Open access • Journal Article • DOI:10.1109/TII.2016.2532118

A Portable Implementation on Industrial Devices of a Predictive Controller Using Graphical Programming — Source link

Silviu Folea, George Mois, Cristina I. Muresan, Liviu Miclea ...+2 more authors

Institutions: Technical University of Cluj-Napoca, Ghent University, Anglia Ruskin University

Published on: 18 Feb 2016 - IEEE Transactions on Industrial Informatics (IEEE)

Topics: Control theory, Visual programming language and Model predictive control

A Portable Implementation on Industrial Devices of a Predictive Controller Using Graphical Programming

Silviu C. Folea, *Member, IEEE*, George Moiş, *Member, IEEE*, Cristina I. Muresan, Liviu Miclea, *Member, IEEE*, Robain De Keyser, and Marcian N. Cirstea, *Senior Member, IEEE*

***Abstract*—This paper presents an approach for developing an extended prediction self-adaptive controller employing graphical programming of industrial standard devices for controlling fast processes. For comparison purposes, the algorithm has been implemented on three different field-programmable gate arrays (FPGAs) chips. This paper presents research aspects regarding graphical-programming controller design, showing that a single advanced control application can run on different targets without requiring significant program modifications. Based on the time needed for processing the control signal and on the application, one can efficiently and easily select the most appropriate device. To exemplify the procedure, a conclusive case study is presented.**

***Index Terms*—Benchmark testing, field-programmable gate arrays (FPGAs), predictive control, real-time systems.**

I. Introduction

PREDICTIVE control has been used successfully in control applications in all fields of industrial activity, a fact that has triggered an increasing interest in the methodology during the last decade. The choice for predictive control, rather than other modern control concepts, is based on some series of important benefits such as its intuitive principles, performance-oriented design parameters, the ability to handle nonlinearities, and its capability of taking into account various constraints (such as actuator constraints, safety constraints, and quality constraints). Typically, predictive control has been used in the control of slow dynamics processes, such as thermal and chemical plants [1]. However, more and more model-predictive control (MPC) applications are directed toward dynamical systems with fast response times [2]–[4]. Until recently, the main MPC limitation resulted from the long computational time needed for performing the optimization. The use of digital signal processors (DSPs) and field-programmable gate arrays (FPGAs) led to the reduction of the time needed for solving the constrained optimization problem with a period of tens or hundreds of microseconds [5], [6]. The large real-time computational complexity was managed until now using industrial computers, and this paper presents results especially obtained from personal computer implementations.

The purpose of this paper is to present an efficient and robust control solution for fast dynamic systems, using FPGA devices and the LabVIEW graphical-programming environment. The motivation for using this solution is based on the fact that compared to hardware description languages, such as very high speed integrated circuit hardware description language (VHDL), graphical programming is a more user-friendly configuration environment and offers a very short project-development time [7], [8]. An application for a dc motor was chosen for validation and testing purposes. The dc motor supports a wide range of command rates and of execution time variations, without being damaged or broken. The particular application here refers to the control of the dc motor, as a part of the vacuum pumps used to maintain an efficient thermal isolation in the vacuum jacket of a train of three carbon isotopes separation columns. The efficiency of the isotope-separation process, occurring at very low temperatures, is strongly dependent upon a strict operation of these vacuum pumps. These need to be carefully controlled since a failure of the vacuum leads to the compromise of the entire separation process [9]. The efficiency of the proposed solution was highlighted by comparing it to several different implementations, a PC-based control system, one implemented on a real-time target, and one on an advanced RISC machine (ARM) microcontroller, all of them running the controller. Finally, the same predictive-control algorithm was implemented on three different FPGA chips: a Virtex-II, a Spartan-6, and a Zynq. The comparison between the three systems has shown that this type of complex algorithms can operate on cheaper FPGA chips, such as the Spartan-6 and Zynq, achieving not only the same levels of computational performance as their more complex and more expensive counterparts but also important power savings. This comparison

Manuscript received July 21, 2015; revised November 13, 2015; accepted January 26, 2016. Date of publication February 18, 2016; date of current version March 29, 2016. This work was supported by the Romanian National Authority for Scientific Research National Council for Development and Innovation - Executive Agency for Higher Education, Research, Development and Innovation (CNDI-UEFISCDI) under Project PCCA 155/2012. Paper no. TII-15-1090.

S. C. Folea, G. Moiş, C. I. Muresan, and L. Miclea are with the Department of Automation, Technical University of Cluj-Napoca, Cluj-Napoca 400114, Romania (e-mail: Silviu.Folea@aut.utcluj.ro; George.Mois@aut.utcluj.ro; Cristina.Pop@aut.utcluj.ro; Liviu.Miclea@aut.utcluj.ro).

R. De Keyser is with the Department of Electrical Energy, Systems and Automation, Ghent University, Gent 9052, Belgium (e-mail: Robain.DeKeyser@UGent.be).

M. N. Cirstea is with the Department of Computing and Technology, Anglia Ruskin University, Cambridge, CB1 1PT, U.K. (e-mail: Marcian@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2016.2532118

regarding the implementation on various FPGA targets and microcontrollers represents one of the main contributions and original elements in comparison to previous research [10], also including comparative tables and benchmarks for resource allocation in FPGAs. This paper also presents the diagrams implemented using LabVIEW.

The choice of using an FPGA instead of a processor-based solution was motivated by several advantages. FPGAs have already been used in industrial control systems, being capable of providing an increased level of performance, while at the same time reducing the cost, size, and power consumption of the actual implementation and improving reliability [11]–[15]. The ever-increasing sophisticated control algorithms can take advantage of the natural parallelism and increased resource density of the FPGA chips [16]. Thus, complex architectures, fully dedicated to the control algorithm to implement, can be developed [17]. The design and real-time implementation of control loops running at frequencies above 1 MHz is now possible with the use of these system-on-chip digital reconfigurable platforms. Although still more expensive than DSPs and microcontrollers, they compensate through their compactness, all the building parts of the competitor solutions (CPU, RAM, bus) being placed inside a single capsule. While DSPs are aimed at implementing signal-processing applications and can perform large amounts of computations, FPGA chips offer higher flexibility levels and transfer the printed circuit board (PCB) complexity inside the device, on-chip. The work in [18] presents a systematic comparison between these two technologies along with their main advantages and drawbacks when used in control applications. FPGAs also provide the possibility of in-the-field programming, which allows the addition of other features to the controller and the implementation of further data postprocessing algorithms. They can also be dynamically reconfigured, enabling the controller to adapt to the needs of the plant. Thus, adaptation to changes in environmental conditions becomes possible.

A wide range of applications in the field of electrical systems employ FPGAs [19]–[21]. The authors in [19] developed a reliable low-complexity reusable digital controller, by using an FPGA implementation. The work in [20] presents an FPGA-based adaptive digital PI controller and emphasizes the advantages provided by FPGAs in the control of complex industrial processes. MPC was addressed for the control of power converters [22] and electric drives [23], FPGA-based solutions showing good control performance [24], [25].

This paper is structured as follows. Section II describes the extended prediction self-adaptive control (EPSAC) principles, while Section III shows the steps completed for finalizing the EPSAC design and the methodology used for the FPGA implementation. Then, Section IV provides information regarding the details of the hardware and of the software setup. The testing and validation of the proposed solution along its performance evaluation are synthesized in the Section V, and finally, the concluding remarks are outlined in Section VI.

II. EPSAC Control Principles

The EPSAC methodology is a typical member of the model-based predictive-control (MBPC) family. MBPC is a type of control which uses an online process model (in the control computer) for calculating predictions of the future plant output and for optimizing future control actions. The two key principles of MBPC consist in the explicit online use of the process model for forecasting the process output at future time instants and in the calculation of an optimal control action based on the minimization of a cost function [26]. The principle of the EPSAC control, presented in [26], is based on the minimization of the error between the specified reference trajectory and a future predicted process output. A cost function having the form

$$\sum_{k=N_1}^{N_2}\left[r(t+k/t)-y(t+k/t)\right]^2+\lambda\sum_{k=0}^{N_u-1}\left[\Delta u(t+k/t)\right]^2 \tag{1}$$

will be minimized. The design parameters of the cost function are N_2—the maximum prediction horizon, N_u—the control horizon, N_1—the minimum prediction horizon, λ—the weight parameter, $y(t)$—the (measured) process output, $u(t)$—the process input, and $r(t)$—the reference trajectory. The control signal in (1) is given by

$$\Delta u(t+k/t)=u(t+k/t)-u(t+k-1/t), \text{with} \tag{2}$$

$$\Delta u(t+k/t)\equiv 0, \quad \text{for } k\geq N_u. \tag{3}$$

For minimizing (1), the choice of N_2 and N_1 plays an important role, as well as the estimation of the process output, $y(t)$, over the prediction horizon N_1 to N_2. In the EPSAC approach, the prediction of the process output is done based on previous measurements of the process output and input signal, as well as some future values of the input signal.

For predicting the output, the generic model in (4) can be used

$$y(t+k/t)=x(t+k/t)+n(t+k/t) \tag{4}$$

where $x(t)$ represents the process model output, while $n(t)$ is the process/model disturbance.

To predict the process output $y(t)$, $x(t+k|t)$ is computed based on an existing model of the process, while $n(t+k|t)$ is predicted using filtering techniques.

Assuming the process model for a single-input-single-output system is given by

$$x(t)=\frac{B(q^{-1})}{A(q^{-1})}u(t) \tag{5}$$

the output model x may be predicted k samples ahead using previous values of the process model and of the control input u, considering that polynomials $B(q^{-1})$ and $A(q^{-1})$ in (5) are fully known.

The algorithm for computing the control signal required to minimize (1) uses also the concepts of free and forced response

$$y(t+k/t)=y_{\text{free}}(t+k/t)+y_{\text{forced}}(t+k/t). \tag{6}$$

The component $y_{\text{free}}(t+k/t)$ can be easily computed using (4), by simply putting $u(t/t)=\cdots=u(t+N_2-1/t)=u(t-1)$. The component $y_{\text{forced}}(t+k/t)$, however, is the effect

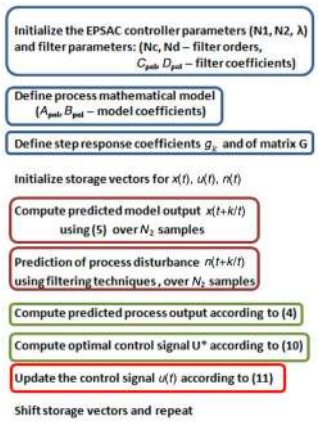

Fig. 1. Pseudo-code of the EPSAC control algorithm.

of a sequence of step inputs. In matrix notation, y_{forced} may be computed as

$$\begin{bmatrix} y_{\text{forced}}(t+N_1/t) \\ y_{\text{forced}}(t+N_1+1/t) \\ ..... \\ ..... \\ y_{\text{forced}}(t+N_2/t) \end{bmatrix} = \mathbf{G} \cdot \begin{bmatrix} \Delta u(t/t) \\ \Delta u(t+1/t) \\ ... \\ \Delta u(t+N_u-1/t) \end{bmatrix} = \mathbf{G} \cdot \mathbf{U} \tag{7}$$

where $G = \begin{bmatrix} g_{N_1} & g_{N_1-1} & \cdots & \cdots \\ g_{N_1+1} & g_{N_1} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ g_{N_2} & g_{N_2-1} & \cdots & g_{N_2-N_u+1} \end{bmatrix}$ and the parameters g_k are the coefficients of the unit step response. Using matrix notation, replacing the result in (7) into (6) gives

$$\mathbf{Y} = \mathbf{Y}_{\text{free}} + \mathbf{Y}_{\text{forced}} = \mathbf{Y}_{\text{free}} + \mathbf{G} \cdot \mathbf{U}. \tag{8}$$

The cost function in (1) can be written in matrix notation as

$$\begin{aligned} &(\mathbf{R}-\mathbf{Y})^{\mathbf{T}}(\mathbf{R}-\mathbf{Y}) + \lambda \mathbf{U}^{\mathbf{T}}\mathbf{U} \\ &= [(\mathbf{R}-\mathbf{Y}_{\text{free}}) - \mathbf{G}\mathbf{U}]^{\mathbf{T}}[(\mathbf{R}-\mathbf{Y}_{\text{free}}) - \mathbf{G}\mathbf{U}] + \lambda \mathbf{U}^{\mathbf{T}}\mathbf{U}. \end{aligned} \tag{9}$$

Minimizing (9) with respect to U leads to the optimal solution

$$\mathbf{U}^* = (\mathbf{G}^{\mathbf{T}}\mathbf{G} + \lambda\, \mathbf{I})^{-1}\mathbf{G}^{\mathbf{T}}(\mathbf{R} - \mathbf{Y}_{\text{free}}). \tag{10}$$

The first element, $\Delta u(t/t)$, in U* is then used to update the control signal

$$u(t) = u(t-1) + \Delta u(t/t). \tag{11}$$

The procedure is then repeated at the next sampling instant, when $u(t+1)$ is computed based on the new measurement $y(t+1)$. A pseudo-code of the EPSAC algorithm is given in Fig. 1.

III. EPSAC DC Motor Controller

The EPSAC-predictive algorithm can be used for controlling various types of electrical systems [22], [27]. The dc motor provided the possibility of building a flexible stand for running the tests and of achieving rapid performance comparisons.

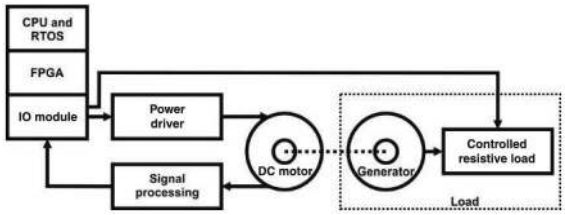

Fig. 2. System block diagram.

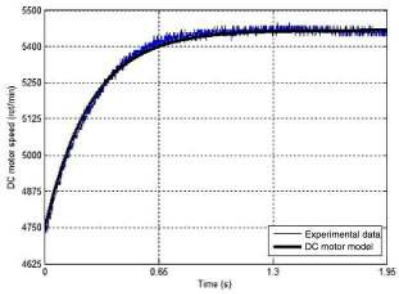

Fig. 3. Experimental data for process identification—speed rises.

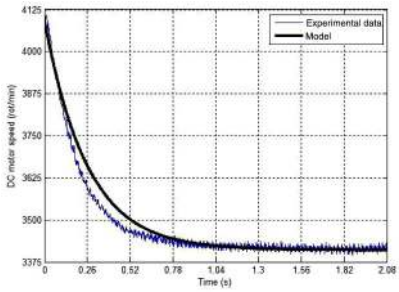

Fig. 4. Experimental data for process identification—speed decreases.

This is just a case study for testing and for validating the FPGA-based implementation, which clearly demonstrates that the EPSAC-predictive controller can be used in a wide range of applications. The block diagram of the system, including the controller, the driver, the signal-processing module, the dc motor, and the load implemented using a generator and a controlled resistive load, is presented in Fig. 2.

The CompactRIO embedded system used for implementation is a reconfigurable control and acquisition system providing high performance and reliability, and is programmable with LabVIEW. The device includes a PowerPC real-time controller running at 400 MHz and an extension module with digital input–output lines. Three different systems were used, one having a chassis with a Virtex-II FPGA, one having a chassis with a Spartan-6 device, and another one with a Zynq-programmable system on chip, including a real-time dual core processor running at 667 MHz.

The special architecture of the embedded system is built around two chips: the first one, on which a real-time operating system runs, and the second, the FPGA.

A. Extraction of DC Motor Parameters for Finalizing EPSAC Design

The EPSAC control strategy implemented in the FPGA has been tested in the closed-loop trajectory control of a dc motor. The first step in the FPGA implementation of the EPSAC

consists in determining a mathematical model of the process, i.e., the polynomials $A(q^{-1})$ and $B(q^{-1})$ in (5). To determine these polynomials, experimental identification techniques were employed.

Figs. 3 and 4 present the experimental data used for identification of the dc motor model and the output of the identified process model compared to experimental data, when the speed increases and decreases, respectively.

The dc motor output is its rotation speed, represented as experimental data in these figures, while the control input is the dc voltage supplied to the rotor. Prior to the experiment, the input voltage supplied was 70%. A step input of $+10\%$ was then applied to the rotor. For decreasing the speed, a -10% step was applied to the input.

Based on the shape of the step response, a transfer function was selected to model the process. The gain, as well as the time constants, is determined through identification techniques.

Using the determined transfer function, the polynomials $A(q^{-1})$ and $B(q^{-1})$ are computed based on a zero-order hold discretization, considering the sampling time $Ts = 0.015\,\text{s}$, chosen according to Shannon theorem

$$A(q^{-1}) = 1 - 0.94q^{-1}, \quad B(q^{-1}) = 1.54q^{-1}. \tag{12}$$

In the EPSAC controller design, the maximum prediction horizon is chosen for the predicted signal to capture around 60% of the process dynamics [26]. Since there is no process time delay, the minimum prediction horizon may be chosen $N_1 = 1$ sample, while $N_2 = 10$ samples, $\lambda = 0$, and $N_u = 1$. With this choice of the prediction horizons, the designed controller was first tested on the MATLAB simulation environment, using the transfer function of the model as the mathematical representation of the dc motor.

The controller designed and tested in the simulation environment was further implemented in a FPGA module, using the guidelines given in Sections IV and V, and employed in the closed-loop control of the dc motor previously described.

B. FPGA Implementation of EPSAC

This section shows the methodology that can be used for achieving the FPGA implementation of different types of control algorithms through graphical programming. The steps that have to be followed for reaching an optimal implementation method on the FPGA of the various control methods, realized using specific analysis and simulation environments, are briefly described below.

1) Rewriting the code used for simulation in the LabVIEW environment on the PC or on the real-time target.
2) Program testing using control vectors that were generated during simulation, for the control and for the controlled unit.
3) Data conversion from floating-point format to fixed-point format (FXP) or integer (INT).
4) The comparative testing of the implementations using the control vectors and the data available in the second step.
5) Go through steps 3 and 4 again until the stationary errors are acceptable; in the case of this paper, it is assumed that a small error is acceptable.

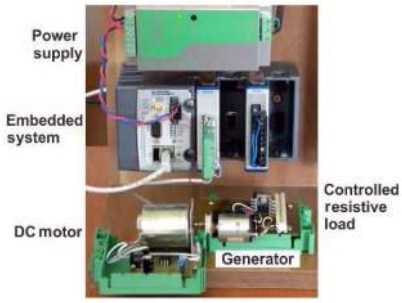

Fig. 5. Experimental stand.

The use of MATLAB sequences of code using MathScript was avoided, because it is not supported on the FPGA target.

IV. Hardware and Software Setup

Setting up the hardware and the software for implementing real-life control systems can be a troublesome task. On one hand, the hardware part requires taking into account various parameters including component compatibility, signal conditioning, placing and routing problems, while on the other hand, the software part must consider the architecture of the equipment. However, in this case, the software application takes advantage of the facilities provided by graphical programming [28]. The following two sections will present how the hardware and software had been developed in the case of the example application.

A. Hardware Setup

For interfacing the dc motor, two PCBs have been developed. The first PCB performs the processing of the signal received from the speed transducer, implying the amplification of the signal from the encoder (speed transducer) and signal filtering and its formatting for obtaining rectangular pulses. The board also includes the power driver for commanding the motor. The second PCB represents the load of the dc motor and consists of a digitally controlled resistive load. It is, in fact, a motor acting as a generator, with the same characteristics as the dc motor used, connected to a controlled resistive load. The stand used for verification consists in the embedded system, a power supply, the dc motor, and the components described above (Fig. 5).

B. Software Setup

The FPGA implementation of EPSAC consists of three different while loops: the first loop is used for measuring the speed of the motor; the second loop is used for generating the PWM that changes the speed of the motor using a digital output line; and the third loop represents the main loop, where the control algorithm is implemented.

The first loop measures the speed of the motor using a digital input line and can run at different speeds depending on the sensors that are used. This feature is made possible by including a time delay function, the implementation being based on a sequence of functions, which forces the execution order: two rising edges determine the time period.

The *while* loop is specific to LabVIEW FPGA implementations and is used for representing the continuous operation mode of the circuit to be realized.

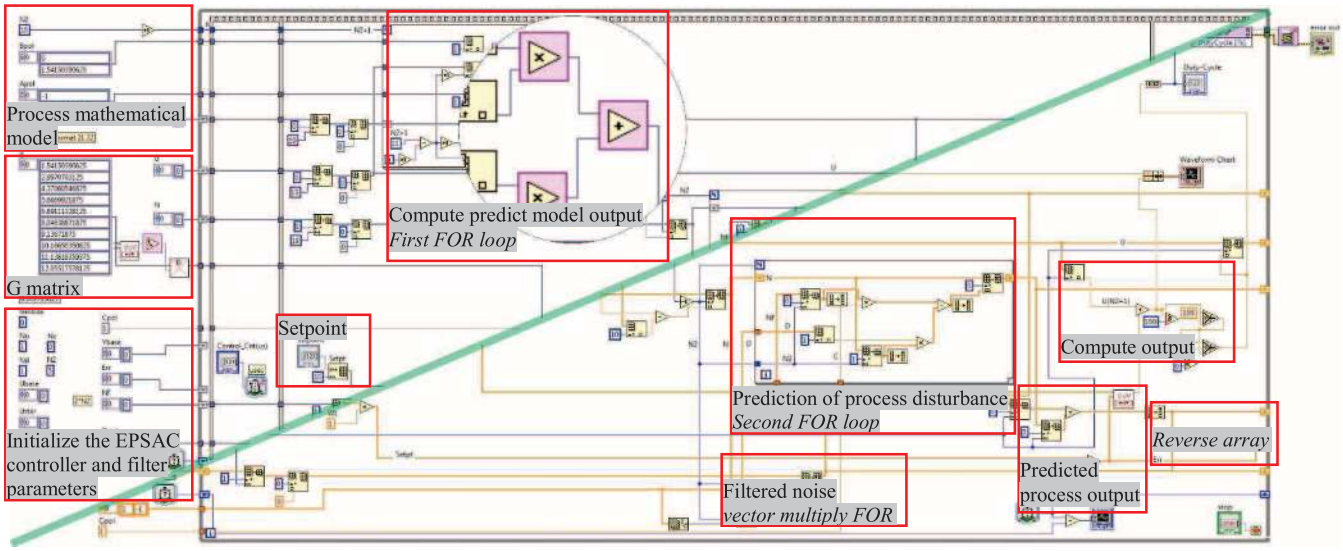

Fig. 6. Third loop—EPSAC control with fixed-point data (upper part) and floating-point data (lower part).

The application running on the real-time target implies the opening of a connection to the FPGA program. The values of the parameters are set using property and method nodes, while the measured values are read inside a loop. Data are not transferred between the real-time target and the FPGA through direct memory access (DMA) first-in, first-out (FIFO) because only a small amount is transmitted; only for the graphical representation of the involved values, the entire control algorithm is being implemented in the FPGA.

The loop that implements the EPSAC method can be seen in Fig. 6. The part below the diagonal of the picture shows the virtual instrument (VI) using floating-point values, while the part above the diagonal shows the program using fixed-point data. The scope presents some of the special fixed-point functions, high-throughput multiplication, and addition.

Fig. 6 includes the blocks from Fig. 1, where the pseudo-code for the EPSAC control algorithm is presented, and the occupied FPGA resources, listed in Table II.

V. Testing and Validation

The testing and validation of the design represent an important step in the development of FPGA-based systems and are usually performed using simulator-specific environments. In the case of this paper, several benchmark programs were developed and run on various targets for comparing the computation performances achieved.

LabVIEW provides functions that access the real-time timers of the systems that were tested, offering resolutions in the order of milliseconds, microseconds, or tens of nanoseconds for execution time or jitter measurement. Special frameworks, inside which the application could be tested, were developed. In the end, histograms including the execution time and jitter were realized for analysis.

First, a comparison between different platforms running the controller was done: a PC, a real-time controller, an FPGA, an FPGA including DSP blocks, a dual-core ARM, and an ARM microcontroller. After this, the different implementation

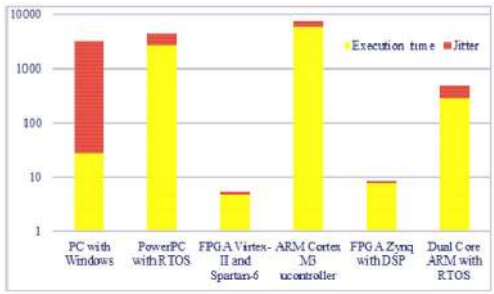

Fig. 7. Execution time and jitter for all targets (μs).

options offered by the graphical-programming environment in the case of FPGA devices were studied. In the end, a parallel between the performances offered by three different FPGA technologies used for implementing the controller was made: a more expensive, but relatively old Virtex-II device and cheaper and newer Spartan-6 and Zynq chips. For the first benchmark, the EPSAC algorithm code was compiled on a PC and run locally.

In the second test, the benchmark was transferred and run on the real-time target. For the third set of tests, the benchmarks ran on the FPGA, and finally, the same program was downloaded to a microcontroller. The first limitation of this solution consists of the data representation. In the case of the simulation and for the implementations on the PC, on the real-time target or on the microcontroller, the data are represented on floating point, double-type. When the FPGA is used, the data representation is a fixed point, using different formats, such as 14 bits for the integer word length and 32 bits for the entire word length.

The computational performance differs depending on the tested platform and the jitter is different from the datasheet value, depending on the implementation. The results, presented in Fig. 7, are the maximum reachable values and lead to the conclusion that the FPGA-based EPSAC controller can be used for fast dynamic processes.

TABLE I
FPGA, DEVICE UTILIZATION

Virtex-II	Total slices (14 336) (%)	Slice regs (28 672) (%)	Slice LUTs (28 672) (%)	MULT 18X18 (96) (%)	Exec. time (µs)	Clock (MHz)
R VIs, FXP 14.32	49.8	29.2	41.0	86	5	40.0
N-R VIs, FXP 14.32	49.5	37.2	41.0	57	9	40.0
Spartan-6	Total slices (6822) (%)	Slice regs (54 576) (%)	Slice LUTs (27 288) (%)	MULT 18X18 (58) (%)	Exec. time (µs)	Clock (MHz)
R VIs, FXP 14.32	Could not compile because not enough multipliers available					
N-R VIs, FXP 14.32	42.2	18.9	33.9	89.7	8.975	40.0
Zynq 7010 Artix-7	Total slices (4400) (%)	Slice regs (35 200) (%)	Slice LUTs (17 600) (%)	DSP48 s (80) (%)	Exec. time (µs)	Clock (MHz)
only R VIs, FXP 14.32	Could not compile because not enough multipliers available					
only N-R VIs, FXP14.32	Could not compile because not enough multipliers available					
R & N-R VIs, FXP14.32	79.0	46.8	79.5	80.0	7.775	40.0

TABLE II
FPGA FUNCTIONS, RESOURCES USED, AND EXECUTION TIMES

Virtex-II	Total slices (14 336) (%)	Slice regs (28 672) (%)	Slice LUTs (28 672) (%)	MULT 18X18 (96) (%)	Exec. time (µs)
Vect. scalar multiply	11.9	7.7	7.0	41	0.125
Vect. multiply *for*	12.9	10.2	5.8	4	1.125
Add vectors	14.4	9.7	7.6	33-bit adder 10	0.125
Subtract vectors	14.8	9.7	7.9	33-bit sub. 10	0.125
Reverse array	9.3	6.3	5.7	–	0.1
1st *for* loop	19.2	13.9	14.4	8	2.6
2nd *for* loop	17.0	12.8	11.4	41	3.375
Spartan-6	Total slices (6822) (%)	Slice regs (54 576) (%)	Slice LUTs (27 288) (%)	MULT 18X18 (58) (%)	Exec. time (µs)
Vect. scalar multiply	9.6	4.2	4.1	69	0.125
Vect. multiply *for*	12.2	5.6	5.9	6.9	1.125
Add vectors	11.5	5.3	7.9	33-bit adder 10	0.125
Subtract vectors	11.2	5.3	8.1	33-bit sub. 10	0.125
Reverse array	7.7	3.5	4.9	–	0.1
1st *for* loop	16.3	7.6	10.1	13.8	2.6
2nd *for* loop	14.7	7.0	10.3	69	3.375
Zynq 7010 Artix-7	Total slices (4400) (%)	Slice regs (35 200) (%)	Slice LUTs (17 600) (%)	DSP48s (80) (%)	Exec. time (µs)
Vect. scalar multiply	52.5	29.2	50.7	50	0.125
Vect. multiply *for*	24.4	31.2	52.7	5.0	1.125
Add vectors	20.5	30.8	53.2	–	0.125
Subtract vectors	20.5	30.8	53.3	–	0.125
Reverse array	19.4	26.0	45.8	–	0.1
1st *for* loop	29.0	34.4	62.7	10.0	2.6
2nd *for* loop	53.2	33.5	59.9	50	3.375

The FPGA target is more than five times faster than the PC, for the same control-performance parameters. Another important parameter tested here is the variation on the loop execution time (jitter). In the case of the PC, the jitter varies depending on the tasks that run in parallel with the application at a specific point of time. The minimum resolution of the function used to measure the execution time in the case of the FPGA is 25 ns. For achieving the maximum execution speed, the FPGA program includes mathematical operations specially designed for the FPGA target, allowing the specification of the data representation and its configuration, for both the input and output. The difficulty here consists in the computations involving arrays and in choosing the proper format for the data in fixed-point representation (integer word length and entire word length).

The use of reentrant or nonreentrant VIs in the control loop leads to different percentages in the FPGA resource utilization, in the case of the multiplier blocks, as it can be seen in Table I.

The conclusion that can be drawn from Table I is that it is difficult to predict which of the methods will always occupy less hardware resources, but, in general, as can be deduced from the presented cases, nonreentrant VIs lead to overall implementations requiring less area, which could be compiled successfully. Certainly, reentrant VIs lead to FPGA implementations offering faster execution speeds, as can be seen in this table.

In the case of the Zynq 7010 Artix-7 FPGA, which belongs to a more recent generation and for which a compiler from the year 2014 was used, a new situation emerged: the VI could be compiled only after a part of the VIs were configured experimentally as nonreentrant and the others as reentrant. Here, the occupied resources are far from the limit, and many possible configuration cases were obtained. The execution time represents a median value when compared with the one in the other cases, where the other two types of FPGAs were used and where all the VIs were set up to be either reentrant or nonreentrant. Other types of implementations on Artix-7 FPGA were not possible to be compiled due to an increase of the number of slices or DSP48s multipliers above the maximum limit.

The execution time can be improved by choosing reentrant VIs and preallocating clones for each instance of the blocks, by this way instantiating each one of them. In Table I, R stands for reentrant and N-R for nonreentrant VIs. The use of nonreentrant subVIs (subroutines) requires less multipliers, but more other FPGA resources are needed in this case, leading to an increase in the overall program-execution time. The amount of occupied resources in the FPGA is specific to the LabVIEW implementation and can be different than that of a VHDL implementation.

Furthermore, the area occupied by the controller can differ depending on the device and software version. However, the advantage of the approach used in this paper lies in the short project-completion time [29]. Experiments indicate that the system used for algorithm implementation allows clock speeds between 3 and 40 MHz. Therefore, if the process dynamics permits it, the clock frequency can be decreased, so that power savings can be achieved. This is also the case of the example application, where the execution time can be extended without affecting the control.

Table II presents the resource requirements and the execution time of some of the functions used in the control algorithm written in the FPGA. Based on data presented in this table, optimization can be performed regarding the resources in the FPGA that are used and regarding the execution time.

The blocks presented in Table II can be seen in Fig. 6, and can be found in the EPSAC implementation and in the algorithm presented as pseudo-code in Fig. 1.

The operations performed on vectors occupy more FPGA resources as the ones performed on scalars, and the loops, in the current case *for* loops, significantly increase the execution

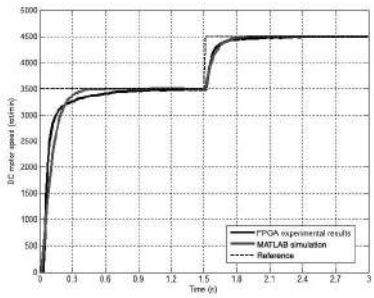

Fig. 8. Comparison between simulation and experimental data—output amplitude (rot/min).

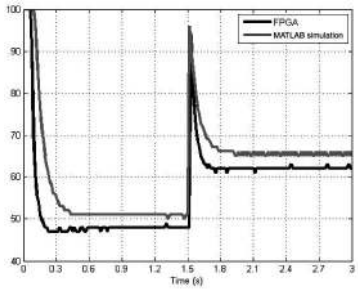

Fig. 9. Comparison between simulation and experimental data—input (%).

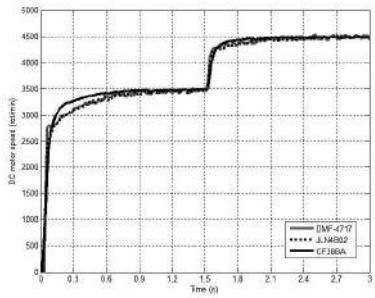

Fig. 10. Closed-loop experimental results obtained using three different motors—output amplitude (rot/min).

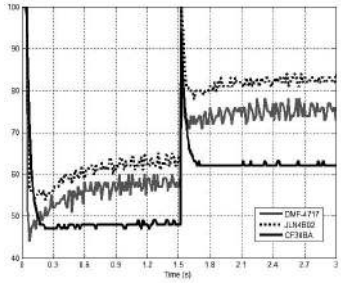

Fig. 11. Closed-loop experimental results obtained using three different motors—input (%).

time. The information in this table allows the user to perform optimization actions in case the achievement of an application that requires less FPGA resources is desired, which compiles faster or which provides shorter execution times. Although different technologies, with release dates separated by several years, are compared, the differences between the results are relatively small. The reasons for choosing the newer technology, Spartan-6 or Zynq, consist in the reduced cost and power consumption of the FPGA, but the industrial equipment embedding these state-of-the-art devices is still expensive.

Two data vectors, one for command and one for speed, generated through simulations, were used for evaluating the correctness of the proposed solution. The closed-loop experimental results are presented in Figs. 8 and 9 together with the simulation results.

The simulated EPSAC controller reaches the new prescribed setpoint within 0.4 s with no overshoot, while the experimental results with the FPGA-based EPSAC controller show that a similar performance is obtained with a settling time of 0.5 s and zero overshoot. The dc motor rotation speed is given in Fig. 8, while the corresponding control input, required to drive the dc motor to its new prescribed position, is presented in Fig. 9.

The validation of the proposed implementation, first on the PC, then on the real-time target, and finally, on the FPGA, was made possible by using the data vectors generated through simulation. The validation step was important, especially for the FPGA implementation, because additional changes in the behavior of the controller, caused by the translation from double precision data type (DBL) to FXP representation, occurred. Taking into account the fact that the program compilation time lasts for approximately 10 min, the simulation of the FPGA program was also an important action.

The behavior of the FPGA-based solution and the robustness of the controller are emphasized using three different dc motors from the same power class (Fig. 10). Fig. 11 shows that the command varies in different ways, because of the differences between the motors' parameters.

The execution time and jitter on the targets that were used are presented in Fig. 7. The execution time of the control loop (sampling time) in the FPGA or running on the real-time target has a constant value, 15 ms, while a variation of $\pm 200\ \mu s$ appears on the real-time target. The PC implementation has loop execution-time variations, which can reach up to tens of milliseconds. The performance is higher when the execution time is shorter, but at the same time, the jitter should be as low as possible.

In the vast majority of cases, for a numerical control system or for a "time critical" process, a better control system is obtained when the jitter is at its minimum. The PC does not belong to this category, having a short execution time, but a rather large jitter value. A jitter value which is hundreds of times smaller than the value of the execution time does not affect the control system, but a jitter having the same magnitude as the execution time negatively affects the entire system.

VI. Conclusion

For the case of the EPSAC control strategy, this paper demonstrates the feasibility of the graphical-programming controller design methodology as a fairly elegant, effective, and user-friendly method. Different implementations were compared against each other regarding speed, hardware resources, real-time performance, and programming aspects, under the following circumstances: graphical-programs portability on as many industrial standard devices as possible, program

scalability providing the possibility of running on resource-limited and relatively cheap devices or on high-performance systems. The results show that the FPGA solution offers a good compromise considering computational speed, hardware-resource usage, power consumption, and real-time performance. These advantages provide the possibility of using predictive control for fast dynamic processes. The results obtained justify the use of a graphical-programming environment in industry for realizing fast synthesis of control algorithms and for shortening time to market dedicated solutions.

References

[1] R. Zhang, A. Xue, and F. Gao, "Temperature control of industrial coke furnace using novel state space model predictive control," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2084–2092, Nov. 2014.

[2] F. Xu, H. Chen, X. Gong, and Q. Mei, "Fast nonlinear model predictive control on FPGA using particle swarm optimization," *IEEE Trans. Ind. Electron.*, vol. 63, no. 1, pp. 310–321, Jan. 2016.

[3] H. Guzman *et al.*, "Comparative study of predictive and resonant controllers in fault-tolerant five-phase induction motor drives," *IEEE Trans. Ind. Electron.*, vol. 63, no. 1, pp. 606–617, Jan. 2016.

[4] S. Chai, L. Wang, and E. Rogers, "A cascade MPC control structure for a PMSM with speed ripple minimization," *IEEE Trans. Ind. Electron.*, vol. 60, no. 8, pp. 2978–2987, Aug. 2013.

[5] M. A. Stephens, C. Manzie, and M. C. Good, "Model predictive control for reference tracking on an industrial machine tool servo drive," *IEEE Trans. Ind. Informat.*, vol. 9, no. 2, pp. 808–816, May 2013.

[6] C. Wang, M. Yang, W. Zheng, J. Long, and D. Xu, "Vibration suppression with Shaft Torque limitation using explicit MPC-PI switching control in elastic drive systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 11, pp. 6855–6867, Nov. 2015.

[7] M. Kaminski and T. Orlowska-Kowalska, "FPGA implementation of ADALINE-Based speed controller in a two-mass system," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1301–1311, Aug. 2013.

[8] L. Gomes, E. Monmasson, M. Cirstea, and J. J. Rodriguez-Andina, "Industrial electronic control: FPGAs and embedded systems solutions," in *Proc. 39th Annu. Conf. Ind. Electron. Soc. (IECON'13)*, 2013, pp. 60–65.

[9] C. I. Muresan, E. H. Dulf, and R. Both, "Comparative analysis of different control strategies for a train of cryogenic ^{13}C separation columns", *Chem. Eng. Technol.*, vol. 38, pp. 619–631, 2015.

[10] S. Folea, G. Mois, C. I. Muresan, L. Miclea, R. De Keyser, and M. Cirstea, "Implementation of an extended prediction self-adaptive controller using Labview^{TM}," in *Proc. 13th Int. Conf. Ind. Informat. (INDIN'15)*, Jul. 22–24, 2015, pp. 883–888.

[11] M. Ricco, P. Manganiello, E. Monmasson, G. Petrone, and G. Spagnuolo, "FPGA-based implementation of dual kalman filter for PV MPPT applications," *IEEE Trans. Ind. Informat.*, doi: 10.1109/TII.2015.2462313.

[12] E. Monmasson, L. Idkhajine, and M. Naouar, "FPGA-based controllers," *IEEE Ind. Electron. Mag.*, vol. 5, no. 1, pp. 14–26, Mar. 2011.

[13] E. Jamshidpour, P. Poure, and S. Saadate, "Photovoltaic systems reliability improvement by real-time FPGA-based switch failure diagnosis and fault-tolerant DC–DC converter," *IEEE Trans. Ind. Electron.*, vol. 62, no. 11, pp. 7247–7255, Nov. 2015.

[14] M. Shahbazi, P. Poure, S. Saadate, and M. R. Zolghadri, "FPGA-based reconfigurable control for fault-tolerant back-to-back converter without redundancy," *IEEE Trans. Ind. Electron.*, vol. 60, no. 8, pp. 3360–3371, Aug. 2013.

[15] E. Monmasson and M. Cirstea, "Guest editorial special section on industrial control applications of FPGAs," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1250–1252, Aug. 2013.

[16] L. Idkhajine, E. Monmasson, and A. Maalouf, "Fully FPGA-based sensorless control for synchronous AC drive using an extended Kalmanfilter," *IEEE Trans. Ind. Electron.*, vol. 59, no. 10, pp. 3908–3918, Oct. 2012.

[17] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "FPGAs in industrial control applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 224–243, May 2011.

[18] C. Sepulveda, J. Munoz, J. Espinoza, M. Figueroa, and F. C. Baier, "FPGA v/s DSP performance comparison for a VSC-based STATCOM control application," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1351–1360, Aug. 2013.

[19] A. Dinu, M. N. Cirstea, and S. E. Cirstea, "Direct neural-network hardware-implementation algorithm," *IEEE Trans. Ind. Electron.*, vol. 57, no. 5, pp. 1845–1848, May 2010.

[20] J. Rodriguez-Araujo, J. Rodriguez-Andina, J. Farina, F. Vidal, J. Mato, and M. A. Montealegre, "Industrial laser cladding systems: FPGA-based adaptive control," *IEEE Ind. Electron. Mag.*, vol. 6, no. 4, pp. 35–46, Dec. 2012.

[21] A. Oliveri *et al.*, "Two FPGA-oriented high speed irradiance virtual sensors for photovoltaic plants," *IEEE Trans. Ind. Informat.*, doi: 10.1109/TII.2015.2462293.

[22] Amin, R. T. Bambang, A. S. Rohman, C. J. Dronkers, R. Ortega, and A. Sasongko, "Energy management of fuel cell/battery/supercapacitor hybrid power sources using model predictive control," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 1992–2002, Nov. 2014.

[23] S. Vazquez *et al.*, "Model predictive control: A review of its applications in power electronics," *IEEE Ind. Electron. Mag.*, vol. 8, no. 1, pp. 16–31, Mar. 2014.

[24] A. Damiano, G. Gatto, I. Marongiu, A. Perfetto, and A. Serpi, "Operating constraints management of a surface-mounted PM synchronous machine by means of an FPGA-based model predictive control algorithm," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 243–255, Feb. 2014.

[25] Z. Ma, S. Saeidi, and R. Kennel, "FPGA Implementation of model predictive control with constant switching frequency for PMSM drives," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2055–2063, Nov. 2014.

[26] R. D. Keyser, "Model based predictive control," in *UNESCO Encyclopaedia of Life Support Systems (EoLSS)*, vol. 83. Oxford, U.K.: Eolss Publishers Co Ltd, 2003, article 6.43.16.1, ISBN 0 9542 989 18-26-34.

[27] C. S. Lim, N. A. Rahim, W. P. Hew, and E. Levi, "Model predictive control of a two-motor drive with five-leg-inverter supply," *IEEE Trans. Ind. Electron.*, vol. 60, no. 1, pp. 54–65, Jan. 2013.

[28] T. Orlowska-Kowalska and M. Kaminski, "FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system," *IEEE Trans. Ind. Informat.*, vol. 7, no. 3, pp. 436–445, Aug. 2011.

[29] A. Hace and M. Franc, "FPGA implementation of sliding mode control algorithm for scaled bilateral teleoperation," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1291–1300, Aug. 2012.

Silviu C. Folea (M'07) received the B.Sc. and Ph.D. degrees in control systems from the Technical University of Cluj-Napoca, Cluj-Napoca, Romania, in 1995 and 2005, respectively.

He is currently an Associate Professor with the Department of Automation, Technical University of Cluj-Napoca. His research interests include embedded and reconfigurable systems, data acquisition systems, wireless sensor networks, and graphical programming.

George Moiş (M'08) received the B.Sc. and Ph.D. degrees in control systems from the Technical University of Cluj-Napoca, Cluj-Napoca, Romania, in 2008 and 2011, respectively.

He is currently a Lecturer with the Department of Automation, Technical University of Cluj-Napoca. His research interests include embedded system design, digital design, and wireless sensor networks.

Cristina I. Muresan received the B.Sc. and Ph.D. degrees in control systems from the Technical University of Cluj-Napoca, Cluj-Napoca, Romania, in 2007 and 2011, respectively.

She is currently a Lecturer with the Department of Automation, Technical University of Cluj-Napoca. Her research interests include modern control strategies, such as predictive algorithms, fractional order control, time delay compensation methods, and multivariable systems.

Liviu Miclea (M'00) received the Ph.D. degree in automatic systems from the Technical University of Cluj-Napoca, Cluj-Napoca, Romania, in 1995.

He is currently a Professor in the Department of Automation with the Technical University of Cluj-Napoca. His research interests include design for testability, automatic testing, computer aided design, distributed systems, agent systems, cyber-physical systems (CPSs), and cloud computing.

Robain De Keyser received the M.Sc. degree in electromechanical engineering and the Ph.D. degree in control engineering from Ghent University, Gent, Belgium, in 1974 and 1980, respectively.

He is currently a Senior Professor of Control Engineering with the Faculty of Engineering, Ghent University. His research interests include model-predictive control, auto-tuning and adaptive control, modeling and simulation, and system identification.

Marcian N. Cirstea (M'97–SM'04) received the M.Eng. degree in electrical engineering from the Transilvania University of Brasov, Brasov, Romania, in 1990, and the Ph.D. degree in digital control of power converters from Nottingham Trent University, Nottingham, U.K., in 1996.

He is currently a Professor of Industrial Electronics and Head of the Department of Computing and Technology with Anglia Ruskin University, Cambridge, U.K. He was previously at De Montfort University, Leicester, U.K. He has authored over 135 works in this field and has delivered a number of international tutorials/presentations. His research interests include digital controllers for power electronics.

Dr. Cirstea was the recipient of the prestigious award of the Doctor Honoris Causa title by the Transilvania University of Brasov in January 2016.