A Brushless DC Motor Control Software C Library based on ATmega64M1 Applied to Teaching

Himar A. Fabelo, José Cabrera, Aurelio Vega, Víctor Déniz
Dept. of Electronics Engineering and Automatics (DIEA)
Institute for Applied Microelectronics (IUMA)
University of Las Palmas de Gran Canaria (ULPGC), Spain
hfabelo@iuma.ulpgc.es, jose.cabrera@ulpgc.es, avega@iuma.ulpgc.es, vdgonzalez@iuma.ulpgc.es

***Abstract*— This paper describes the architecture of a C library developed for the control of a brushless direct current motor. The library has been made using a modular programming methodology. The control system is based on ATmega64M1 microcontroller integrated into a controller for 3-phase brushless direct current motors. The controller has been especially designed and manufactured for this project. This library has been mainly created to be used as an educational resource in teaching of practical sessions of microcontrollers programming and motor control systems. With it, students can learn the structure and the operation of the most used control systems currently that are replacing to the traditional direct current motors with brushes.**

Keywords*— brushless direct current motor (BLDCM), BLDC motor control C library, microcontroller ATmega64M1, analog to digital converter (ADC), Power Stage Controller (PSC).

I. INTRODUCTION

BLDC motors (Brushless Direct Current Motors) operate by an electronic commutation signalized by solid state Hall sensors instead of a brushes commutation. These sensors indicate to the microcontroller (µC) the position of the motor rotor. The controller, from these signals, must excite the motor coils by a right switching logic. Software to perform these functions is needed. It must also control the speed adjustment of the system, the motor temperature sensor (if the motor has one) or the communications to the outside.

At present the trend at the time of performing almost any type of software, is to develop it in a modular way. One advantage of this development methodology is to obtain the ability to reuse code developed, achieving high productivity and reduced time in future work.

By the modular programming, large and complex problems are divided into several smaller and simple subproblems. These subproblems, in turn, are divided into simpler ones. These steps must be carried out again and again to get items that are simple enough to be easily resolved. This technique is called "successive refinement" or "top-down design".

By this project, students of electronics and robotics will understand and study the modular programming and the BLDC motor control systems. Thus, by the experiments with this software and the complete control system, they will understand its functioning and, moreover, they will learn the C programming of µCs.

II. THE MOTOR CONTROL SOFTWARE C LIBRARY

The µC chosen for the development of this project is the ATmega64M1 [1], which has a 64 kB flash memory and a SRAM (Static Random Access Memory) of 4 kB. Within this series of ATmega, there are also the Atmega16M1 and ATmega32M1 with a flash memory capacity and a SRAM of 16 kB and 1 kB, and 32 kB and 2 kB respectively. The µC model is chosen according to the application and the size of the software. The ATmega64M1 was chosen for this project due to its larger flash memory capacity in which future software upgrades will be able to be placed.

The development of the function library for controlling BLDC motors is based on the open-loop control with speed adjustment technique [2][3][4][5]. Fig. 1 shows the most important components of the µC ATmega64M1 for this application.

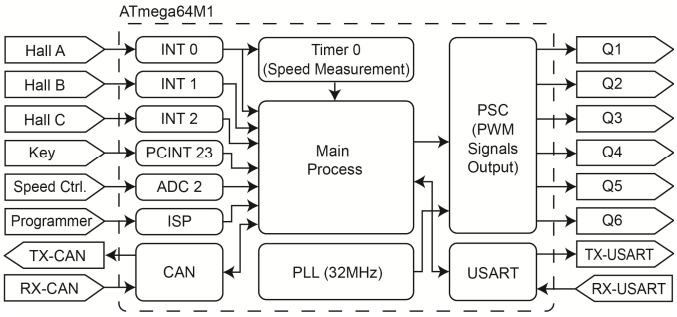

Fig. 1. Diagram of the parts of the µC used in the BLDCM controller

The Hall sensor signals (A, B and C) from the motor enter in the µC through the ports that have the external interrupts with higher priority levels. These interrupts act on the main system process activating the switching of the Q1 to Q6 outputs of the PSC (Power Stage Controller) module in the correct order. The interrupt of the A Hall sensor carries out the measurement of motor speed by the timer 0.

The input signal "Key" starts or stops the motor. This active low signal is introduced through the pin change interrupt 23 (PCINT 23). This interrupt is executed when a signal level

978-1-4799-6002-6/14/$31.00 ©2014 IEEE

change occurs. In the main process, it disables or enables the PSC module. This module generates the 6 PWM (Pulse Width Modulation) signals, from the Q1 to the Q6, which attack the drivers of MOSFETs transistors. Then these transistors excite the BLDC motor coils.

The motor speed adjustment is performed by a digitized analog signal through one of the ADC (Analog to Digital Converter) channels. This digitalized signal establishes the duty cycle of the PWM signal used by the PSC module. This PWM signal of 16 kHz is generated from the 32 MHz PLL (Phase-Locked Loop) inside the µC.

Moreover, there are both the CAN (Controller Area Network) and the UART (Universal Asynchronous Receiver and Transmitter serial) communication modules with their respective transmit and receive data signals.

A. Main Process

This process performs the initialization of variables and the necessary hardware of the microcontroller. By *micro_modules_initialization()* function, the initial configuration of input and output ports of the µC is realized and the following functions are executed:

- *adc_initialization()*: It configures and enables the ADC module to read the analog speed adjustment potentiometer by the ADC2.
- *timer1_initialization()*: It initializes the timer 1 to perform the task of sampling speed and the PWM duty cycle adjustment. The compare match interrupt is enabled to run every 256 microseconds.
- *timer0_initialization()*: It configures the timer 0 as a counter for the motor speed measurement. It enables the overflow interrupt to convert the 8-bit timer in a 16-bit timer by an auxiliary variable that is incremented each time the interrupt is executed. Thus, higher accuracy in the motor speed measurement is obtained.
- *uart_initialization()*: It sets the µC UART module as transmitter and receiver. It uses the parameters set up by the user in the configuration file.
- *start_pll_32mhz()* y *wait_pll_ready()*: Functions that configures and activates the PLL to a frequency of 32 MHz. Once they are activated, they keep waiting until the PLL is ready to continue executing the program.
- *psc_initialization()*: It makes the PSC module configuration according to the parameters detailed in the user configuration file.
- *external_interrupt_initialization()*: It enables the external interrupts used by the Hall sensors (INT1, 2 and 3). Also, it enables the pin change interrupts for the ignition switch and the motor rotation direction switch (if this option is enabled).

When the system has been initialized, the *adc_launch()* function is executed every 256 µs in an infinite loop. This function reads the speed adjustment potentiometer. The value is stored and used to set the motor reference speed. By this reference speed, the control loop is running in open loop to set the value of the PWM duty cycle.

When the value of the PWM duty cycle has been obtained, the PSC module registers are updated with that value. After, the value of the measured speed is sent through the UART module for its display on the monitoring software (Fig. 2).

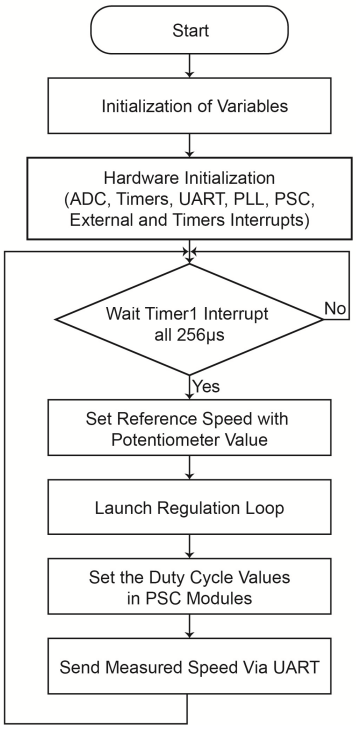

Fig. 2. Flow diagram of the main process

B. Hall Sensors Interrupts

These interrupts detect the rising edges of the Hall sensors signals. They are the highest priority level external interrupts of the program. In each one, the *output_psc_switch_commutations (value)* function is executed. This function requires as parameter the value at that time of the Hall sensors properly formatted. Using this value and the rotation direction variable at that time, the activation of the corresponding Q outputs of the PSC module is performed. These PWM output signals make the switching of the BLDC motor coils and their duty cycle is set by the speed adjustment potentiometer.

Fig. 3 shows the switching sequence of PSC module outputs according to the Hall sensor signals when the rotation is CW (*ClockWise*). The three possible states in which the U, V and W motor coils can be found are the following:

- VCC: coil is connected to the supply voltage.
- GND: coil is connected to the ground.
- NC: coil is not connected.

Table I details the switching sequence of PSC outputs in relation to the value of the Hall sensors and the CCW (*CounterClockWise*) or CW rotation motor.

In the A Hall sensor interrupt (INT 0) the calculation of revolutions per minute of the motor for each rising edge is performed by the *calculate_estimated_speed ()* function. This function uses the 16-bit value obtained from the timer 0 and a constant, which is defined by the user based on the motor used, to calculate the revolutions per minute of the motor (Fig. 4).

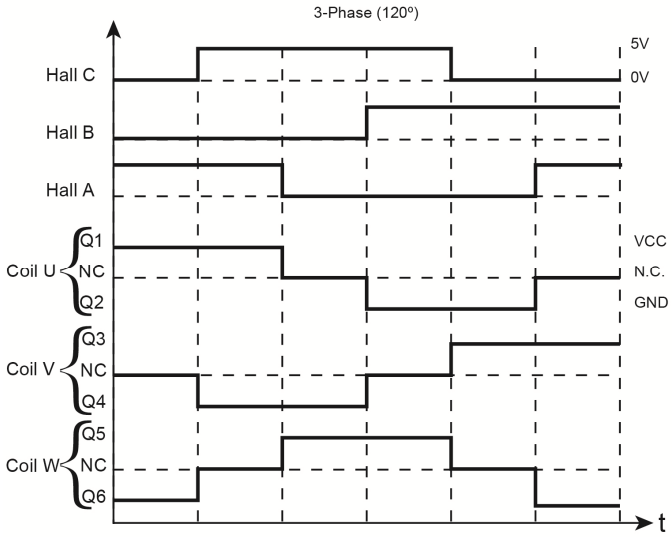

Fig. 3. Hall sensors signals and PSC outputs for CW rotation

TABLE I. SWITCHING SEQUENCE OF PSC OUTPUTS

Hall Sensors Value (C,B,A)	PSC Outputs Active (CCW)	PSC Outputs Active (CW)
001	Q5 – Q2	Q1 – Q6
101	Q3 – Q2	Q1 – Q4
100	Q3 – Q6	Q5 – Q4
110	Q1 – Q6	Q5 – Q2
010	Q1 – Q4	Q3 – Q2
011	Q5 – Q4	Q3 – Q6

C. ADC interrupt

This interrupt reads of the speed adjustment potentiometer and converts the value obtained to a digital value (Fig. 5). This interrupt is executed when the potentiometer signal conversion connected to the analog input channel 2 (ADC2) is finalized.

D. Timer 0 overflow interrupt

As already explained above, this interrupt increases the value of auxiliary variable that converts the 8-bit timer 0 in a 16-bit timer (Fig. 6). If the auxiliary variable is above than a calculated value, then the variable and the value of the reference speed are reset. If the variable that indicates whether the motor is active has a true value, the motor is tried to run again. To do this, it is used the *retry_run_motor()* function, which launches the control loop, updates the value of the PWM duty cycle and runs the activation function of the PSC outputs.

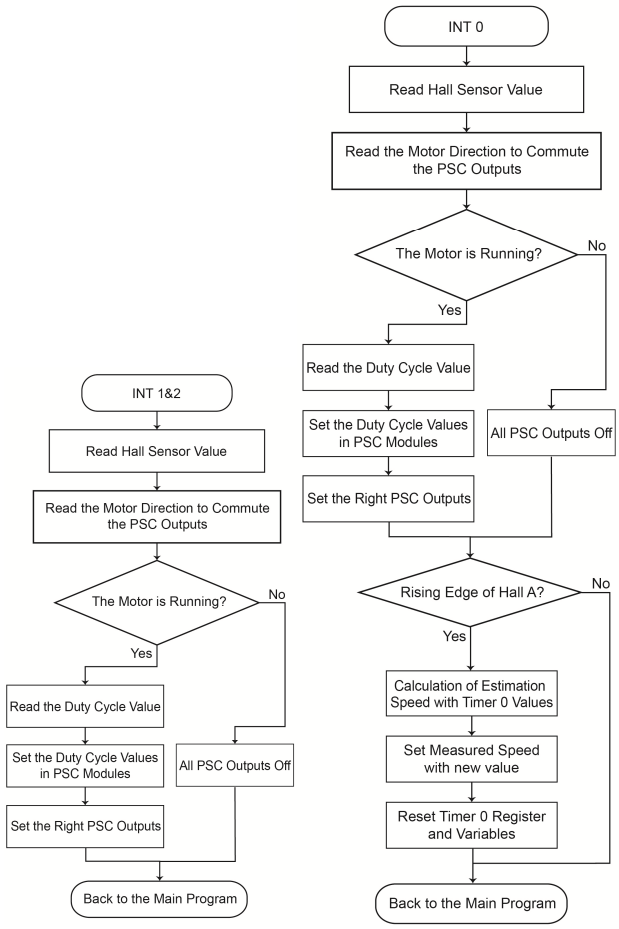

Fig. 4. Flow diagram of the Hall sensors interrupts

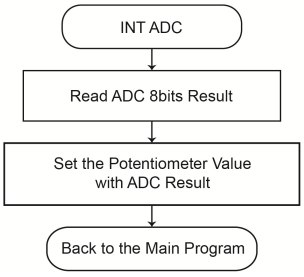

Fig. 5. Flow diagram of the ADC interrupt

E. Pin change interrupt 1 (PCINT 1)

It controls the starting and the stopping of the motor. When the interrupt detects a change in the signal level from the ignition switch, it checks if the level is high or low. When a high level is present on the pin, all PSC outputs are disabled. On the contrary, when a low level is present, the *output_psc_switch_commutations(value)* function is called with the value of the Hall sensors in that instant (Fig. 7).

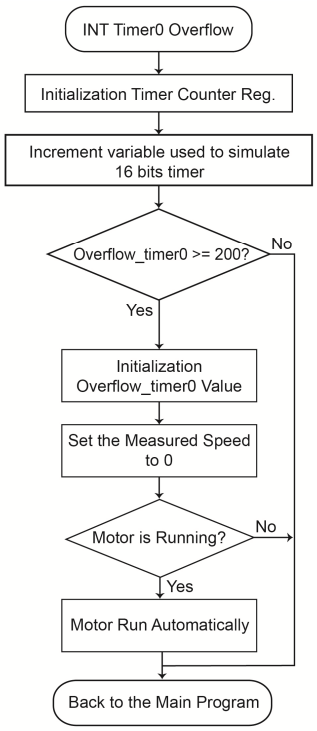

Fig. 6. Flow diagram of the Timer 0 overflow interrupt

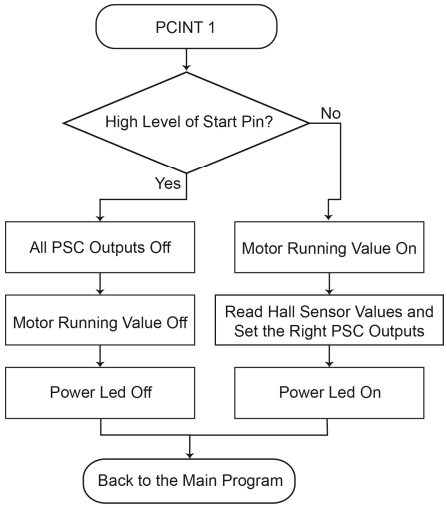

Fig. 7. Flow diagram of start and stop motor interrupt

F. *Configuration file*

In this file there are the constants that the users must set depending on their needs. For the UART module, they must set the baud rate, the number of stop bits, and the number of bits to transmit that they will use in the serial connection to the PC.

Another aspect that must be configured is the speed constant of the motor. This constant is calculated using the equation (1). This equation depends on the value, in seconds, set for the timer 0 (*t_timer0*), the maximum speed (*max_speed*), in revolutions per minute, and the number of pairs of poles of the BLDC motor used.

$$speed\_const = \frac{60 \cdot 255}{n \cdot t\_timer0_{(s)} \cdot \max\_speed_{(rpm)}} \quad (1)$$

A limited data in a range of 0 to 255 (8 bits) from the measured value of the motor speed is got by this constant. This data will be the value sent to the monitoring software to display the motor speed.

G. *ATmega64M1 programming*

Fig. 8 shows the flash memory map of the ATmega64M1 µC used in the project. The application code occupies a memory space of 10.8 kB. It is observed that there is clearance in memory for future expansion of the project.

Microcontroller programming is done by the Atmel Studio 6.0 software [6]. This software can be downloaded for free from the µC manufacturer's web and it is used to develop and compile codes made in C/C++ or assembly language.

A low-cost BLDC motor controller has been developed to carry out the tests in this project [7]. The application testing has been performed by this controller (Fig. 9). On the other hand, the load of the program has been carried out through the ISP (In-System Programming) interface of the µC and the AVRISP mkII programmer device [8].

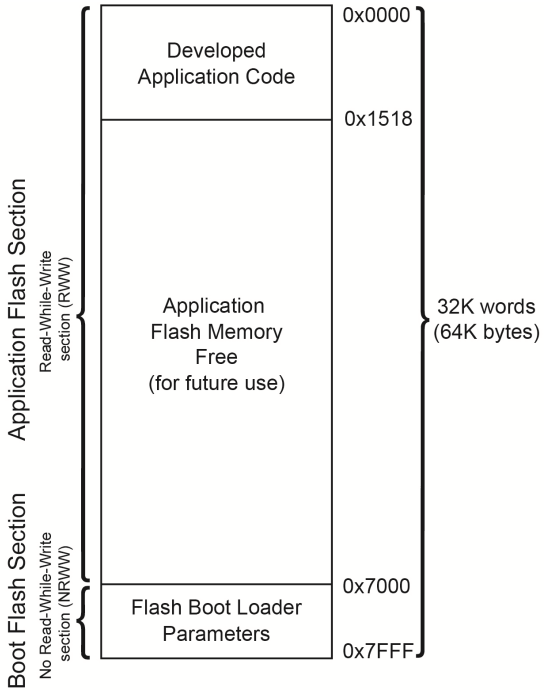

Fig. 8. Flash memory map of the ATmega64M1

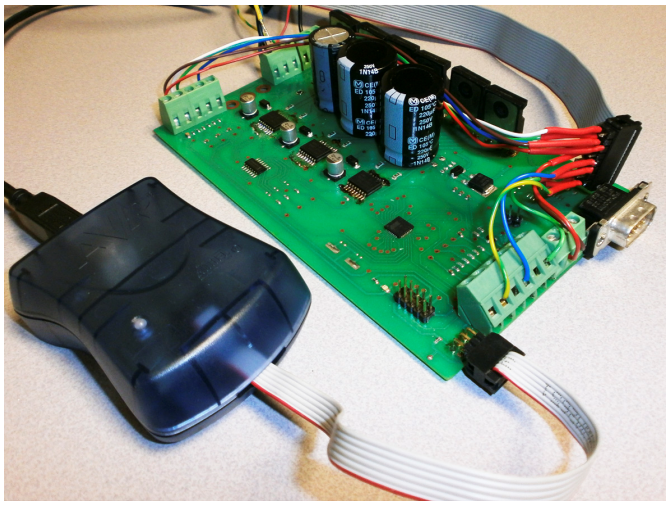

Fig. 9. View of the BLDCM controller and the AVRISP mkII programmer

III. Control and Monitoring Software

In this section, the developed software, using NI LabVIEW [9], for the control and monitoring of the BLDC motor from a PC is described. The communication protocol used is the RS-232. The software consists of two parts: a control and monitoring panel of the BLDC motor and a configuration panel of the UART connection and the constant for calculating the motor speed in revolutions per minute.

A. Control and monitoring

In this panel there are the controls and the display of revolutions per minute of the motor (Fig. 10). It has controls on and off (ON / OFF), change of direction of rotation (CCW / CW) and a virtual potentiometer for speed adjustment. These virtual controls have been incorporated to give the possibility to control the motor from the application. The configuration panel has a switch that enables these controls. If this switch is set, the physical controls of the test bench will be locked.

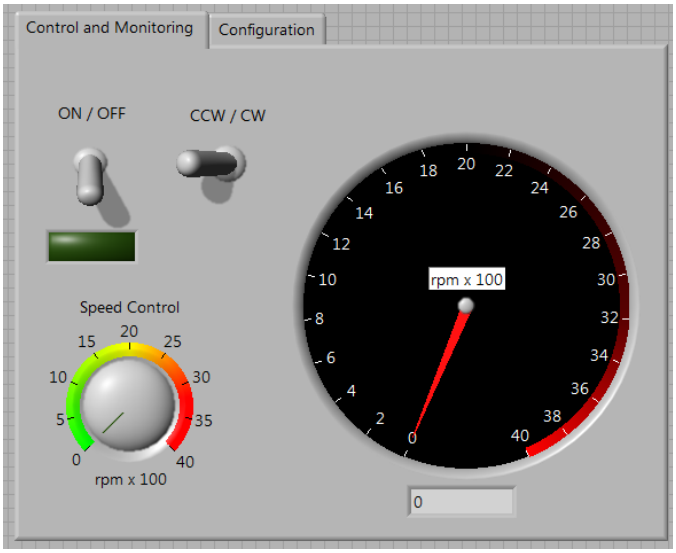

Fig. 10. View of the control and monitoring panel

B. UART configuration

This panel shows the controls to configure the serial port connection of the PC to the µC (Fig. 11). The parameters to configure this protocol are the following:

- Serial port used on the PC.
- Baudrate used in the connection.
- Number of bits per packet in the connection.
- Type of parity.
- Number of stop bits.
- Flow control used in the connection.
- Start transmission character (*XON*).
- Stop transmission character (*XOFF*).
- End of frame character.
- Maximum wait time in milliseconds (*Timeout*).

Furthermore, there is a field where must be entered the *alpha* constant used to find the real value of the motor speed. This constant is calculated by the expression (2), where *n* is the number of pairs of poles of the BLDC motor, *speed_const* is the constant calculated above and *t_timer0* is the timer0 value in seconds.

$$alpha = \frac{60}{n \cdot speed\_const \cdot t\_timer0_{(s)}} \tag{2}$$

The real speed of the BLDC motor is calculated by the expression (3), where *alpha* is the constant obtained in equation (2) and *measured_speed* is the 8-bit value of the measured speed received by the serial port.

$$real\_speed_{(rpm)} = alpha \cdot measured\_speed \tag{3}$$

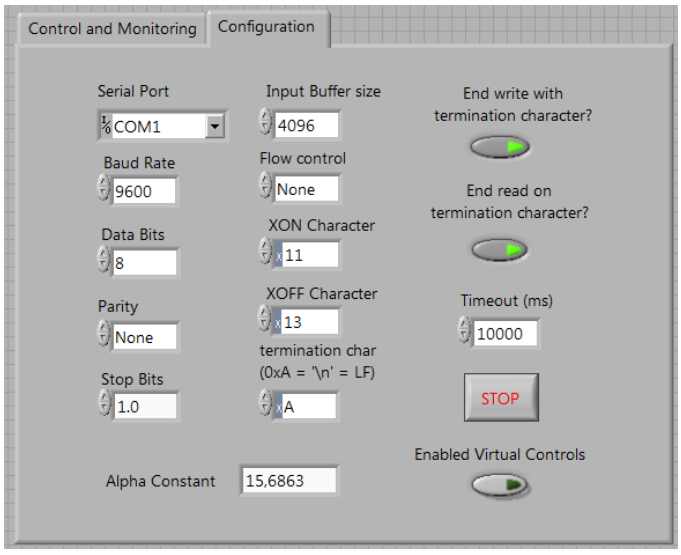

Fig. 11. View of the configuration panel

CONCLUSIONS AND ONGOING WORK

Nowadays this type of BLDC motors are increasingly used due to their high performance and excellent features, replacing traditional DC motors (Brushed Direct Current Motors). Hence, students must acquire knowledge in this field. By this system, they can learn to make mathematical calculations of the BLDC motor control and, moreover, settings and calculations to program the μC simultaneously. In this way, students will be more motivated and get a greater benefit from the practices.

A possible future action to expand this project is implementing the functions for speed and current loop control of BLDC motors. Thus, students will be able to learn everything about the PID control (Proportional Integral Derivative controller) of a BLDC motor.

On the other hand, the basic functions of the CAN protocol may be used in future as communication system for control and monitoring of the motor. This protocol is more robust and reliable than the RS-232 one. Students will be able to do internships and understand the operation of this communication bus by implementing the necessary functions to configure the CAN module of the microcontroller. After, the application layer would be added to the project using CANopen [10].

Multiple controllers are being currently manufactured for use in practice the next academic year 2014/2015.

REFERENCES

[1] Atmel ATmega16/32/64M1 microcontroller's family: http://www.atmel.com/devices/ATMEGA64M1.aspx, Last accessed 2013, December.

[2] Wang Dongmei, Guo Haiyan, and Yu Jing, "Modeling and Simulation Research of Brushless DC Motor open-loop Speed-adjustment System", The 2nd International Conference on Intelligent Control and Information Processing, ISBN 978-1-4577-0816-9, pp.: 394 – 398, 2011.

[3] Alphonsa Roslin Paul, and Prof. Mary George, "Brushless DC motor control using digital PWM techniques", Proceedings of 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN 2011), ISBN 978-1-61284-653-8, pp.: 733 – 738, 2011.

[4] Hai-tao Wang, Ze Zhang, and Xiang-yu Liu, "Design of control system for brushless DC motor based on TMS320F28335", Third International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), ISBN 978-1-4244-9010-3, pp.: 954 – 958, 2011.

[5] Radu Duma, Petru Dobra, Mirela Dobra, and Ioan Valentin Sita, "Low cost embedded solution for BLDC motor control", 15th International Conference on System Theory, Control, and Computing (ICSTCC), ISBN 978-1-4577-1173-2, pp.: 1 – 6, 2011.

[6] Atmel Studio 6.0 Manual. [Online]. Available: http://atmel.no/webdoc/atmelstudio/

[7] H. A. Fabelo, J. M. Cabrera, A. Vega, and V. Déniz, "A Low-Cost Control System for BLDC Motors Applied to Teaching", XI TAEE, 2014.

[8] Atmel AVRISP mkII microcontroller programmer: http://www.atmel.com/tools/avrispmkii.aspx, Last accessed 2013, December.

[9] NI LabVIEW: http://www.ni.com/labview/esa/, Last accessed 2013, December.

[10] CANOpen protocol: http://www.can-cia.org/, Last accessed 2013, December.