19th Symposium IMEKO TC 4 Symposium and 17th IWADC Workshop
Advances in Instrumentation and Sensors Interoperability
July 18-19, 2013, Barcelona, Spain

Automatic and Systematic Control of Experimental Data Measurements on ASICs

E. Tena, J. Castro, A. J. Acosta

Department of Electronics and Electromagnetism, University of Seville, Spain
Microelectronics Institute of Seville, IMSE-CNM-CSIC, Seville, Spain
E-mail: {erica, casram, acojim}@imse-cnm.csic.es

Abstract-This paper presents a methodology to perform automatic and systematic characterization test on application specific integrated circuits (ASICs). The proposed methodology is based on the automatic control of all laboratory equipment and the data processing with Matlab. The ASIC, or integrated system, is connected to controllable test equipment to generate patterns and collect the output data provided by the ASIC. The methodology that provides the Matlab script controlling the equipment, test process, making the analysis of the results and supervising the whole process, can be easily adapted to different experiments and ASIC features. The test of a piecewise affine (PWA) ASIC controller has been used to experimentally prove the automatic control in both open-loop as well as in closed-loop configurations, reducing the risk of manual measurement errors.

Keywords- Test automation, CAD tools for instrumentation and measurements, ASIC test and characterization, Matlab supervised experiments.

I. Introduction

All ASICs need a test and characterization stage after fabrication. This stage is usually a very complex and repetitive task, even more with the increasing of their complexity and integration scale. Therefore, an optimization and automation of the test is mostly required to save time and reduce the risk of manual measurement errors. There are a lot of ways to face the automatic test tasks. Between handmade solutions requiring the participation of skill engineers and the commercial automatic test machines (ATEs), the gap is very wide. ATEs perform systematic test on digital ASICs, but they are very expensive and need skilled people due to ATE complexity, besides they are more related to production. The handmade test is error-prone and extremely depending on the skilled test engineer. In the middle, there are different solutions. Among others, low-cost FPGA-based test environments have been proposed but are limited by the performances of the virtual test equipments [1]. On the other hand, low-cost ATEs machines are now in the market [2], but needing microprocessors or microcontrollers introducing additional complexity.

This paper presents a systematic and automatic test control methodology able to be used in a conventional laboratory. Using controllable equipments, the methodology uses a Matlab [3] script controlling the equipment, the test process, making the analysis of the results and supervising the whole process. A comparison between expected results obtained by simulation, and measured experimental data is made within the characterization process. The methodology can be easily adapted to different experiments and ASIC features. Our contribution tries to emulate an ATE using a laboratory with basic instrumentation. With a PC running Matlab scripts, our low-cost solution does not need skilled people, no extra instruments, and is totally customizable and adaptable for any digital ASIC under test. The proposed methodology has been applied to the test and characterization of an ASIC implementing a Generic PWA (PWAG) function [4].

The paper is organized as follows: Section II presents the proposed methodology, Section III considers an application example of the methodology. Results for such example are included in Section IV. Finally Section V presents the most interesting conclusions.

II. Methodology

The proposed methodology is fully designed in a unique design environment managed by Matlab in real time. There are three main domains, related each other, but contemplated in the methodology in different abstraction levels, as it is shown in Figure 1. In the lowest level, the instrumentation control is contemplated via specific communication protocols. For each instrument, a preliminary analysis has to be done to know the available connectivity that can be done with a PC to control the instrument automatically. Once known the types of connections available, the most feasible and appropriate connection is chosen for the setup. For instance, some types of connections used in the application example considered in this paper are COM [5] and GPIB [6] (Figure 2).

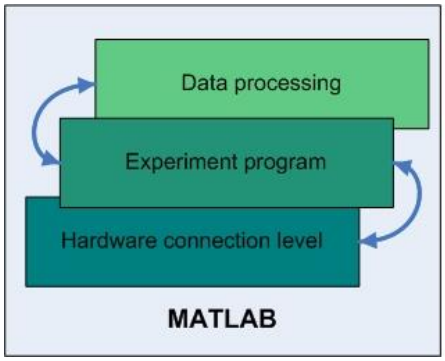

Figure 1. Methodology abstraction levels.

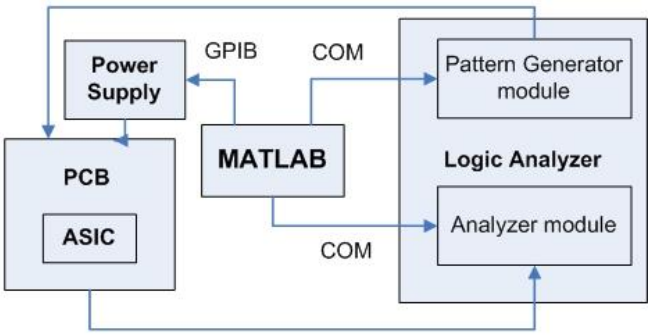

Figure 2. Setup and instrumentation control scheme.

In the intermediate level, the experiment is programmed in the script, handling the equipments automatically and performing in the correct sequence the tasks to be done. In this level, the Matlab script interfaces not only the instruments, but also the information provided by external CAD tools as, for instance, those providing the simulation results used as expected data. To do this, all the instruments are controlled as objects under Matlab. The control of the functions of the instruments is done using SCPI (Standard Commands for Programmable Instruments) commands or the specific ones for each instrument. In this way, we can configure each instrument adapting them to our setup and transfer data between the PC and the instruments.

Finally, both simulation and experimental data are processed accordingly to the operation required, made in a highest software level. If needed, further data processing is made at this level. However, a close iteration with the intermediate level is usually done to perform a complete test, for example, in a closed loop test in which the captured data is processed and converted into the new entries to the ASIC, as it will be explained in next section.

All the interactions between levels have to be done in the correct sequence, without interfering each other, to avoid error-prone automatic tests. The scheme of the methodology is shown in Figure 3.

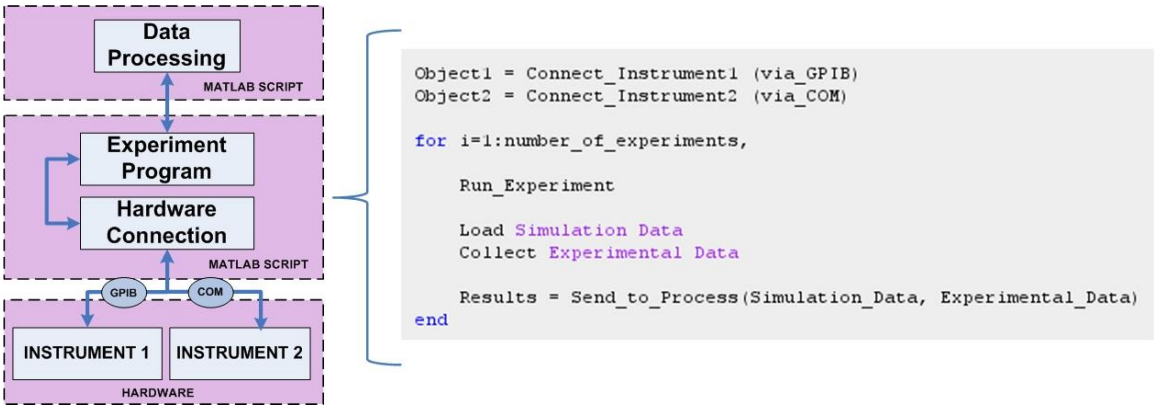

Figure 3. Methodology Scheme.

The three-level methodology can be tailored to any application example. The first level is intended for interaction and communication between the user and the controllable equipment through the Matlab Kernel. The second level contains the experiment itself. The highest level is required to process the information provided by the experiment or the CAD tools supporting the experiment. The way these mechanisms are implemented depends on the specific Matlab scripts are made as operating tools.

III. Application Example

Our methodology has been proven performing a test and characterization of 20 samples of an ASIC in a 90nm technology which implements a PWAG function. Three types of tests are carried out: one to verify the functionality of the samples (go/no-go test), another one for determining the maximum operating frequency

ISBN-10: 84-616-5438-2 | ISBN-13: 978-84-616-5438-3

(frequency test) and a final closed-loop test where the ASIC feeds a Matlab model that provides a feed-back output to the ASIC.

A. Digital ASIC

Piece-Wise Affine (PWA) functions are very useful in the design of controllers and virtual sensors for various applications. The configurable and programmable ASIC tested with this methodology implements the PWAG (Generic PWA) function. For more information about this ASIC, please refer to [4].

B. Instrumentation

In the application example, our laboratory setup needs the following equipment (Figure 2 and Figure 4):

1. Logic Analyzer Agilent 16823A, that includes the PC and the Logic Analyzer application software:
   - The PC runs Matlab controlling all the instruments. In our case, the Logic Analyzer runs a Windows Xp Service Pack 2 operating system. The Matlab script makes the connection with all the laboratory instruments, processes the output data, simulates the plant and provides the results.
   - The Logic Analyzer application software generates input patterns to the DUT (pattern generator module) and collects the output data (analyser module). A COM (Component Object Model) automation is used to communicate the Logic Analyzer environment with Matlab [7].

2. Power supply source that supplies electric power to the PCB (5V). The Matlab script controls the power supply using GPIB interface [8].

3. Digital ASIC that implements a PWAG controller (DUT).

4. PCB used as physical interface between the ASIC and the laboratory instruments. It includes special circuitry to adequate:
   - 5V from the power supply to 2.5V (ASIC pad ring voltage) and 1.2V (ASIC core voltage).
   - 3.3V input patterns provided by the Logic Analyzer's pattern generator, to 2.5V inputs needed by the ASIC.

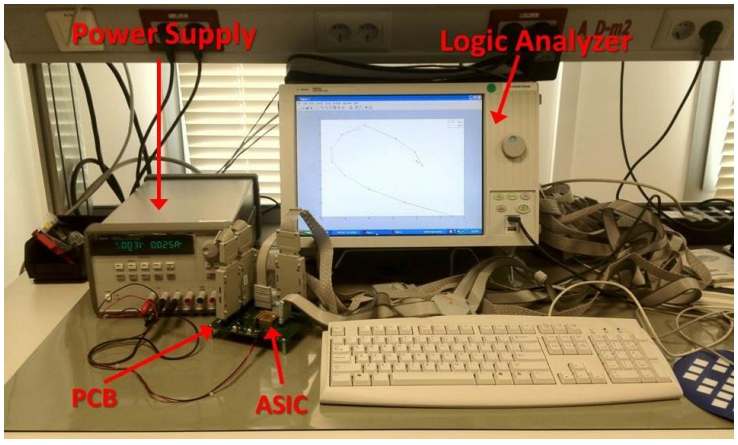

Figure 4. Experimental setup.

The proposed methodology is fully designed in a unique design environment managed by Matlab. The adaptation of the methodology to our application example implies the following steps. Before executing the Matlab script, we need some configuration files. The first one, is the ASIC configuration, obtained in our case automatically from the Moby-Dic Toolbox [9]. This toolbox also generates a Verilog file with the testbench that programs the controller. The simulation of the testbench generates automatically two CSV (Comma Separated Values) files. One includes the input patterns to the ASIC, with the exact format to directly be interpreted by the logic analyzer. The other one, includes the output simulation data to verify the experimental results. The second file, is a logic analyzer configuration file, which contains all the information of the channels used in both pattern generator and analyzer modules, clock frequency, trigger options, etc.

ISBN-10: 84-616-5438-2 | ISBN-13: 978-84-616-5438-3

At this point we can execute the Matlab script defining the test. The first step in the script is to make the connection with all the laboratory instruments creating an independent object for each one (COM objet for the logic analyzer and GPIB object for the power supply). All the instruments are going to be controlled as objects using instrument specific or SCPI commands. After fixing the correct voltage for the power supply, the script loads the logic analyzer configuration file. Then, Matlab loads the input 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 input 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 stopped. Matlab asks the analyzer module for the obtained data and processes them depending on the test that is being performed. Finally, Matlab displays the results and deletes all the created objects to avoid connection errors in future tests. This simplified process is shown in Figure 4.

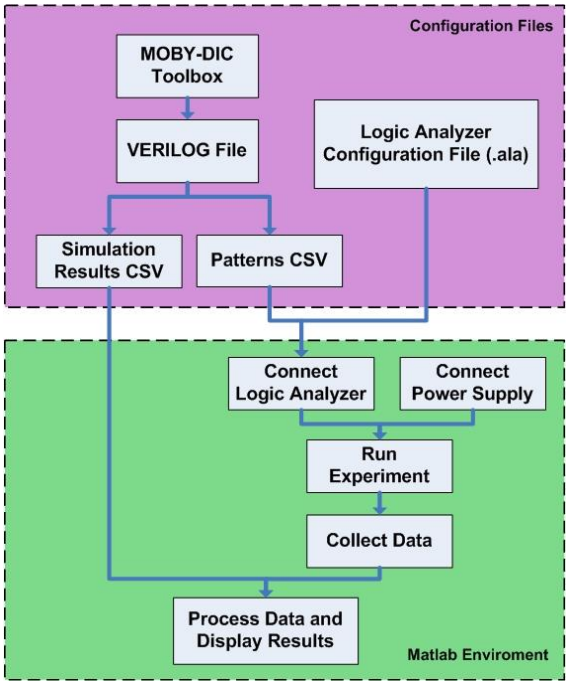

Figure 4. Application example test process scheme.

IV. Application Example Test Results

A. Go/no-go Test

The methodology can be easily adapted to most of common tests. The go/no-go test determines if the sample works as it is expected, comparing the measured results and the simulation results as expected data.

A Matlab script is created to perform the go/no-go test. First, it makes the connection with the laboratory instruments, loading the configuration file onto the logic analyzer. The CSV file containing the input patterns is loaded in the pattern generator. At the same time, in Matlab is loaded the other CSV file containing the output simulation data. After running the pattern generator module, as well as the analyzer module, the experimental data from the DUT are collected. Two matrixes are created, one called Real obtained after processing the data from the logic analyzer, and the second one called Sim, with processed data from the CSV file generated after a ModelSim simulation, used as expected data. Both matrixes are composed with the same type of data, including input data and the output data for the ASIC. The verification is satisfied when both matrixes are exactly the same (Matlab relational operator “equal to” is used). The simplified scheme of the go/no-go test is shown in figure 5.

This process can be iterated as many times as different experiments are needed for the complete go/no-go characterization of the ASIC. In our example, we consider three different experiments or case studies: a DI (Double Integrator), ACC (Adaptative Cruise Control) and a DCDC controller, obtaining a perfect matching for the 20 samples in the three case studies.

ISBN-10: 84-616-5438-2 | ISBN-13: 978-84-616-5438-3

B. Maximum Frequency Test

The maximum frequency test is performed substantially similar to the go/no-go test, except on the step of loading the logic analyzer configuration file. The determination of maximum frequency measurements has been made by using an iterative process. It can be seen as a go/no-go test, increasing in each iteration the frequency of the pattern generator.

First, the connection is established with the instruments, fixing the correct voltage to the PCB with the power supply. In this case, the loading of the logic analyzer configuration file will vary depending on the frequency, because on this test, the frequency of the pattern generator will be slightly higher in each iteration. Different .ala files are loaded containing the frequency parameter of the pattern generator. The first file contains the lowest frequency and the following will increase the frequency value gradually (2MHz increase per execution). At this point an iterative process starts, performing a go/no-go test, loading in each iteration a new configuration file. Logic analyzer configuration files are configured within a frequency range that allows to reach the maximum operating frequency of the chip. The starting frequency is provided by designers taking into account the limitations of the ASIC. Once loaded the logic analyzer configuration file, both CSV files are loaded, one to the pattern generator of the logic analyzer, and the second one on the Matlab environment. The experiment program runs both pattern generator and analyzer modules of the logic analyzer. When the whole pattern runs at least one time, Matlab asks the logic analyzer for the experimental data.

The data obtained experimentally and the one obtained from the simulation are processed in the same way. Processing the data, two matrixes are created, one called 'Real' obtained from the experimental data, and another one from the simulation data called 'Sim'. If the Real and Sim matrixes match, a new iteration starts, loading a new logic analyzer configuration file containing a slightly higher frequency. This iterative process finish when the Real and Sim matrixes mismatch, being the maximum frequency of the sample the one loaded in the previous iteration after the mismatch. All this process can be performed as many times as experiments or case of study considered. The scheme of the maximum frequency test is shown in Figure 6.

The maximum frequency obtained for each case of study after testing the 20 samples were: 107.5MHz for the DI, 96.7MHz for the ACC and 97.5MHz for the DCDC controller [4].

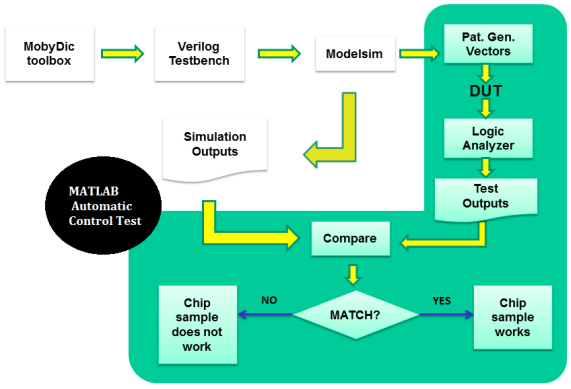

Figure 5.Go/no-go test scheme.

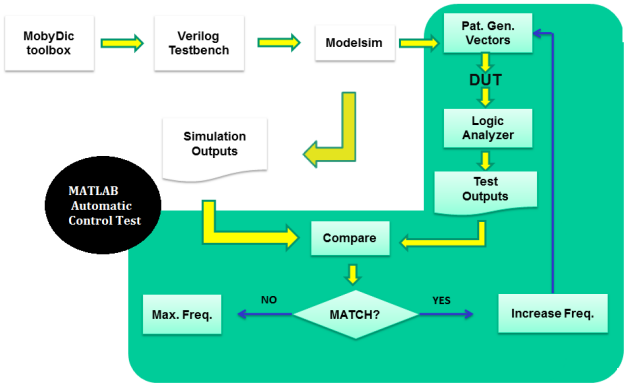

Figure 6. Frequency test scheme.

C. Close-Loop Test

The generic process of close-loop circuits and control systems implies introducing initial values to the DUT, capturing the output data of the DUT corresponding to initial data, processing it, so that the processed new values are the new input values for the DUT. The scheme for the close-loop test can be seen in Figure 7.

The close-loop test has been carried out to test the ASIC performance when it is connected to the plant to be controlled, using the DI case of study as experiment.

The steps to carry out the close loop test are the following ones. Firstly, the Matlab script makes the connection with the laboratory instruments. Then, the corresponding configuration files are loaded. After that, Matlab

ISBN-10: 84-616-5438-2 | ISBN-13: 978-84-616-5438-3

provides to the ASIC the initial values of the state of the plant. This is done loading into the pattern generator the initial values, and running the logic analyzer modules. The output provided by the ASIC on the first iteration, which is acquired by the logic analyzer, is processed by the plant model described in Matlab. The plant model calculates the new input values for the ASIC and a new iteration starts providing them to the ASIC. This operation can be repeated as many times as desired by the test.

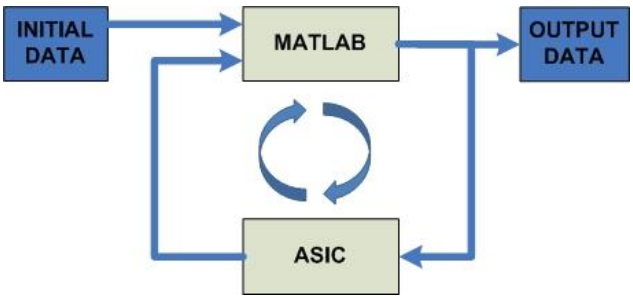

Figure 7.Close-loop test scheme.

The behavior obtained in the close-loop operation of the ASIC was exactly as expected [10].

V. Conclusions

This paper has presented a three-level methodology to automatically test and characterize digital ASICs. The adaptation of the methodology to an application example with experimental results is shown. In our specific case, the methodology has been used for the validation and determination of maximum frequency in an open-loop and closed-loop configuration of 20 samples of a PWAG controller. It is fully automatized within a unique framework managed by Matlab, saving test time (from weeks to days) and reducing the risk of manual measurement errors. Our methodology does not require special laboratory equipment, and can be carried out with common controllable laboratory instruments. The methodology can be applied and extended to different experiments, ASIC features and characteristics, and controllable equipments.

Acknowledgments

This work was partially supported by the Spanish Government under CITIES project TEC2010-16870, by European Community under the MOBY-DIC Project FP7-IST-248858 (www.mobydic-project.eu), and by Junta de Andalucía under the CRIPTOBIO Project P08-TIC-03674 (with support from the PO FEDER-FSE).

References

[1] L.Mostardini, L.Bacciarelli, L.Fanucci, L.Bertini, M.Tonarelli, M. Marinis, "FPGA-based Low-cost Automatic Test Equipment for Digital Integrated Circuits", *IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, 21-23 September 2009, Rende (Cosenza), Italy.
[2] http://www.teseda.com/
[3] http://www.mathworks.es/
[4] P. Brox, J. Castro, 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*", Accepted IEEE Transactions on Circuits, and Systems Part-I, 2013.
[5] http://www.microsoft.com/com/default.mspx
[6] *IEEE Standard Digital Interface for Programmable Instrumentation*, Institute of Electrical and Electronics Engineers, 1987, ISBN 471-62222-2, ANSI/IEEE Std 488.1-1987
[7] Agilent 16823A Com Automation Manual, 2008.
[8] HP E3631A USER MANUAL, 2000.
[9] MOBY-DIC Toolbox user manual, June 2012. [Online]. Available:http://ncas.dibe.unige.it/software/MOBY-DIC_Toolbox/MOBYDIC_Toolbox_Documentation/doc/html/index.html.
[10]M.C.Martínez-Rodríguez, P.Brox, J.Castro, E.Tena, A.J.Acosta, I.Baturone, "*ASIC-in-the-loop methodology for verification of piecewise affine controllers*", Proc. IEEE Int. Conf. on Electronics, Circuits, and Systems (ICECS'2012), Sevilla, Spain, Dec. 9-12, 2012.