AN1327 APPLICATION NOTE

BRUSHLESS DC MOTOR: SENSORLESS FUZZY CONTROL BY ST52X420

Author: G. Grasso, V. Marino

ABSTRACT

This application note describes the very low-cost implementation of a Brushless DC Motor actuator. The cost reduction is obtained by means of an 8-bit fuzzy microcontroller which implements both a Sensorless phase driving and a closed loop speed control.

The application is designed for high-voltage motors suitable for 220V_{ac} direct use.

Keywords: BLDC, Sensorless, Permanent Magnet Rotor, Speed control, Main supply.

1. INTRODUCTION

In the last years the home appliance market is becoming a strong competitive field. The growing performances/costs ratio is leading the industry to new approaches in the motor control field in order to satisfy EMI/EMC laws, Quality and reliability standards, etc.

The results of these constraining factors are: long life motors, maintenance-free motors and sensors removal requirements; high efficiency driving, and so on.

Brushless DC motors well fit these requirements and are becoming widely used as small horsepower motors.

Moreover, when the application does not require high dynamic performances in the handling of the load torque, a sensorless control of the BLDC is a suitable low-cost implementation.

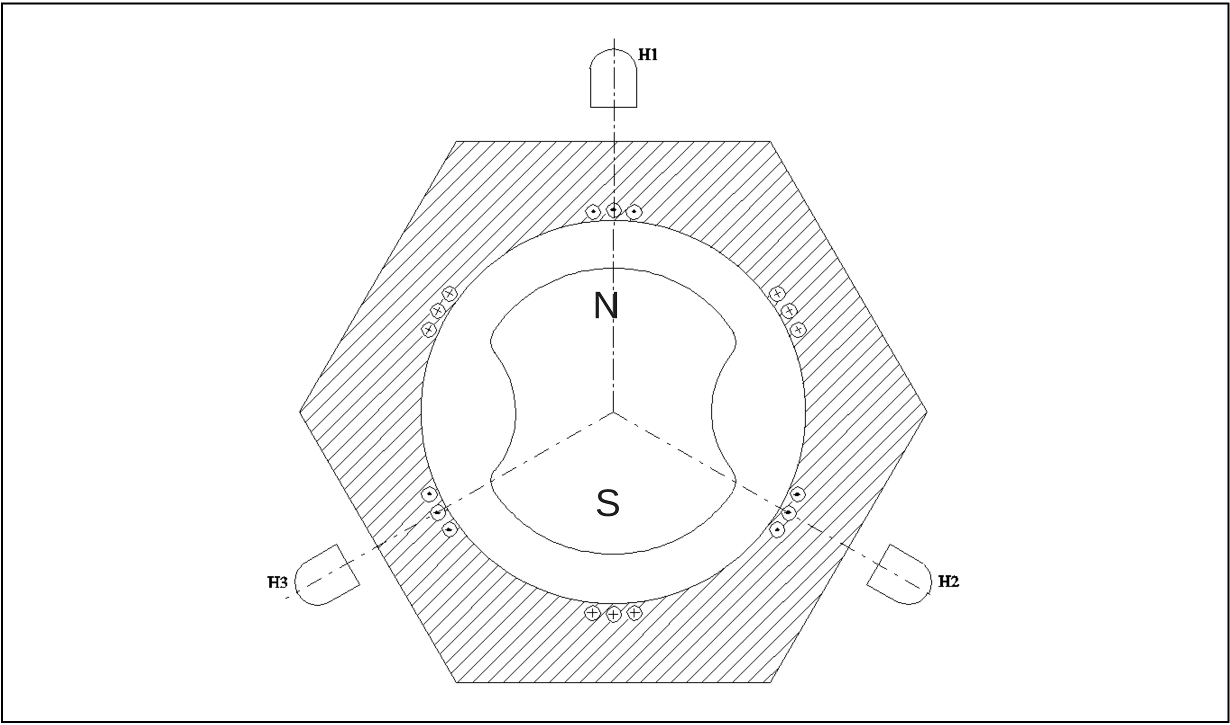
Adjustable-speed drives for home appliances systems, large fans, compressors, hydraulic pumps, … etc are good examples of system in which the load does not change suddenly and then a sensorless approach could be enough.

2. BRUSHLESS MOTOR OVERVIEW

Brushless DC (BLDC) motors are referred to by many aliases: brushless permanent magnet, permanent magnet DC motors, permanent magnet synchronous motors, etc. This type of motors 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. Electronics replaces the function of commutator and energize the proper winding. The energized stator winding leads the rotor magnet and switches as the rotor aligns with the stator.

The motor used in the current application is a three-phase motor, two pole pairs, Vrms=200V, Irms=300mA, used in a radial fan split for air conditioning system.

Figure 1. BLDC section- 3-phase, 1 pole pair for each phase



The following theoretical tips are not an outline on synchronous machines but a transposition of the concept in our application. For details on the BLDC motor principles the reader can refer to any book on electrical machines (see references).

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:

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

where ω_S (rad/s) is the flux synchronous speed or the rotor speed.

In our case the stator has two pole pairs for each phase (*p*=4).This means that:

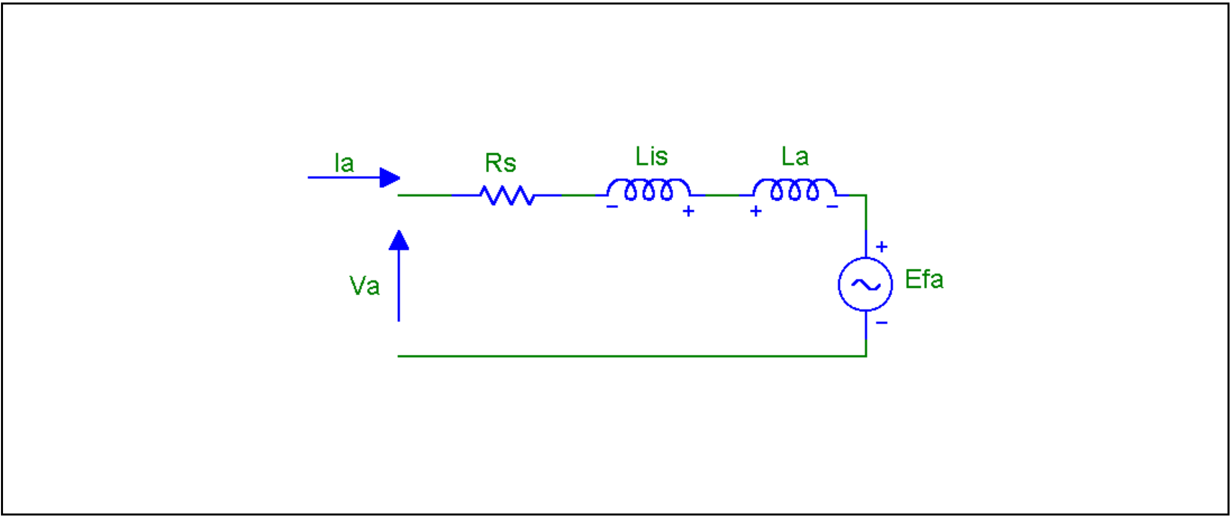
$$\omega = 2\pi f = \frac{4}{2}\omega_S = 2\omega_S$$

In other words, for one turn of the rotor (360° mechanical), the inverter must supply two trapezoidal cycles for each phase.

By using the per-phase equivalent model of the motor it is possible to make some considerations.

In the diagram *R*s is the phase stator winding resistance, L_a is the armature inductance and L_{is} is the winding leakage inductance.

Figure 2.



E_{fa} represents the Back-EMF (bemf). This is the voltage that the permanent magnet rotor induces in the stator coil when motion takes place. This EMF is always present during the normal driving of the motor and opposes a counter-force to the motion of the shaft that is proportional to the rotor speed.

Main end of this application note is to detect the BEMB. The expression of the electromechanical torque is:

$$T_{EM} = k_t \phi_f I_a \sin\delta$$

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_f and the relative stator current I_a.

Now, if the electronic controller is able to supply the phase P_{h1} in order to maintain δ=90° the fields decoupling takes place then the (2) can be simplified as:

$$T_{EM} = K_T I_S \qquad (3)$$

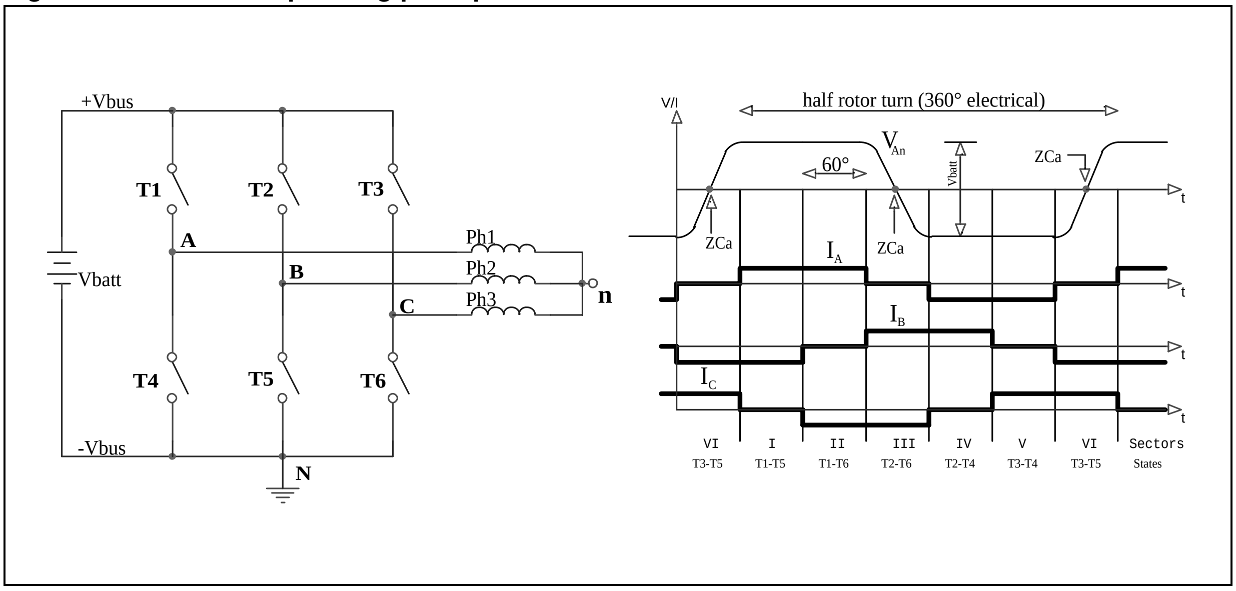
where K_T is called motor torque constant and I_S is the amplitude of the stator phase currents. In these specific conditions (i.e. stator flux in quadrature with the rotor flux) the equation (3) shows that the output torque is proportional only to the supplied current. This is the same case of a simple DC machine.

Main task of this application note is to maintain the torque angle δ=90° without the use of any position sensor.

3. INVERTER DRIVER TOPOLOGY

The widely used circuit to perform the trapezoidal wave driving of a DC brushless motor 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.

Figure 3. Drive basic operating principle



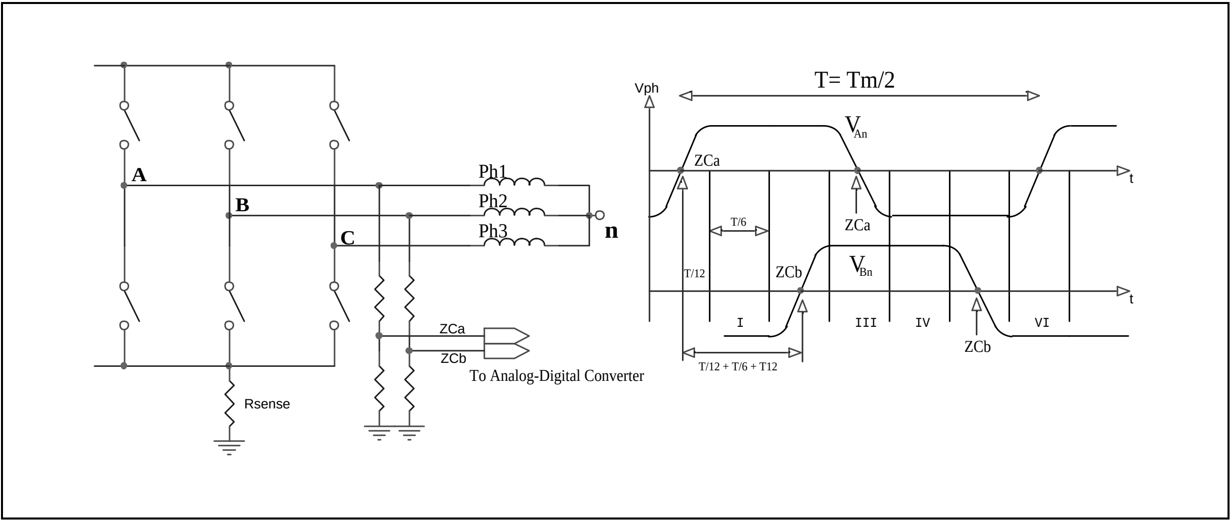
The name "six-steps" arises from the finite time-steps in which the whole period $0 \div 2\pi$ can be shared. Being our motor a two pole pairs, the electrical period $0 \div 2\pi$ represents half turn of the rotor.

During each time step, the current direction does not change whereas current amplitude can increase or decrease in the windings. This topology drives each winding for the 240°electrical degrees, allowing to one phase at a time to be "floating" for 120°. The floating state is the mandatory condition to read the BEMF produced by the rotor flux in the phase under evaluation. For instance in the third sector T2 and T6 are conducting; node ’A’ is floating because switches T1 and T4 are open. If an oscilloscope is connected between node ’A’ and ’n’ during the III sector, an inducted voltage will be displayed. This is the voltage that a rotating permanent magnet induces on a fixed coil. Polarity inversion of this voltage means that the rotor flux inverts its direction from North-pole to South-pole or viceversa. This flux inversion (voltage zero-crossing) is a useful information to well know the rotor position. Like in the sensored applications we can take advantage of this position information in order to drive the next coils.

4. HARDWARE DESCRIPTION

As already discussed, the main target of this application note is to reduce the overall cost of the system. For this reason, a simple voltage scaling in conjunction with the AD converter is used to detect the zero crossing occurrency. Fig. 4 shows the resistor partition used to limit the overvoltage on the AD inputs.

Figure 4. BEMF sensing principle



Only two phases are monitored in order to detect their zero-crossing. In fact, only two temporal information are enough to know position and speed of the rotor with enough accuracy.

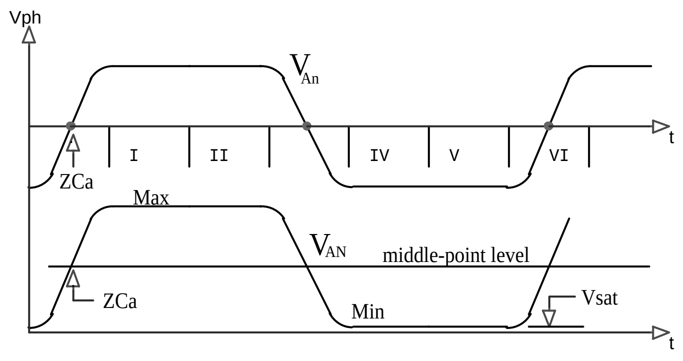
The AD converter will continuosly read the phase voltages and will produce an interrupt to the CPU when the Zero-Crossing is reached.

At this point, a brief theoretical consideration on the acquired voltages must be done. As shown in fig. 5 the zero crossing event is referred to the phase voltages V_{An}, V_{Bn} (i.e. between phase terminal and center tap). The converted values of the scaled voltages are referred to the ground (line voltages). It is possible to transform the line voltages into the phase voltages using the center tap voltage and the translation equations. This is tipycally done with a DSP approach.

In this application note a different method is used in order to detect the zero crossing event. For the six step inverter (discontinuous mode), it is easy to show that the center tap voltage is a constant value during the six sectors. Its level depends from the chopping time of the High Side switches when the PWM is used to control the speed.

Assuming that the center tap voltage is constant, the AD can detect the ZC by comparing the actual converted value with a "middle-point" value. This middle value is the half of the BEMF ranges detected in the previous sectors.

Figure 5. Line to center and line to neutral voltages



The AD-subroutine (see later for details) is shared in several sub-tasks, one for each sector, in order to minimize the overall AD-interrupt time-task. For instance: during the I^ sector only the Max-value is sampled, filtered and divided to obtain ’middle’ value; during the II^ sector the phaseB voltage is converted and compared with ’middle’ value in order to flag the ZCb event; during the VI^ sector for the ZCa event. The ’Min’ value is about zero volt being the dropout voltage on the low-side switches of the bridge.

Another feature of the driver is the total current monitoring. A very little resistor value Rsense is used to sense the current level in the phases during each step of the inverter. This current measurement will be used to detect wrong motor operation and rotor asynchronism (pull-out torque).

5 SOFTWARE DESCRIPTION

The following paragraphs describe the implementation of the Application note realized by using the ST52x420 microcontroller. The easy use of the microcontroller support tools allows to simply adjust and optimize the microcontroller code

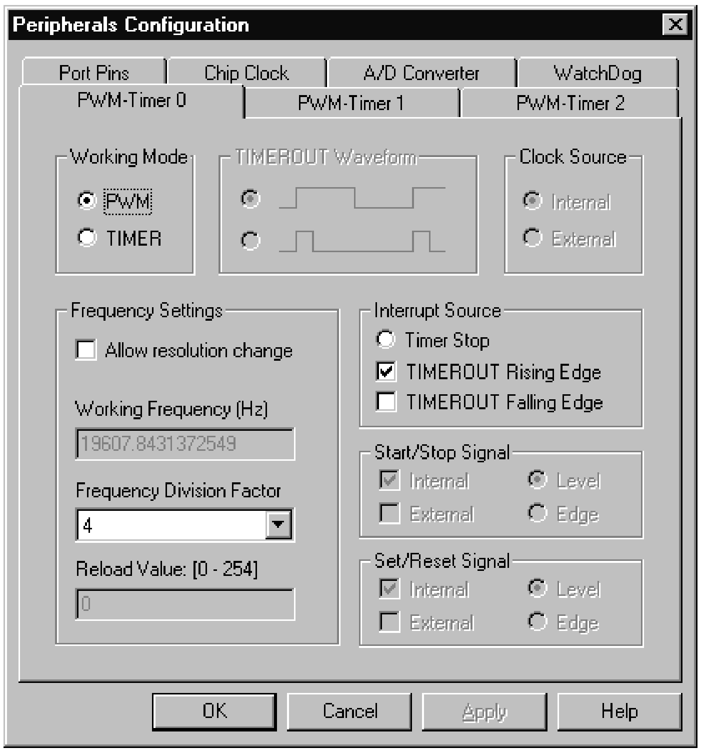
The main program presentation is carried out in a graphical way in order to describe in a user-friendly and intuitive manner the flow of the program.

After a paragrah dedicated to the configuration of the ST52x420 peripherals (please refer to ST52x420 data sheet for further information), the flowchart of the main program and of the relative subroutines is provided.

5.1 ST52x420 Settings

In the following picture is shown the configuration of the peripherals of ST52x420 used in the current application (*cfr:* peripheral configuration chapter in ST52x420 data sheet).

Figure 6. PWM configuration



Peripherals Configuration

Port Pins | Chip Clock | A/D Converter | WatchDog | PWM-Timer 0 | PWM-Timer 1 | PWM-Timer 2

Working Mode: [✓ PWM] TIMER

TIMEROUT Waveform

Clock Source: [✓ Internal] External

Frequency Settings
- ☐ Allow resolution change
- Working Frequency (Hz): 19607.8431372549
- Frequency Division Factor: 4
- Reload Value: [0 - 254]: 0

Interrupt Source
- Timer Stop
- ☑ TIMEROUT Rising Edge
- ☐ TIMEROUT Falling Edge

Start/Stop Signal: ☑ Internal ☐ External; [✓ Level] Edge

Set/Reset Signal: ☑ Internal ☐ External; [✓ Level] Edge

PWM-Timer 0 is used in PWM mode, with a switching frequency of 19.6 KHz; T0_OUT is AND-ed with the three high-side driver signals in order to modulate the voltage on the motor and then establish its speed.

PWM-Timer 1 and PWM-Timer 2 are used in Timer mode. The first timer is used to calculate the time between the zero-crossing of two phases (that is 1/3 of the electrical period T); because the prescaler is 2048 and the clock master is 20 MHz, the Timer 1 clock period is 102.4 µs, then the counted period T/3 is given by 102.4 µs*PWM_1_COUNT.

Instead, the second timer is used to generate the driver signals. Because in each period T there are six switches command, Timer 2 is used to scan 6 temporal events. Of course these 6 events must be scanned at each T/6 and synchronized with the rotor position. For this purpose, Timer2 prescaler is set to divide by 1024 achieving a Timer2 clock input at 51,2µs. By imposing:

$$51{,}2\mu s \times PWM2\_COUNT = T/6$$

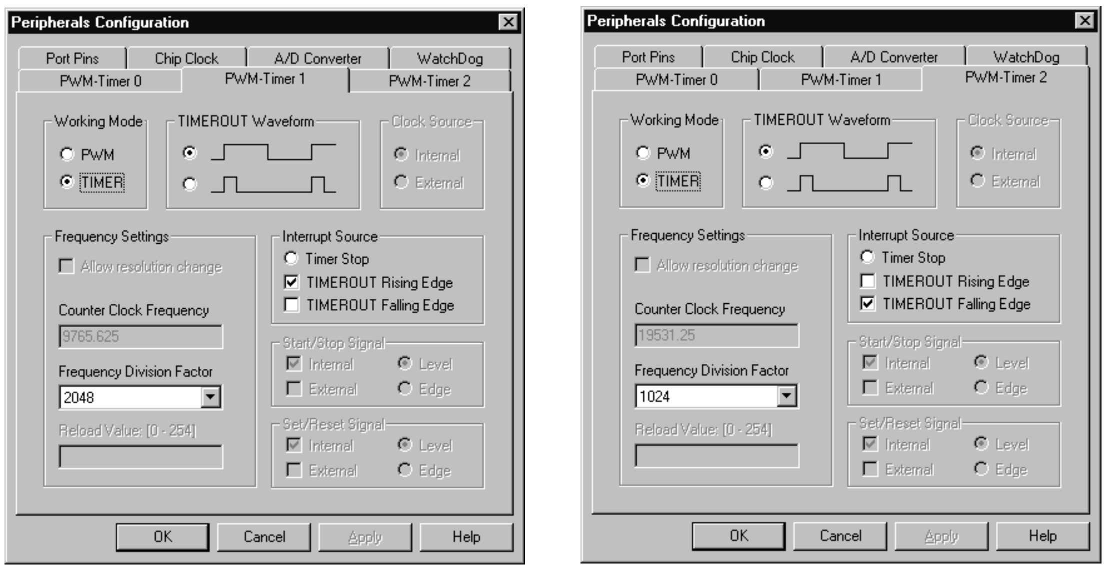
it is easy to see that it is enough to equal:

$$PWM2\_COUNT = PWM1\_COUNT$$

This is a simple method to obtain a period division (from T/3 to T/6) without the use of a DIV instruction.

In other words, Timer2 counts at a speed rate that is the double of Timer1; then, with the same count value, Timer1 reads T/3 whether Timer2 imposes T/6.

Figure 7. Timers configuration



Peripherals Configuration — PWM-Timer 1

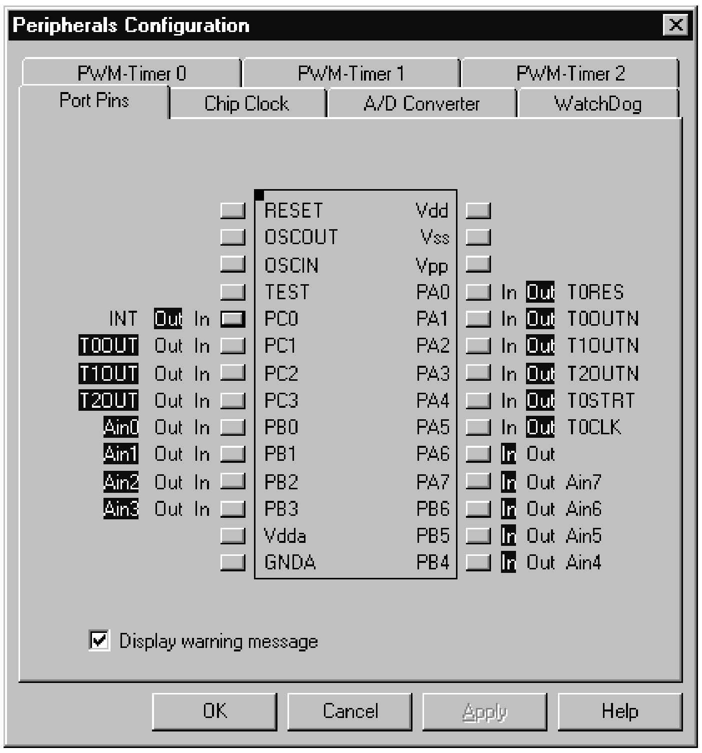
- Working Mode: PWM [✓ TIMER]
- TIMEROUT Waveform: first waveform selected
- Clock Source: [✓ Internal] External
- Frequency Settings: ☐ Allow resolution change
- Counter Clock Frequency: 9765.625
- Frequency Division Factor: 2048
- Reload Value: [0 - 254]
- Interrupt Source: ☐ Timer Stop; ☑ TIMEROUT Rising Edge; ☐ TIMEROUT Falling Edge
- Start/Stop Signal: ☑ Internal, ☐ External; [✓ Level] Edge
- Set/Reset Signal: ☑ Internal, ☐ External; [✓ Level] Edge

Peripherals Configuration — PWM-Timer 2

- Working Mode: PWM [✓ TIMER]
- TIMEROUT Waveform: first waveform selected
- Clock Source: [✓ Internal] External
- Frequency Settings: ☐ Allow resolution change
- Counter Clock Frequency: 19531.25
- Frequency Division Factor: 1024
- Reload Value: [0 - 254]
- Interrupt Source: ☐ Timer Stop; ☐ TIMEROUT Rising Edge; ☑ TIMEROUT Falling Edge
- Start/Stop Signal: ☑ Internal, ☐ External; [✓ Level] Edge
- Set/Reset Signal: ☑ Internal, ☐ External; [✓ Level] Edge

The six driver signals are provided by the ST52x420 parallel port; these are supplied on the pins PA0-PA5, configured as output pins as shown in fig. 8.

Figure 8. I/O configuration



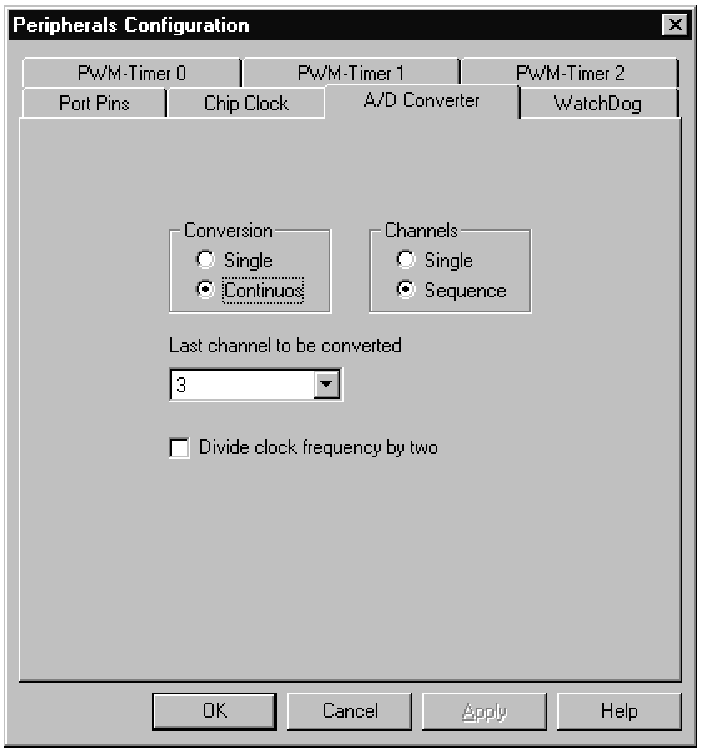
Peripherals Configuration — Port Pins

		RESET	Vdd		
		OSCOUT	Vss		
		OSCIN	Vpp		
		TEST	PA0	In [✓ Out]	T0RES
INT	[✓ Out] In	PC0	PA1	In [✓ Out]	T0OUTN
T0OUT	Out In	PC1	PA2	In [✓ Out]	T1OUTN
T1OUT	Out In	PC2	PA3	In [✓ Out]	T2OUTN
T2OUT	Out In	PC3	PA4	In [✓ Out]	T0STRT
Ain0	Out In	PB0	PA5	In [✓ Out]	T0CLK
Ain1	Out In	PB1	PA6	[✓ In] Out	
Ain2	Out In	PB2	PA7	[✓ In] Out	Ain7
Ain3	Out In	PB3	PB6	[✓ In] Out	Ain6
		Vdda	PB5	[✓ In] Out	Ain5
		GNDA	PB4	[✓ In] Out	Ain4

☑ Display warning message

AD converter is configured with four channels, which are converted continually: Ain0 and Ain1 are used to detect ZCa and ZCb; Ain2 is used to convert the information coming from Rsense that is related with the overall maximum current; Ain3 is used to convert the voltage coming from the potentiometer that sets the target speed.

Figure 9. AD Converter configuration



Conversion: Single [✓ Continuos]

Channels: Single [✓ Sequence]

Last channel to be converted: 3

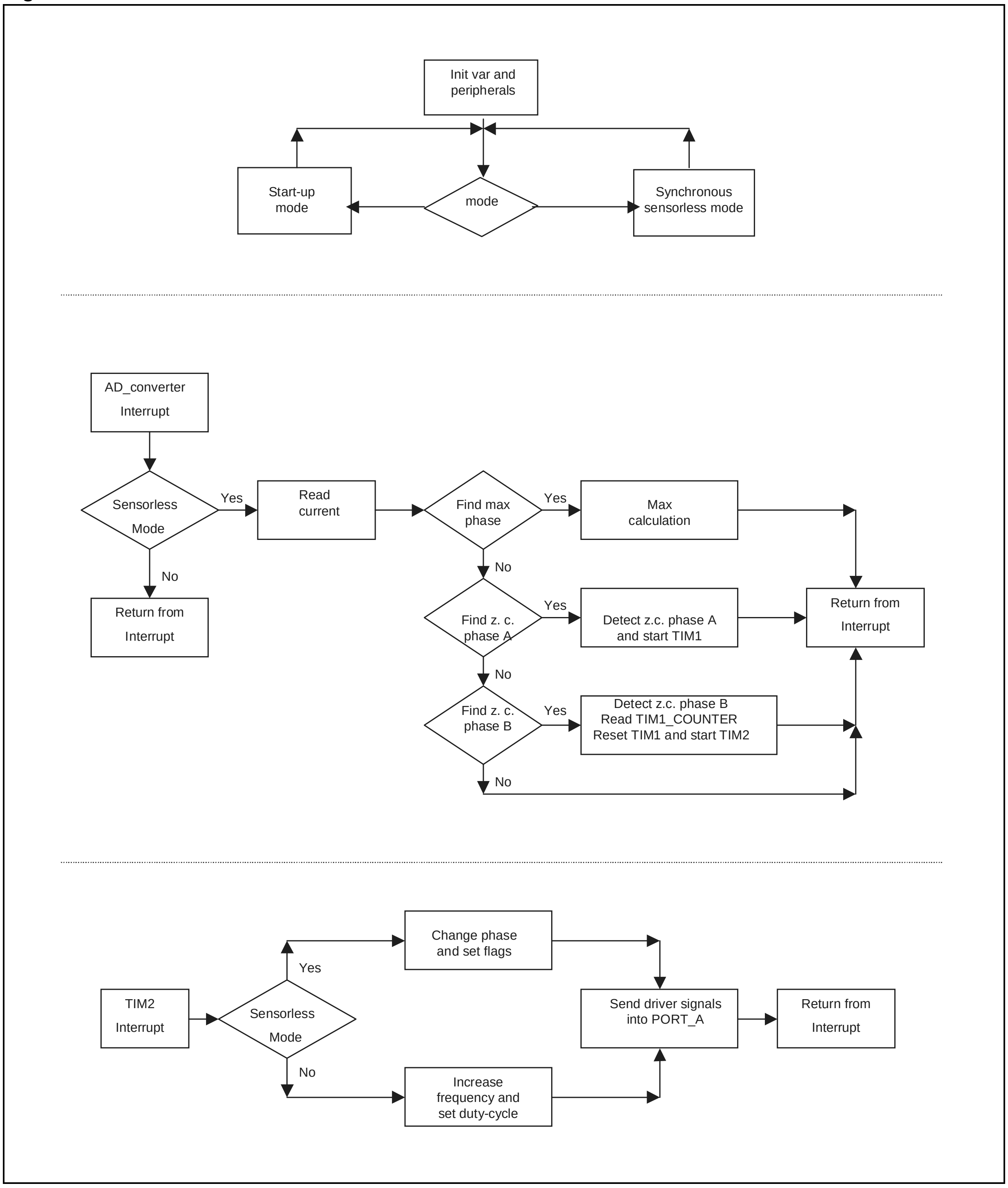
☐ Divide clock frequency by two

5.2 Algorithm Description

The following figure summarises all the tasks implemented in the ST52x420 program. At the top is shown the main program while on the following parts are shown the AD and Timer2 interrupt service subroutines.

The main program mainly consists of 2 different loops activated one after the other according to the variable mode: start-up and sensorless modes.

Figure 10. Flow-chart .


Init var and peripherals
Start-up mode
mode
Synchronous sensorless mode
AD_converter Interrupt
Sensorless Mode
Yes
No
Read current
Find max phase
Max calculation
Return from Interrupt
Find z. c. phase A
Detect z.c. phase A and start TIM1
Find z. c. phase B
Detect z.c. phase B Read TIM1_COUNTER Reset TIM1 and start TIM2
TIM2 Interrupt
Change phase and set flags
Increase frequency and set duty-cycle
Send driver signals into PORT_A

a) *Start-up mode*

The main goal of this procedure is to lead the rotor from 0 rpm to almost 1000 rpm, in order to have some BEMF waveforms detectable by AD converter.

During this operation mode the six driver signals are supplied with a set 6 of commands with duration of T/6. Progressively this starting period T is storten (stator flux speed increases) in order to force the rotor movement. For this puropose, Timer2 is used to synchronize the witches command from one secotr to another. As referred in the Timer2 flowchart,at each falling edge of Timer2_out, the variable 'phase' (that represents the sector) is incremented, in order to send into PORT_A the pattern related.

Varying the PWM_2_COUNT value, the frequency of the driver signals is increased to reach the desired value; note that, in this mode, the duty-cycle of PWM0 is increased (phase voltage increases) according to the frequency value, by means of the fuzzy block present in the Timer2 interrupt routine.

b) *Synchronous mode*

After a programmed delay, the sensorless mode is started; in this mode a speed loop control is performed as shown in fig. 11, in order to reach the reference speed value; a fuzzy block, reading the speed error (the difference between the desired and the actual speed), allows to adjust the duty-cycle value (PWM), modulating the voltage on the motor and then adjusting the speed rotation.

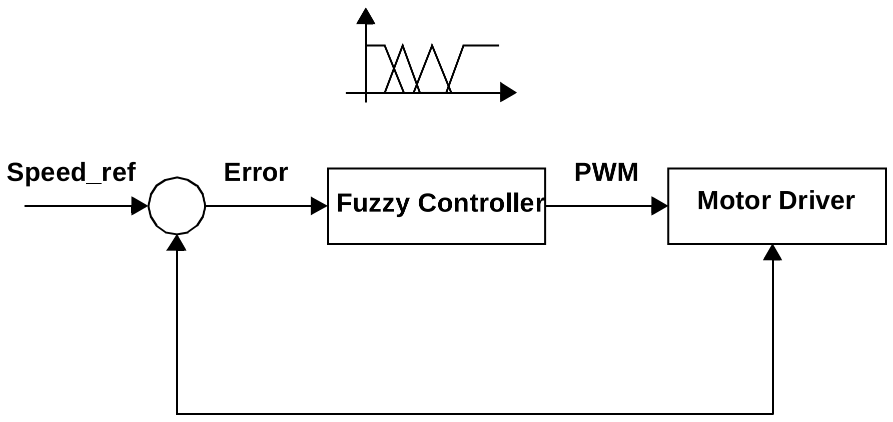
To read the rotation speed, a sensorless algorithm is implemented. By using the AD_converter to detect two subsequent zero-crossing events, a time measurement is done by means of Timer1. timer1 starts at the first zero-crossing and is stopped at the second event.

This period represents, as preivously discussed, 1/3 of the electrical period T, and then, to have the time T/6, (used to commutate the driving signal in Timer2 interrupt routine), it is enough to load the value PWM_1_COUNT in PWM_2_COUNT.

At each commutation, besides producing the six driving signals, some flags are set to establish the actions to perform (find max, find zero-crossing on phase A or on phase B).

Finally, in the AD interrupt routine, the maximum current value Imax is sampled. If Imax overcomes a fixed current value the driver immediately turns off and the microcontroller is reset (thanks to the watchdog) to restart the program.

Figure 11. Control topology

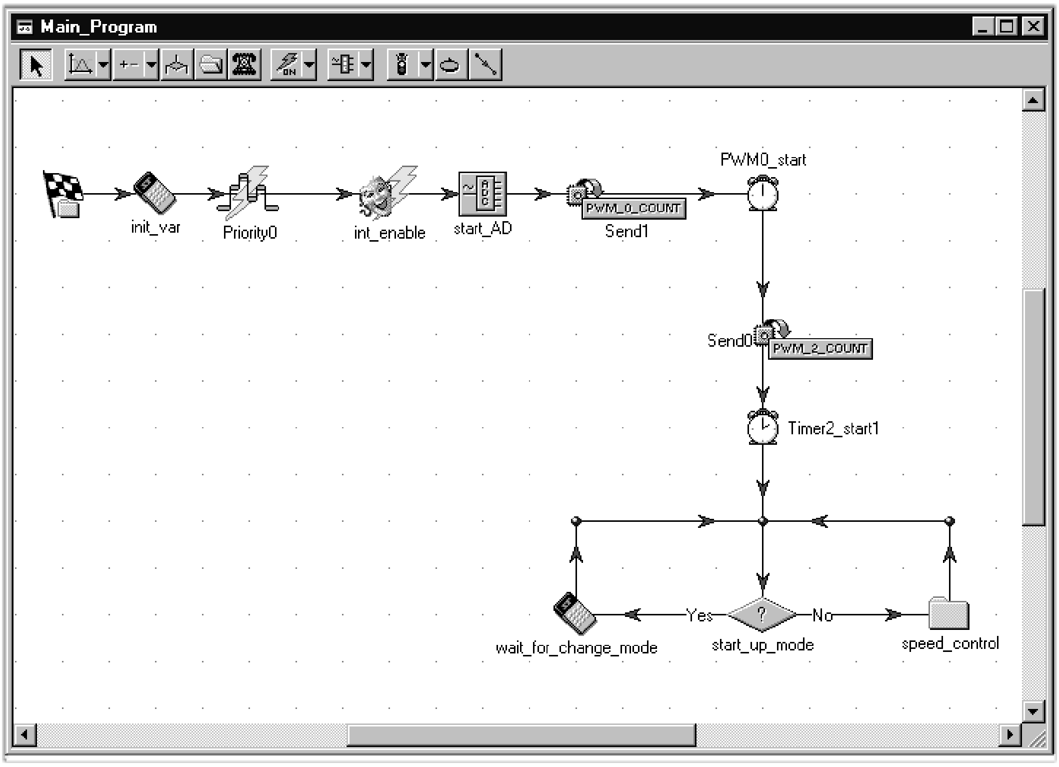


5.3 Main program window

Let us discuss about the main program; '*init_var*' and '*int_enable*' blocks are used to initialize global variables and interrupt mask (only ADC and Timer2 interrupts are enabled). The following blocks let ADC, PWM0 and Timer2 start. After starting, PWM2 will count for an initial period imposed from the variable named '*period_start_up*'. Nexy, this variables will be decreased (see Timer 2 Interrupt routine) in order to increment the rotor speed as discussed before.

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

Figure 12. Main program



5.4 Fuzzy Control Window

By implementing the control topology described in fig. 11, the following figures (13 and 14) show the graphical code implementation.

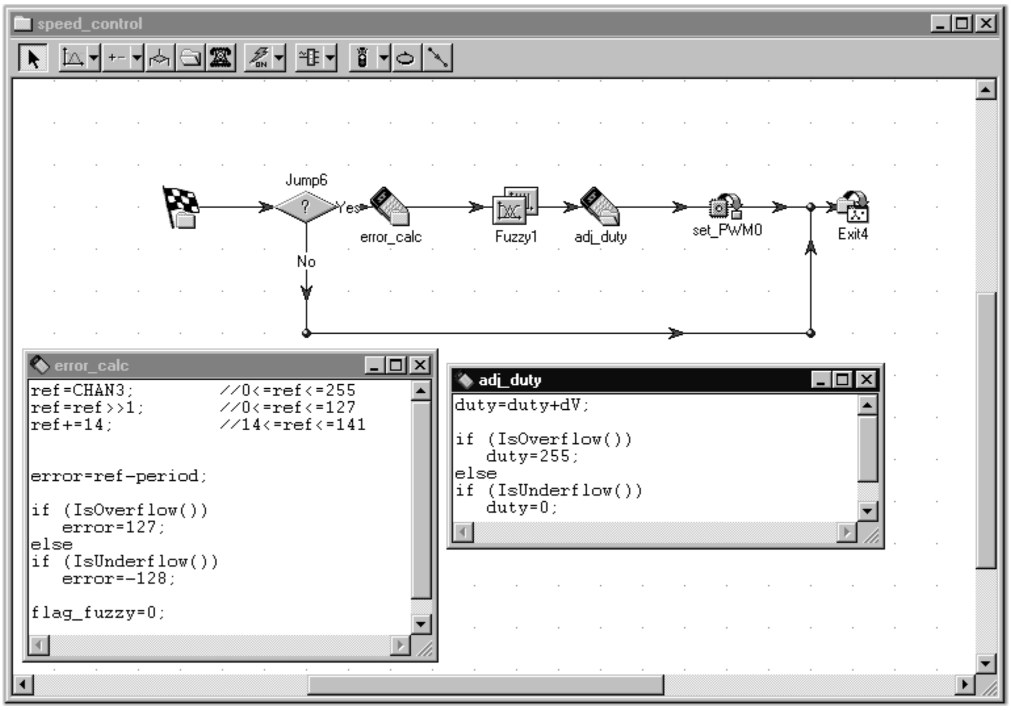
The block '*error_calc*' performs error calculation as 'error=ref-period', where 'period' is the variable containing the value T/3 counted by Timer2.

The '*error*' variable is sent to the Fuzzy engine. This '*fuzzy block*' produces the incremental value 'dv' that is added 9with sign) to the current PWM0 value. the updated PWM0 duty-cycle is sent to the PWM peripheral by the block named 'set_PWM0'.

In the calculation blocks of fig. 13 and 14 the 'IF' 'THEN' operators are used to avoid overflow or underflow in the variables '*error*' and '*duty*' during the control.

Figure 14 shows a classical integrative control described with the fuzzy linguistic aproach. Five Mbfs are chosen and five rules are set in order to implement a single input and output control block.

Figure 13. Fuzzy Control



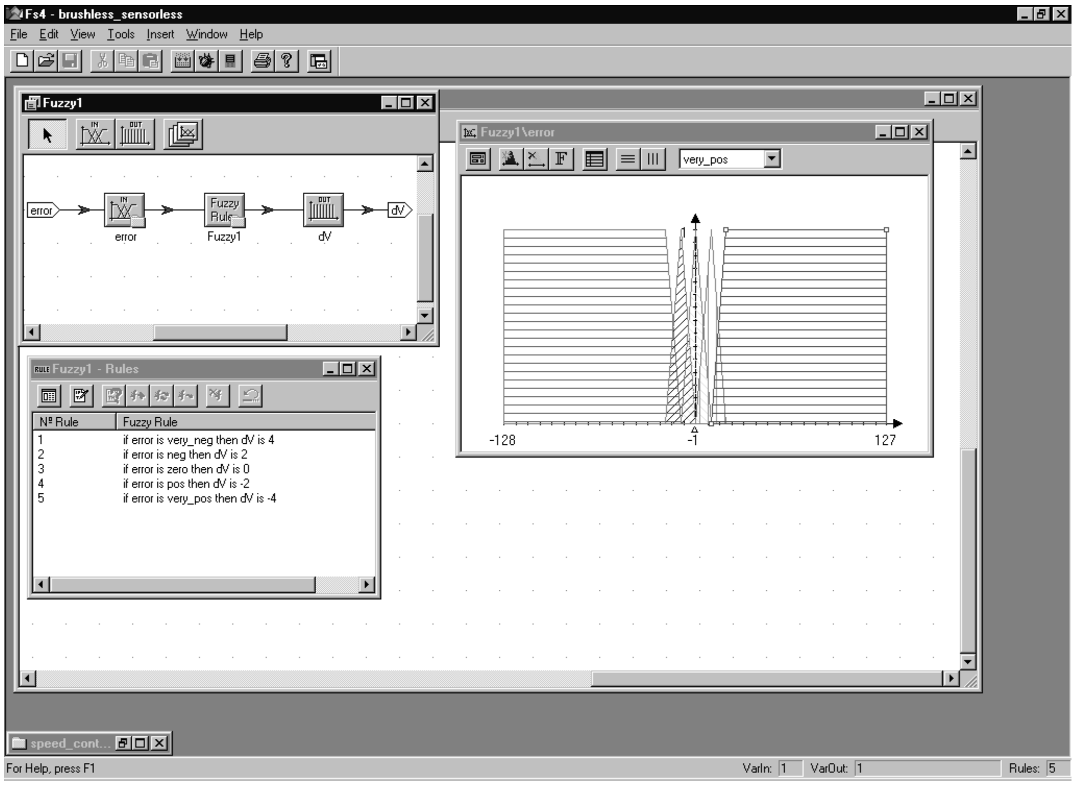
error_calc

```
ref=CHAN3;          //0<=ref<=255
ref=ref>>1;         //0<=ref<=127
ref+=14;            //14<=ref<=141


error=ref-period;

if (IsOverflow())
   error=127;
else
if (IsUnderflow())
   error=-128;

flag_fuzzy=0;
```

adj_duty

```
duty=duty+dV;

if (IsOverflow())
   duty=255;
else
if (IsUnderflow())
   duty=0;
```

Figure 14. Fuzzy rules



Fuzzy1 - Rules

Nº Rule	Fuzzy Rule
1	if error is very_neg then dV is 4
2	if error is neg then dV is 2
3	if error is zero then dV is 0
4	if error is pos then dV is -2
5	if error is very_pos then dV is -4

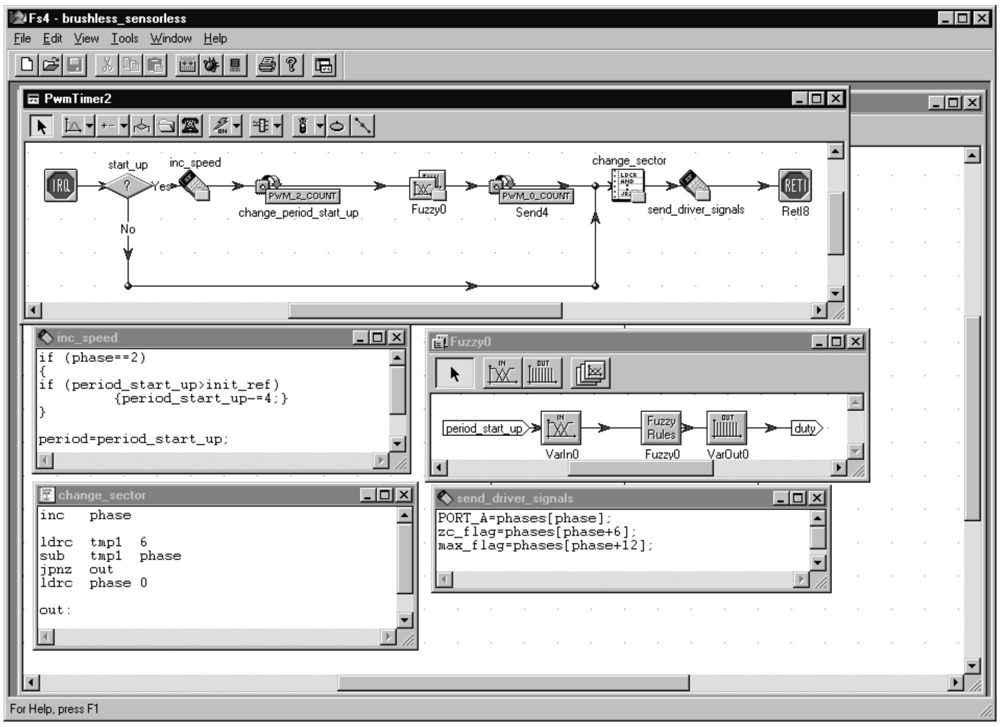
VarIn: 1 VarOut: 1 Rules: 5

5.5 Timer 2 Interrupt window

The task of this subroutine is to generate the signals used to drive the switches T1-T6, either in start-up and synchronous mode.

In the block '*change_sector*' displayed in fig. 15, the variable '*phase*' is incremented in order to switch to the next sector. The block '*send_driver_signals*' is used to write a Boolean pattern on the PORT_A pins.

Figure 15. Interrupt routine



inc_speed

```
if (phase==2)
{
if (period_start_up>init_ref)
        {period_start_up-=4;}
}

period=period_start_up;
```

change_sector

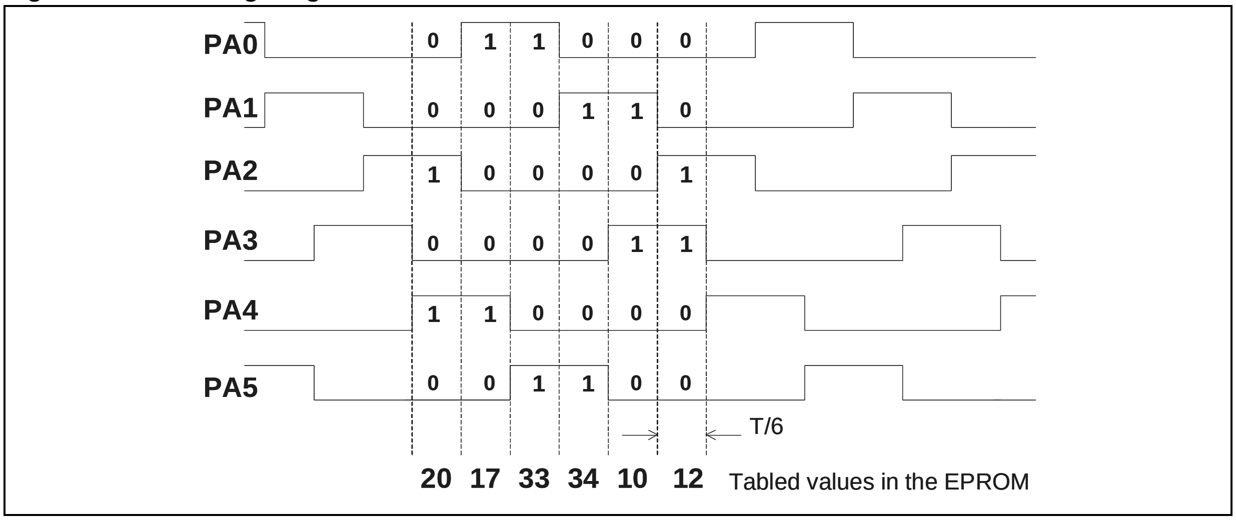
```
inc   phase

ldrc  tmp1  6
sub   tmp1  phase
jpnz  out
ldrc  phase 0

out:
```

send_driver_signals

```
PORT_A=phases[phase];
zc_flag=phases[phase+6];
max_flag=phases[phase+12];
```

This pattern is available in a look-up table stored in the ST52 x420 EPROM. Since the Timer2 subroutine is executed at each T/6 period then PORT_A will change its pattern at each T/6. The following figure shows the pattern for the driver topology previously discussed.

Figure 16. Driver signal generation



The same look-up table is used to store some flags that are addressed during the variable's status change. These flags are used to indicate to the AD interrupt routine the task to be performed in that moment (find max, find ZCa or ZCb) accordingly to the related sector.

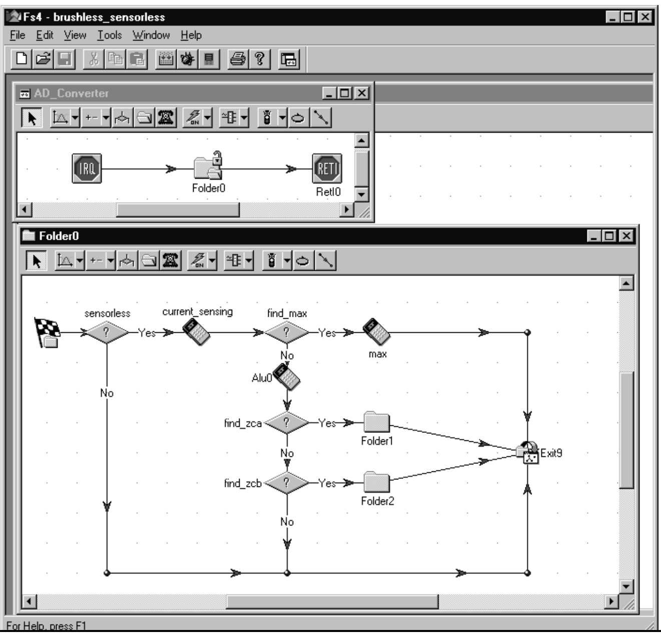
Fig. 15 also shows a fuzzy routine (*Fuzzy0*) used only for the start-up phase. The fuzzy block implements a non linear relation between '*period_start_up*' and '*duty*' variables. As already stated, the stator flux speed is increased to let the rotor start.

5.6 ADC Interrupt window

Figure 17 shows the AD interrupt routine main window. Fig. 18 and 19 give more details on Folder1 and Folder2. In fact, this AD interrupt routine subroutine is shared into three sub-tasks, each one corresponding with the sectors I ('find_max'), II ('find_zcb') and VI ('find_zca').

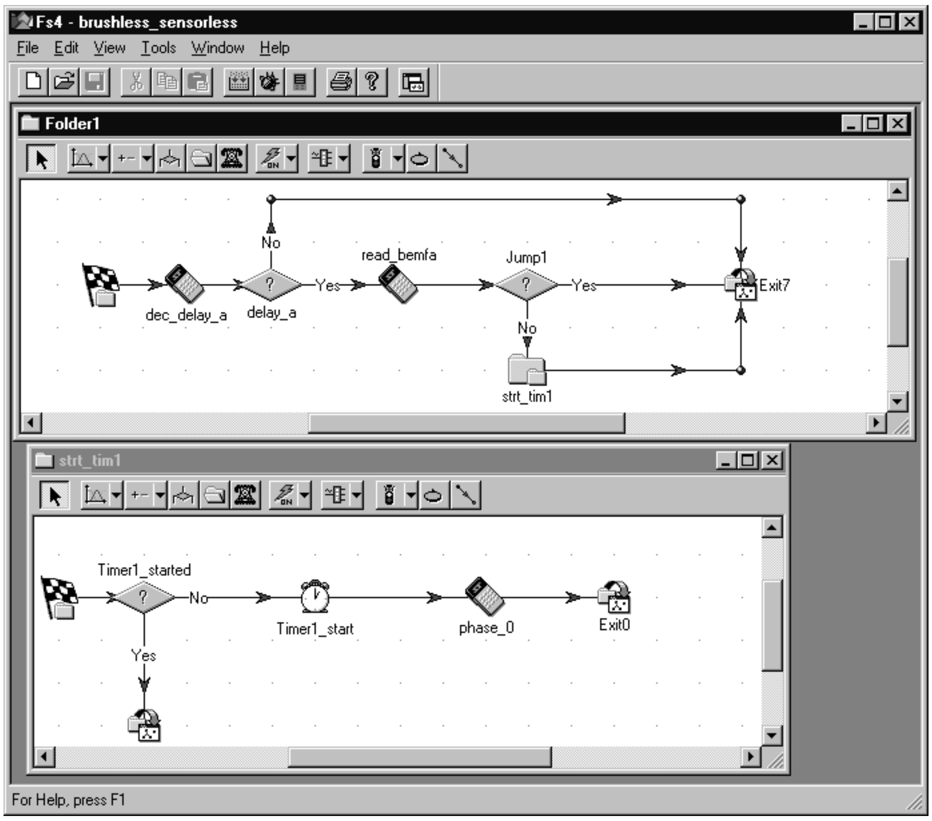
During the first sector, when the phase A of the motor is connected to Vbus (T1 is on), the maximum value is continually acquired by Ain0 pin. The average value of these samples, divided by two (in the block 'max'), represents the value '*middle*' to be compared to detect the zero-crossing on the phases A and B.

Figure 17. AD interrupt routine



A different task has to be performed in the sixth sector (find Zca) as shown in fig 18.

Figure 18. Find Zero Crossing on phase A

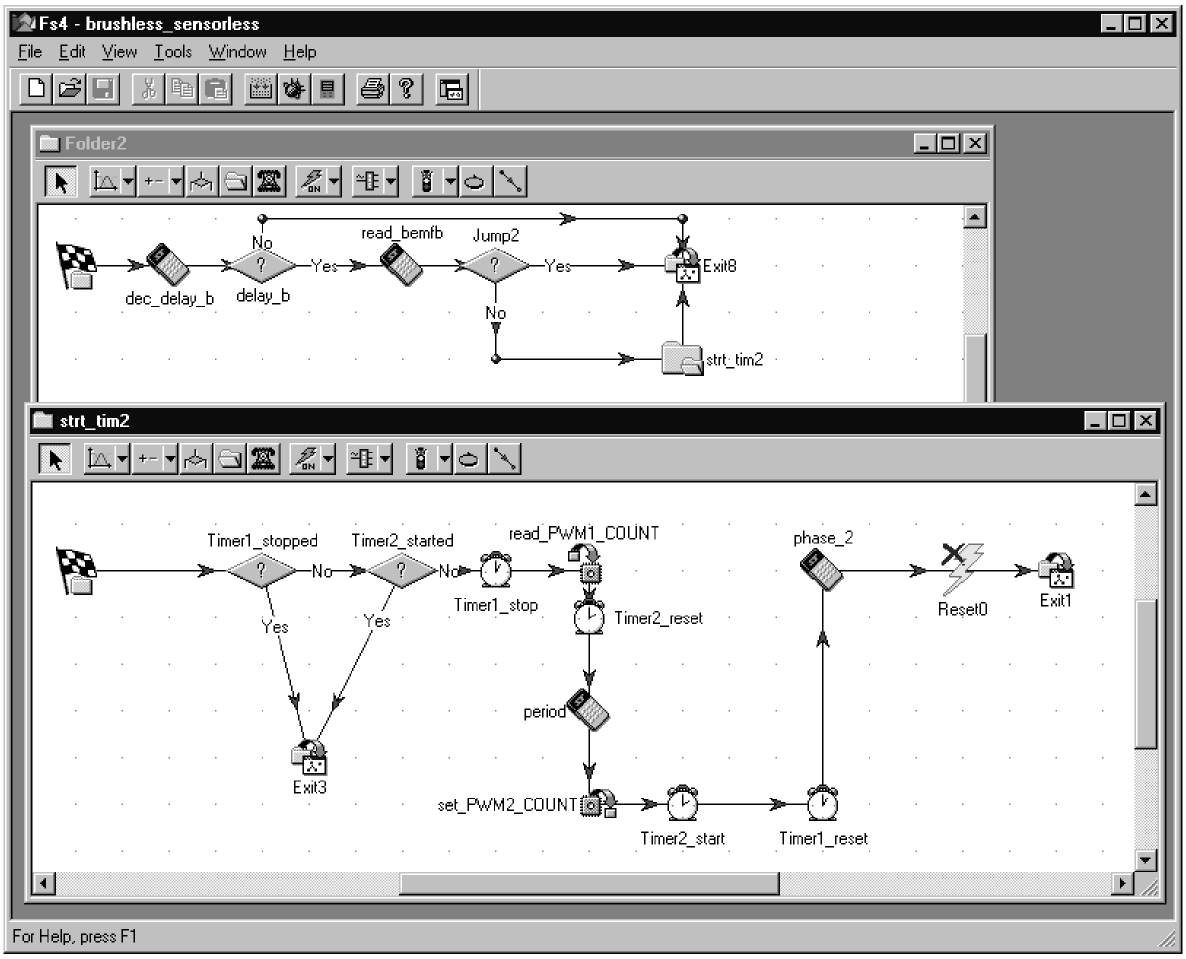


The blocks ‘dec_delay_a’ and ‘delay_a’ implement a blanking time necessary to avoid the sampling of commutation spike. This spike arises from the coil opening in correspondence to the switching sector (See technical literature for further reference).

After this blanking time, the value read on Ain0 is compared with the value 'middle' (block 'Jump1'). When sampled value on phase A voltage reaches the 'middle' value, the block 'strt_tim1' is executed, and Timer1 is started.

Analogously, in the second sector, the zero-crossing on phase B is detected, and when this occurs, the block 'strt_tim2' is executed (fig. 19). In this block Timer1 is stopped, its value (PWM_1_COUNT) is read and Timer1 is reset. At the same time, Timer2 is preloaded with the read value of PWM_1_COUNT and then started.

Figure 19.



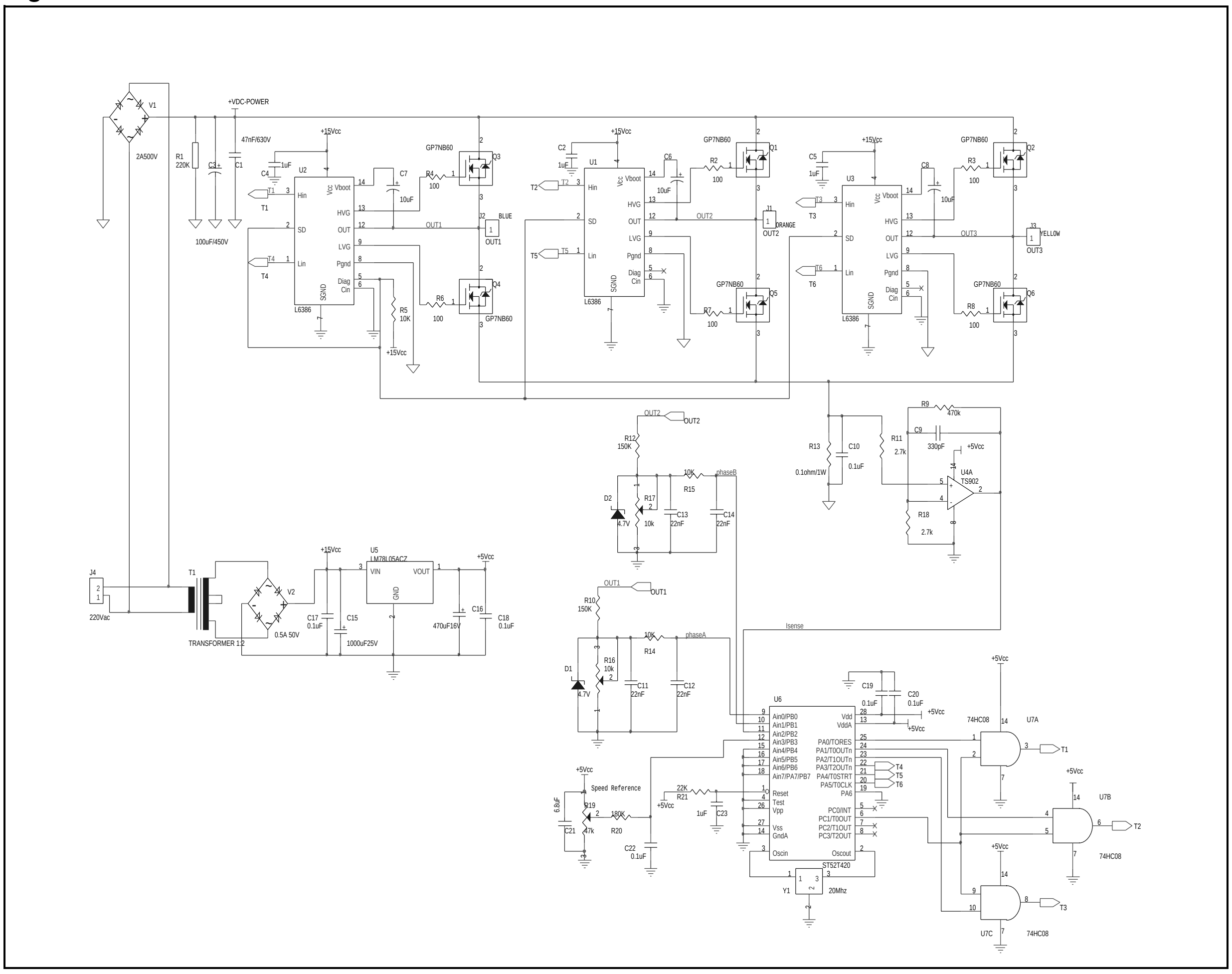
The timers tasks described above allow to synchronize each BEMF zero-crossing with the rising/falling-edge of Timer2. Then the Timer2 edge events to control the commutation of the driver signals to be sent to Port A. Since Timer1 starts and stops on two BEMF events, its content also gives information about the rotor speed to implement the PID fuzzy controller.

6. ELECTRICAL SCHEMATICS

Figure 20 shows the hardware implemented in the current application. The active devices are designed for a brushless motor of 70W and speed rate of 5.000 rpm.

The user can easily modify the device values in order to meet his own specifications.

Figure 20.



7. CONCLUSIONS AND RESULTS

By using ST52x420 8-bit microcontroller it is easy to implement a low-cost approach to achieve a real time control. The brusless 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 non-expert programmers.

The overall system has been implemented to control the speed of a fan in an air-conditioning system with good results.

REFERENCES

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

[2] STMicroelectronics "Power products- Application manual"

[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^{TM}4.0- User Manual, STMicroelectronics 2000