AN1113
APPLICATION NOTE

Brushless Motor Fuzzy Control by using ST52x301

Authors: G. Grasso, M. Di Guardo

INTRODUCTION

Brushless DC motors (BLDC) are becoming widely used in the field of control motors. These kind of synchronous motors are used as servo drives in applications such as computer peripherals equipment, robotics, and as adjustable-speed drives in load-proportional capacity-modulated heat pumps, large fans , compressors and so on.

Brushless DC motors are referred to by many aliases as brushless permanent magnet, permanent magnet AC motors, permanent magnet synchronous motors, etc. The confusion arises because a brushless motor does not directly operate off a dc voltage source. It is generally driven (supplied) from an inverter which converts a constant voltage to a 3-phase voltage with a frequency corresponding instantaneously to the rotor speed.

One of the advantages of BLDC motor is the sparks absence. The brushes of a DC motor have several problems as regards to brushes' life and dust residues, maximum speed and electrical noise. BLDC motors are potentially cleaner, faster, more efficient, less noisy and more reliable. However, BLDC motors require a more complex electronic control.

This application note will show how this complexity can be reduced by using ST52x301 Fuzzy controller.

AN OUTLINE OF BRUSHLESS MOTORS

The Brushless motor has the physical appearance of a 3-phase permanent magnet synchronous machine.

The brushes and commutator have been eliminated and the windings are connected to the control electronics. Electronics replaces the function of the commutator and energizes the proper winding. The energized stator winding leads the rotor magnet and switches just as the rotor aligns with the stator.

In synchronous motor drives, the stator is supplied with a set of balanced three-phase currents, whose frequency is f.

If p is the number of the poles in the motor, then:

$$f = \frac{p}{4\pi}\omega_S \quad (1)$$

where ω_S (rad/s) is the flux synchronous speed or, that is the same, the rotor speed. This equation links the rotor speed to the phases switching frequency of the electronic drive.

The above currents produce a constant amplitude flux ϕ_S in the air gap, which rotates at the synchronous speed ω_S. Since the flux amplitude is proportional to the current amplitude, it is enough to manage winding current level to control the rotor torque.

From Brushless theory [3-4] it is possible to demonstrate that

$$T_{em} = K_t \Phi_f I_{ph} \sin(\delta) \quad (2)$$

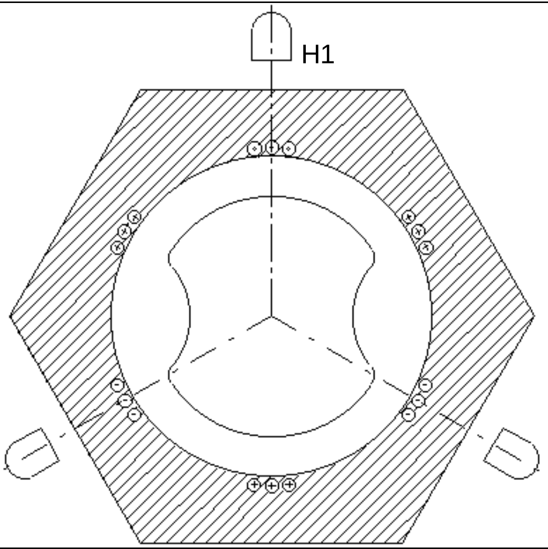
where k_t is a constant, ϕ_f is the field-flux, δ is called torque angle. δ represents the angle between the phase linked flux ϕ_{fph1} and the relative stator current I_{ph1}.

Now, if the electronic controller is able to supply phase *Ph1* in order to maintain δ=90° the equation above (2) can be simplified as

$$T_{em} = K_T\, I_S (3)$$

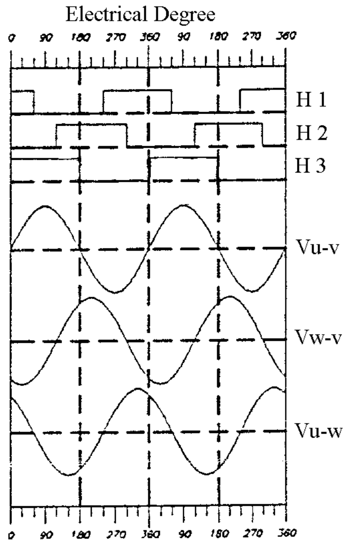
where K_T is called motor torque constant and $\mathbf{I}_S$ is the amplitude of the stator phase currents. From the previous considerations it is clear the reason why a BLDC motor needs always position sensors to know exactly rotor position. Fig. 1 shows a transversal section of a typical three phase brushless motor.

Figure 1 . Three-phase brushless motor



In the diagram, three position sensors (Hall sensors) are placed around the rotor in order to detect the shaft position. Fig. 2 reports motor data sheet supplied by the motor builder.

Figure 2. Motor Data Sheet



Linked Voltage for Counterclockwise direction

By observing the motor data sheet, these concepts become clear. In fact, it is possible to see the relation between shaft position, hall sensors response and voltage profile to be supplied.

This way to supply the stator phases is very complex because it is necessary to produce a sine wave with a proper period and delay with respect to the sensors information. As we will show later on, a simpler method is possible.

INVERTER DRIVER TOPOLOGY

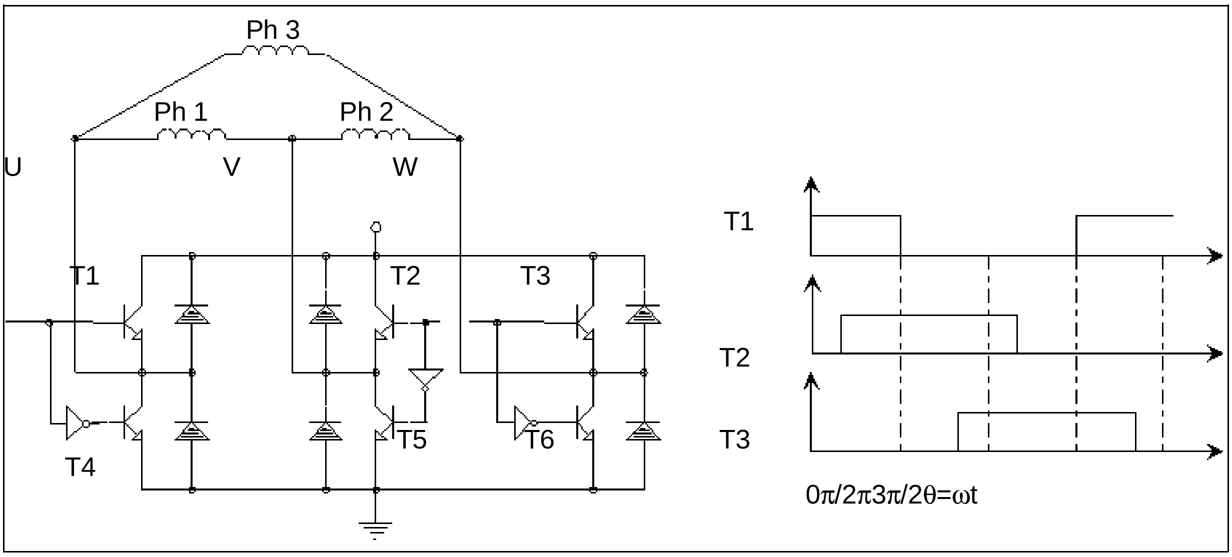
The stator of a brushless DC motor is generally supplied by an inverter which converts a DC voltage to a 3-phase AC voltage whose frequency is related to the speed of the rotor. Speed control is achieved by a Pulse Width Modulation (PWM) of the phases voltage accomplished by periodically switching the phase voltage to zero. The widely used driver to perform this switching is the six-step inverter where each phase is driven by means of a couple of transistors. Fig.3 shows the basic operating principle of this drive.

The name "six-steps" arises from the finite time-steps in which the whole period can be shared. During each time step, current direction does not change whereas current amplitude can increase or decrease in the coil.

To better explain this operating principle, let us consider the action of one leg (phase) of the inverter, such as for example T1 and T4.

Transistor T1 is turned on at θ= - 90°and turned off at θ=90° while T4 is turned on at θ=90°. When T1 is

Figure 3. Inverter driver operating principle



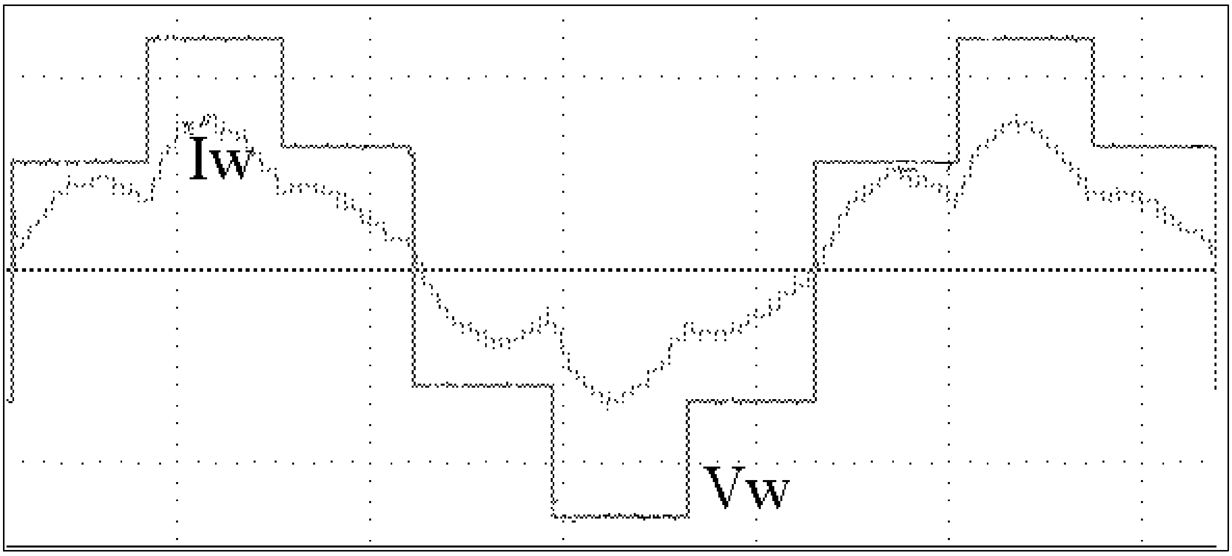
turned off, the current it was carrying is immediately diverted to the diode in parallel with T4. This diode re-circulates the instantaneous current of the winding until it decreases to zero.

Once the phase current reverses direction is carried by T4. In term of voltage it is easy to draw the phases voltages by conceiving the transistors as switches. Due to the triangular connection of the phases, each phase voltage depends by the status of two legs of the bridge. Figure 4 shows one real phase star voltage and one phase current. Six voltage steps are evident in the look like sine wave.

The same happens in the other legs of the bridge in different times. This topology drives the windings for the whole period, avoiding the phase to be "floating" (Six-Steps Continuous Mode Inverter).

FUZZY CONTROLLER

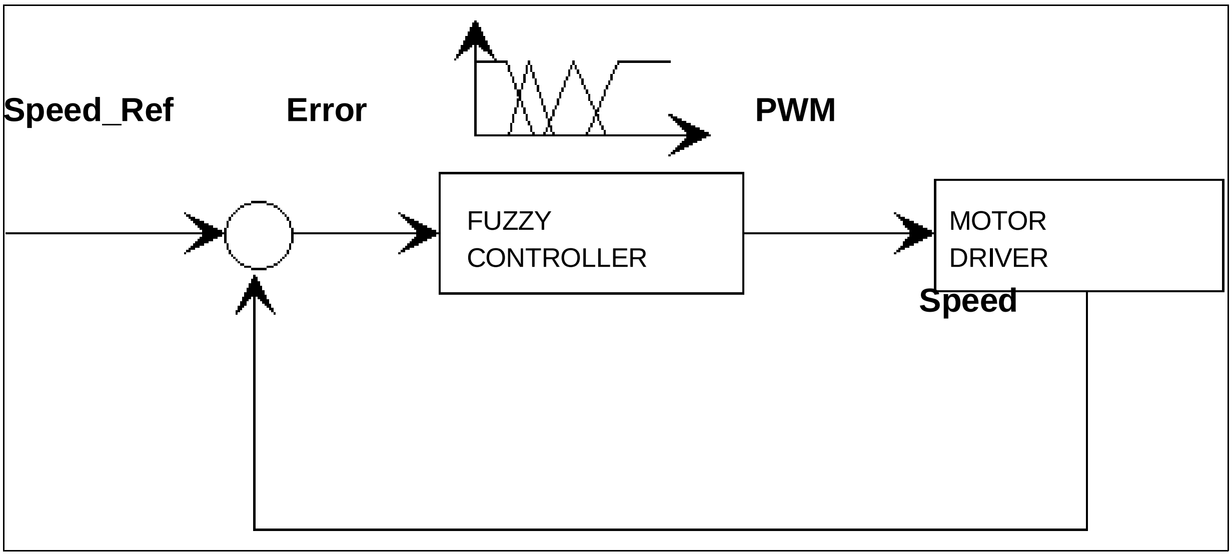
Figure 4. Phase voltage and current



The aim of the control is to maintain the desired Speed regardless of the applied load to the shaft. When a resistance torque is applied to the shaft, a reduction of the speed takes place. This implies an increment in the wave period of the Hall sensors signals and then a decrement of the sine frequency in the phases, to follow sensors information.

In this case, the only way to lead the rotor at the previous speed is to increase the winding current in order to balance the load torque. Fuzzy Controller, in fig. 6, performs this task. ST52x301 reads the speed

Figure 5. Control topology



"Ref" value from AD Channel0 and the instantaneous Hall signal period by means of a digital port. A software task performs the "error" calculation

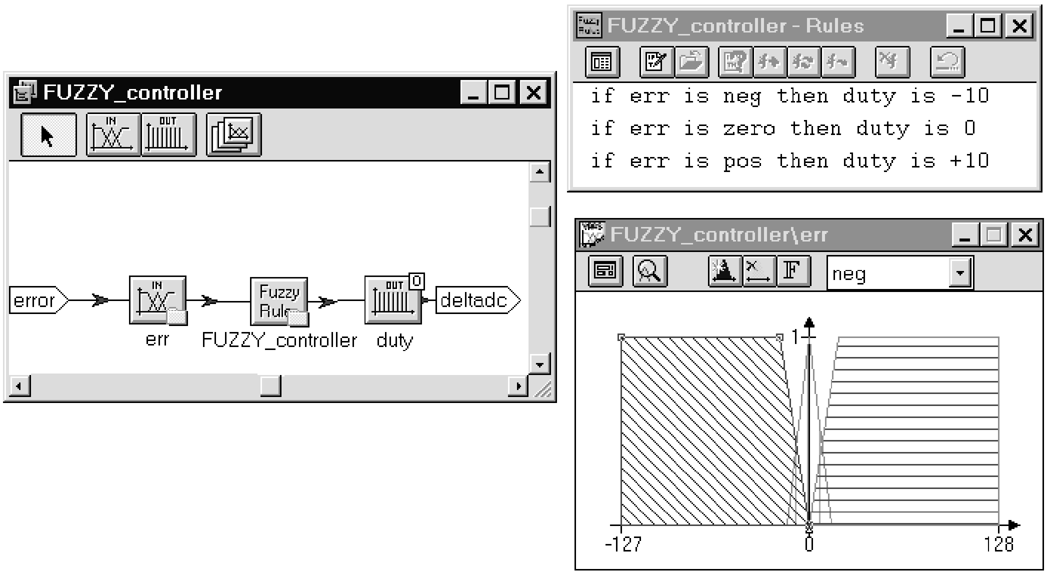
$$error = VREF - speed.$$

The variable "*Error*" also forms the Fuzzy Input for the "*Fuzzy Controller*" block. A Fuzzy algorithm uses three rules to compute the fuzzy-out, achieving an incremental variable to drive the inverter in real-time mode.

This incremental method allows to manage the speed in a closed loop real-time control, since software task time is very short.

The first rule is fully activated when "error" value is "neg", i.e. when Vref <<speed. This implies that the actual speed of the shaft is higher than "Ref". Then the action to carry out is to reduce the phase current

Figure 6. Fuzzy algorithm



I. To achieve this, it is necessary to decrease the PWM duty-cycle. The displayed value "-10" is a good compromise between system stability and step response of the system. This value was assigned after some trials by using only human reasoning. Instead, If the duty step is higher, for example "-20", the system will sooner reach the "Speed_Ref" but the overshoot in a step response could lead to instability. The above explanation is similar for the other rules.

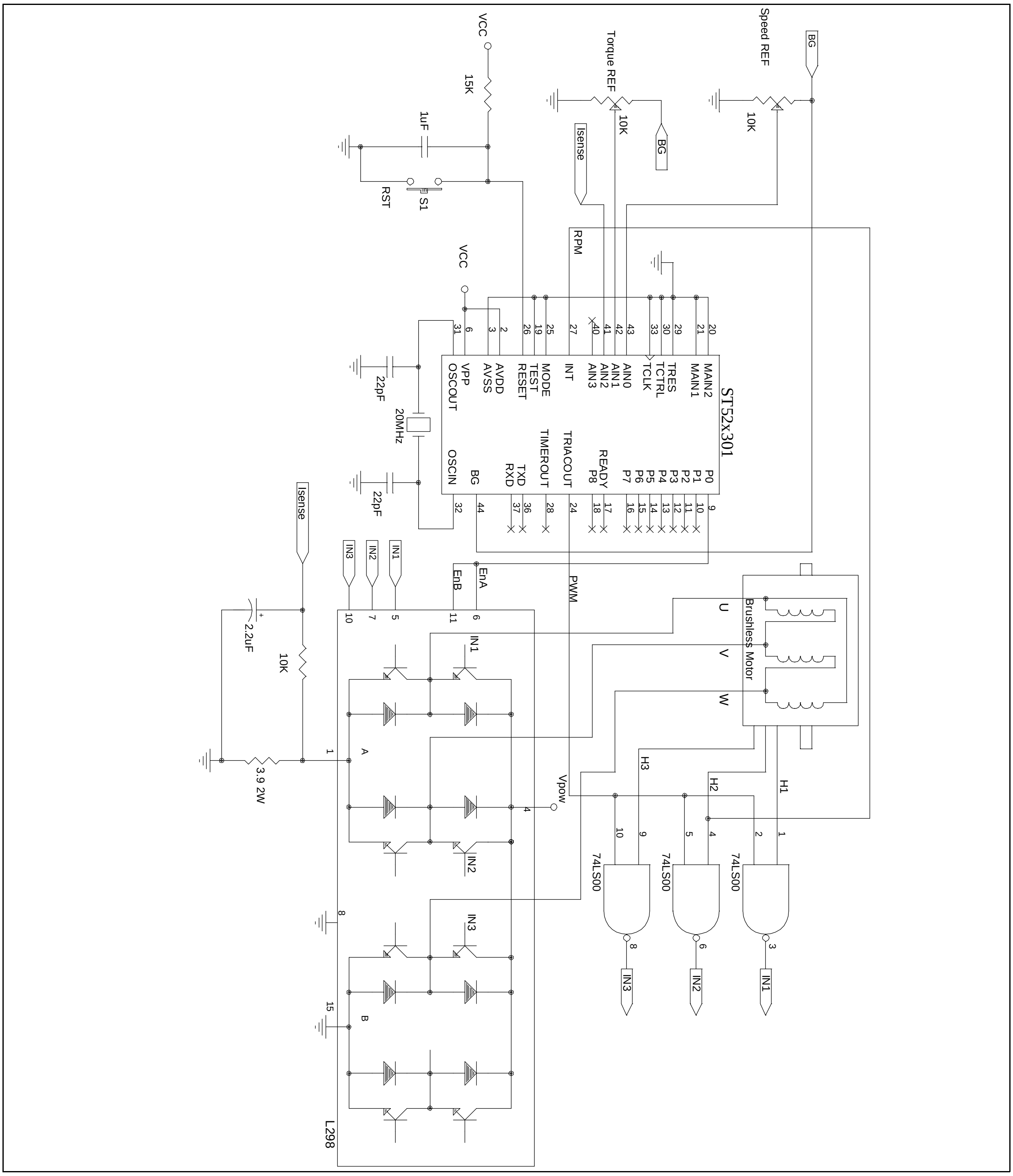
The implemented 3-rules fuzzy algorithm is the simplest way to control the speed. A more accurate control could be done by using first derivative of the speed to improve the action strenght by means of other rules.

HARDWARE DESCRIPTION

The real implemented system is shown in the schematic below. ST52x301, the integrated full-bridge dual drive and three AND’s are enough to control the BLDC motor.

The basic idea used in this implementation employs the Hall sensors in a direct way. The Hall signals are "AND-ed" directly with a PWM wave produced by ST52x301 in order to supply the proper winding. In

Figure 7. Electrical schematic



fact, a high frequency PWM wave produces, in a coil, a voltage whose amplitude value is the mean value of the square wave.

The motor data sheet displayed in fig. 2, clearly shows that "U-W" phase must be supplied positive when sensor "H1" is high, "W-V" when sensor "H2" is high and so on. This is true because the triangular connection was preferred in the coils arrangement. AND output is, then, a Pulse train whose duration is the same of the Hall signal. This pulses train is used to drive each leg of the bridge L298.

L298 is a monolithic dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Enable input signals are available to allow a software protection. Internal circuitry provides the appropriate dead-time in order to avoid a "cross-conduction" along the leg.

ST52x301 provides, by means of an internal peripheral, a PWM wave that can be varied by software. ST52x301 PWM frequency is chosen as compromise between acoustic noise in the motor and losses in the power stages of the bridge. A 19 KHz wave frequency was used in the implemented application.

SOFTWARE DESCRIPTION

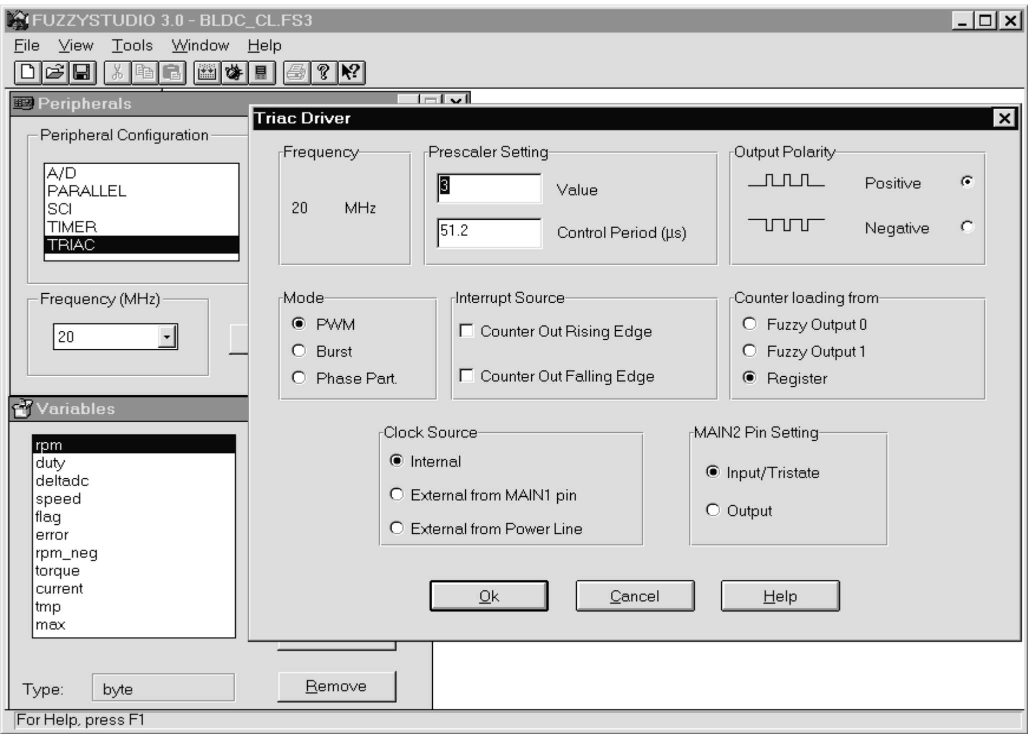
Before to discuss about ST52x301 software configuration, it is important to note some HW connections in the schematic. Bit "0" (pin 9) of the parallel port is used to enable the power stage only after power-on reset, then parallel port must be configured in OUT mode. The analog input AIN0 (pin 43) is used to read the voltage reference. A voltage between 0 + 2.5 V present on this pin, is converted in the range 0 + 255.

External INTerrupt pin (27) is used to read one Hall sensor signal period in order to calculate the instantaneous speed. This digital input will be configured both in negative or positive edge trigger to produce an internal software interrupt.

Fig. 8 displays how to configure A/D peripheral with 3 inputs, TRIAC peripheral in PWM mode at 19 KHz, the used global variables.

The following figure reports the main program in term of graphical programming. The appendix at the

Figure 8 . Peripheral configuration

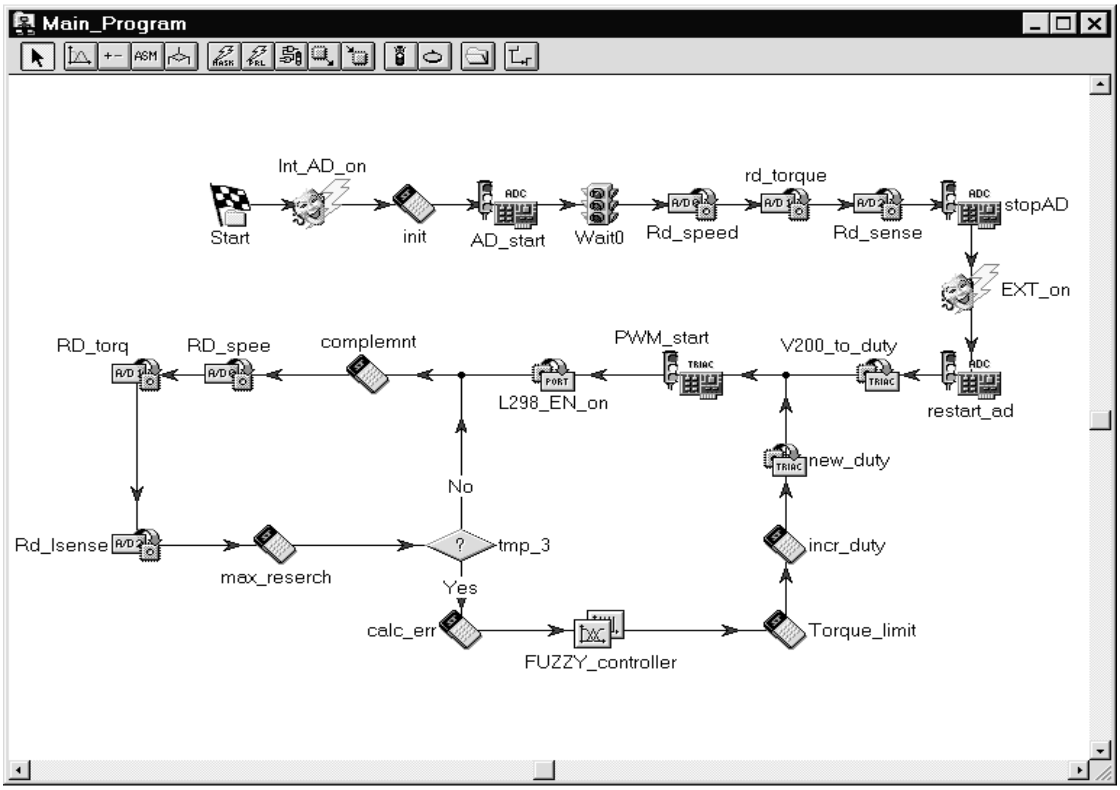


end of this application note contains the whole assembler code generated by the compiler.

Let us discuss about the main program. "*Int_AD_on*" and "*init*" blocks in fig. 9 are used to initialize global variables and interrupts mask (only AD Int is enabled). The two following blocks start the converter and wait for the results of the conversion. After that, the converter values are stored in the variable called "speed, "torque", "current" and a new Interrupt mask is enabled (only Ext Int).

Block "*V200_duty*" loads the Triac counter with a default value and the block "*PWM_start*" allows the Tri-

Figure 9 . Main view



ac peripheral to run. At this time it is already possible to see a PWM wave on pin 24 of ST52x301.

Since pin 27 reads a time period related to the speed it is necessary to perform a mathematical inversion to achieve the frequency. Just to simplify, a complementation of the period byte will be made instead of the inversion. In this way, an error will be introduced in the spin frequency calculation. If a more accurate precision is requested, an alternative method to implement the inversion is to use a fuzzy implementation of the function 1/T. The block "L298_EN_on" enables the bridge driver.

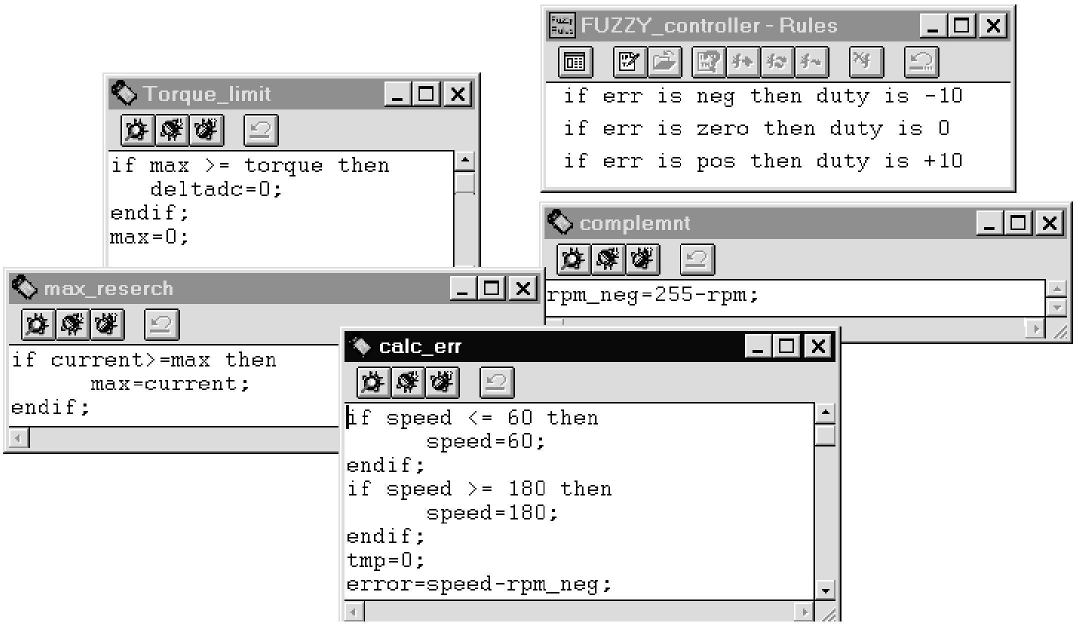
The Block "*complemnt*" performs the above task. Following the loop, a new value of the speed Ref "*torque_ref*" and "*current*" are read. The block "*max_research*" catches the maximum value of the current during a turn of the shaft. This loop is performed until the condition "tmp>=2" is false. Variable tmp is used to create a time inertia in the fuzzy control. "Tmp" is incremented each time the Ext-Interrupt routine is executed (each 1/3 turn of the shaft) and this implies a real time control in about one turn of the shaft.

Block "*calc_err*" performs error calculation as "*error = speed - ref - speed* ", error variable is sent to the fuzzy input, Fuzzy block "*fuzzy controller*" produces the incremental value "*delta_DC*".

Next figure shows the content of the 5 mathematics blocks. In the second, "delta_DC" is added (or subtract, if negative) to the current duty-cycle before to refresh Triac counter. The operators *If .. then* are introduced to avoid overflow or underflow in the counter registers during the control.

Key point in the program is the Ext_Int routine (fig. 11). This task performs the period measurement of

Figure 10 . Arithmetic blocks

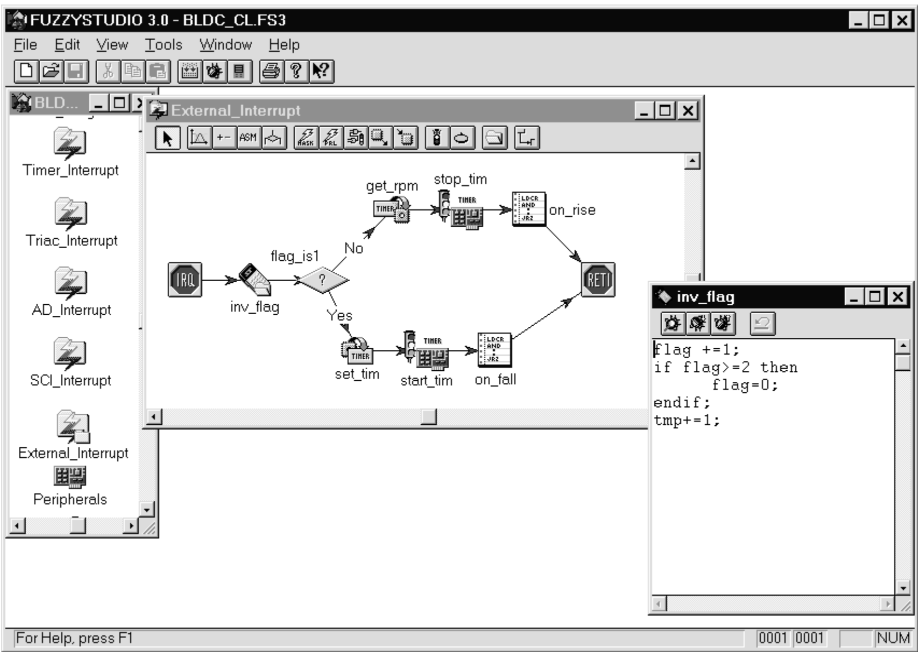


the square wave supplied by one Hall sensor. The period is measured by counting the time between a positive edge and the following negative edge of the H2 sensor signal.

A flag is used to select the edge. If the coming edge is positive, the Timer peripheral will be started. If negative, the current Timer counter value will be read and the Timer stopped.

Before to escape from Ext_Int routine two assembler blocks will set the edge select mask in order to sense the proper next edge.

Figure 11. External Interrupt Routine



CONCLUSIONS AND RESULTS

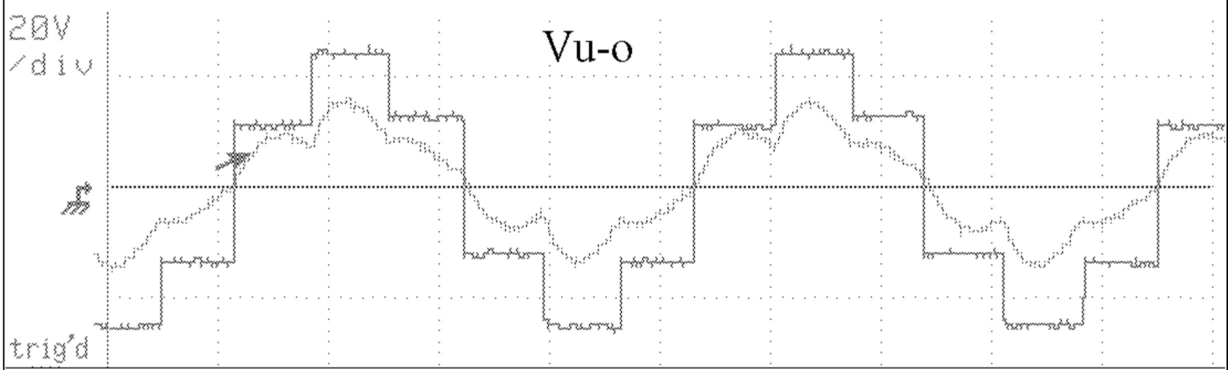
By using ST52x301 Fuzzy Controller it is easy to implement a real time control with few components.The brushless motor control described in this application note represents a good compromise between system costs and motor performances. The graphical programming environment reduces the development time also for not expert programmers.

Fig. 12 displays a phase current and the star voltage Vu-o. PWM square wave is filtered by the oscilloscope.

From this picture it is possible to observe the six steps on the current wave that yields a distortion of the theoretical sine wave. At higher speed rates the distortion becomes lower, although at low speed, motor performance does not degrade.

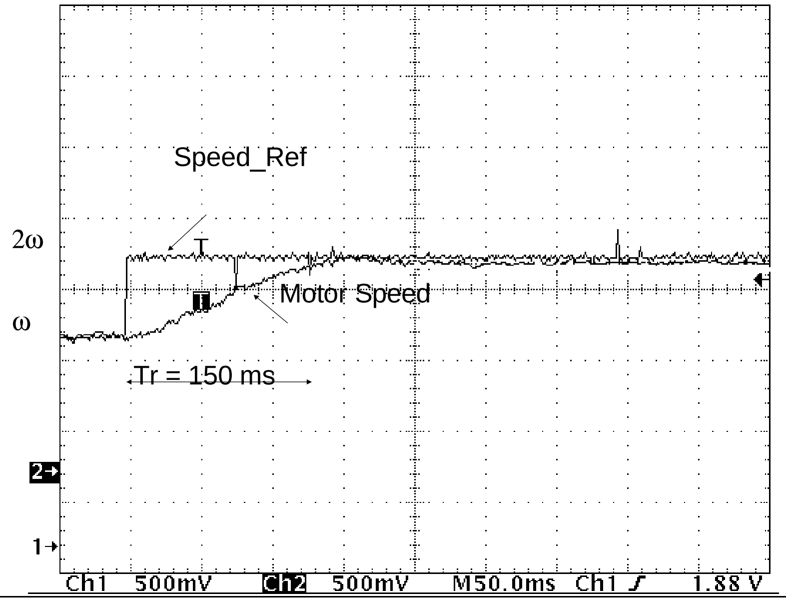
To evaluate acceleration characteristics and control goodness, some trials were made during the software development. Fig. 13 shows free acceleration characteristics starting from a given speed of the shaft to reach a double speed by changing suddenly the speed_Ref.

Figure 12 . Phase current



The dynamic performances of this system were compared, in term of speed and load response, with a traditional controller with six steps drive. No substantial differences were issued in the dynamic performances, but the costs of the traditional system was higher.

Figure 13. Dynamical performances



REFERENCES

[1] Paul C. Krause "Analysis of Electric Machinery" - McGraw-Hill

[2] "Power Products- Application Manual", STMicroelectronics

[3] Mohan, Undeland, Robbins "Power Electronics: Converters, Applications and Design" John Wiley & Sons

[4] Yasuhiko Dote, Sakan Kinoshita "Brushless Servomotors - Fundamentals and Applications" Oxford Science Publications

[5] FUZZYSTUDIO™ 3.0 - User Manual, STMicroelectronics, 1998

APPENDIX: ST52X301 ASSEMBLER CODE

```
;   Source file: C:\TEMP\BLDC_CL.wcl
;   Compile time:Mon Sep 28 11:01:26 1998
;   Device type: ST52x301
;   Compiler version: 01.00 (02.06.98)

            data    0 0 20 147 0
            data    0 1 15 127 15
            data    0 2 0 107 20
            stop
            irq     3       Timer_Interrupt
            irq     4       Triac_Interrupt
            irq     1       AD_Interrupt
            irq     2       SCI_Interrupt
            irq     0       External_Interrupt
            stop
@WCLStart@@:
            ldcf    0       255
            ldcf    1       4
            ldcf    2       10
            ldcf    3       0
            ldcf    4       59
            ldcf    5       15
            ldcf    6       40
            ldcf    7       8
            ldcf    8       3
            ldcf    9       0
            ldcf    10      4
            ldcf    11      0
            ldcf    12      64
            ldcf    13      0
            ldcf    14      0
            ldcf    15      228
Start:
Int_AD_on:
            ldcf    14      2
init:
            ldrc    13      128
            ldrc    14      128
            ldrc    6       0
            ldrc    11      0
```

```
                ldrc    15      100
                ldrc    5       0
AD_start:
                ldcf    2       11
Wait0:
                waiti
Rd_speed:
                ldri    12      0
rd_torque:
                ldri    8       1
Rd_sense:
                ldri    7       2
stopAD:
                ldcf    2       10
EXT_on:
                ldcf    14      1
restart_ad:
                ldcf    2       11
V200_to_duty:
                mdgi
                ldrc    0       200
                ldpr    1       0
                megi
PWM_start:
                ldcf    11      2
                ldcf    10      7
L298_EN_on:
                mdgi
                ldrc    0       1
                ldpr    2       0
                megi
complemnt:
                mdgi
                ldrc    9       255
                sub     9       15
                megi
RD_spee:
                ldri    12      0
RD_torq:
                ldri    8       1
Rd_Isense:
                ldri    7       2
```

```
max_reserch:
            mdgi
            ldrr    0       7
            sub     0       5
            megi
            jps     @@00000
@00001:
            ldrr    5       7
@00000:
@00002:
tmp_3:
            mdgi
            ldrc    0       3
            sub     0       6
            megi
            jpz     @@00004
            jpns    @@00003
@00004:
            jp      calc_err
            jp      @@00005
@00003:
            jp      complemnt
@00005:
calc_err:
            mdgi
            ldrc    0       60
            sub     0       12
            megi
            jps     @@00006
@00007:
            ldrc    12      60
@00006:
@00008:
            mdgi
            ldrc    0       180
            sub     0       12
            megi
            jpz     @@00010
            jpns    @@00009
@00010:
            ldrc    12      180
@00009:
```

```
@00011:
                ldrc    6       0
                mdgi
                ldrr    10      12
                subo    10      9
                megi
controller:
                ldrr    0       10
                stop
                ldp     0       2
                ldp     0       2
                fzand
                con     117
                ldp     0       1
                ldp     0       1
                fzand
                con     127
                ldp     0       0
                ldp     0       0
                fzand
                con     137
                out     0
                stop
                ldri    13      9
Torque_limit:
                mdgi
                ldrr    0       5
                sub     0       8
                megi
                jps     @@00012
@00013:
                ldrc    13      128
@00012:
@00014:
                ldrc    5       0
incr_duty:
                mdgi
                ldrc    0       128
                add     14      13
                add     14      0
                megi
                mdgi
```

```
            ldrc    0       244
            sub     0       14
            megi
            jpz     @@00016
            jpns    @@00015
@00016:
            ldrc    14      244
@00015:
@00017:
            mdgi
            ldrc    0       11
            sub     0       14
            megi
            jps     @@00018
@00019:
            ldrc    14      11
@00018:
@00020:
new_duty:
            ldpr    1       14
            jp      PWM_start
External_Interrupt:
inv_flag:
            mdgi
            ldrc    0       1
            add     11      0
            megi
            mdgi
            ldrc    0       2
            sub     0       11
            megi
            jpz     @@00022
            jpns    @@00021
@00022:
            ldrc    11      0
@00021:
@00023:
            mdgi
            ldrc    0       1
            add     6       0
            megi
flag_is1:
```

```
                mdgi
                ldrc    0       1
                sub     0       11
                megi
                jpz     @@00025
                jpns    @@00024
@00025:
                jp      set_tim
                jp      @@00026
@00024:
                jp      get_rpm
@00026:
get_rpm:
                ldri    15      4
stop_tim:
                ldcf    6       40
on_rise:
                rint    0
                ldcf    14      1
                rint    0
IRET0:
                reti
set_tim:
                mdgi
                ldrc    0       255
                ldpr    0       0
                megi
start_tim:
                ldcf    6       41
                ldcf    6       43
on_fall:
                rint    0
                ldcf    14      129
                rint    0
                jp      IRET0
AD_Interrupt:
IRET2:
                reti
SCI_Interrupt:
IRET1:
                reti
Timer_Interrupt:
```

```
IRET4:
          reti
Triac_Interrupt:
IRET3:
          reti
          stop
```