EasyChair Preprint

№ 2194

An Application-specific Instruction Set Processorfor Microgrid Simulation

Edgar Mauricio Brenes and Carlos Meza Benavides

December 18, 2019

An Application-specific Instruction Set Processor for Microgrid Simulation

Edgar Mauricio Brenes
Aruba Networks
Hewlett Packard Enterprise
Heredia, Costa Rica
Email: edgar.brenes@hpe.com

Carlos Meza
Electronic Engineering School
Instituto Tecnologico de Costa Rica
Cartago, Costa Rica
Email: cmeza@itcr.ac.cr

***Abstract*—Microgrid represents a new paradigm in the power sector that offers more reliability and flexibility for electricity delivery. A microgrid consists of different types of power generation units, loads and energy storage systems that are controlled and coordinated. Smart power processing units play an important role in a microgrid. The present paper presents an application-specific instruction set processor for the study of power converters operating in isolated microgrids. The proposed processor has been developed in order to solve differential algebraic equations that describe the dynamical behavior of microgrid power processor. Due to the inherently parallelism of the proposed processor the simulation time is reduced considerably with respect to a general purpose processor.**

***Index Terms*—microgrid, off-grid system, photovoltic system**

I. Introduction

A microgrid is a subsystem of a larger electrical grid which consists of a collection of distributed power generators and loads located in a nearby area [1]. A typical diagram of a microgrid is depicted in Figure 1.

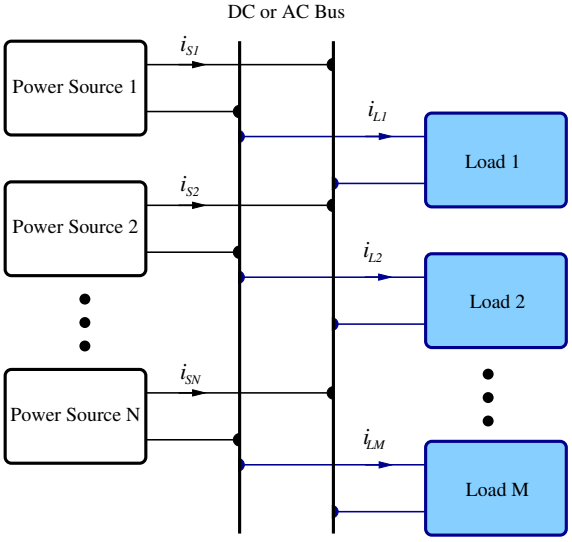

Figure 1. Diagram of a typical microgrid

One of the biggest challenges in the design and operation of microgrids is that they are inherently asymmetrical and heterogeneous [2], i.e., they present several different types of microgenerators and loads. Additionally, it is also desired that a microgrid continues operating with the loss of any component and/or any generator (plug-and-play capability) [1]. Another important challenge in plug-and-play microgrid is to have a dynamic fast-acting load balancing system. A microgrid can operate connected to the utility grid or as a stand-alone system. This paper deals only with stand-alone microgrids.

Smart power processing devices (e.g. power inverters and converters) can provide the necessary functionality to deal with the aforementioned challenges. Such power processing units are considered intertialess [2] and can potentially intervene to correct a failure or imbalance almost instantly. A multi-level hierarchical management and control system is able to operate the power processing units in a coordinated way [3].

Giving the importance of power processing units in microgrids it is useful to study in detail their interaction in a microgrid. A simulation environment that is able to accomplish the aforementioned should make explicit the behavior of the system in a wide range of time scales, i.e., it should be able to simulate the dynamics of the system from miliseconds to hours or even days. Large processing power is required in order to simulate the system in the required time scales considering also the inherently complexity and nonlinearity of the microgrid.

The cost reduction and performance increment of field-programmable gate arrays (FPGA) in the last years [4] have make it possible to design and implement application-specific instruction set processor (ASIP) that can be used for a wide range of applications such as video processing [5], [6], advanced encryption [7], power monitoring [8] and monitoring seismic activity [9]. The advantages offered by ASIPs is that they can be fine tunned to perform specific tasks more efficiently and faster than general purpose processors.

In this regards, the present work proposes an ASIP designed to be the calculation core of a simulation environment for microgrids. This proposed system is able to simulate concurrently several microgenerators and loads in a few milliseconds, solving several non-linear differential algebraic equations.

The rest of the paper is structured as follows: First, a simple stand-alone microgrid structure is presented. This microgrid topology will be used for the design and evaluation of the ASIP-based simulation environment. In section III-A the ar-

chitecture of the ASIP is described. Next, a simple microgrid case of study is presented and simulated using the proposed simulation environment. Finally, section V presents the main conclusions and the future activities of this work.

II. A Stand Alone Microgrid Structure

The proposed ASIP and simulation environment have been designed based on a specific microgrid structure. The idea behind this approach is to validate the usefulness of an ASIP-based simulation environment for a simple microgrid configuration so that it can be later extended to more complex microgrid topologies.

The microgrid structure used in this work consists of n power sources connected in parallel to an alternate current (AC) bus, as depicted in Figure 2. Each power source (PS) injects current to the AC bus through a power unit (PU) that is controlled by a local current controller (LCC). The current reference for each LCC is provided by a global current controller (GCC) as seen in Figure 2. Notice that by having the GCC define the current references of each LCC, ($i^*_{S_i}$), all the currents generated by the power sources, i_{S_i} will be in phase. More specifically, the GCC is in charge of:

- identifying the total current required by the load such that the AC bus voltage, v_{AC}, is equal to a reference value, v^*_{AC},
- determining the power contribution of each power source to the load by providing the desired value of the current (i_{S_i}) that each power source needs to inject to the AC bus.

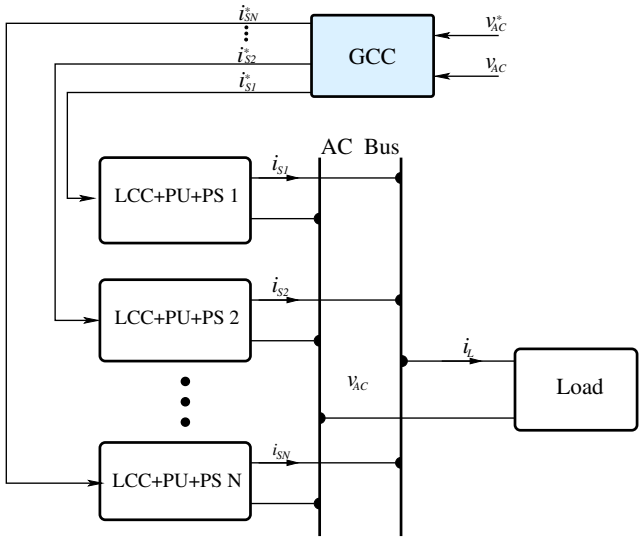

Figure 2. Structure of an Isolated Microgrid. GCC=Global Current Controller, LCC=Local current controller, PU=Power Unit, PS=Power source

As seen in Figure 2 each power source is associated to a power unit and a local current controller (LCC+PU+PS). A more detailed representation of this subsystem is shown in Figure 3, where the power source is a photovoltaic generator and the power unit is a full-bridge inverter. The LCC provides the required signals, μ_{1_i}, μ_{2_i}, μ_{3_i}, μ_{4_i}, to turn on the MOSFETS such that the full-bridge output current, i_{S_i}, is equal to the reference value, $i^*_{S_i}$. This subsystem can be modeled according to the following set of differential equations

$$C_i \frac{dv_{C_i}}{dt} = i_{\mathrm{pv}_i} - u_i i_{S_i} \tag{1}$$

$$L_i \frac{di_{S_i}}{dt} = u_i v_{C_i} - v_{AC} \tag{2}$$

here C_i, L_i represent the capacitance and inductance, respectively, of the i-th PVG+Full bridge unit. $u_i \in \{1, -1\}$ is the output of the LCC used to generate μ_{1i}, μ_{2i}, μ_{3i}, μ_{4i} in the following way

$$u_i = 1 \Leftrightarrow \mu_{1i} = \mu_{4i} = 1 \Leftrightarrow \mu_{2i} = \mu_{3i} = 0 \tag{3}$$

$$u_i = -1 \Leftrightarrow \mu_{1i} = \mu_{4i} = 0 \Leftrightarrow \mu_{2i} = \mu_{3i} = 1 \tag{4}$$

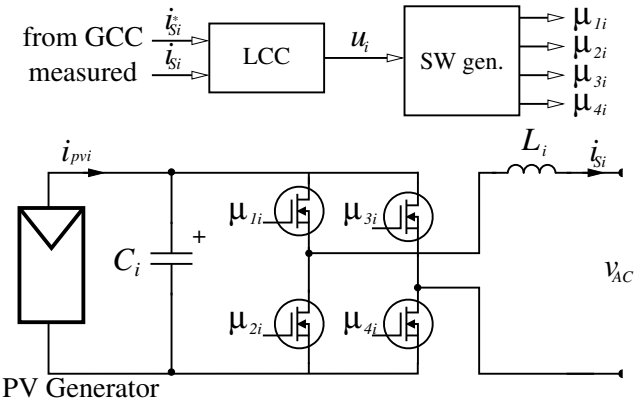

Figure 3. PV Power Source Diagram

Each photovoltaic generator (PVG) represents a string of r PV modules with the electrical characteristics of Fig.4 which can also be described as follows

$$i_{pvi} = I_{gi} - I_{si}\left(\mathrm{e}^{\alpha_i v_{pvi}} - 1\right) \tag{5}$$

where i_{pvi} and v_{pvi} are the current generated and the voltage at the i-th PV generator and I_{gi}, I_{si} and α_i are its parameters.

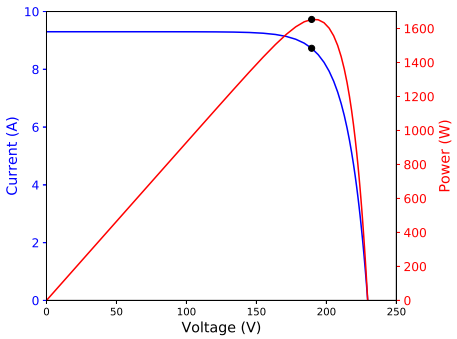

Figure 4. i-v and P-v curves of a PV Generator

As mentioned previously, the control scheme, i.e., the GCC and LCC, is in charge of making the current delivered to the grid such that the AC bus voltage is sinusoidal with constant frequency and amplitude. Moreover, the AC bus voltage can be defined as a function of the delivered current, i.e., $v_{AC} = f(i_L)$. In this regard, the control scheme must be designed considering $f(i_L)$. In this work, the load is modeled as a resistor, i.e.,

$$v_{AC} = R_L \sum_{i=1}^{i=n} i_{S_i} \tag{6}$$

Nevertheless, modifying (6) and redesigning the controller it is possible to consider reactive and/or nonlinear loads.

Notice that the proposed case of study can be simulated solving a set of non-linear differential algebraic equations (DAE), i.e., equations (1) to (6). In this case the order of the DAE will be $2n$, where n is the number of PVG+Full-bridges presented in the system.

III. ASIP-based Simulation Environment

The proposed simulation environment consists of an ASIP, and ARM processor and general purpose computer. The ASIP is controlled by an ARM processor, which is connected to a PC for interfacing with the user. A general diagram of the system is shown in figure Fig 5. Notice that the ARM processor and the ASIP can be easily integrated in a single silicon device, i.e., a field-programmable gate array (FPGA) or an application specific integrated circuit (ASIC). The ARM

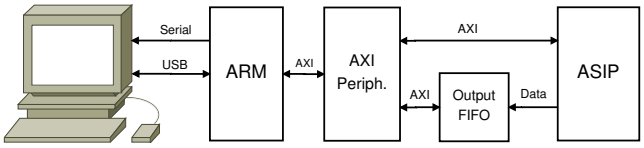

Figure 5. System Architecture

processor sends to the ASIP the simulation program to be executed and controls the execution of the program. The ASIP program memory and control registers are memory-mapped in the ARM memory space. The ARM processor also polls the Output FIFO and if there is data available it pops the data from the FIFO and sends it to the computer. Communication between ARM processor and ASIP occurs through an AXI interface. The ASIP executes the program which has the instructions to generate the simulation data, it also sends the data to the Output FIFO when the print instruction is executed.

The ASIP communicates with a general purpose computer to transmit all the simulated data for detailed data analysis and visualization. The communication is also required so that the computer can download the simulation program that will be executed in the ASIP. The computer program includes the information about the configuration of the microgrid such as value of the load, the number of active energy sources and the photo-generated current in each PV power source unit.

A. The ASIP Architecture

The ASIP architecture is shown in figure Fig 6. The architecture allows execution of mathematical operations concurrently in order to solve equations (1),(2) and (6) in a short period of time. Operations are executed using IEEE 754 double precision floating point format. This number representation was preferred over fixed point because operands in (5) can be of order -12 and +12, needing at least 40 bits for integer value and 40 bits for fractional value.

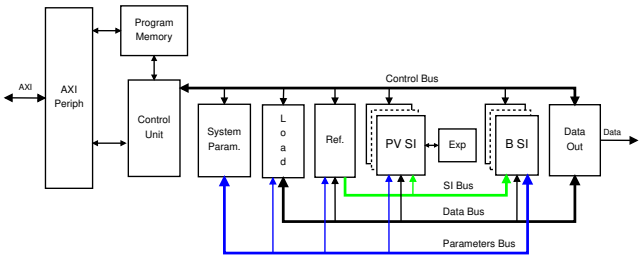

Figure 6. ASIP Architecture

The following subsections describe the main modules of the ASIP.

1) Control unit: The control unit fetches instructions from the Program Memory and executes them. This module communicates with other modules of the processor through a data bus and a control bus. The data bus is used to read and write parameters in each module. The control bus is used by the control unit to indicate a module when it has to execute an operation and to be notified when a module has complete it. There is one dedicated start signal for each module in the processor so the control unit can indicate to different modules to start an operation in the same clock cycle. The simulation program instructions are stored in the Program Memory by the ARM processor through an AXI interface. This memory is mapped in the ARM memory space so an access to an address in that memory region will access the Program Memory directly.

2) PV Source Inverter and Controller: The photovoltaic Source Inverter module (PVSI) calculates the output current of a photovoltaic generator and the voltage on its internal capacitor by solving equations (1) and (2). This unit implements in the ASIP the LCC+PU+PS depicted in Figure 3. In Figure 7 the structure of the PVSI implemented in the ASIP is shown.

Submodule VC Integral solves equation (1) and submodule IL Integral solves equation (2). Each of these two modules have a floating point multiplier, a floating point adder, and a finite state machine with the corresponding sequence of mathematical calculations and data movement to solve the equations.

PVSI has a finite states machine (FSM) to control the other submodules. When the ASIP Control Unit asserts the start signal the FSM triggers execution of mathematical operations to get values of output current and voltage capacitor in the photovoltaic generator, these two operations are executed in

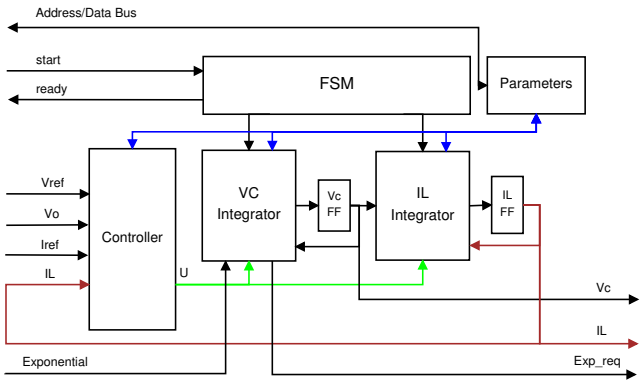

Figure 7. PVSI Block Diagram

parallel. When both calculations are complete the FSM asserts the ready signal indicating to the ASIP Control Unit that a calculation cycle has complete.

The parameter module in PVSI store the values specific to a photovoltaic generator that are needed for the mathematical calculations, these values are: capacitance, inductance, reverse saturation current, initial voltage in capacitor, initial current in inductor, and number of solar cells in the photovoltaic generator. These values are written by the ASIP Control Unit through the data bus.

The controller in PVSI determines the value of u which is used to control the inverter switches as shown in Figure 3. It has two modes of operation which can be selected through a PVSI parameter: (1) based on reference voltage or (1) based on current reference. The control signal u is then defined according to the set reference value.

Additionally, the number of PVSI modules in the ASIP is determined by a parameter, so it can be changed at synthesis time.

3) Battery Source Inverter and Controller: Battery Source Inverter and Controller unit (BSI) calculates the output current of a battery and an inverter. It works similar to a PVSI with the difference that it does not include VC Integrator submodule as the voltage is provided by the battery.

4) Exponential Unit: Exponential module receives a floating point value and returns its exponential value also in floating point format. Exponential value is needed by the VC Integrator in PVSI module, as shown in Fig 7, to calculate the output current (5). In order to reduce simulation time a set of lookup tables were implemented to generate the exponential value of a given number. There are different lookup tables for different operation ranges. When this unit receives a request it converts the operand to an integer and a fractional number, then depending on the values this unit extracts the exponential value from the corresponding lookup table. Given that this module takes a lot of resources there is only one exponential unit shared by all the PVSI units, so an arbitration method was implemented to return the corresponding exponential value to the different requesters.

5) Reference Unit: The Reference Unit generates the voltage of reference or the current of reference used by the PVSI and BSI units to calculate their corresponding value for u. This unit implements the GCC depicted in Figure 2. The generated voltage of reference is a sinusoidal waveform with a peak amplitude of 170V or 120V RMS. All PVSI and BSI units receive the same voltage of reference and each of them generates the corresponding value of u in order to follow this input voltage. This unit can also be configured to generate a different current of reference for each PVSI and BSI units. Based on the configured parameters this module calculates the current that each source have to inject

6) Load Unit: Load unit calculates voltage on the load by adding the current generated by all the energy sources and multiplying it by the value of the load as described by (6).

7) Instruction Set: The ASIP is controlled by instructions stored in the Program Memory. Instructions are one, two or three words long, where every word is 64 bits width. Every instruction is composed by an eight bits operation code and the arguments. Supported instructions are shown in Table I

Table I
ASIP INSTRUCTIONS

Instruction	Mnemonic	Arguments
Write parameter	WR	address, data
Read parameter	RD	address
Set cycles counter	SETC	number of cycles
Decrement and jump if not zero	DJNZ	address offset
Enable source	ENAB	modules
Execute calculation	EXEC	modules
Wait calculation	WAIT	modules
Print data	PRINT	index
Stop program execution	STOP	

IV. VALIDATION AND SIMULATION RESULTS

The ASIP and ARM processor were implemented in a FPGA Xilinx Zynq-7000 AP SoC XC7Z020-CLG484. Table II shows the resources utilized when synthesizing the ASIP with eight PVSI units and one BSI unit.

Table II
FPGA RESOURCES UTILIZATION

Resource	Utilization
Slice LUTs	43056 (81%)
Slice Registers	25088 (24%)
RAM Blocks	133 (26%)
DSPs	18 (8%)

A typical program that the ASIP processor runs is shown in Figure 8.

A. Validating the proposed simulation environment

In order to validate the developed ASIP-based simulation environment two cases have been tested. Such cases were simulated in the proposed environment and in a Python program

```
WR 0x00, 0x00, 0x4034000000000000
// Circuit load = 20 Ohms
WR 0x00, 0x01, 0x3eb0c6f7a0b5ed8d
// delta t = 1us
WR 0x00, 0x03, 0x0000000000000009
// sampling = 1 every 10 cycles
WR 0x02, 0x00, 0x4069000000000000
// PVSI0 Capacitor = 5mF (1/200)
WR 0x02, 0x01, 0x403546ce01000201
// PVSI0 Inductor = 47mH (1/21.2766)
WR 0x02, 0x02, 0x402299999999999a
// PVSI0 Photocurrent = 9.3A
WR 0x02, 0x03, 0x4070e00000000000
// PVSI0 Initical Vc = 270V
WR 0x02, 0x04, 0x0000000000000000
// PVSI0 Initial IL = 0A
WR 0x02, 0x05, 0x3eb21e908ed8f651
// Dark sat. curr. = 1,08uA
WR 0x02, 0x06, 0x3fadddddddddddde
// q/kTn = 0,05833
WR 0x0A, 0x01, 0x403546ce01000201
// Bat0 Inductor = 47mH (1/21.2766)
WR 0x0A, 0x03, 0x4070e00000000000
// Bat0 Vbat = 370V
WR 0x0A, 0x04, 0x0000000000000000
// Bat0 Initial IL = 0A
ENAB 0x00000101 // Enable PVSI0 and Bat0
SETC 0x0186A0 // Cycles counter = 100.000
EXEC 0x0404     // Calc PVSI0 & Bat0 IL
WAIT 0x0404     // Wait calculation
EXEC 0x0002     // Calculate load voltage
WAIT 0x0002     // Wait calculation
PRINT 0x02      // Print V load
PRINT 0x03      // Print V reference
PRINT 0x04      // Print PVSI0 IL
PRINT 0x05      // Print PVSI0 Vc
PRINT 0x14      // Print Bat0 IL
PRINT 0x15      // Print Bat0 Vbat
DJNZ  0x0F      // Dec and jump if not zero
STOP            // Stop simulation
```

Figure 8. ASIP Program for Simulation Case A (Table III)

running in a general purpose processor (GPP) of a typical personal computer.

The general purpose processor used was the Intel(R) Core(TM) i5-4200M CPU. The differential equations that describe the system were simulated in the GPP with Python 2.7 executed in a machine with Fedora Linux.

The details of the cases simulated in the aforementioned platforms are shown in Table III. Each PV panel has the electrical characteristics shown in Fig. 4.

Figures 9 and 10 show the results of Case A. The system simulated in GPP solved the differential equations using the

Table III
PARAMETERS OF THE CASE SIMULATED

Case	Number of PV Generators	Composition of PVG	Load
A	1	1 string with 6 PV panels	20 Ω
B	4	2 strings with 6 PV panels and 2 strings with 5 PV panels	20 Ω

Runge-Kutta method of the Odespy Pythons library [10]. Figures 11 and 12 show the results of Case B. As mentioned previously, Case B represents a microgrid with 4 PV power sources, where each power source is associated with a full-bridge converter. The aforementioned figures shown that the proposed FPGA-based simulation environment generates identical responses as a conventional software based simulation.

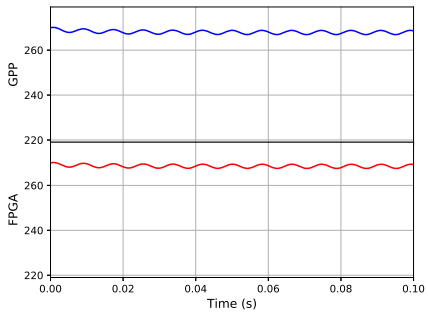

Figure 9. Simulation results: voltage in the PVG for Case A using a FPGA-based simulation environment and a conventional general purpose processor.

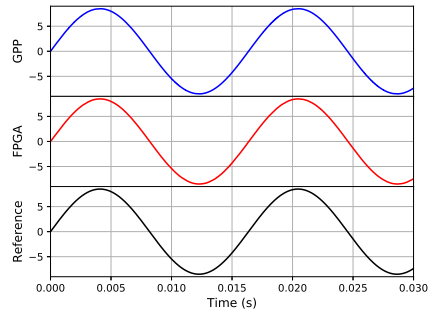

Figure 10. Simulation results: output current for Case A using a FPGA-based simulation environment and a conventional general purpose processor.

B. Comparing the simulation time

In order to compare the execution time of both simulation a microgrid with 1, 2, 4, 8 and 16 photovoltaic power sources were simulated. For these cases, both simulation systems solved the differential equations using the Forward-Euler method. The simulation time for each case are shown in Table IV, as expected the FPGA-based simulation environment is around 10 times faster than the GPP. Additionally, as shown

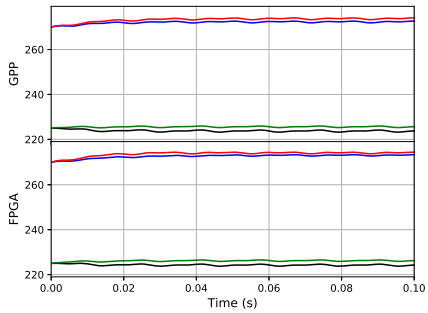

Figure 11. Simulation results: voltage in each PVG for Case B using a FPGA-based simulation environment and a conventional general purpose processor.

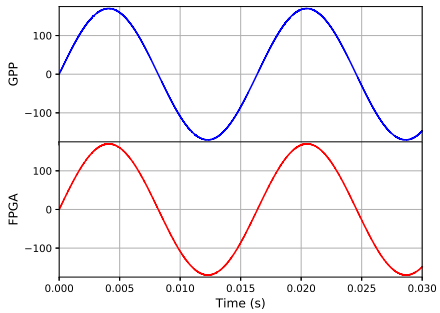

Figure 12. Simulation results: voltage at the load for Case B using a FPGA-based simulation environment and a conventional general purpose processor.

in 13, for the FPGA-based system, the simulation time grows with respect to the quantity of PV generators with a smaller slope than the GPP environment.

Table IV
SIMULATION TIME

PVSIs	GPP (ms)	FPGA (ms)
1	956	130
2	1179	142
4	1475	167
8	2242	219
16	3352	320

V. CONCLUSIONS

A dynamical model useful for the analysis of power processing units that operates in microgrid has been presented. For a proof-of-concept test such model has been implemented in a FPGA-based simulation environment. The simulation results with the proposed FPGA-based systems have been proved to be identical than those obtained with a high level simulation language using a high-order numerical method. Finally, due to its concurrent nature, the FPGA-based system is able to execute the simulation in a much lower time than a General Purpose Process. The simulation time of the proposed system

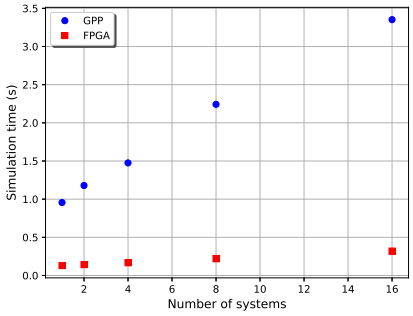

Figure 13. Simulation Circuit Block Diagram

is less dependent on the amount of energy sources considered. The results of the present paper suggests that FPGA-based systems are suitable environments for the simulation of microgrid systems with several energy sources and loads of different types.

REFERENCES

[1] R. H. Lasseter and P. Paigi, "Microgrid: A conceptual solution," in *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, vol. 6. IEEE, 2004, pp. 4285–4290.

[2] A. S. Meliopoulos, "Challenges in simulation and design of/spl mu/-grids," in *Power Engineering Society Winter Meeting, 2002. IEEE*, vol. 1. IEEE, 2002, pp. 309–314.

[3] A. Merabet, K. T. Ahmed, H. Ibrahim, R. Beguenane, and A. M. Ghias, "Energy management and control system for laboratory scale microgrid based wind-pv-battery," *IEEE Transactions on Sustainable Energy*, vol. 8, no. 1, pp. 145–154, 2017.

[4] L. Guan, "Fpga and digital signal processing," in *FPGA-based Digital Convolution for Wireless Applications*. Springer, 2017, pp. 5–23.

[5] H. Peters, R. Sethuraman, A. Beric, P. Meuwissen, S. Balakrishnan, C. A. A. Pinto, W. Kruijtzer, F. Ernst, G. Alkadi, J. Van Meerbergen *et al.*, "Application specific instruction-set processor template for motion estimation in video applications," *IEEE transactions on circuits and systems for video technology*, vol. 15, no. 4, pp. 508–527, 2005.

[6] J. Antikainen, P. Salmela, O. Silvén, M. Juntti, J. Takala, and M. Myllylä, "Application-specific instruction set processor implementation of list sphere detector," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 054173, 2008.

[7] T. Good and M. Benaissa, "Very small fpga application-specific instruction processor for aes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 7, pp. 1477–1486, 2006.

[8] S. Vaas, M. Reichenbach, and D. Fey, "An application-specific instruction set processor for power quality monitoring," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 2016, pp. 181–188.

[9] D. Bora, B. Thomas, S. Nandi, and G. Trivedi, "Application specific processor design implementation to monitor seismic activity," in *Accessibility to Digital World (ICADW), 2016 International Conference on*. IEEE, 2016, pp. 9–14.

[10] "The odespy software package," https://github.com/hplgit/odespy, accessed: 2018-01-14.

ACKNOWLEDGMENT

This paper has been financed by Project 1360042 *"Diseño de Arquitecturas Multinúcleo para Aplicaciones de Procesamiento Masivo de Datos"* of the Costa Rica Institute of Technology.