# ASIC-in-the-loop methodology for verification of piecewise affine controllers

M.C. Martínez-Rodríguez[1,2], P. Brox[2], J. Castro[1,2], E. Tena[1,2], A. J. Acosta[1,2], I. Baturone[1,2]
[1]Department of Electronics and Electromagnetism. University of Seville, Spain
[2]Microelectronics Institute of Seville. IMSE-CNM-CSIC, Seville, Spain
Email: {macarena,brox,casram,erica,acojim,lumi}@imse-cnm.csic.com

***Abstract*—This paper exposes a hardware-in-the-loop methodology to verify the performance of a programmable and configurable application specific integrated circuit (ASIC) that implements piecewise affine (PWA) controllers. The ASIC inserted into a printed circuit board (PCB) is connected to a logic analyzer that generates the input patterns to the ASIC (in particular, the values to program the memories, configuration parameters, and values of the input signals). The output provided by the ASIC is also taken by the logic analyzer. A Matlab program controls the logic analyzer to verify the PWA controller implemented by the ASIC in open-loop as well as in closed-loop configurations.**

## I. Introduction

The concept known as Hardware-In-the-Loop (HIL) is a technique that is used to develop and test complex embedded systems. HIL communicates the physical hardware under test and links it to a computer model that simulates the other aspects of the system. HIL has been successfully applied to the development and testing of embedded controllers in a wide variety of engineering fields: automotive applications [1], power electronics systems [2]-[3], radar systems[4]-[5], and robotics [6].

In the particular case of embedded controllers, the ideal condition is to test the controller against the real plant but this could impose severe limitations. One solution is to model the plant under control by adding the mathematical representations that is referred as plant simulation. Therefore, the embedded controller to be tested interacts with this plant simulation [7]-[9].

Piece-Wise Affine (PWA) functions are very useful in the design of controllers and virtual sensors in many applications [10]-[13]. Several digital realizations of PWA functions have been proposed in the literature during the last years to implement four canonical forms: Generic PWA (PWAG) [10], Simplicial PWA (PWAS) [11], Lattice PWA (PWAL) [12], hyperRectangular PWA (PWAR) [13]. Digital architectures for the implementation of all these canonical forms have been proposed through FPGA implementations. More recently, a configurable and programmable architecture to implement PWAG functions has been integrated in a nanometric VLSI technology [14]. This ASIC is able to implement PWA functions with a configurable number of inputs (from one to four) and one output.

FPGA manufacturers (Xilinx and Altera), in collaboration with The MathWorks, offer commercial tools that enable the HIL simulation of FPGA implementations directly within Simulink [15]-[16]. This strategy provides an efficient technique to verify the FPGA implementation of the PWA controllers [17].

The work presented in this paper describes an ASIC-in-the-loop methodology for the verification of PWA embedded controllers. The paper is organized as follows. Section II explains the proposed ASIC-in-the-loop methodology. Section III addresses the open-loop verification of the controller whereas the closed-loop verification is detailed in Section IV. The methodology is applied to verify a controller that regulates a Double Integrator System in Section V. Finally, conclusions of this work are expounded in Section VI.

## II. ASIC-in-the-loop methodology

The proposed methodology has been proven with a digital ASIC that implements PWA controllers based on the generic canonical form [14].

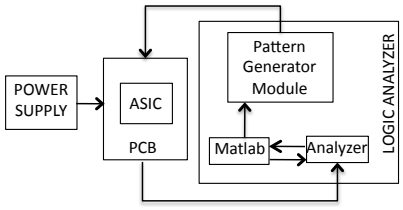
A PWA controller, $f_{PWA} : D \rightarrow \mathbb{R}$, provides a linear (affine) control law for each region in which the input domain, $D$, is partitioned ($D \subset \mathbb{R}^n$):

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

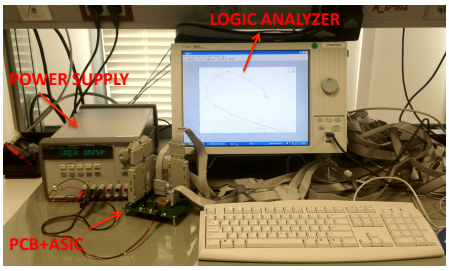
where $x \in \mathbb{D}$, $[f_i \ \ g_i] \in \mathbb{R}^{n+1}$, and $P_i \in \mathbb{R}^n$. $P_i$ are $P$ non overlapping regions, called polytopes, that induce a polyhedral partition of the domain.

The architecture contains two memories, one memory implements the search tree whereas the other one stores the coefficients ($f_i$, $g_i$) to generate the affine control laws to each polytope and its edges for each input value ($x$). The architecture is both programmable and configurable [14]. Configurability allows changing the partition of the input domain and the number of inputs (from one to four). Therefore, the proposed architecture is able to work with different number of dimensions, memory sizes, determining the complexity of the search tree and number of parameters, etc. Programmability allows, once the desired configuration is selected, to work with different control surfaces, by reprogramming the memory contents. The architecture that is implemented on the ASIC has different working modes; two of them enable the writing process of the memories and the other one is the operation mode in which the ASIC works as a controller.

The main components to perform the ASIC-in-the-loop methodology are (see Fig. 1):




(a) Simplified scheme




(b) Photograph of the experimental set up

Fig. 1. Setup of the ASIC-in-the-loop methodology

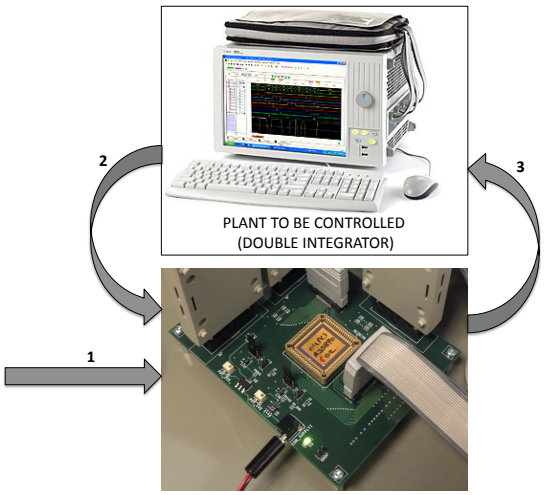



Fig. 2. Scheme of the closed-loop verification

- Power Supply that supplies electric power to the PCB (5V).
- PCB is used as a physical interface between the ASIC and the laboratory instruments. It includes a special circuitry to adequate 5V from the power supply to 2.5V (ASIC pad ring voltage) and 1.2V (ASIC core voltage). Input patterns to the ASIC are provided by 3.3V pods of the Logic Analyzer. Then, two level converters are used to adequate the input signal to 2.5V needed by the ASIC.
- Digital ASIC that implements the PWA controller. The maximum clock frequency is 107.5 MHz (with 2 inputs) and 98 MHz (with 4 inputs). The power consumption is 38.08 mW@107.5 MHz and 41.91 mW@98 MHz. For the application example in Section V, the working frequency is 37.5 MHz with 18.1 mW of power consumption.
- Logic Analyzer Agilent 16823A that is used for the following functions:
  - Controller Configuration: a Matlab description of the controller that provides the data to be stored in the ASIC memories. The communication protocol with the ASIC is implemented by Matlab R2008 installed on the analyzer, using Microsoft COM (Component Object Model) automation.
  - Pattern Generator: this module of the instrument is used to provide the input values for the controller.
  - Analyzer: this mode provides a clock internal that is used to sample data. Additionally, it captures data that corresponds to the input and output values of the controller. For the application example in Section V, the sampling time is 26.67 ns.
  - Plant: a Matlab program iterates the plant. Hence, it generates a new set of inputs to the controller, by taking into account the previous plant state and controller outputs.

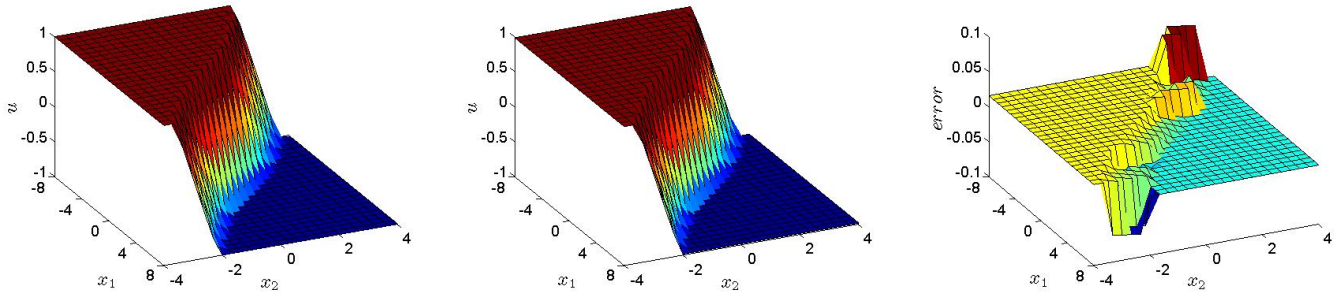
The proposed ASIC-in-the-loop methodology is fully designed in a unique design environment managed by Matlab. Firstly, the controller configuration is obtained from the Moby-dic Toolbox [18]. This toolbox generates a Verilog file with the testbench that programs the controller. Automatically, this testbench generates a Comma Separated Values (CSV) file that contains the values to write the data memories. Next, this CSV file fixes the operation mode giving the corresponding inputs depending on what is going to be verified. This CSV file is directly interpreted by the logic analyzer. Each line corresponds to the value of the inputs of the system in each semi-period of the sampling clock.

A Matlab script provides the connectivity with the logic analyzer. Next, the logic analyzer configuration files are loaded with a description of the channels used by the pattern generator and analyzer modules. These channels are connected to the corresponding PCB pins. The working frequency, the trigger value, and the sample conditions are also fixed by this Matlab script.

The next step is to load the CSV file that has been generated previously. At this point the logic analyzer can be set to run. The pattern generator module sends the same data pattern repeatedly, whereas the analyzer module runs for a time slot in which the pattern generator has sent the complete data pattern at least once. After that, the analyzer module is stopped firstly and afterwards the pattern generator is also stopped. The part of the Matlab program that implements the plant requires the data acquired by the analyzer module. Finally, the removal of the software connection is required since if not, when a new connection is launched, a new software connection should be created. This new connection will enter into conflict with the old existing software connection.

## III. Open-loop verification

In this case, the Verilog file generates a CSV file that writes both memories and configures the operation mode. This file




(a) Simulation (b) Real (c) Error

Fig. 3. Control Surface of a Double Integrator Controller

sweeps all the values in the input domain. The Matlab script generates a matrix, called Real, with the data captured by the logic analyzer, where the first columns are the input values to the controller and the last column is the output obtained from the controller. Each row represents a different inputs-output value (inputs sweep over the entire domain). To verify the correct functionality of the PWA controller, a second matrix, called Sim, is composed with the same info coming from the simulation results obtained with ModelSim simulator. The open-loop ASIC verification is satisfied when the content of both matrixes is the same (Matlab relational operator "equal to" is used).

A second verification is to calculate the RMSE between the ASIC and the simulation results obtained from the Moby-dic Toolbox. The RMSE is expressed as:

$$RMSE = \sqrt{\frac{1}{N_{pts}} \sum_{1}^{N_{pts}} \left(\hat{u}(x_i) - \tilde{u}(x_i)\right)^2} \tag{2}$$

where $\hat{u}(x_i)$ is the result given by the Moby-dic Toolbox and $\tilde{u}(x_i)$ is the output provided by the ASIC and $N_{pts}$ is the number of cells in the matrices.

## IV. Closed-loop verification

In the Matlab script, a starting point is fixed and it is converted from its real value to a new base used by the pattern generator (arrow 1 in Fig. 2). This new value is written in the corresponding lines of the pattern. Then the script runs the pattern generator that captures the data with the analyzer module. The resulting data are processed generating a vector that contains the input data and its corresponding output.

Once the vector of the inputs from the last state and its corresponding output is captured (arrow 2 in Fig. 2), a Matlab program implements the plant to generate the new state. The plant model in Matlab calculates the new inputs for the controller taking into account the previous output of the controller and the previous state(s) of the plant (in case of dynamical plants). These inputs have to be written in the pattern and they are provided to the ASIC controller (see arrow 3 in Fig. 2). Once again the new output is captured and processed. This operation will be repeated as many times as it is required so that the plant could be properly controlled. Finally, the last step is to remove the connection with the logic analyzer.

## V. Application Example: Double Integrator

To illustrate the methodology, let us consider a controller to regulate a Double Integrator system to the origin. The discrete-time double integrator with sampling of one time unit is represented as follows:

$$\begin{aligned} \mathbf{x}_{k+1} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \\ y_k &= \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}_k \end{aligned} \tag{3}$$

where $\mathbf{x}_k \in \mathbb{R}^2$ denotes the state variable and $u_k$ the control input at discrete time $k \in \mathbb{N}$. The control objective is to regulate the system to the origin under the hard constraint $-1 \leq u_k \leq 1$, $k \in \mathbb{N}$. The state domain is given by $S = \{\mathbf{x} \in \mathbb{R}^2 : -8 \leq x_1 \leq 8, -4 \leq x_2 \leq 4\}$. An optimal explicit control law (PWAG) has been obtained using the Moby-dic Toolbox [18], which provides the Verilog file. The PWAG control law is characterized by a number of edges plus polytopes equal to 163, and a search tree that has 191 nodes and a maximum depth of $d = 8$. For this example, the ASIC has been configured to work with 191 words for the search tree and 163 words for the edges and affine functions.

An open-loop verification has been performed, loading a CSV file that contains the values over the entire input domain. For each component of the bi-dimensional domain, 25 points have been taken resulting 625 points ($N_{pts}$). After running the logic analyzer, the results are stored in the Real matrix and compared with the simulation results of the controller obtained with the Moby-dic Toolbox. The resulting RMSE is 0.018. In Fig. 3 the control surface has been illustrated in the ASIC case (see Fig. 3(a)), simulation case (see 3(b)), and the error obtained as the difference between them (see 3(c)).

For the closed-loop verification the plant corresponding to the Double Integrator System is modeled in Matlab according to eq. (3). Fig. 4 illustrates the evolution of the closed-loop system in the state space (test), starting from the state [3, -4], and it is compared with the simulation of the controller in the Moby-dic Toolbox (sim). Fig. 5 shows the evolution of the

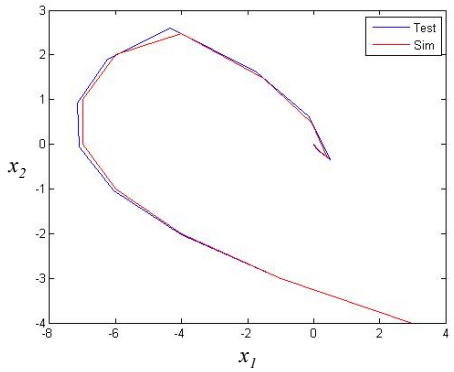



Fig. 4. Closed-loop verification: state-space evolution

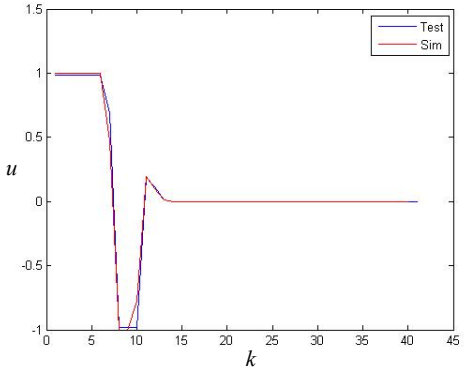



Fig. 5. Closed-loop verification: control signal evolution

control signal (test and sim output). Finally, other trajectories with different starting states in the ASIC are shown in Fig. 6 (test results).

## VI. Conclusions

This paper presents a novel ASIC-in-the-loop methodology for verification of PWA controllers, which allows both open-loop and closed-loop verifications. It is fully automatized within a unique framework managed by Matlab.

## Acknowledgment

This work was partially supported by MOBY-DIC project FP7-INFSO-ICT-248858 (www.mobydic-project.eu) from European Community, TEC2011-24319 project from the Spanish Government, and P08-TIC-03674 project from the Andalusian Regional Government (with support from the PO FEDER). P. Brox is supported under the post-doctoral program called Juan de la Cierva from the Spanish Ministry of Science and Innovation.

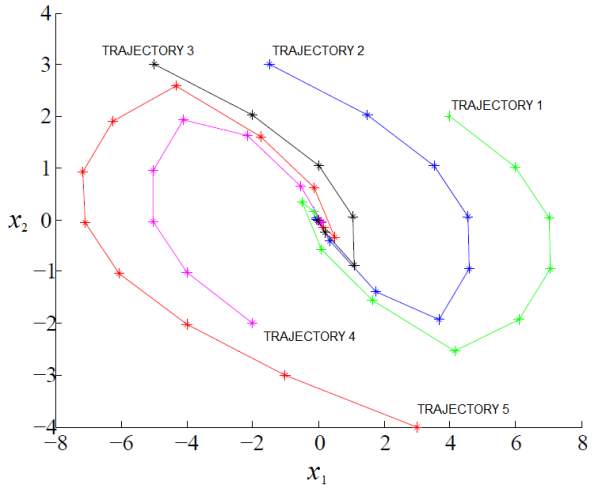



Fig. 6. State-space evolution with different starting states

## References

[1] A. Hentunen, J. Suomela, A. Leivo, M. Liukkonen, P. Sainio, *Hardware-in-the-loop verification environment for heavy-duty hybrid electric vehicles*, IEEE Vehicle Power and Propulsion Conference (VPPC), vol., no., pp.1-6, 1-3 Sept. 2010.

[2] M. O, O. Faruque and V. Dinavahi, *Hardware-in-the-Loop Simulation of Power Electronic Systems Using Adaptive Discretization*, IEEE Transactions on Industrial Electronics, vol.57, no.4, pp.1146-1158, April 2010.

[3] B. Lu, X. Wu, H. Figueroa, and A. Monti, *A low-cost real-time hardware in-the-loop testing approach of power electronics controls*, IEEE Trans. Ind. Electron., vol. 54, no. 2, pp. 919931, Apr. 2007.

[4] G.-L. Wang, D.-L. Su, Y.-L. Zhao, X.-D. Gao, D. Di, *Study on time-sharing methods for hardware-in-the-loop radar netting simulation*, 8th International Symposium on Antennas, Propagation and EM Theory (ISAPE), pp.1252-1255, 2-5 Nov. 2008.

[5] W. S. Sward, D. E. Reed, *Phase continuous radar test set*, AUTOTESTCON, 2010 IEEE , vol., no., pp.1-5, 13-16 Sept. 2010.

[6] A. Martin, M. Reza Emami, *Dynamic load emulation in Hardware-in-the-Loop Simulation of Robot Manipulators*, IEEE Trans. Ind. Electron., vol. 58, no. 7, pp. 2980 2987, July 2011.

[7] L. Kis, G. Regula, B. Lantos,*Design and hardware-in-the-loop test of the embedded control system of an indoor quadrotor helicopter*, Intelligent Solutions in Embedded Systems, vol., no., pp.1-10, 10-11 July 2008.

[8] A. Sánchez, A. de Castro, J. Garrido, *A comparison of simulation and hardware-in-the-loop alternatives for digital control of power converters*, IEEE Transactions on Industrial Informatics, vol.8, no.3, pp.491-500.

[9] X. Wu, H. Figueroa, A. Monti,*Testing of digital controllers using real-time hardware in the loop simulation*, Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual , vol.5, no., pp. 3622- 3627 Vol.5, 20-25 June 2004.

[10] A. Oliveri, T. Poggi, M. Storace, *Circuit implementation of piecewise-affine functions based on a binary search tree*, European Conference on Circuit Theory and Design, 2009. ECCTD 2009, pp.145-148, 23-27 Aug. 2009.

[11] M. Storace and T. Poggi, *Digital architectures realizing piecewise-linear multi variate functions: Two FPGA implementations*, Int. J. Circuit Theory Appl., vol. 37, 2010.

[12] 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), Linkping, Sweden, August 29-31, pp. 644647, 2011.

[13] F. Comaschi, B.A.G. Genuit, A. Oliveri, W.P.M.H. Heemels and M. Storace, *FPGA implementations of piecewise affine functions based on multi-resolution hyperrectangular partitions*, IEEE Transactions on Circuits and Systems I, 2012, accepted.

[14] A. J. Acosta, I. Baturone, J. Castro-Ramírez, C.J. Jiménez-Fernández, P. Brox, M.C. Martínez-Rodríguez, *Método para generar funciones multivariables afines a tramos con computación on-line del árbol de búsqueda y dispositivo para implementación del método*. No. 201200608. Oficina Española de Patentes y Marcas.

[15] http://www.xilinx.com/tools/sysgen.htm

[16] http://www.altera.com/products/software/products/dsp/dsp-builder.html

[17] M.C. Martínez-Rodríguez, I. Baturone and P. Brox, *Design methodology for FPGA implementation of lattice piecewise-affine functions*, in Field-Programmable Technology (FPT), 2011 International Conference on, December 12-14 2011, pp. 14.

[18] *http://ncas.dibe.unige.it/software/MOBY-DIC_Toolbox/*