BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 63 (67), Numărul 2, 2017
Secția
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ

ALTERNATIVE SOLUTION FOR MAJORITY FUNCTION USED BY MICROCHIP FOR BRUSHLESS DIRECT CURRENT SENSORLESS MOTOR CONTROL

BY

EUSEBIU BIVOL, CRISTIAN AGHION[*] and OVIDIU URSARU

Technical University “Gheorghe Asachi” of Iaşi,
Faculty of Electronics, Telecommunications and Information Technology

Received: May 18, 2017
Accepted for publication: June 21, 2017

Abstract. Nowadays, motors are more and more used in various fields where the number of electronics components tends to overcome mechanical components, *e.g*. automotive industry. The main targets in developing methods for motor control are to obtain a better control, smaller loss, increasing efficiency, low power use, reduced number of components and all of them must be integrated in a cost-effective design. At this moment, there is a vast field of techniques for controlling all kinds of machines in all manners we want.

Key words: Power electronics; PWM control; BEMF sensing; Switching circuit.

1. Introduction

In the following article a comparison will be presented, between the majority function technique used in the application note AN1160 (AN1160 Microchip, 2008) as a non-linear digital filter and our proposed solution for accuracy speed control of an BLDC motor (Bose, 1987; Dimitriu, 2003), without position sensors (*e.g*. Hall, Encoder), using only BEMF (Back ElectroMotive Force) voltage (Cetin & Sazak, 2009).

In AN1160 a sensorless BLDC (Brushless Direct Current) motor control algorithm is described, implemented on one of their platform (Duma 2001; Duma, 2004). This technique consists in four main steps:

[*]Corresponding author: *e-mail*: aghion@etti.tuiasi.ro

- sampling the BEMF signal, which has a trapezoidal form, using microcontroller's A/D converter;
- comparing converted signal to reconstructed neutral point to detect zero crossing points;
- filtering the output signals of the comparator to eliminate false zero cross detections;
- commute the motor driving voltages.

For a better understanding of the algorithm we will further discuss the working principle of BLDC machines (Neacşu, 2013).

In Fig. 1, a simplified illustration of a Brushless Direct Current Motor construction is exemplified.

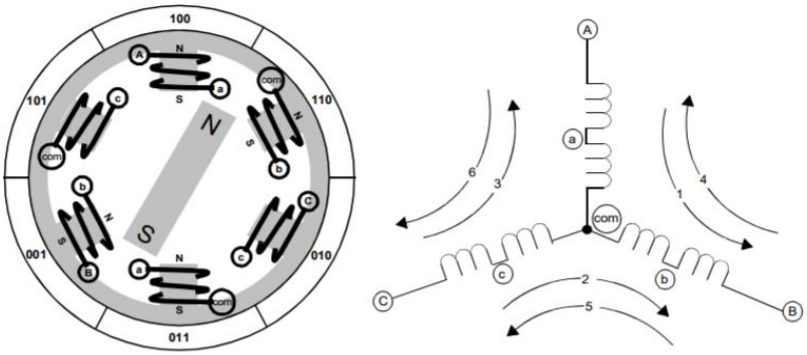

Fig. 1 – Simplified BLDC construction scheme.

A BLDC motor is composed by two parts: the static one called the stator and the moving one called the rotor. In the above example the rotor is a permanent magnet with one pole pair and the stator has three pairs of wounded wires tied around the rotor (Neacşu, 2013 CRC Press; Raţă, 2012).

As the supply current flows through the stator coils, they are generating a rotating magnetic which is converted into mechanical energy by attracting the permanent magnet rotor and making it rotate (Aghion, 2010; Valachi, 2009).

Shown in the left part of Fig. 1, is the electrical circuit representing a standard three phase topology in star connection (Erfidan, 2008). This topology consists in driving the motor by energizing two phases at a time (Hava, 1998; Aghion, 2010). For example, the static alignment shown in the right diagram can be achieved by creating a path for current from point A to point B (path 1 noted in the figure). Next, by changing the current flow path from C to B (noted as path 2) the rotor can be made to rotate clockwise 60 degrees. In practice, maximum torque can be obtained when the rotor is at 90 degrees away from the alignment with the stator magnetic fields (Comsa, 2012).

The main task in driving a BLDC motor is to sense the rotor position and to energize the next pair of phases that will produce the most amount of torque. The transition from a pair of phases to another is called commutation.

One commutation step is equal to 60 electrical degrees. There are only six possible combinations of electrical connections and commuting through all of them at the rights moments will pull the rotor through one electrical revolution.

Giving a 3-phase BLDC motor, in Fig. 2 the phase input signals are presented.

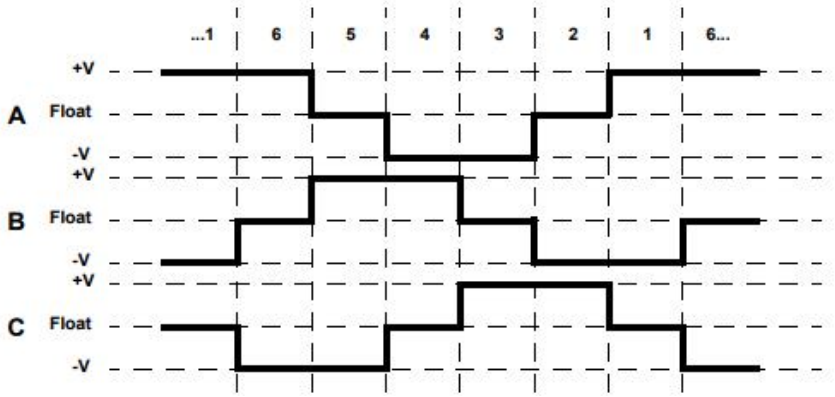

Fig. 2 – Phase input signals in trapezoidal control of a BLDC motor.

Depending on number of rotor's magnetic pole pairs, one electrical revolution could be equal to one mechanical revolution if the rotor has only one magnetic pole pair (Bârleanu, 2012).

The easiest way to obtain the commutation moments is by using position sensors. The most common solution practiced by motor manufacturers is including Hall Sensors composed of three elements, each delivering to the output a digital signal with high level value for 180 electrical degrees of one rotation and low level value for the other 180 degrees as it is presented in Fig. 3.

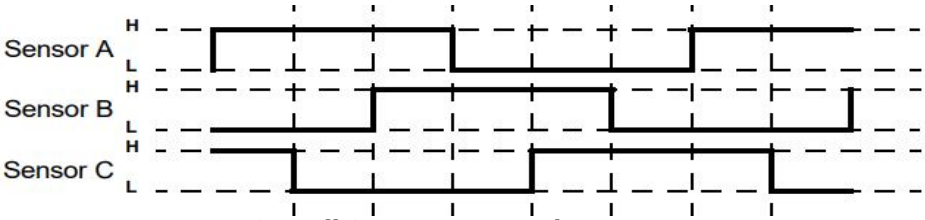

Fig. 3 – Hall Sensors outputs with respect to Fig. 2.

In Fig. 4 is presented an inverter used to provide the supply voltage to motor terminals thereby: each sector consists of one motor terminal driven high, one terminal driven low and one motor terminal left floating. To separately drive each phase to high, low or floating point, it is needed to individually control each transistor (Ursaru, 2009).

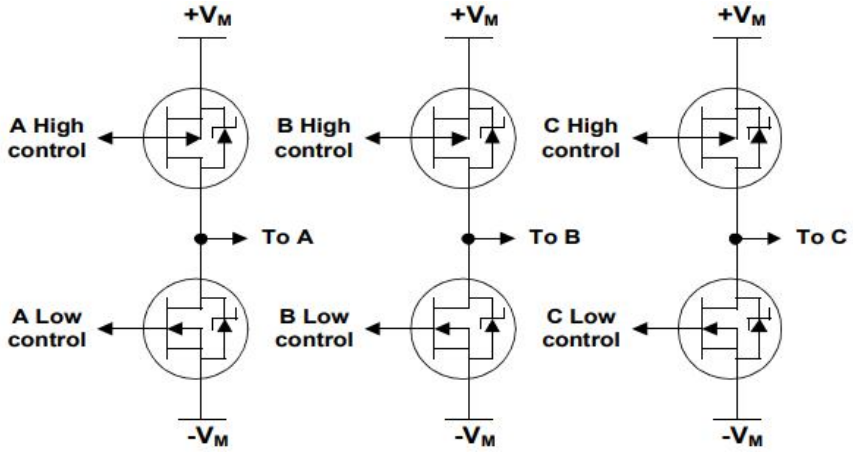

Fig. 4 – Three phase inverter.

2. BEMF Sensing

The use of Hall Sensors implies a higher cost for the system. If the low cost is a primary objective, sensorless control may be a better choice, but in exchange for a low price there comes some disadvantages such as no control at low speed and unknown behavior at fast load changes.

Finding the right moment to commutate the motor voltages is possible by sensing the voltage on the undriven motor terminal during one of the drive phases. The measured voltage is called BEMF voltage.

The BEMF is a voltage that opposes the change of the current that induced it. For example, the voltage drop on a coil is determined by the fluctuating current flow through it, as the voltage formula through an inductor is:

$$U = L \times \mathrm{d}I/\mathrm{d}t. \tag{1}$$

Suddenly changing the direction of the current I will cause an overvoltage drop called BEMF. The same theory applies to the motor windings which basically are coils. Not driving one winding means suddenly cutting the current flow through it resulting in returning a BEMF voltage opposing the main voltage supply. BEMF calculation formula is:

$$\mathrm{BEMF} = NlrB\omega, \tag{2}$$

where: N is the number of windings per phase; l – length of the rotor; r – internal radius of the rotor; B – rotor magnetic field; ω – angular velocity.

By considering the rotor magnetic field constant, the only variable remains the angular velocity. Therefore the magnitude of the BEMF voltage is directly proportional to the speed of the motor. Because of this the control is nearly impossible at low speed, making the sensorless control useful only in

applications that do not require good control at low speed values. The higher the frequency of the commutations is, the higher the mechanical speed of the motor will be.

For an efficient drive of the motor the voltage applied to windings must be modulated to not waste any amount of energy. If the voltage is applied at it's maximum value constantly the motor will be driven inefficiently and the energy will be unnecessarily wasted. The solution is to use PWM (Pulse Width Modulation) transferring only a percentage of the total supply power to the motor. This way, there will be less dissipated energy on the windings and the functioning will be more efficient.

The most important question at this point is how to use BEMF voltage to do a sensor job. By analyzing the voltage and current on a winding we can observe that for maximum torque both signals must be in phase.

The intersection between voltage and current is called zero crossing. This point is where the BEMF crosses it's middle value and it's equal to half of the supply voltage value. Zero crossing occurs at 30 electrical degrees from the end of the last commutation, which is also at 30 electrical degrees from the start of the next commutation. A detection of a zero crossing can trigger a precise schedule to calculate the exact time where the next commutation should occur.

There are multiple methods to detect the zero crossing. A practical one is by comparing the BEMF voltage to the neutral point of the motor. As a physical connection to the neutral voltage of the motor is usually not available it has to be reconstructed. Hardware, this can be done by connecting three resistors in parallel, with one terminal connected to a motor winding and the second one to the other resistors, as described in the Fig. 5.

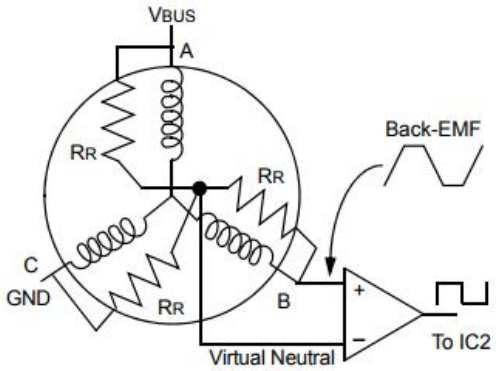

Fig. 5 – BEMF signal compared to a virtual neutral point of the motor.

This method will attract more components which implies more money. The alternative is to recreate the neutral voltage in a software manner. This can be done by reading the three phase voltages and averaging them. This value compared to the BEMF signal will give the exact moment of zero crossing events.

3. Filtering Zero Crossing Detection Events

As we mentioned above, the PWM method is used to apply only the voltage we need to run the motor. By looking at Fig. 3, the BEMF voltage is severely affected by PWM commutations. The ripple induced by fast switching of the transistors can cause faulty zero crossing detections.

In application note AN1160 the filtering method used is called majority function and it is, as described there, "a nonlinear digital filter".

Digital filtering eliminates the need of an external hardware filter which implies a higher cost for the system.

Moreover, a hardware filter will insert a delay in the signal which will need special treatment by the software code.

Here is an example of how majority function works with 3 inputs (Table 1).

Table 1
Example of a Majority Function Using Three Inputs

A	B	C	Majority
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

The formula used to calculate the majority is:

$$Majority = (A \,\&\, B) \mid (A \,\&\, C) \mid (B \,\&\, C) \tag{3}$$

where: & is the bitwise AND operator; | – bitwise OR operator.

Basically, *Majority* is equal to the most frequent value that appears in the equation. It is very important to know that being a digital filter, majority functions accepts only digital inputs (logic 1's and logic 0's). The Microchip method involves some pre-filtering preparations which determine on which phase we have active BEMF signal and the sign of the slope of the BEMF signal. In the software code this is done by using two masks vectors called ADC_MASK and ADC_XOR:

```
const unsigned int ADC_MASK[8] = {0x0000, 0x0002, 0x0001, 0x0004,
0x0002, 0x0001, 0x0004, 0x000};
const unsigned int ADC_XOR[8] = {0x0000, 0x0000, 0xFFFF, 0x0000,
0xFFFF, 0x0000, 0xFFFF, 0x0000};
```

If the result of the masking is a true value, then the filtering is prompted to find if the extracted sample satisfies the conditions of a true zero crossing detection. An array of 64 elements is initialized at the beginning of the code.

In the application note, after a zero crossing detection, the filter check the next three samples to see if they meet a true condition of detection.

The formula used to calculate the array elements is:

First Half: $Array\ Value[N] = N * 2$
Second Half: $Array\ Value[N] = (N - 32) * 2$

The resulting array is presented in Table2.

Table 2
Majority Function Array

Array Index [N]	Array Value	Array Index [N]	Array Value
0	0	32	0
1	2	33	2
2	4	34	4
3	6	35	6
4	8	36	8
5	10	37	10
6	12	38	12
7	14	39	14
8	16	40	16
9	18	41	18
10	20	42	20
11	22	43	22
12	24	44	24
13	26	45	26
14	28	46	28
15	30	47	30
16	32	48	32
17	34	49	34
18	36	50	36
19	38	51	38
20	40	52	40
21	42	53	42
22	44	54	44
23	46	55	46
24	48	56	48
25	50	57	50
26	52	58	52
27	54	59	54
28	56	60	56
29	58	61	58
30	60	62	60
31	62	63	62

In the software code it translates in:

```
const unsigned char ADC_BEMF_FILTER[64] =
        {0x00,0x02,0x04,0x06,0x08,0x0A,0x0C,0x0E,0x10,0x12,0x14,0x16,0x
        18,0x1A,0x1C,0x1E,
        0x20,0x22,0x24,0x26,0x28,0x2A,0x2C,0x2E,0x01,0x01,0x01,0x36,0x0
        1,0x3A,0x3C,0x3E,
        0x00,0x02,0x04,0x06,0x08,0x0A,0x0C,0x0E,0x01,0x01,0x01,0x16,0x0
        1,0x1A,0x1C,0x1E,
        0x01,0x01,0x01,0x26,0x01,0x2A,0x2C,0x2E,0x01,0x01,0x01,0x36,0x0
        1,0x3A,0x3C,0x3E};
```

Only 16 of these elements represent a true condition, that concludes to real zero crossing event detection. These elements get the value 1 and when reached, the filtering ends and the program is ready to take another sample.

These elements together with their values are presented in Table 3.

Table 3

Index Numbers Indicating a True Condition

Number	6-Bit Binary Value
24	011000
25	011001
26	011010
28	011100
40	101000
41	101001
42	101010
44	101100
48	110000
49	110001
50	110010
52	110100
56	111000
57	111001
58	111010
60	111100

```
if((ComparatorOutputs^ADC_XOR[ADCCommState])&
ADC_MASK[ADCCommState])
        adcBackEMFFilter|=0x01;
//Majority detection filter
adcBackEMFFilter = ADC_BEMF_FILTER[adcBackEMFFilter];
if (adcBackEMFFilter&0b00000001)
        PreCommutationState();
```

The software works exactly as described above. The variable *ComparatorOutputs* stores the logic value resulted by comparing motor phases voltage values with neutral voltage value and memories the results on the three last significant bits, each bit for each phase. If the result is "1" it means that the BEMF value was higher than neutral voltage value.

Then, the XOR mask establish if the detection was on the falling edge or on the rising edge and the AND mask is used to clear bits that not correspond to BEMF signal. The index used to select the masks from vectors is also used to select the commutation state, so we can anticipate the phase with BEMF signal.

If a BEMF signal is successfully detected the filtering begins with the steps mentioned above and when the filter outputs the value 1 the commutation sequence begins.

4. Filtering without Majority Function

In order to simplify the majority function theory, we replace it with a basic median filter. This filter takes a number of samples (of logic 1's and logic 0's) given by the width of its window and returns the most frequent value, which is the same return as the majority function has. Median filters are usually used to eliminate impulse noise which is the same noise we have on our unfiltered BEMF signal, almost instantaneous sharp spikes.

Taking at least 3 samples from BEMF signal and comparing it with the neutral voltage value we obtain 3 output logic values. These values are used as inputs for the median filter and the return of it will be the true value of Zero Crossing Detection.

If we consider that our BEMF signal has a positive slope, we expect a zero logic output from the comparator after the Zero Cross Detection.

Using an AD converter we take 5 samples of a BEMF signal, three of them being the real value of the BEMF and two of them being values from spikes (not necessary in this order). It is very important to know the frequency of spikes because the sampling frequency must be higher. Noise frequency is given by switching frequency of the transistors from the inverter.

Comparing these samples with the neutral point value we will obtain 3 true values which are the correct ones, and 2 false values which are incorrect (output sequence example: 1,0,1,0,1). Without filtering the code will set the Zero Cross immediately when it detects the transition from logic 1 to logic 0.

We memorize these 5 values into a buffer and then call a function which returns the output of the median filter for these samples. Applying the filter's logic, the returned value will be logic 1, which is not the expected value we want to set a Zero Cross Detection so we do not execute the commutation yet. The larger the filter's window width is the more precise will be the output, but it will take more time to sample the signal and to calculate the output logic value.

With the explanations above, this is how the replacement code will look like:

```
if((ComparatorOutputs^ADC_XOR[ADCCommState])&
ADC_MASK[ADCCommState])
        buffer[++i] = 1;
else
        buffer[++i] = 0;
if(i>= 4)
        i = 0;
if(!MedianFilt(buffer))
        PreCommutationState();
```

We kept the masking process because it is a simple and clean way to determine signal's slope and phase. Then we added a buffer of *BUFFER_LENGTH* elements (in our case 5 elements), all initialized with 0, with variable *i*as an index, also initialized with value 0 and resetting when it reaches *BUFFER_LENGTH - 1*.

In the buffer we keep memorizing and overlapping the last 5 results after masking the comparator output value. If the masking result is true we add a 1 in the buffer marking a zero crossing detection (the validity of a true detection is unknown at this moment) and, respectively, if the result is false we add a 0 in buffer to mark that no detection happened.

Until the buffer is first filled with values after initialization we assume that no zero crossing detection occurred. After that, with every new value inserted in buffer a median filter function is called having as argument the same exact buffer filled with our values.

The *MedianFilt* function, which is called in the *if* statement, has the next definition:

```
intMedianFilt(unsigned char *buffer)
{
        inti,j,temp;
        for (i = 0; i< (5 - 1); ++i)
        {
                for (j = 0; j < (5 - 1 - i); ++j )
                {
                        if (*(buffer + j) > *(buffer + j + 1))
                        {
                                temp = *(buffer + j + 1);
                                *(buffer + j + 1) = *(buffer + j);
                                *(buffer + j) = temp;
                        }
                }
        }
        return *(buffer + 2);
}
```

In this function the buffer vector is covered element by element, being ordered ascending using the bubble sort technique.

This method of sorting consists in taking an element at a time and comparing it with all others elements giving it's exact place in the vector. After one element is placed the next one is taken and placed and so on till the last element is in place resulting in an ascending or descending order of the elements.

First for statement cover all buffer's elements except the last one because after ordering all elements, the last one is automatically placed in the single empty spot remained.

The second for statement cover buffer's elements that were not ordered yet. So it takes the first unordered element and compares it to the next ones and if it is bigger, the compared element is moved to the end of the vector step by step.

After the buffer is ordered it will be divided in two parts: left side will be occupied by 0 values and right side will be populated by 1 values. The middle value is the most frequent one and that is the explanation why we chose an odd value for filter's window width. This function returns the middle value of the ordered buffer which is the same result as majority function output described above.

5. Conclusions

In the vast field of motor control techniques, users tend to opt to easiest ways to understand and to personalize. The method described in this article, used as an alternative solution, is a simple, customizable and low price software solution which along with supported development platforms can provide a full package for an user to start developing his own personalized control. In each case is a win-win situation where the producer sold his product and the user can exploit his purchase at its best.

REFERENCES

Aghion C., Ursaru O., Lucanu M., *Software Implementation for ACIM Motor Control*, International Review of Electrical Engineering (IREE), **5**, *2*, 433-436 (2010).

Aghion C., Ursaru O., Lucanu M., *Three-Phase Motor Control using Modified Reference Wave*, Electronics and Electrical Engineering. – Kaunas: Technologija, *3*(*99*), 35-38 (2010).

Bârleanu A., Băițoiu V., Stan A., *FIR Filtering on ARM Cortex-M3*, Proc. of the 6th WSEAS European Computing Conf., *9*, WSEAS Press, 2012, 490-494.

Bose K.B., *Microcomputer Control of Power Electronics and Drive*, IEEE Press, New York, 1987.

Cetin S., Sazak B.S., *Triple Half Bridge Series Resonant Inverter for Home Cooking Applications*, Internat. Rev. of Electrical Engng. (IREE), **4**, *2*, 168-173 (2009).

Comsa C., Haimovich A.M., *Performance Bound for Time Delay and Amplitude Estimation from Low Rate Samples of Pulse Trains*, Signal Proc. Conf. (EUSIPCO), 2012 Proceedings of the 20th European, pp. 455-459.

Dimitriu L., Lucanu M., Aghion C., Ursaru O., *Control with Microcontroller for PWM Single-Phase Inverter*, Signals, Circuits and Systems, 2003. SCS 2003. Internat. Symp. on, **1**, pp. 265-268.

Duma P. *Microcontrolerul INTEL 8051.Aplicatii*, Ed. TEHNOPRESS, Iaşi, 2004.

Duma P., *Arhitectura sistemelor cu microprocesor. Microcontrolere*, Ed. Univ. Tehnice Gh. Asachi, Iaşi, 2001.

Erfidan T., Urgun S., Hekimoglu B., *Low Cost Microcontroller Based Implementation of Modulation Techniques for Three-Phase Inverter Applications*, Electrotechnical Conf., 2008. MELECON 2008. The 14th IEEE Mediterranean, pp. 541-546.

Hava A., *Carrier Based PWM-VSI Drives in the Overmodulation Region*, Ph. D. Diss., Univ. of Wisconsin, MADISON, 1998.

Neacşu D.O., *Power Converter Topologies with Reduced Component Count for Automotive AC Auxiliary Power*, Signals, Circuits and Systems (ISSCS), 2013 Internat. Symp. on, DOI: 10.1109/ISSCS.2013.6651170, INSPEC Accession Number: 13879807, 11-12 July 2013, Iaşi, Romania, pp. 1-4.

Neacşu D.O., *Switching Power Converters: Medium and High Power*, CRC Press, 2013/12/13

Raţă G., *The Study of the Deforming Regime of AC/AC Converter using Fourierand Multiresolution Analysis*, Electronics and Electrical Engineering, Kaunas, Technologija, *5*, pp.7-12, 2012, DOI: 10.5755/j01.eee.121.5.1643.

Ursaru O., Aghion C., Lucanu M., Tigăeru L., *Pulse Width Modulation Command Systems Used for the Optimization of Three Phase Inverters*, Advanced in Electrical and Computer Engng. J., Suceava, Romania, **9**, *1*, 22-27 (2009).

Valachi A., Timiş M., Danubianu M., *Some Contributions to Synthesis and Implementation of Multifunctional Registers*, 11th WSEAS Internat. Conf. on Automatic Control, Modelling & Simulation (acmos'09), Istanbul, Turkey, May 30 - June 1, 2009, pp.146-149.

* * * *Sensorless BLDC Control with Back-EMF Fitlering Using a Majority Function*, AN1160, Microchip Technology Inc., 2008

* * * *Brushless DC Motor Control Made Easy*, AN857, Microchip Technology Inc., 2002.

SOLUŢIE ALTERNATIVĂ LA FUNCŢIA DE MAJORITATE UTLIZATĂ DE MICROCHIP PENTRU CONTROLUL MOTOARELOR DE CURENT CONTINUU FĂRĂ PERII COLECTOARE

(Rezumat)

Motoarele de curent continuu fără perii colectoare (BLDC) oferă mai multe avantaje decât motoarele de curent continuu cu perii colectoare. Partea electronică de alimentare şi control, împreună cu senzorii de poziţie, înlocuiesc periile colectoare oferind astfel un timp de funcţionare mai lung al motorului, întreţinere scăzută şi zgomot datorat periilor colectoare – nul. Soluţia tehnică prezentată în acest articol a fost comparată cu cea descrisă în documentaţia Microchip An1160 făcându-se referire în special la simplitatea şi robusteţea soluţiei găsite de autori. Rezultatele au fost prezentate şi analizate prin comparaţie. Avantajul mare al acestei soluţii este ca un astfel de control al motorului BLDC este simplu şi uşor de implementat acolo unde este nevoie de control fără senzori de poziţie (encoder, Hall, etc.).