge, and B. Ninness. "Fast Linear Model Predictive Control Via Custom Integrated Circuit Architecture". In: *IEEE Transactions on Control Systems Technology* 20.1 (Jan. 2012), pp. 59–71.
- [13] A. Oliveri, G.J.L. Naus, M. Storage, and W. P. M.H. 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.
- [14] M. Mönnigmann and M. Kastsian. "Fast explicit model predictive control with multiway trees". In: *18th IFAC world congress 2011 : Milan, Italy*. Vol. 2. Sept. 2011, pp. 1356–1361.
- [15] F. Bayat, T. A. Johansen, and A. A. Jalali. "Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control". In: *Automatica* 47.3 (2011), pp. 571–577.
- [16] R. Rovatti, C. Fantuzzi, and S. Simani. "High-speed DSP-based implementation of piecewise-affine and piecewise-quadratic fuzzy systems". In: *Signal Processing* 80.6 (2000), pp. 951–963.
- [17] I. Baturone, F. J. Moreno-Velo, V. Blanco, and J. Ferruz. "Design of Embedded DSP-Based Fuzzy Controllers for Autonomous Mobile Robots". In: *IEEE Transactions on Industrial Electronics* 55.2 (Feb. 2008), pp. 928–936.
- [18] I. Baturone, S. Sánchez-Solano, A. A. Gersnoviez, and M. Brox. "An automated design flow from linguistic models to piecewise polynomial digital circuits". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. May 2010, pp. 3317–3320.
- [19] P. Echevarria, M. V. Martínez, J. Echanobe, I. del Campo, and J.M. Tarela. "Digital Hardware Implementation of High Dimensional Fuzzy Systems". In: *Applications of Fuzzy Sets Theory: 7th International Workshop on Fuzzy Logic and Applications, WILF 2007, Camogli, Italy, July 7-10, 2007. Proceedings*. Ed. by Francesco Masulli, Sushmita Mitra, and Gabriella Pasi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 245–252.
- [20] S. Sánchez-Solano, A. Cabrera, I. Baturone, F. J. Moreno-Velo, and M. Brox. "FPGA implementation of embedded Fuzzy Controllers for

- Robotic Applications". In: *IEEE Trans. on Industrial Electronics* 54.4 (2007), pp. 1937–1945.
- [21] R. Rovatti, M. Borgatti, and R. Guerrieri. "A geometric approach to maximum-speed n-dimensional continuous linear interpolation in rectangular grids". In: *IEEE Trans. on Computers* 47.8 (1998), pp. 894–899.
- [22] M. Rubagotti, T. Poggi, A. Oliveri, et al. "Low-complexity piecewise-affine virtual sensors: theory and design". In: *International Journal of Control* 87.3 (2014), pp. 622–632.
- [23] A. Oliveri, L. Cassottana, A. Laudani, et al. "Two FPGA-Oriented High Speed Irradiance Virtual Sensors for Photovoltaic Plants". In: *IEEE Transactions on Industrial Informatics* PP.99 (Sept. 2015), pp. 1–1.
- [24] M. Storace, F. Bizzarri, and M. Parodi. "Cellular non-linear networks for minimization of functionals. Part 1: Theoretical aspects". In: *International Journal of Circuit Theory and Applications* 29.2 (2001), pp. 151–167.
- [25] R. E. Groff, D. E. Koditschek, and P. P. Khargonekar. "Piecewise linear homeomorphisms: the scalar case". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 3. 2000, pp. 259–264.
- [26] L. O. Chua and A. Deng. "Canonical piecewise-linear analysis: Generalized breakpoint hopping algorithm". In: *International Journal of Circuit Theory and Applications* 14.1 (1986), pp. 35–52.
- [27] D. M. W. Leenaerts. "Further extensions to Chua's explicit piecewise linear function descriptions". In: *International Journal of Circuit Theory and Applications* 24.6 (1996), pp. 621–633.
- [28] M. Storace and M. Parodi. "Towards analog implementations of PWL two-dimensional non-linear functions". In: *International Journal of Circuit Theory and Applications* 33.2 (2005), pp. 147–160.
- [29] T. Kettner, C. Heite, and K. Schumacher. "Analog CMOS realization of fuzzy logic membership functions". In: *IEEE Journal of Solid-State Circuits* 28.7 (July 1993), pp. 857–861.
- [30] M. Delgado-Restituto, A. Rodríguez-Vázquez, and F. Vidal. "Non-linear synthesis using ICs". In: *Wiley Encyclopedia of Electrical and Electronics Engineering* 14 (1999). Ed. by Webster J. G, pp. 472–502.
- [31] A. Rodríguez-Vázquez, R. Navas, M. Delgado-Restituto, and F. Vidal-Verdu. "A modular programmable CMOS analog fuzzy controller

- chip". In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 46.3 (Mar. 1999), pp. 251–265.
- [32] M. Parodi, M. Storace, and P. Julián. "Synthesis of multiport resistors with piecewise-linear characteristics: a mixed-signal architecture". In: *International Journal of Circuit Theory and Applications* 33.4 (2005), pp. 307–319.
- [33] I. Baturone, S. Sanchez-Solano, A. Barriga, and J. L. Huertas. "Implementation of CMOS fuzzy controllers as mixed-signal integrated circuits". In: *IEEE Transactions on Fuzzy Systems* 5.1 (Feb. 1997), pp. 1–19.
- [34] 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.
- [35] A. Oliveri, C. Gianoglio, E. Ragusa, and M. Storace. "Low-complexity digital architecture for solving the point location problem in explicit Model Predictive Control". In: *Journal of the Franklin Institute* 352.6 (2015), pp. 2249–2258.
- [36] 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". In: *IEEE Transactionis on Circuits and Systems I* 59.12 (Dec. 2012), pp. 2920–2933.
- [37] M. Storace and T. Poggi. "Digital architectures realizing piecewise-linear multi-variate functions: two FPGA implementations". In: *Int. Journal of Circuit Theory and Applications* 39.1 (2009), pp. 1–15.
- [38] A. Gersnoviez, M. Brox, and I. Baturone. "High-Speed and Low-Cost Implementation of Explicit Model Predictive Controllers". In: *IEEE Transactions on Control Systems Technology* PP.99 (2017), pp. 1–16.
- [39] M. Di Federico, T. Poggi, P. Julián, and M. Storace. "Integrated circuit implementation of multi-dimensional piecewise-linear functions". In: *Digital Signal Processing* 20.6 (2010), pp. 1723–1732.
- [40] J.A. Rodríguez, O. D. Lifschitz, V. M. Jiménez-Fernández, P. Julián, and O. E. Agamennoni. "Application-Specific Processor for Piecewise Linear Functions Computation". In: *IEEE Trans. on Circuits and Systems I: Regular Papers* 58.5 (2011), pp. 971–981.
- [41] 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.

- [42] G.J.L. Naus, J. Ploeg, M.J.G. Van de Molengraft, W.P.M.H. Heemels, and M. Steinbuch. "Design and implementation of parameterized adaptive cruise control: An explicit model predictive control approach". In: *Control Engineering Practice* 18.8 (2010), pp. 882–892.
- [43] C.K. Chui, B. P. Nguyen, Y. Ho, et al. "Embedded Real-Time Model Predictive Control for Glucose Regulation". In: *World Congress on Medical Physics and Biomedical Engineering May 26-31, 2012, Beijing, China*. Ed. by Mian Long. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1437–1440.
- [44] A. Malinowski and H. Yu. "Comparison of Embedded System Design for Industrial Applications". In: *IEEE Transactions on Industrial Informatics* 7.2 (May 2011), pp. 244–254.
- [45] I. Baturone, A. Barriga, S. Sánchez-Solano, C. J. Jiménez-Fernández, and D. R. López. *Microelectronic Design of Fuzzy Logic-based Systems*. Boca Raton, FL, USA: CRC Press, Inc., 2000.
- [46] P. Tøndel, T.A. Johansen, and A. Bemporad. "Evaluation of piecewise affine control via binary search tree". In: *Automatica* 39.5 (2003), pp. 945–950.
- [47] J.M. Tarela and M.V. Martínez. "Region configurations for realizability of lattice Piecewise-Linear models". In: *Mathematical and Computer Modelling* 30.11 (1999), pp. 17–27.
- [48] C. Wen, X. Ma, and B. E. Ydstie. "Analytical expression of explicit MPC solution via lattice piecewise-affine function". In: *Automatica* 45.4 (2009), pp. 910–917.
- [49] M. Storace, L. Repetto, and M. Parodi. "A method for the approximate synthesis of cellular non-linear networks—Part 1: Circuit definition". In: *International Journal of Circuit Theory and Applications* 31.3 (2003), pp. 277–297.
- [50] L. Repetto, M. Storace, and M. Parodi. "A method for the approximate synthesis of cellular non-linear networks—Part 2: Circuit reduction". In: *International Journal of Circuit Theory and Applications* 31.3 (2003), pp. 299–313.
- [51] 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". In: *IET Control Theory Applications* 6.8 (May 2012), pp. 1015–1028.
- [52] A. N. Fuchs, C. N. Jones, and M. Morari. "Optimized decision trees for point location in polytopic data sets - application to explicit

- MPC". In: *Proceedings of the 2010 American Control Conference*. June 2010, pp. 5507–5512.
- [53] F. Bayat. "Comments on "Analytical expression of explicit MPC solution via lattice piecewise-affine function" [Automatica 45 (2009) 910–917]". In: *Automatica* 48.11 (2012), pp. 2993–2994.
- [54] C. Wen, X. Ma, and B. E. Ydstie. "Reply to "Comments on 'Analytical expression of explicit MPC solution via lattice piecewise-affine function' [Automatica 45 (2009) 910–917]"". In: *Automatica* 48.11 (2012), pp. 2995–2996.
- [55] J.M. Tarela, J.M. Perez, and V. Aleixandre. "Minimization of lattice polynomials on piecewise linear functions (Part I)". In: *Mathematics and Computers in Simulation* 17.2 (1975), pp. 79–85.
- [56] J.M. Tarela, L.A. Bailon, and E. Sanz. "Minimization of lattice polynomials on piecewise linear functions (Part II)". In: *Mathematics and Computers in Simulation* 17.2 (1975), pp. 121–127.
- [57] M. Chien and E. Kuh. "Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision". In: *IEEE Transactions on Circuits and Systems* 24.6 (June 1977), pp. 305–317.
- [58] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan. "Linear matrix inequalities in control theory". In: *Studies in Applied Mathematics, SIAM* 15 (1994).
- [59] 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.
- [60] M. Lazar and W.P.M.H. Heemels. "Predictive control of hybrid systems: Input-to-state stability results for sub-optimal solutions". In: *Automatica* 45.1 (2009), pp. 180–185.
- [61] A. Alessio and A. Bemporad. "A Survey on Explicit Model Predictive Control". In: *Nonlinear Model Predictive Control*. Vol. 384. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2009, pp. 345–369.
- [62] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. "The explicit linear quadratic regulator for constrained systems". In: *Automatica* 38 (2002), pp. 3–20.
- [63] H. Li, D. Yu, and J. E. Braun. "A review of virtual sensing technology and application in building systems". In: *HVAC&R Research* 17.5 (2011), pp. 619–645.



- [64] G. Heredia and A. Ollero. "Virtual Sensor for Failure Detection, Identification and Recovery in the Transition Phase of a Morphing Aircraft". In: *Sensors* 10.3 (2010), pp. 2188–2201.
- [65] J. A. Sánchez, F. Rodríguez, J. L. Guzmán, and M. R. Arahál. "Virtual Sensors for Designing Irrigation Controllers in Greenhouses". In: *Sensors* 12.11 (2012), pp. 15244–15266.
- [66] A. Bustillo, M. Correa, and A. Reñones. "A Virtual Sensor for Online Fault Detection of Multitooth-Tools". In: *Sensors* 11.3 (2011), pp. 2773–2795.
- [67] J. Stephant, A. Charara, and D. Meizel. "Virtual sensor: application to vehicle sideslip angle and transversal forces". In: *IEEE Transactions on Industrial Electronics* 51.2 (Apr. 2004), pp. 278–289.
- [68] C. Novara, F. Ruiz, and M. Milanese. "Direct Identification of Optimal SM-LPV Filters and Application to Vehicle Yaw Rate Estimation". In: *IEEE Transactions on Control Systems Technology* 19.1 (Jan. 2011), pp. 5–17.
- [69] B. Zhang, H. Du, J. Lam, N. Zhang, and W. Li. "A Novel Observer Design for Simultaneous Estimation of Vehicle Steering Angle and Sideslip Angle". In: *IEEE Transactions on Industrial Electronics* 63.7 (July 2016), pp. 4357–4366.
- [70] M. Pinelis. "Automotive sensors and electronics: trends and developments in 2013". In: *Automotive Sensors and Electronics Expo, Detroit, Michigan*. 2013.
- [71] T. Xu, J. B. Wendt, and M. Potkonjak. "Security of IoT systems: Design challenges and opportunities". In: *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2014, pp. 417–423.
- [72] R. Santamarta. "Go Nuclear: Breaking radiation monitoring devices". In: *BlackHat USA 2017, Las Vegas*. July 2017.
- [73] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Berlin Heidelberg, 2009.
- [74] J. Turner. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standards Publication, 2008.
- [75] A. Bogdanov, M. Knežević, G. Leander, et al. "SPONGENT: A Lightweight Hash Function". In: *Cryptographic Hardware and Embedded Systems – CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 312–325.

- [76] J. P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. “Quark: A Lightweight Hash”. In: *Journal of Cryptology* 26.2 (Apr. 2013), pp. 313–339.
- [77] J. Guo, T. Peyrin, and A. Poschmann. “The PHOTON family of lightweight hash functions”. In: *Advances in Cryptology-CRYPTO 2011*. Vol. 6841. Lecture Notes in Computer Science (LNCS), 2011, pp. 229–239.
- [78] S. Eiroa and I. Baturone. “FPGA implementation and DPA resistance analysis of a lightweight HMAC construction based on PHOTON hash family”. In: *23rd International Conference on Field programmable Logic and Applications*. Sept. 2013, pp. 1–4.
- [79] S. Meguerdichian and M. Potkonjak. “Security primitives and protocols for ultra low power sensor systems”. In: *2011 IEEE SENSORS Proceedings*. Oct. 2011, pp. 1225–1227.
- [80] O. Pfeiffer. *Implementing Scalable CAN Security with CANcrypt: Authentication and Encryption for CANopen, J1939 and other Controller Area Network or CAN FD Protocols*. Embedded Systems Academy Inc., 2017.
- [81] *Cryptographic Competitions*. URL: <https://competitions.cryp.to/caesar.html>.
- [82] H. Wu and B. Preneel. “AEGIS: A Fast Authenticated Encryption Algorithm”. In: *Selected Areas in Cryptography – SAC 2013: 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisoněk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 185–201.
- [83] *ASCON: A Family of Authenticated Encryption Algorithms*.
- [84] H. Gross, E. Wenger, C. Dobraunig, and C. Ehrenhöfer. “ASCON hardware implementations and side-channel evaluation”. In: *Microprocessors and Microsystems* 52 (2017), pp. 470–479.
- [85] D. Samyde, S. Skorobogatov, R. Anderson, and J. J. Quisquater. “On a new way to read data from memory”. In: *First International IEEE Security in Storage Workshop, 2002. Proceedings*. Dec. 2002, pp. 65–69.
- [86] J. Guajardo, S. S. Kumar, G. Schrijen, and P. Tuyls. “FPGA Intrinsic PUFs and Their Use for IP Protection”. In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 63–80.
- [87] I. Baturone, M. A. Prada-Delgado, and S. Eiroa. “Improved generation of identifiers, secret keys, and random numbers From SRAMs”.

- In: *IEEE Transactions on Information Forensics and Security* 10.12 (Dec. 2015), pp. 2653–2668.
- [88] A. Oliveri, D. Barcelli, A. Bemporad, et al. “MOBY-DIC: A MATLAB Toolbox for Circuit-Oriented Design of Explicit MPC”. In: *IFAC Proceedings Volumes* 45.17 (2012). 4th IFAC Conference on Nonlinear Model Predictive Control, pp. 218–225.
- [89] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. “Multi-Parametric Toolbox 3.0”. In: *Proc. of the European Control Conference*. <http://control.ee.ethz.ch/~mpt>. Zürich, Switzerland, July 2013, pp. 502–510.
- [90] A. Bemporad. *Hybrid Toolbox - User’s Guide*. <http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox>. 2004.
- [91] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, May 2000. 456 pp.
- [92] M. Lutz, W. Golderer, J. Gerstenmeier, et al. “A precision yaw rate sensor in silicon micromachining”. In: *International Conference on Solid State Sensors and Actuators, 1997. TRANSDUCERS ’97 Chicago*. Vol. 2. June 1997, pp. 847–850.
- [93] D. Kim, Y. Park, and H. Lee. “Sensor offset compensation for a vehicle yaw rate sensor using fuzzy logic”. In: *International Conference on Control, Automation and Systems, 2007. ICCAS ’07*. Oct. 2007, pp. 362–366.