# Hardware Realization of Embedded Control Algorithm on FPGA

Róbert Krasňanský, Branislav Dvorščák and Štefan Kozák
Institute of Automotive Mechatronics, Faculty of Electrical Engineering and IT,
Slovak University of Technology in Bratislava,
Bratislava, Slovakia
{robert.krasnansky, branislav.dvorscak, stefan.kozak}@stuba.sk

***Abstract*—This paper explores an efficient algorithm for design and implementation of Proportional-Integral-Derivative (PID) controller on the Field Programmable Gate Array (FPGA) technology. To create a synthesizable control algorithm, the Very High Speed Integrated Circuits Hardware Development Language (VHDL) was used as a programming tool. The paper points to the possibilities of parallel computation with the aim of speeding up the control implementation. The practical application of proposed control algorithm is illustrated by a test performed on a real laboratory Direct Current (DC) motor system. The results confirm the legitimacy of using the FPGA methodology for design of control algorithms, since it improves speed, accuracy and compactness. In addition, it is cost effective and has a low power consumption, which are desirable attributes in embedded control applications.**

***Keywords-FPGA; PID controller; Spartan 6; DC motor; VHDL language***

## I. INTRODUCTION

Motivated by the practical success of conventional control methods applied in industrial process control, there has been an increasing amount of work on development of effective hardware realizations of these control algorithms. Despite the numerous control design methods that have been proposed in the literature, it is estimated that PID controllers are still employed in more than 92% of the industrial processes today and many control systems using PID control have proved its satisfactory performance [1].

Recently, it has been shown that FPGAs can pose an alternative solution for the realization of digital control systems, previously dominated by the microprocessor systems [2]. The motivation behind using FPGAs to implement a PID controller, rather than microcontrollers or digital signal processors (DSPs), is that they provide a good balance between performance and cost. On the other hand, although the microcontrollers may be cheaper, they do not provide enough processing power to effectively perform complex calculations in real-time. Digital signal processors can implement complex algorithms quickly; however, these implementations are expensive. In addition, the systems designed on FPGA are flexible and can be reprogrammed an unlimited number of times. Unlike processors, FPGA circuits use dedicated hardware for processing commands. FPGAs logical structures can be arranged to execute in a truly parallel manner unlike the inherent sequential execution in microcontrollers, so different processing operations do not have to compete for the same resources. This functionality also makes it possible for multiple control loops to run on a single FPGA device at different rates. Execution time may be this way dramatically reduced, since parallel architectures allow FPGA-based controllers to reach the level of performance of their analog counterparts without their main drawbacks as parameter drifts or lack of flexibility [7]. These features make FPGAs very interesting for rapid prototyping.

The objective of this work is to design and implement a digital PI controller algorithm on FPGA platform and verify its performance as well as assess the FPGA suitability for control application.

The paper is organized as follows. Section II presents the overview to the FPGA architecture and functionality as well as VHDL language features and applications. Section III introduces the technical background of the PID algorithm followed by an approach for designing and implementation of the control system extended with the anti-windup on FPGA technology. In Section IV, an application of the proposed design to a laboratory DC motor system is presented and the experimental results on Xilinx FPGA chip are discussed. Comparisons are made between the implementation on a real system and the simulation results. The conclusion and future work are provided in Section V.

## II. BACKGROUND

### A. FPGA Architecture

The Field Programmable Gate Array (FPGA) represents an integrated circuit containing a two-dimensional array of configurable logic blocks whose interconnection and functionality can be reprogrammed depending upon the requirement of the user [8]. A typical FPGA architecture depicted in the Fig. 1 consists of three major elements:

- Programmable logic blocks, which consist of Configurable Logic Blocks (CLBs) arranged in an array that provides the functional elements and implements most of the logic in an FPGA. Each logic block has two flip flop and can realize any 5-input combinational logic function.
- Programmable interconnect resources provide routing path to connect between individual CLBs and between CLBs and input-output blocks.
- Input-Output Blocks (IOBs) provide the interface between the package pins and internal signal lines and thus the interconnection of external signals and

 ISBN: 978-1-61208-344-5 

internal signals in an array of CLBs. It can be programmed and configured as input, output or bidirectional port.

The CLBs, IOBs and their interconnectors are managed by a configuration program stored in a memory chip.

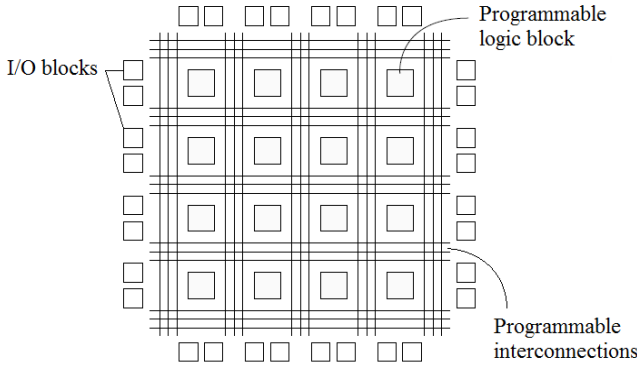


Figure 1. FPGA internal structure

A custom design can be implemented by specifying the function of each logic cell and setting the connection of each programmable switch. The CLBs structures include 2, 4 or more logic cells, also called logic elements. The structure of a logic cell, as the basic grain of the FPGA, is presented in Fig. 2. It consists of a Look-up Table (LUT), which can be configured either as a ROM, RAM or a combinatorial function.

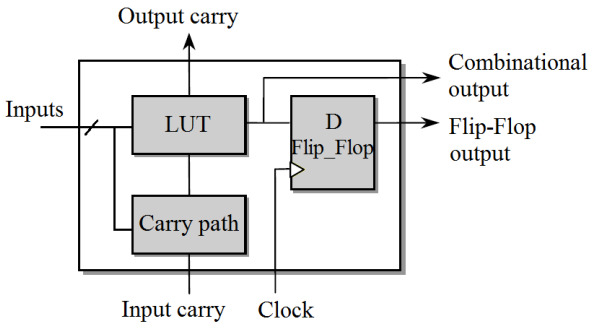


Figure 2. Logic cell [7]

Also, a carry look-ahead data path is included in order to build arithmetic operators and a D-Type Flip-Flop with all its control inputs, allowing registering the output of the logic cell.

### B. *VHDL Programming Language*

FPGAs can be programmed using Very High Speed Integrated Circuits Hardware Development Language (VHDL) [1] specifically developed to describe the behavior and structure of a digital circuit and its attributes. It uses significantly different principles than C language; for instance, the commands in the code are not executed sequentially, from the top to the bottom but in parallel way. VHDL describes the connections of the logic gates together to form adders, multipliers, registers and so on. A custom design can be implemented by specifying the function of each logic cell and setting the connection of each programmable switch.

A circuit design process can be carried out as shown in the Fig. 3. Once a FPGA is programmed, the internal circuitry is connected in a way that creates a hardware implementation of the application defined in the software. The big advantage of FPGA-based algorithms design is the possibility to employ the modular approach. Since there are a lot of I/O ports, it is theoretically possible to design more algorithms on one chip without influencing one another.

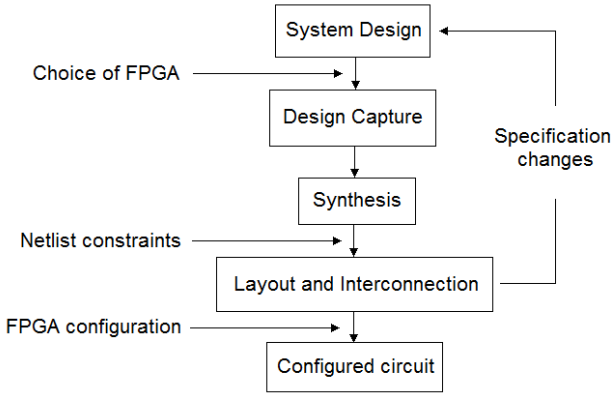


Figure 3. Design process of the circuit

The result is a user programmable piece of hardware with the reliability of dedicated hardware circuitry and the speed of modern microprocessor. Finally, FPGAs are Join Test Action Group (JTAG) compliant, thus the test data can be serially loaded into the device and the test results can be serially read out.

## III. IMPLEMENTATION OF CONTROL ALGORITHM ON FPGA

### A. *Digital PI Controller*

In this paper, the PID algorithm is applied for closed loop control. Among the control structures used in the industrial segment, the classic parallel PID controller depicted in Fig. 4 is one of the most widely used due to its well established practical implementation and tuning. The controller output is computed in continuous time as follows:

$$u(t) = k_p \left\{ e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right\} \qquad (1)$$

where the adjustable parameters are the proportional gain $k_p$, the reset time $T_i$ and the derivative time $T_d$, while $u(t)$ is the control output and $e(t)$ is the error signal (setpoint response level – measured response). The compensation parameters allow an increase in the system performance in a variety of ways.

Proportional control increases gain margin and stabilizes a potentially unstable system. Integral control, on the other

hand, minimizes steady-state error and derivative control increases system speed by increasing system bandwidth.

For a small time sample *T*, (1) can be transformed to a difference equation by discretization using Euler integration method – rectangular integration.

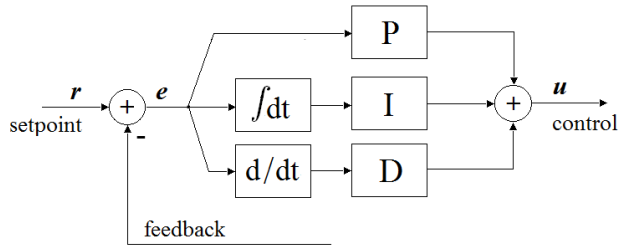


Figure 4. PID controller structure

A difference equation can be implemented by digital systems, either in hardware or software, where the derivative term is replaced by a first-order difference expression and the integral part by a sum, so the equation is given as:

$$u(n) = k_p e(n) + k_i \sum_{j=0}^{n-1} e(j) + k_d (e(n) - e(n-1)) \quad (2)$$

where n is a discrete time instant, $k_i = k_p T/T_i$ is the integral coefficient, $k_d = k_p T_d/T$ is the derivative coefficient and $T$ is sampling time. Using this algorithm called the "position form", all past errors $e(0)$ - $e(n)$ have to be stored to compute the sum. In this paper, we prefer the "incremental form" of the PI algorithm, where the recursive equation describing this algorithm is obtained when (2) for the time instant $n$-1 is subtracted from the same equation for the time instant n. Thus, the expression for $u(n-1)$ is calculated in the following way:

$$u(n-1) = k_p e(n-1) + k_i \sum_{j=0}^{n-1} e(j) + k_d (e(n-1) - e(n-2)) \quad (3)$$

and the correction term as

$$\begin{aligned} \Delta u(n) &= u(n) - u(n-1) \\ &= k_0 e(n) + k_1 (e(n-1) + k_2 e(n-2)) \end{aligned} \quad (4)$$

Subsequently, for the PI controller, the current control input is in the form

$$u(n) = u(n-1) + \Delta u(n) = u(n-1) + k_0 e(n) + k_1 e(n-1) \quad (5)$$

where

$$k_0 = k_p$$

$$k_1 = -k_p + k_i$$

and $k_i = k_p T/\mathrm{Ti}$ is the integral coefficient. The big advantage of this approach is that in software implementation, (5) avoids accumulation of all past errors.

The PI incremental form (5) has to be decomposed into basic arithmetic operations:

$$e(n) = w(n) - y(n) \quad (6)$$

$$p_0 = k_0 e(n) \quad (7)$$

$$p_1 = k_1 e(n-1) \quad (8)$$

$$s_1 = p_0 + p_1 \quad (9)$$

The current control output is then calculated as

$$u(n) = s_1 + u(n-1) \quad (10)$$

### B. *Parallel Design*

For the implementation of the proposed PI algorithm onto FPGA, the parallel design [3] has been used. This design is mainly composed of combinational logic, so each operation has got its own arithmetic unit – adder or multiplier. Such modified control algorithms are then feasible on FPGA circuits. The parallel design architecture of the PI incremental algorithm is depicted in Fig. 5. The design requires a total of 2 combinational logic multipliers, 3 adders and 3 registers [6]. The clock signal *clk* is used to control sampling frequency. The negation of *y* is generated using bit-wise complementing and subsequently adding 1. The difference *w* - *y* generates current error $e(n)$.

Registers are used to store the intermediate results obtained. Multipliers and adders are used for multiplication and addition of input signals according to arithmetic operations described in the previous section A. The block REG stores error values $e(n)$ and $e(n-1)$. Hence, at the rising edge of control, signal $e(n)$ of the last cycle is latched at register REG, thus becomes $e(n-1)$ of this cycle. Similarly, $u(n-1)$ are recorded at REGs by latching $u(n)$ respectively [10].

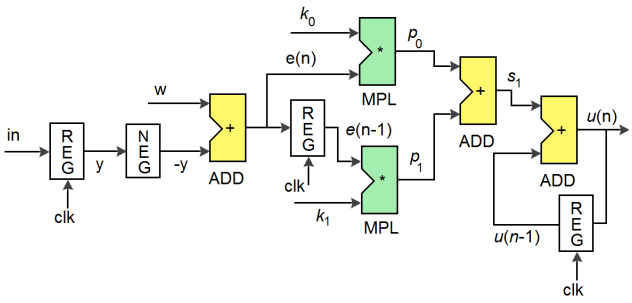


Figure 5. Parallel design of incremental PI algorithm

The values of $e(n)$ and $e(n-1)$ with their polarity indicating whether the calculated value is positive or negative are fed to PI equation (10) and the current control

output is calculated. From Fig. 5, it can be seen that the register blocks (REG) depend on clock frequency. That means that the functionality of these blocks shall be within the process, which responds to the rising edge of *clk* signal. At the same time, registers can be set to initial values of 0 after the first start or by asserting the reset signal. The process code example can be developed as depicted in Fig. 6 below.

```
Regist_process : PROCESS (clk, reset)
BEGIN
    IF  reset = '1' THEN
          en1 <= to_sfixed (0, en1);
          un1 <= to_sfixed (0, un1);
          e <= to_sfixed (0, e);
    ELSIF  clk'EVENT AND clk = '1' THEN
          en1 <= e;
          e <= to_sfixed (to_integer (unsigned (w)) –
                to_integer (unsigned (y)), e);
           un1 <= "0" & Add3 (16 downto -9);
    END IF;
END PROCESS Regist_process;
```

Figure 6. The laboratory model of DC motor

Once the signal *reset* is in logical state 1, the variables $e(n)$, $e(n\text{-}1)$ and $u(n\text{-}1)$ are reset. When the signal *reset* is in logical state 0 and the rising edge of the signal *clk* occurs at the same time, the program assigns to the variable $e(n\text{-}1)$ the value of the variable $e(n)$, similarly to the variable $u(n\text{-}1)$ the value of the variable $u(n)$ and calculates the difference of input signals $w - y$.

### C. Implementation of Anti-windup Control

In the motor speed control, the maximum control output from a PI controller is determined by the converter protection, magnetic saturation and motor overheating. Hence, the saturation is applied even at the cost of introduction a non-linearity into the system. This phenomenon, called windup effect, can lead to a large overshoot, long settling time or even unstable closed-loop system.

The goal of the implementation of anti-windup in the incremental form of the PI controller is to eliminate the wind-up in the error integrator and to provide a strictly aperiodic step response even in case with large input disturbance. The implementation of anti-windup system is easy using incremental PI algorithm. The control action value is being checked and $u_{out}$ is determined according to the following equation [4]:

$$u_{out} = \begin{cases} u_{in} & \text{if } u_{\max} > u_{in} > u_{\min} \\ u_{\max} & \text{if } u_{in} \geq u_{\max} \\ u_{\min} & \text{if } u_{in} \leq u_{\min} \end{cases} \quad (11)$$

where $u_{in}(n)$ represents the control output before saturation and $u_{out}(n)$ is the saturated control output variable.

## IV. CASE STUDY

### A. Laboratory DC Motor System

In this section, the proposed algorithm is applied to control a real laboratory DC motor system (Fig. 7) to demonstrate its high performance and efficacy. The system consists of two co-operating real DC servomotors, where the first one is connected as a drive motor and the other one as a generator. The manipulated variable is the input voltage of DC motor and the output controlled variable is the angular speed represented by the output voltage in range of 0-10V. To obtain a model of the system, input-output relations of the plant have been identified with the help of the software LABREG [5]. The interconnection of the laboratory model with the software LABREG is assured by the Advantech data acquisition card type PCI 1711.

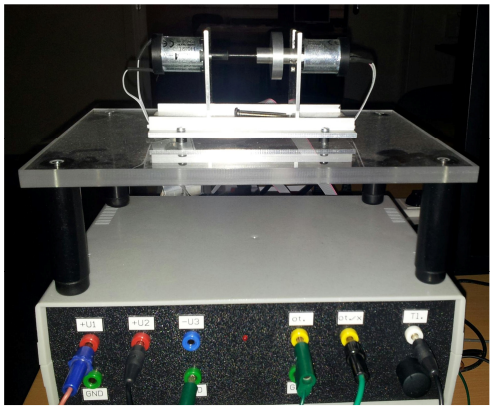

Figure 7. The laboratory model of DC motor

The discrete transfer function we obtained with the selected sampling rate $T_s$ = 0.1s has been converted to the following continuous-time model:

$$G(s) = \frac{0.08047s + 1.677}{0.4142s^2 + 1.053s + 1} \quad (12)$$

The control objective was to drive the angular speed of the motor to track the desired reference signal.

### B. Design of Control Algorithm

The first control algorithm has been developed using VHDL language in the Xilinx ISE Design Suite 14.4 software environment according to incremental PI from (Fig. 5). Firstly, a software implementation was developed and tested to verify the algorithm functionality. By choosing of appropriate sampling period and fixed point format a discrete PI controller has been developed from continuous-time PI controller, whereas the inverse dynamic tuning method has been used to tune the parameters $k_p$ and $T_i$. From the parameter tuning experiment the following results were obtained: proportional gain $k_p$ = 0.2025, integral coefficient $T_i$ = 0.4752, derivative coefficient $k_d$ = 0 and sample period $T$ = 0.1s. The same discrete PI parameters were applied to the

 ISBN: 978-1-61208-344-5 

hardware implementation on FPGA to perform the control tests.

Because of the fact that most FPGAs are limited to finite precision signal processing using fixed-point arithmetic, the bit word-length and radix setting of input and output signals were determined carefully to ensure the fidelity of the algorithm. Since every addition or subtraction causes adding an extra bit as well as every multiplication result will have a bit width equal to the sum of the number of bits in the inputs, this was the most important part of the design. A simple flow diagram (Fig. 8) shows the implementation of the designed algorithm using FPGA. The algorithm has 4 inputs: the motor system output, the reference signal *w* with the same bit width as signal *y*, clock and reset.

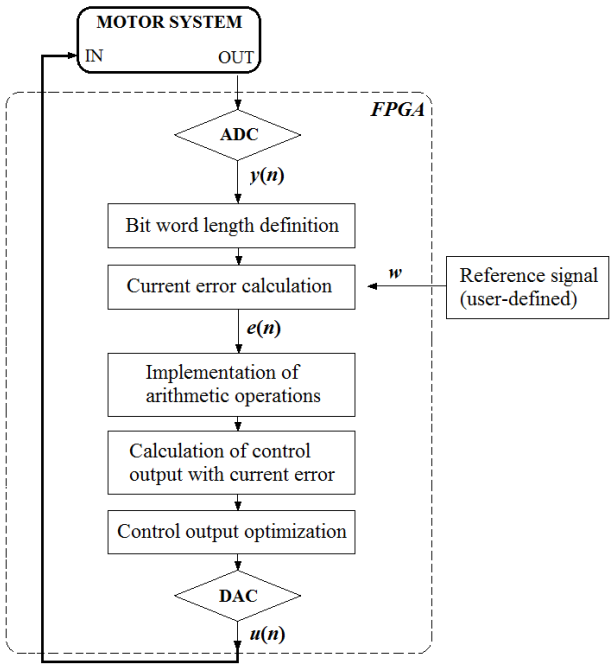


Figure 8. FPGA-based control implementation cycle

The step sequence of the algorithm can be determined as follows:

Step 1: Initialization of the system (set clock frequency, declaration of system variables);

Step 2: Setting of bit width of the signals (input and output signals according to the resolution of A/D conversion results);

Step 3: Calculation of the current error according to the reference signal defined by user ($e = w - y$);

Step 4: Calculation of the control output with the current error based on the combinatorial logic operations according to relation (10) and parallel architecture (Fig. 5);

Step 5: Optimization of the obtained control output for 8-bit D/A converter;

Step 6: The analog output signal obtained is fed back to drive the speed of DC motor system.

The second control algorithm was developed according to section C and (11) and also applied to the real-time speed control of the laboratory motor system. This algorithm is unlike the first one augmented of the anti-windup mechanism. The calculated control output is adjusted according to (10) and after that optimized and fed back to motor system through 8-bit DAC.

### C. *Experimental Results*

The experimental studies were carried out to evaluate the performance of the proposed control algorithm. The algorithm was downloaded into SPARTAN-6 FPGA development kit (Fig. 9) and the complete system was reset.

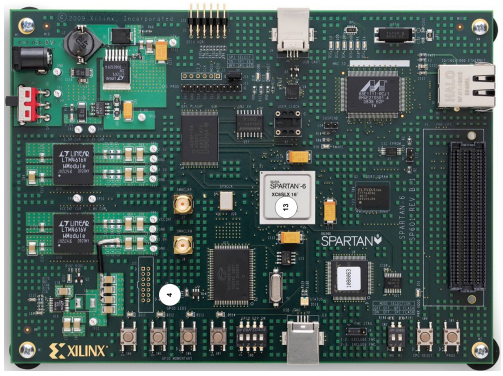


Figure 9. SPARTAN-6 development board

The comparison of the experimental results executed using FPGA with the simulation results obtained from MATLAB are illustrated in Fig. 10 below.

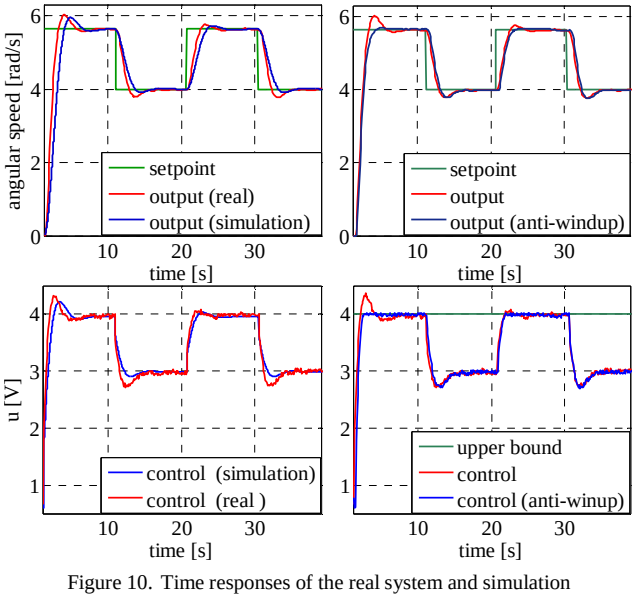


Figure 10. Time responses of the real system and simulation

The comparison of the performance of the proposed anti-windup PI control algorithm with the PI control algorithm without the anti-windup mechanism is also depicted. The

results show the effectiveness and good performance of the FPGA-based controller. The limited vector size of the signals due to the different interpretation of the fixed-point arithmetic has an effect in the calculation and therefore in the shape of the obtained time responses.

The objective evaluation of quality has been performed by meaning of various performance and quality criteria in the time domain (settling time, maximal overshoot and root mean square error (RMSE)) with results expressed in Table 1. The PID algorithm has been demonstrated to be effective for DC motor speed control.

TABLE I. QUANTIFICATION OF QUALITY CONTROL CRITERIA

| Control Performance | | | |
|---|---|---|---|
| ***Controlled output*** | ***Settling time*** | ***Max. Overshoot (%)*** | ***RMSE*** |
| Real system | 8,25 | 6,9449 | 0,8937 |
| Simulation | 9,1 | 5,6301 | 0,8992 |
| Real system with anti-windup | 5,3 | 0,7815 | 0,9217 |

As seen in Table 1, anti-windup mechanism has improved the control performance mostly in the way of overshoot and settling time.

### *D. Resource Usage*

Xilinx tool device utilization summary and percentage of available resources reports, which have been used for the current design using FPGA are shown in Table 2 below.

TABLE II. DEVICES UTILIZATION SUMMARY

| Device Utilization Summary | | | |
|---|---|---|---|
| ***Slice Logic Utilization*** | ***Used*** | ***Available*** | ***Utilization*** |
| Number of Slice Registers | 170 | 18,224 | 1% |
| Number used as Flip Flops | 168 | | |
| Number used as Latches | 2 | | |
| Number of Slice LUTs | 313 | 9,112 | 3% |
| Number used as logic | 305 | 9,112 | 3% |
| Number of occupied Slices | 124 | 2,278 | 5% |
| Number of MUXCYs used | 204 | 4,556 | 4% |
| Number with an unused Flip Flop | 186 | 344 | 54% |
| Number with an unused LUT | 31 | 344 | 9% |
| Number of fully used LUT-FF pairs | 127 | 344 | 36% |
| Number of slice register sites lost to control set restrictions | 94 | 18,224 | 1% |
| Number of bonded IOBs | 33 | 232 | 14% |
| Number of LOCed IOBs | 13 | 33 | 39% |
| Number of RAMB16BWERs | 0 | 32 | 0% |
| Number of RAMB8BWERs | 0 | 64 | 0% |
| Number of BUFG/BUFGMUXs | 5 | 16 | 31% |
| Number of DSP48A1s | 3 | 32 | 9% |

Hardware resources usage was: 168 slice flip-flops, 170 slice registers and 313 slice LUT's. It can be seen that just 5% of the FPGA was used.

## V. CONCLUSION AND FUTURE WORK

In this paper, a closed-loop PI algorithm was proposed, designed and successfully implemented on FPGA platform. The performance was verified and tested for control of laboratory DC motor system. The control algorithm has been improved by using the anti-windup structure in case of considering the input constraints. The overall control algorithm has been programmed using VHDL language and implemented on Xilinx Spartan-6 FPGA development kit. The experimental results show a good set-point tracking and demonstrate that FPGAs are well suited for implementation of complex motor control algorithms due to their high speed execution characteristics. The future work will deal with the design and implementation of more complex predictive control algorithm considering the constraints on input, state and output variables.

## ACKNOWLEDGMENT

The work has been supported by the Slovak Research and Development Agency under grants APVV-0772-12 and APVV-0246-12.

## REFERENCES

[1] P. J. Ashenden, The Designer's Guide to VHDL. Morgan Kaufmann, 1995.

[2] K. J. Astrom and B. Wittenmark, Computer Controlled Systems, Englewood Cliffs, NJ: Prentice-Hall, 1997.

[3] Y. F. Chang, M. Moallem, and W. Wang, "Efficient implementation of PID control algorithm using FPGA technology," Proceedings of 43 IEEE Conference On Decision and Control, vol. 5, Dec. 2004, pp. 4885–4890.

[4] L. Charaabi, E. Monmasson, and I. Slama-Belkhodja, "Presentation of an efficient design methodology for FPGA implementation of control systems: Application to the design of an antiwindup PI controller," Proc. IEEE Ind. Electron. Soc. Annu. Conf., vol. 3, Nov. 2002, pp. 1942–1947.

[5] S. Kajan and M. Hypiusová, "Labreg Software for Identification and Control of Real Processes in Matlab," Technical Computing Prague 2007: 15th Annual Conference Proceedings, Prague, Czech Republic, Nov. 2007, pp. 71.

[6] R. Krasňanský and B. Dvorščák, "Design and Implementation of FPGA-based PID controller," In ACCS'13: 3rd International Conference on Advanced Control Circuits and Systems, ERI, Luxor, Dec. 2013, pp. 43.

[7] E. Monmasson and M. N. Cirstea, "FPGA Design Methodology for Industrial Control Systems-A Review," IEEE Transactions on Industrial Electronics, vol. 54, August 2007, pp. 1824-1842.

[8] J. Oldfield and R. Dorf, Field-Programmable Gate Arrays, John Wiley & Son, 1995.

[9] Xilinx Data Book, 2006, Available online at: www.xilinx.com (accessed March 28, 2014).

[10] W. Zhao, B. H. Kim, A. C. Larson, and R. M. Voyles, "FPGA implementation of closed-loop control system for small-scale robot," In ICAR'05: 12th International Conference on Advanced Robotics, Seattle, WA, July 2005, pp. 70-77.

 ISBN: 978-1-61208-344-5 