AN1291
APPLICATION NOTE

SLIP CONTROL OF AN ASYNCHRONOUS THREE-PHASE MOTOR WITH ST52X420

Author: M. Di Guardo, V. Marino

1. INTRODUCTION

Induction motors with squirrel-cage rotors are the workhorse of industry because of their low cost and rugged construction. The aim of this Application Note is to show how to perform the slip control of an AC three-phase induction motor with ST52x420, in order to obtain minimum input power and maximum efficiency operations.

This type of control can be achieved by adjusting the amplitude of the applied stator voltage versus torque requirement. Efficiency improvement by voltage control is obtained by reducing the applied voltage whenever the torque requirement of the load can be met with less flux.

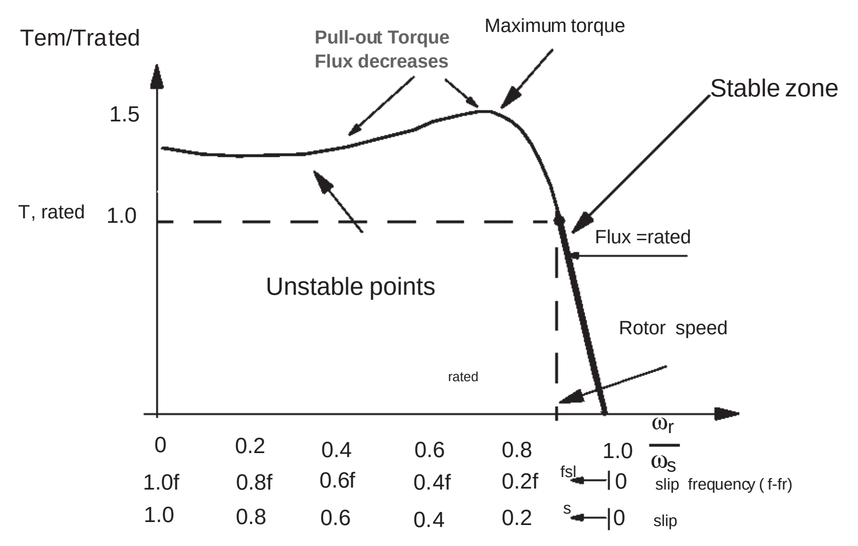
The reduced motor flux results in a reduction of core and stator copper losses since the magnetization component of the stator current is reduced as well. However, it is to note that the minimization of the air gap flux requires a larger slip to produce the torque required if compared with operations at full rated flux.

This application note shows the implementation of the slip control of an aynchronous motor in order to have energy saving of the global power system, representing a convenient solution to reduce the rotor and stator copper losses.

1.1 Torque Characteristics of Asynchronous Three-Phase Motors

Typical torque and current characteristics are represented in figure 1 where the torque T_{em} is plotted as function of rotor speed and f_{sl} (slip frequency is the difference between stator frequency f and rotor frequency). At low values of f_{sl}, T_{em} varies linearly with f_{sl} ,see the line plotted in bold in the stable zone (fig.1). The maximum torque that the motor can produce is represented by the pull-out point shown below.

Figure 1. Torque versus rotor speed at f and Vs constant.

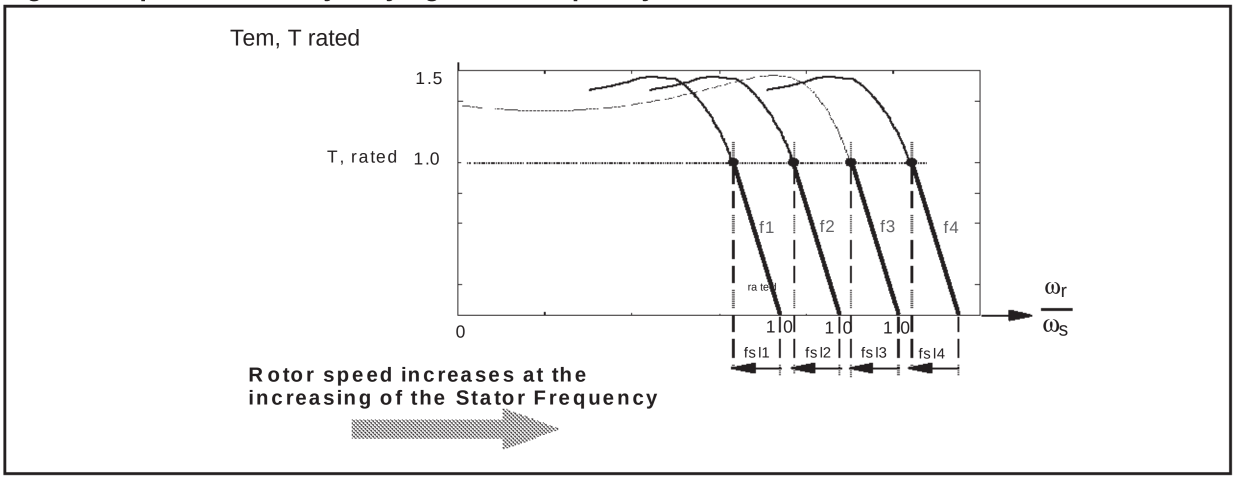


If an induction motor is started directly from the power supply and the load torque is lower than the start-up torque, at maximum slip (slip=1), the motor is able to run and enter in the stable zone. Then the intersection of the motor torque characteristic with load torque determines the steady-state point of operation. If the load torque reaches the maximum torque, the motor enters in the break down domain until the complete stop of the motor.

1.2 Speed Control

Speed can be controlled by varying stator frequency f with power electronic inverter, in order to control synchronous speed and, hence, the motor speed, if the slip is kept small, keeping the flux constant in the air gap, varying stator voltage in linear proportion to stator frequency f (Fig. 2).

Figure 2. Speed control by varying stator frequency

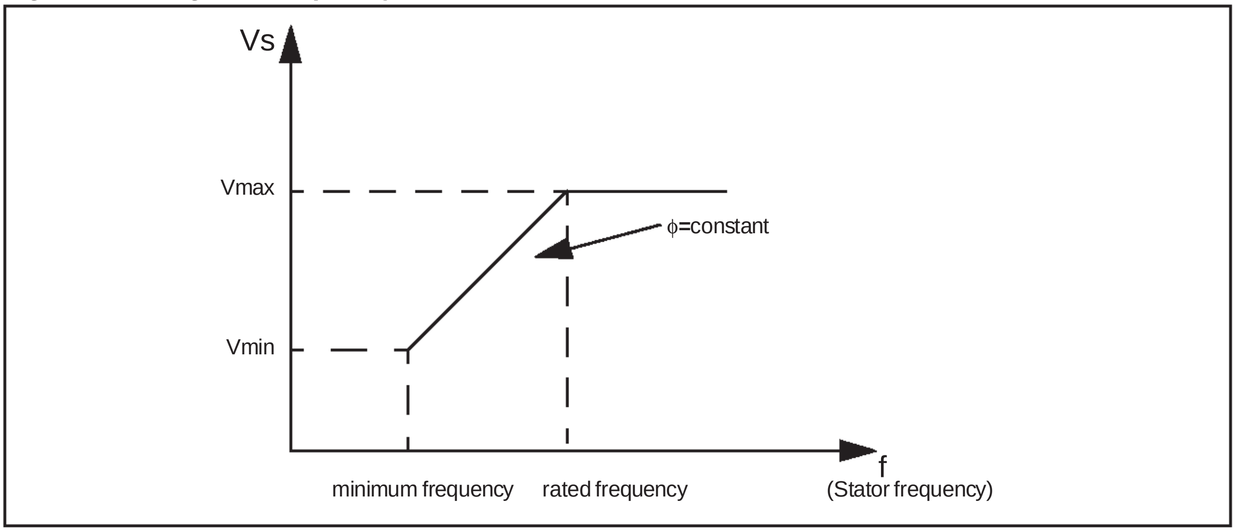


The torque speed characteristics shift horizontally in parallel, as shown in figure 2 for four different values of f_{sl}. Note that, at a constant load-torque, the slip frequency (which is the frequency of the induced voltage and currents in the rotor circuit in hertz) is constant.

1.3 V/F = constant speed control

The simplest method of control is to maintain constant the flux (V/F) with power converters varying the motor speed. This regulation is called torque constant control. If we change the frequency also the voltage applied must vary in a linear way in order to maintain V/F constant. This ratio is dimensionally a flux (fig.3).

Figure 3. Voltage vs Frequency relation



V/F characteristics are listed below:

- It operates at constant flux and Torque
- Motor always supplies the maximum Torque
- Efficiency is not optimized
- Motor is oversized

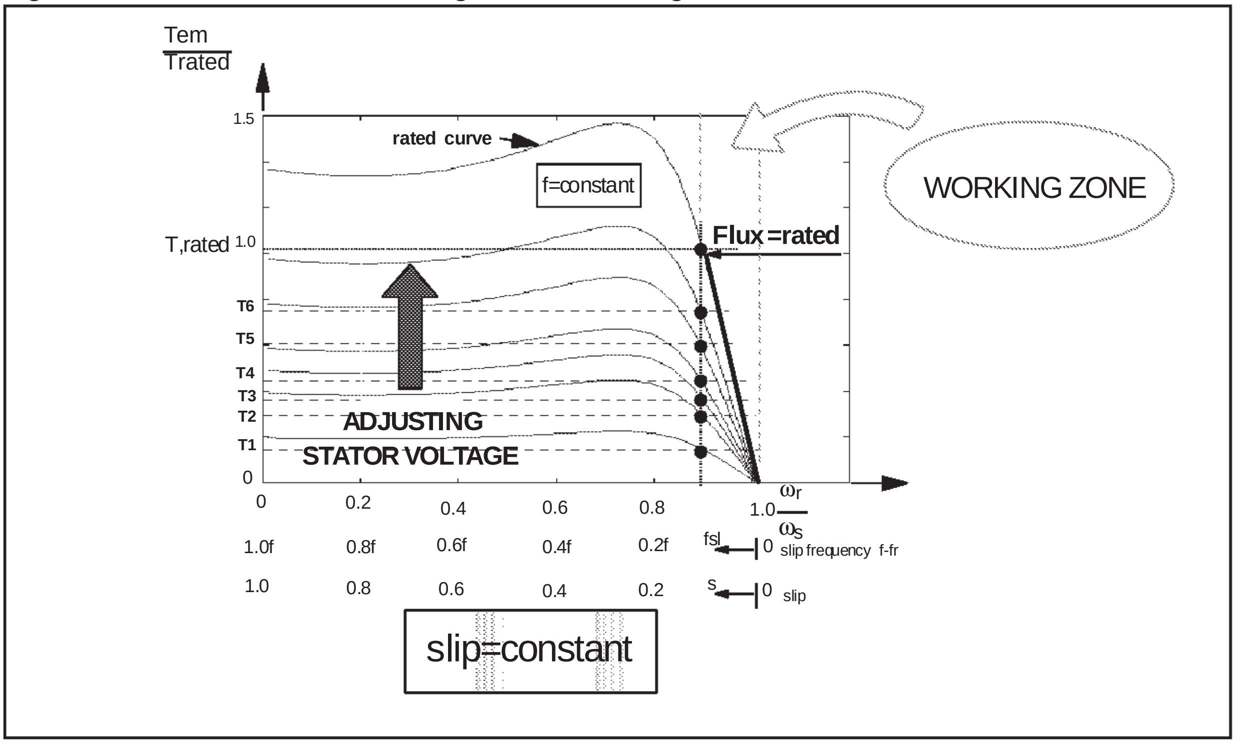
In this application note, the flux minimization control has been implemented instead of the V/F constant method. In this way, it is possible to reach good efficiency and torque regulation.

1.4 Motor Efficiency Optimization with Slip Control

The motor efficiency can be improved by controlling the stator voltage to maintain the slip constant at minimum flux (flux minimization). Adapting the flux in the air gap to have a large slip, but not large enough to reach the pull out torque otherwise the motor would stop, the required torque is generated so as to be compared with operation at full rated flux.

Power loss can be minimized maintaining large and constant the slip adjusting the available torque (Fig.4). This voltage varying method of the phase motor offers limited possibilities of speed regulation. However, combining both voltage control (minimum flux) and frequency control, the motor is well controlled in a wide range of speed.

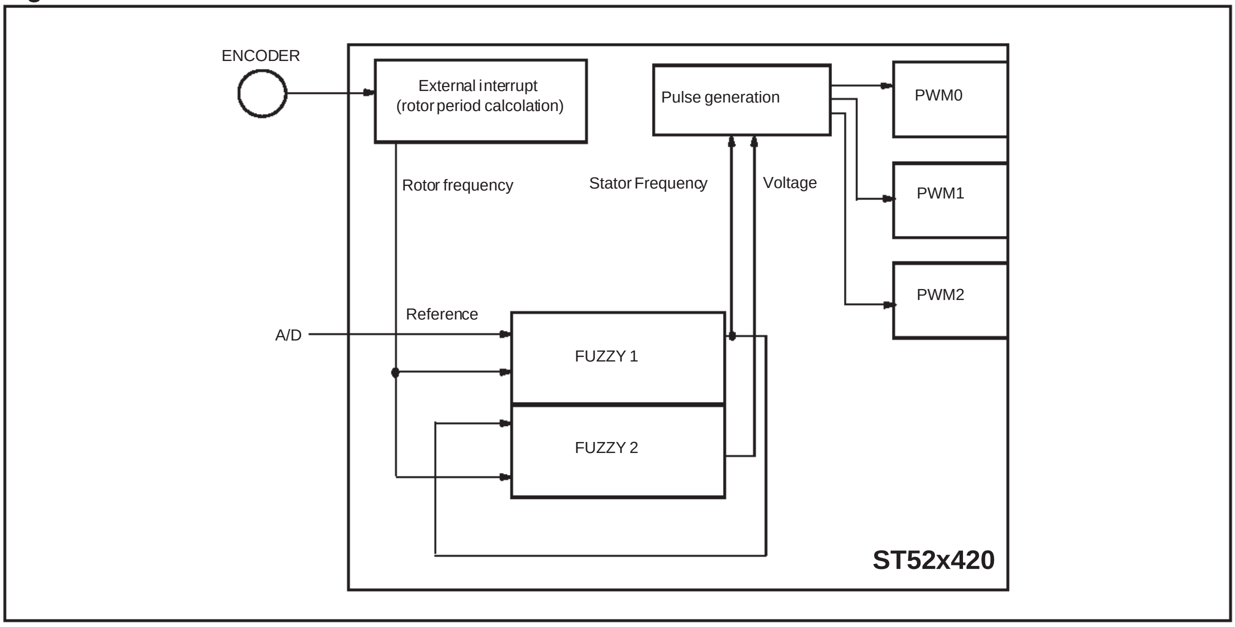
Figure 4. Motor characteristics fixing the stator voltage



2. Control Structure

The aim of the control is to bring the speed of the motor axis to the reference speed maintaining the slip within a certain range fixed by the measures carried out during the modellization phase of the motor. Two fuzzy loops are implemented (Fig.5 and 6):

Figure 5. Control structure

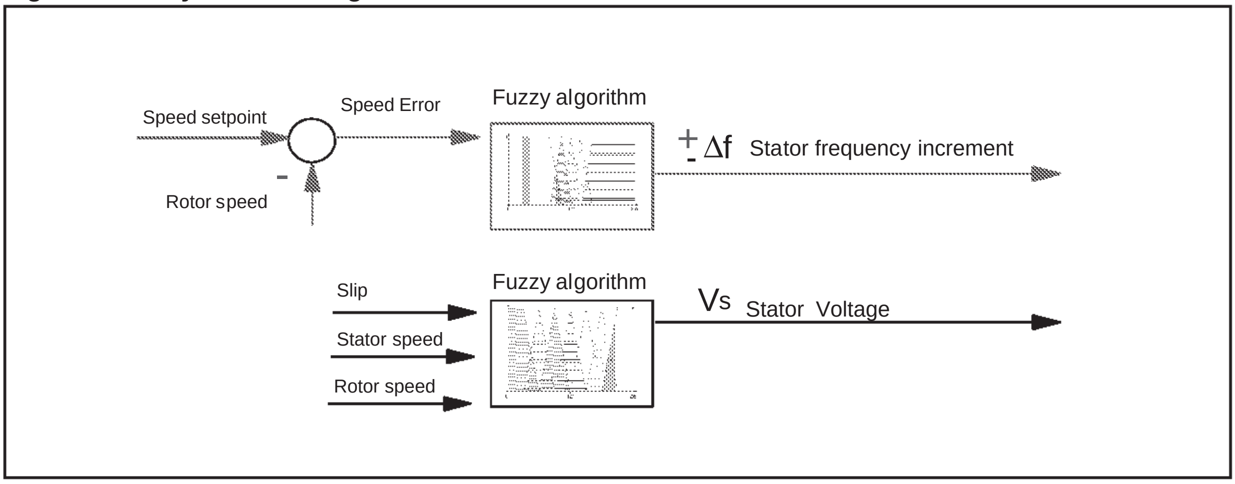


The first fuzzy loop (FUZZY1) is of the incremental type. The input of this Fuzzy block is the speed error given by the difference between the reference speed read through the A/D Converter and the motor speed "*Frotor*" calculated using the external interrupt input where the encoder signal is connected. The output of the block Δf is summed algebraically to the stator frequency Fstator to reach the motor speed to the reference set up.

The second fuzzy loop (FUZZY2) receives in input the difference between "*Fstator*" and "*Frotor*" (slip), and adjusts the voltage level (*voltage*) to optimize the flux and prevent overcurrents in the motor (Fig. 5 and 6).

According to the stator frequency and the desired voltage level, the "*Pulse generator*" block generates three PWM signals to drive the inverter (refer to "*Pulse Construction*" for further information).

Figure 6. Fuzzy Control Diagram



2.1 Fuzzy Controller Algorithm Stator Voltage Loop

The ST52x420 microcontroller thanks to its Fuzzy Logic dedicated architecture, allows the implementation of complex systems such as three-phase motors.

Thanks to the three Timers and to the multiplication and division functions it is possible to obtain the three PWM sinusoidal modulation signals to be supplied to the inverter driver varying, in an independent way, the Frequency and the modulation index.

From a set of preliminary measurements performed on the motor it is possible to build a table that describes the complete functionment of the motor at low and full load in every condition:

rotor frequency [rpm]	accuracy [%]	stator frequency [Hz]	max slip frequency (f-fr) [Hz]	minimum stator voltage [Vs]	max stator voltage [V]	max slip [s%]

More precisely, using the electronic system:

Stator period [msec]	tacho period [msec]	tacho timer value obtained [byte]	max stator timer value [byte]	min stator timer value [byte]	max slip frequency [byte]

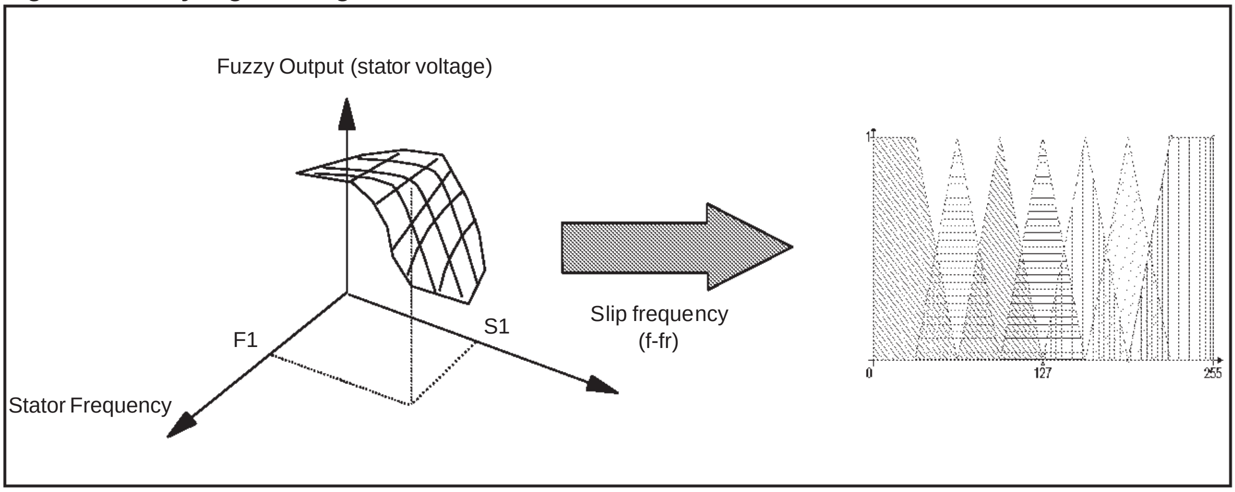
Once the tables have been completed with all the working points we know exactly how to change the stator voltage. Implementing a fuzzy logic interpolation we can modify the voltage by using two Membership Functions inputs that are respectively stator frequency *f* and slip frequency, using a set of rules of the kind:

IF Frequency is Low and slip is Very High then output is High

More precisely, the output of this function, i.e. a function of two variables: Vs=Vs(f,s) that is the required voltage for the motor.

Now, if the loading conditions of the motor are such that the voltage controller is not able to set the motor at the established slip and speed set points, for example under a great increase of the torque in the axis of the motor, then the second controller is activated for the frequency adjustment.

Figure 7. Fuzzy logic voltage control surface



2.2 Fuzzy Controller Algorithm Stator Frequency Loop

Analogously, to build the fuzzy rule for the stator frequency adjustment, we can take into account the speed error to obtain the right increment or decrement for the frequency adjustment. These rules will be of the kind:

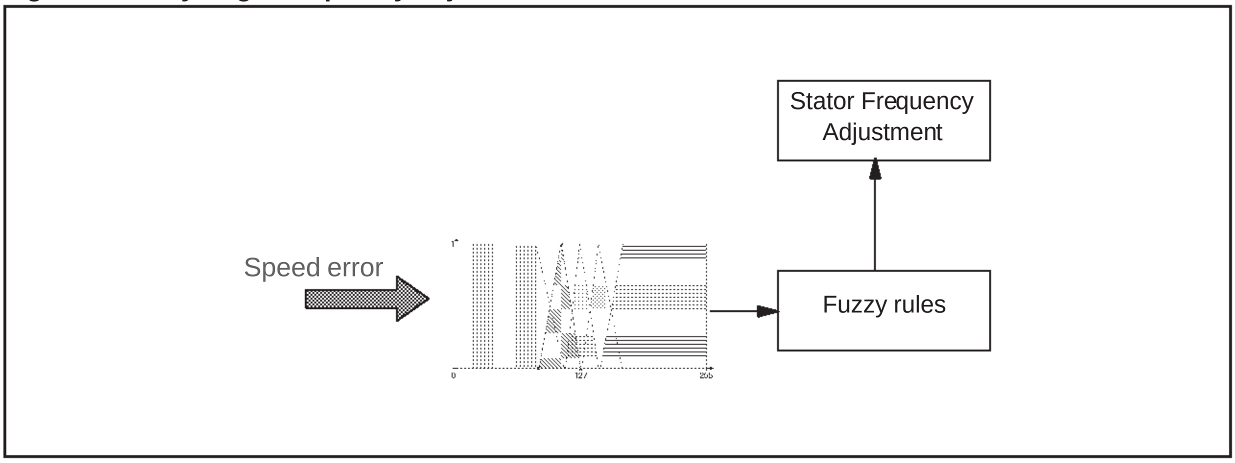
IF Speed Error is Negative Big and then output is Negative Big

More in details, the stator frequency f is equal to:

$$f_{(k)}=f_{(K-1)} \pm \Delta f$$

where Δf is the increment or the decrement provided to the output of the fuzzy controller in order to adjust the rotor speed (Fig.8).

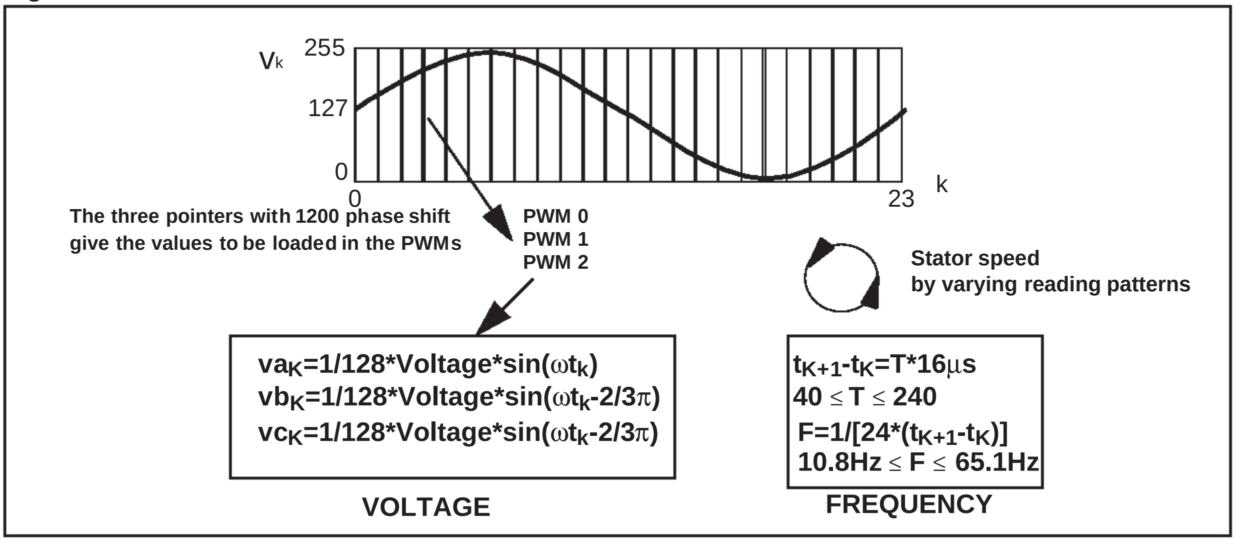
Figure 8. Fuzzy Logic frequency adjustment



2.3 Sinewaves PWM Modulation

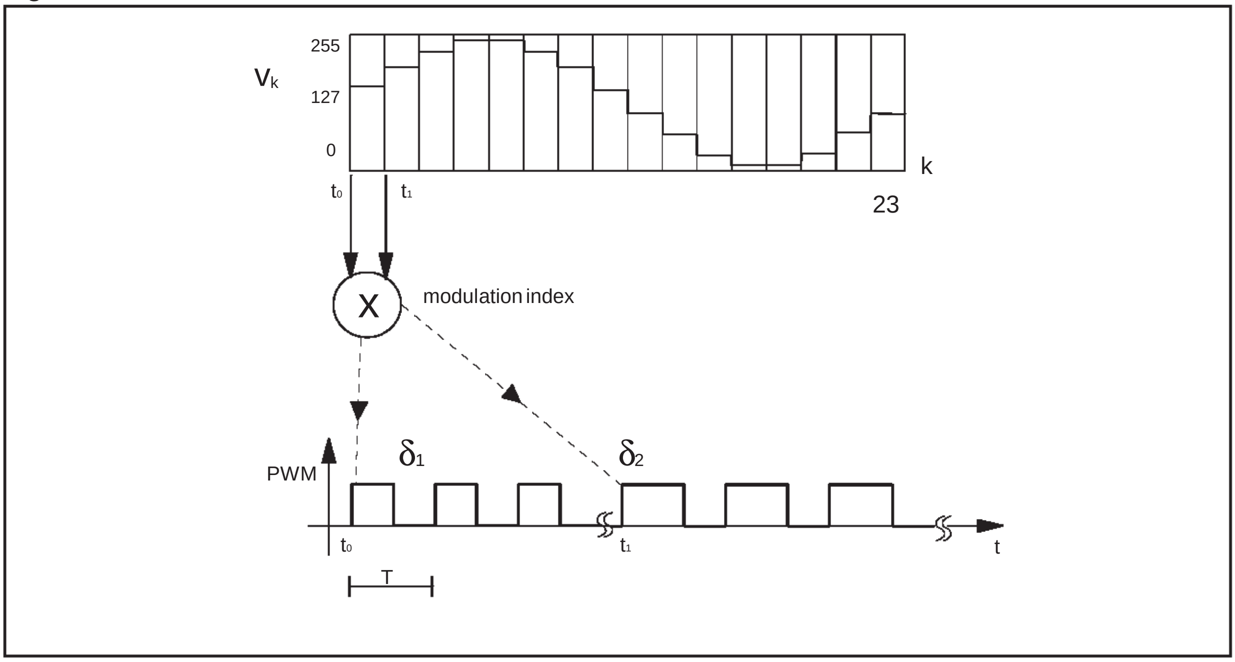
For pulse construction, a 24-byte table, representing the unit sinusoid sampling, is allocated in the internal memory of the MCU. The three sinewaves are drawn by the same table using three 120^{o}-shifted pointers (Fig.9).

Figure 9. Pulses construction



The PWM sinusoidal modulation is obtained scanning the 24 samples with a variable period related to the frequency to be assigned to the motor phases. Each sample is multiplied by the modulation index to change the sinewave amplitude. This is obtained loading this value on the PWM counter thus obtaining a duty-cycle variable that allows to build a sinusoid with an amplitude dependent on the modulation index (Fig.10).

Figure 10. PWM Pulses Construction

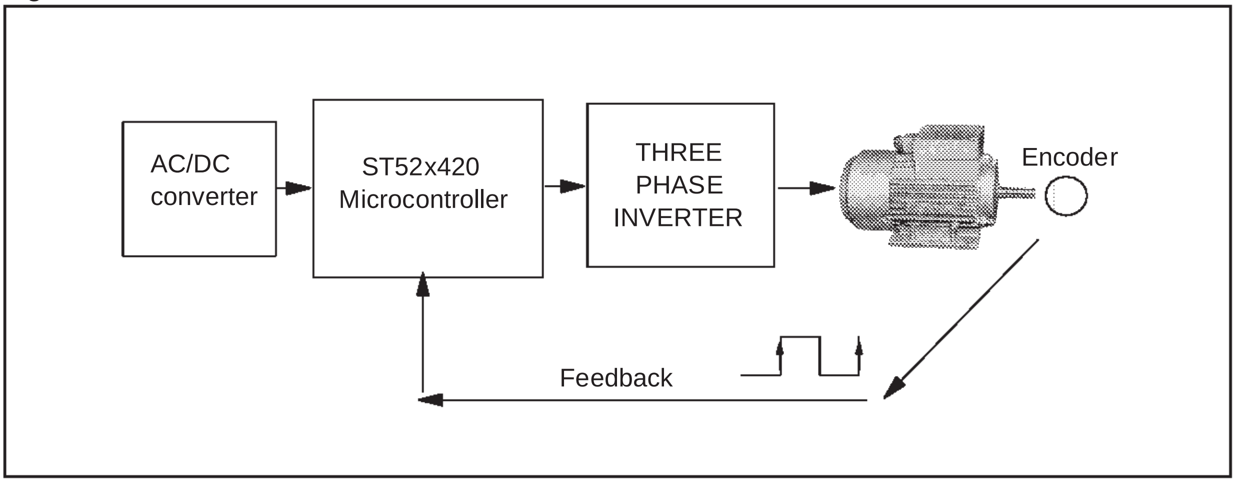


3. HARDWARE IMPLEMENTATION

This application consists of four functional blocks (Fig.11):

- A three-phase asynchronous motor
- A three-phase power inverter
- The closed loop Fuzzy motor control with ST52x420
- An AC-DC converted supplied by the mains

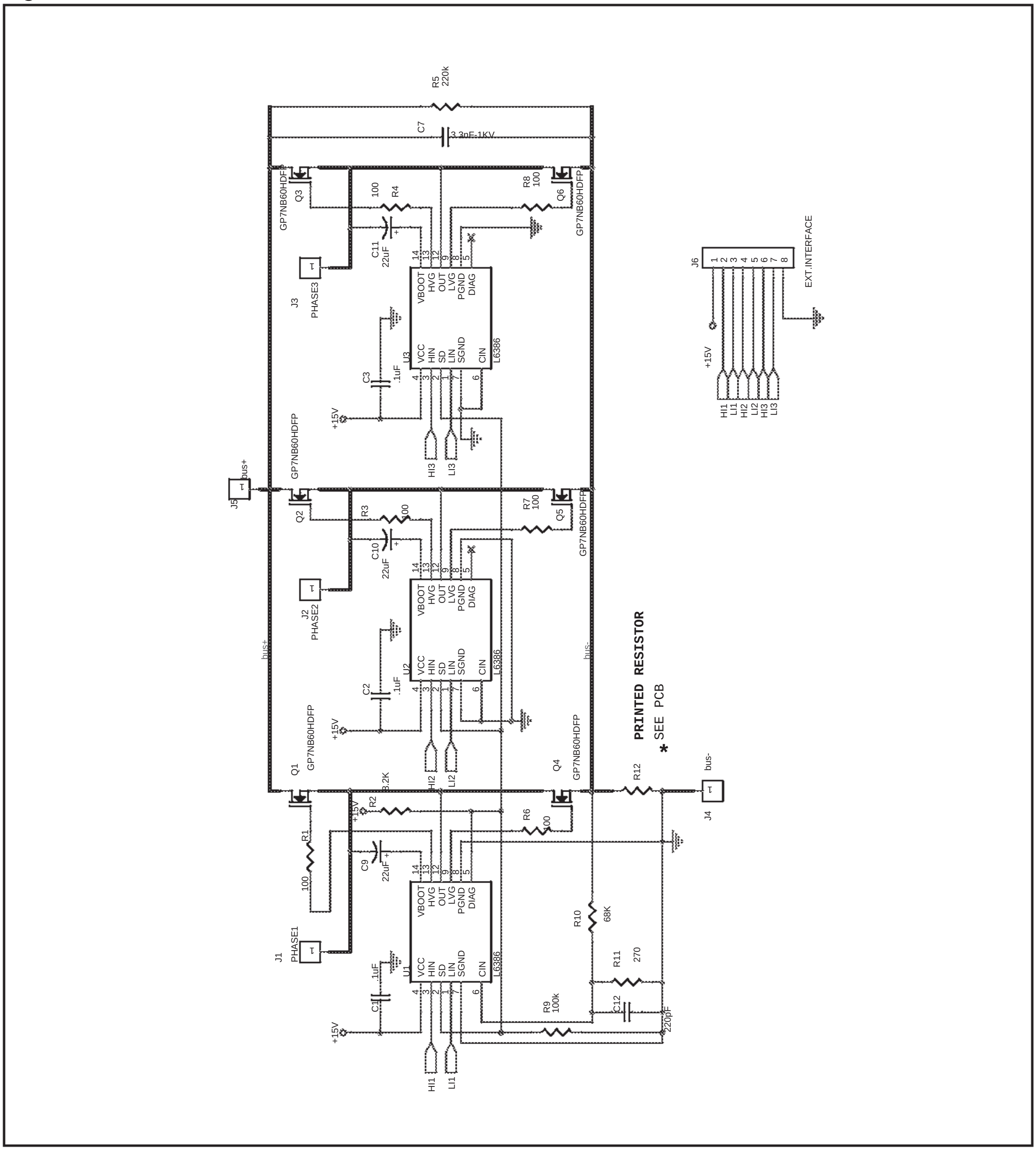
Figure 11. Three-Phase Inverter



3.1 Motor Interface

The motor interface consists of three inverter legs with IGBT or Power Mos, which are driven by the ST L6386 drive (Fig.12).

Figure 12. DC-AC Inverter schematic



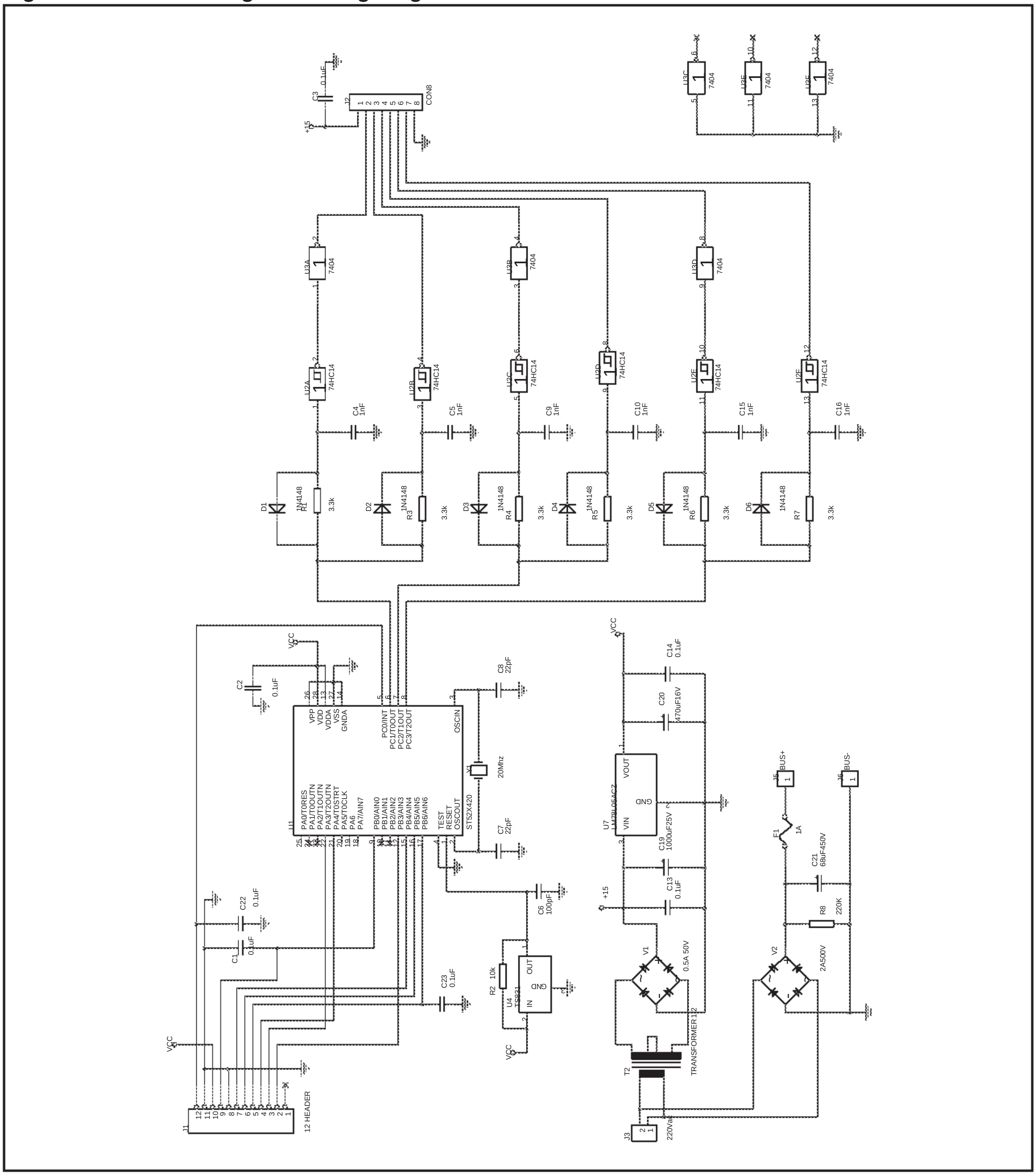
This structure needs DC link voltage +HV, typical value is 325V rectifying AC line voltage. The three-phase motor is connected In the points named R, S, T.

Note: * The PCB can be found on the ST52 Microcontrollers pages at www.st.com/stonline/prodpres/

3.2 Closed Loop Fuzzy Control

In the following figure is shown the complete schematic of the digital control with ST52x420. To generate the six signals to be sent to the inverter section, ST52x420 uses the three-PWM peripheral.

Figure 13. Scheme Diagram for signal generation with ST52x420



These three signals are used from the dead time net in order to obtain all the six signals for the inverter stage to avoid cross conduction in the power switch of each leg.

Figure 14. Components layout of the three-phase inverter

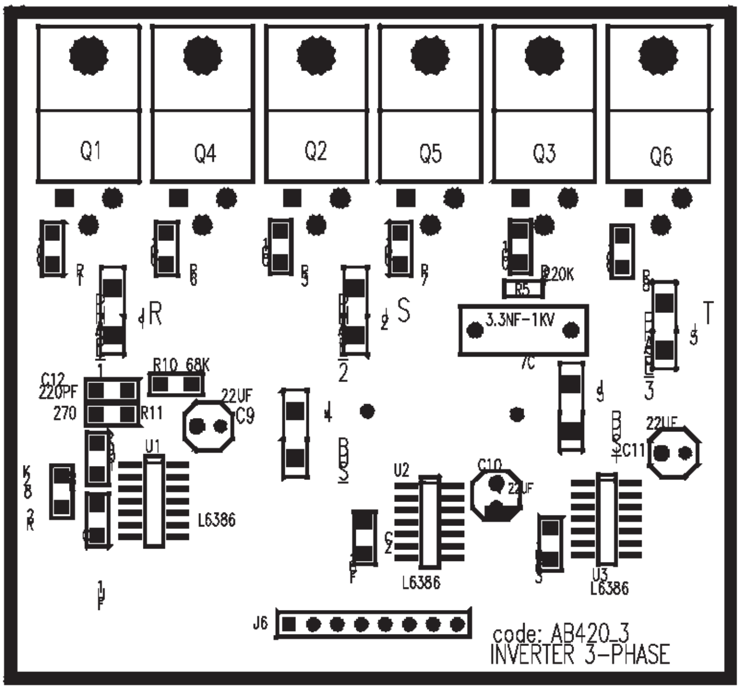


Figure 15. Three-phase inverter board (Power section)

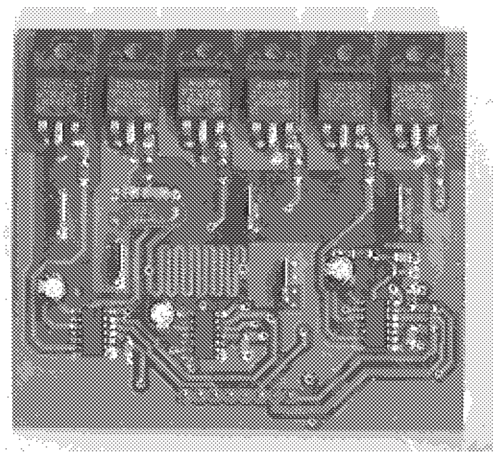


Figure 16. Control board with ST52x420

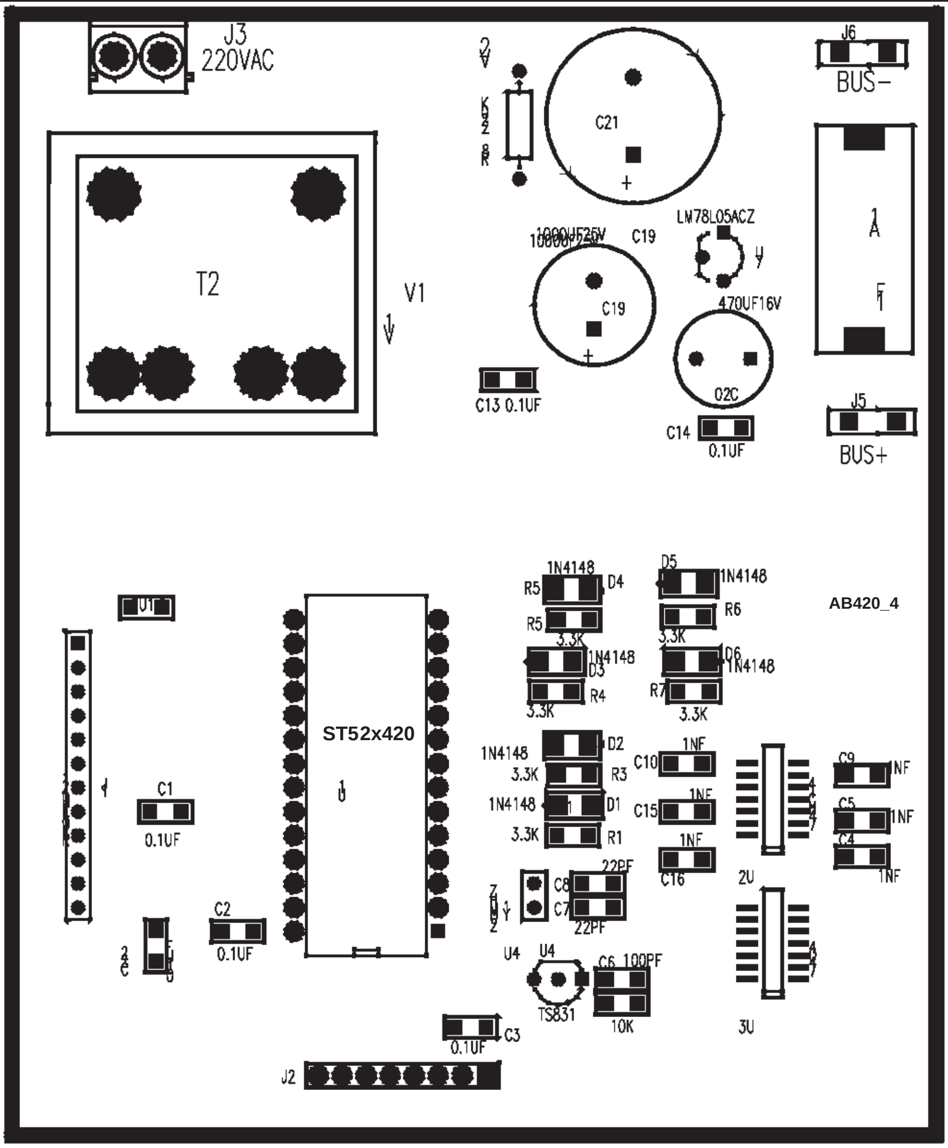
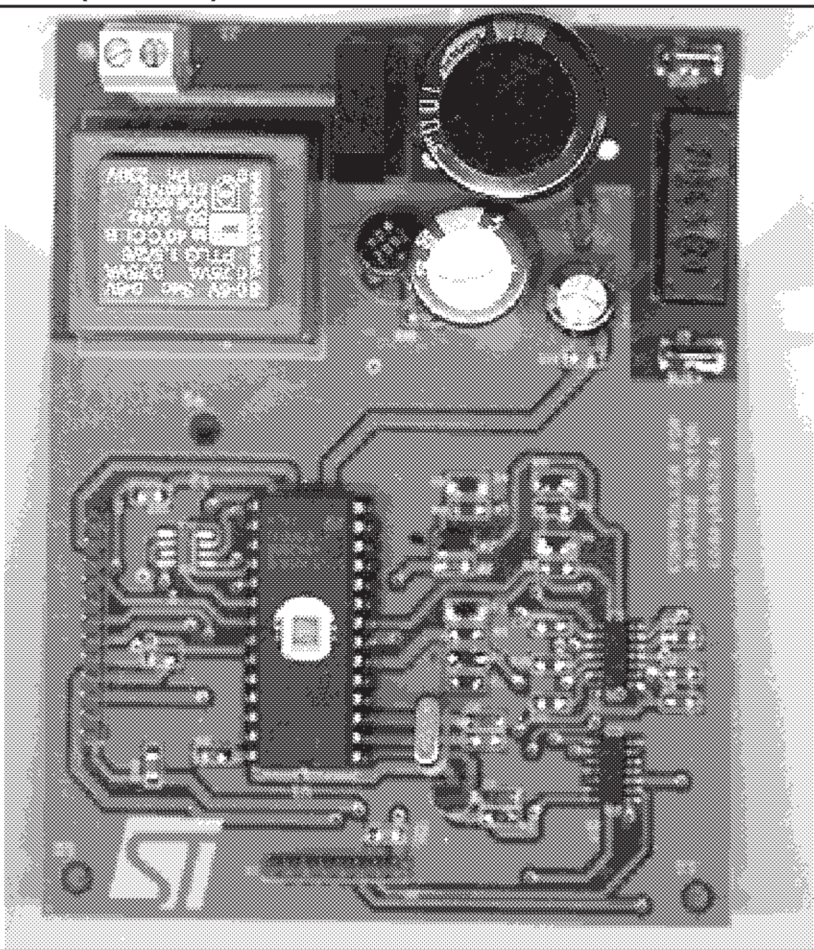


Figure 17. Control board (ST52X420)



4. SOFTWARE DESCRIPTION

Before to analyze the structure of the software project, it is necessary to notice some connections on the schematic. The pins 6, 7 and 8 of ST52x420, (outputs of the three PWM peripherals), are used to drive the three legs of the bridge.

The three-phase voltage is obtained by indexing 3 pointers on the same look-up-table containing the desired PWM pattern at modulation index equal to 1, to reconstruct a sinusoidal signal.

This pattern is recomputed every time for each modulation index, in order to obtain three PWM signals.

One single pointer is shifted on this table, synchronously with one PWM pointer, in order to obtain three phases supplied with 120^{o} phase shift.

Sine period is instead defined by the number N of ADC interrupts:

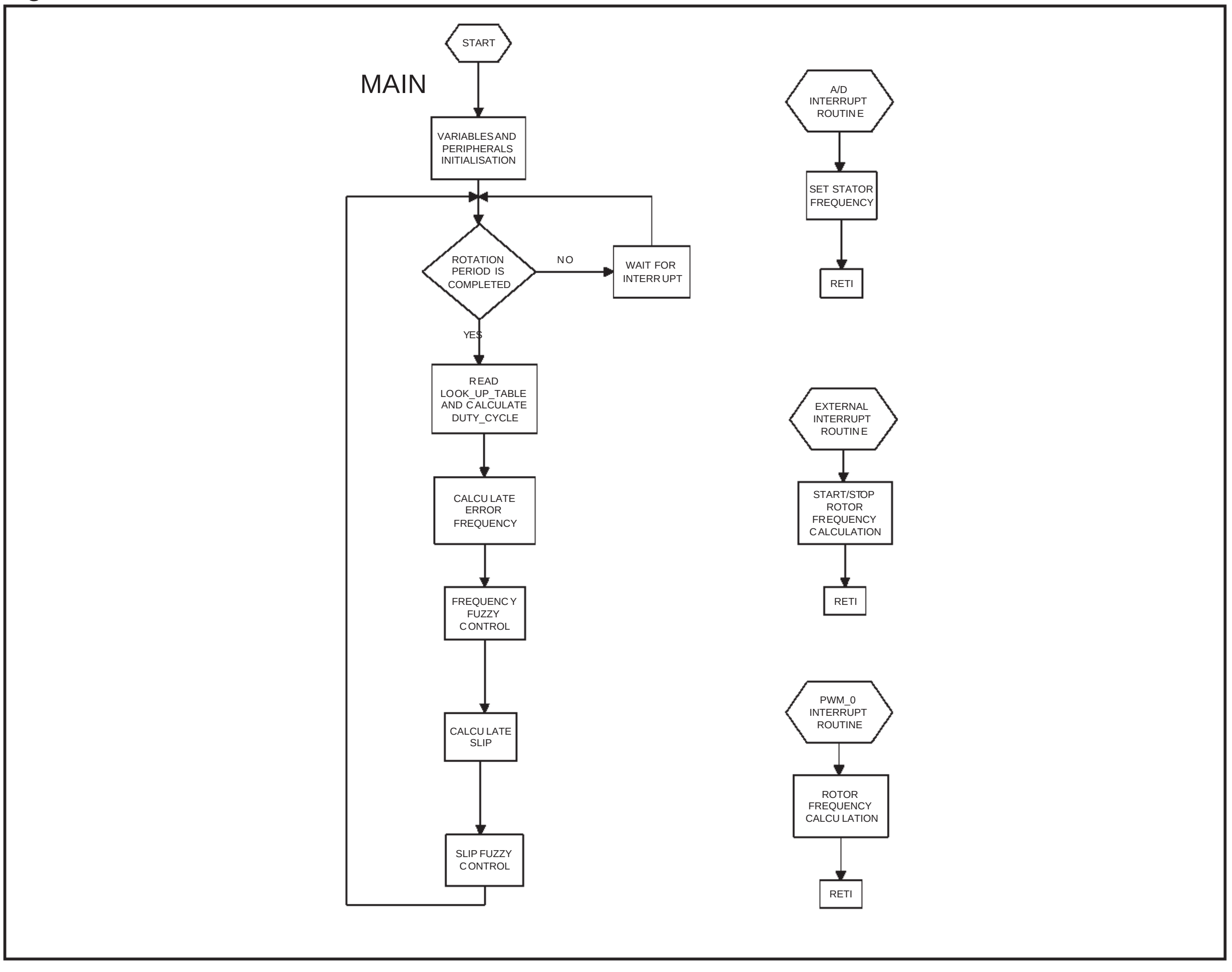
Statoric period = AD_int_counter*AD_int_period*number_of_samples=N*16 µs*24.

In fact, the AD peripheral of ST52x420, besides reading from the pin 9 (Ain0/PB0) the value of reference for the motor speed, is used as time measurer, exploiting the fact that the peripheral requires an interrupt every time that a conversion has been completed (see also paragraph 4). Finally pin 5, configured as external interrupt, is used to measure the instantaneous motor speed by means of a tachometer.

4.1 Main program

The main program is shown in the following flowchart:

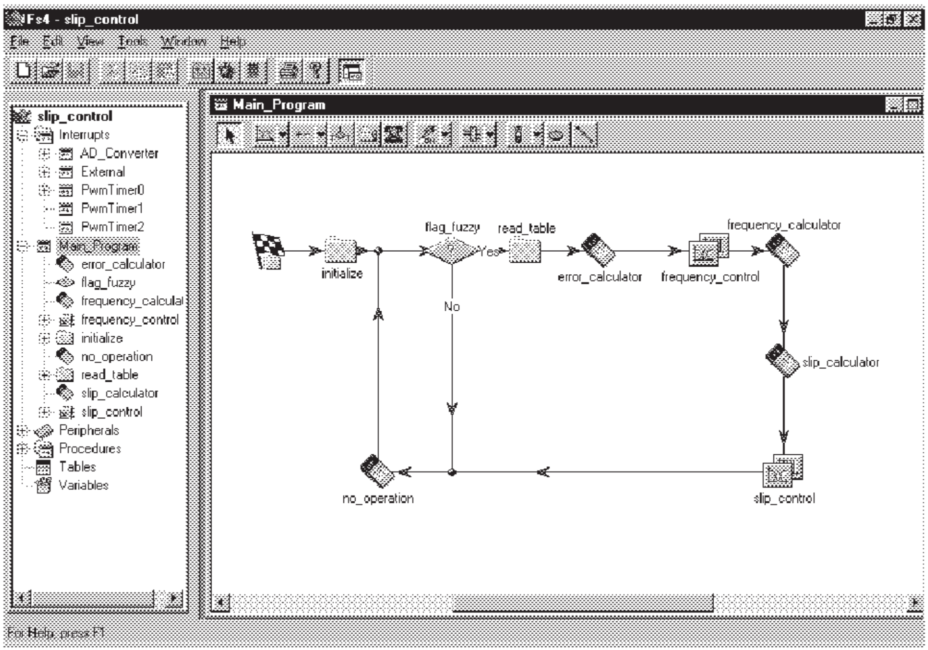
Figure 18. Flowchart



:

In the following figure is shown the main program in FUZZYSTUDIO^{TM}4 environment. The appendix at the end of this application note contains the whole assembler code generated by the compiler.

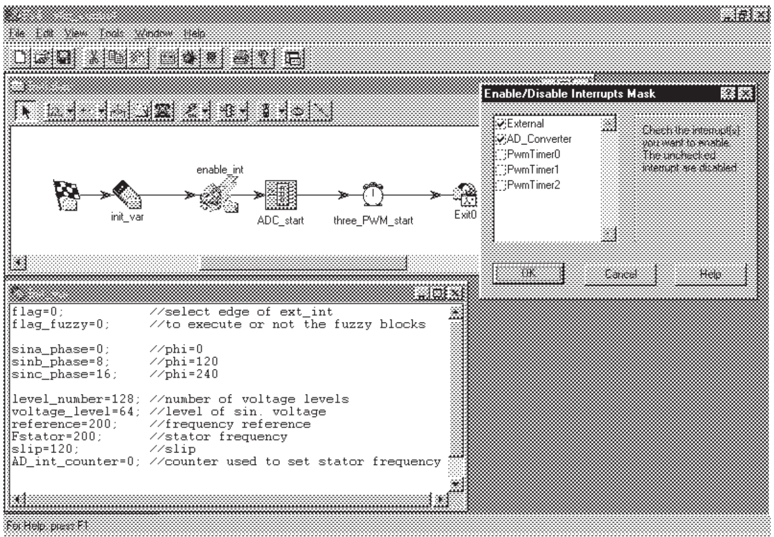
Figure 19. Main Program window



4.2 'Initialize' folder

The folder 'initialize' contains the blocks used to initialize the global variables and the interrupts mask, and to start the peripherals ADC, PWM0, PWM1 and PWM2.

Figure 20. Variables and Peripherals Initialization



4. 3 ’AD interrupt’ routine

The AD interrupt is used as counter; in fact, each 16 µs an interrupt is generated and the counter ’AD_int_counter’ is incremented.

The period between two interrupts is given by the formula:

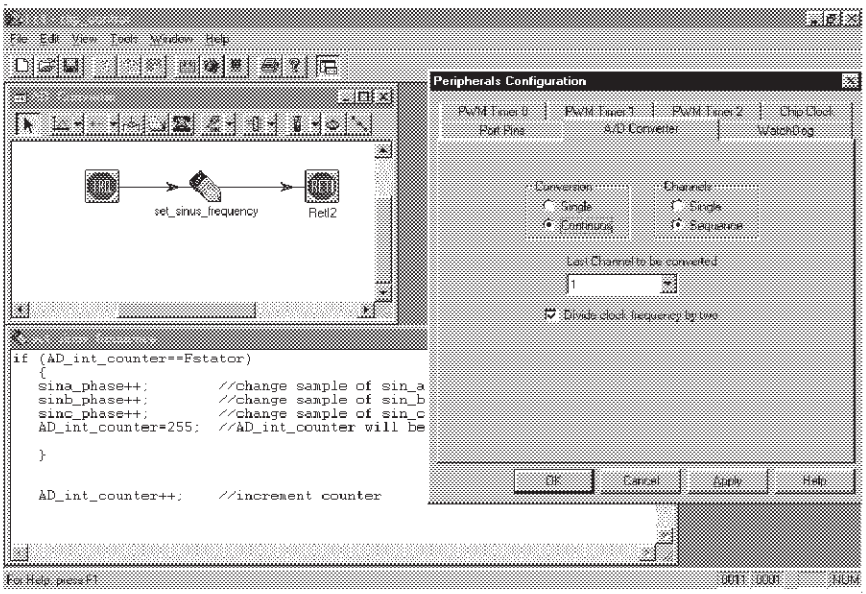
Tconv=number_of_channels*[78*SCK+4]*TCKM

when ’SCK’ is 2 or 1 if AD frequency is divided by 2 or not, and TCKM is the period of the clock master.

In the case described, when the number of channels converted are 2 (0 and 1) and the AD frequency is the clock master frequency (20 Mhz) divided by 2,

Tconv=2*[78*2+4]*50 ns= 16 µs.

Figure 21. A/D Interrupt routine



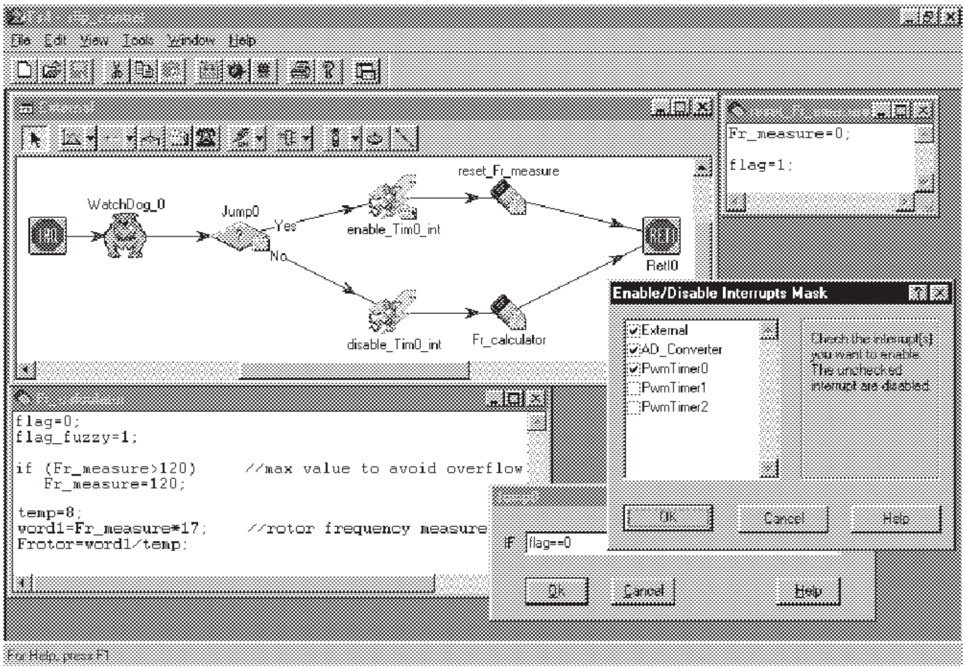
When the counter ’AD_int_counter’ reaches the value ’Fstator’, the pointers are incremented in order to read on the look-up-table the new sample value of the sine wave (see Read_table folder paragraph), and ’AD_int_counter’ is first put to 255, so that, when increased, it is reset.

4.4 ’External interrupt’ and ’PWM0 interrupt’ routines

The external interrupt is used to measure the rotor period; in fact it is measured by counting the time between two positive edges of the square wave supplied by a tachometer system, that is connected to the INT pin; a variable named ’flag’ is used to select the edge. If the variable value is 0, the PWM0_int is enabled, in order to start the calculation of a period, and the variable ’flag’ is set to 1. At the next external interrupt the PWM0_int is disabled and the value reached from the variable ’Fr_measure’ is a measure of the rotor period.

Moreover, the variable ’flag’ is set to 0, in order to restart the calculation at the following edge and the variable ’flag_fuzzy’ is set to 1, in order to perform the fuzzy control (see Fig. 23).

Figure 22. External interrupt routine



Of course the variable 'Fr_measure' is incremented in the 'PWM0_int' routine, and the value obtained between two external interrupt edges must be compared with the value of the stator frequency.

In fact the stator period is:

$$Tstator=Fstator*16\ \mu s*24=(384*Fstator)\ \mu s,$$

instead the rotor period measured with a tacho that gets a period 1/8 of the rotor period is:

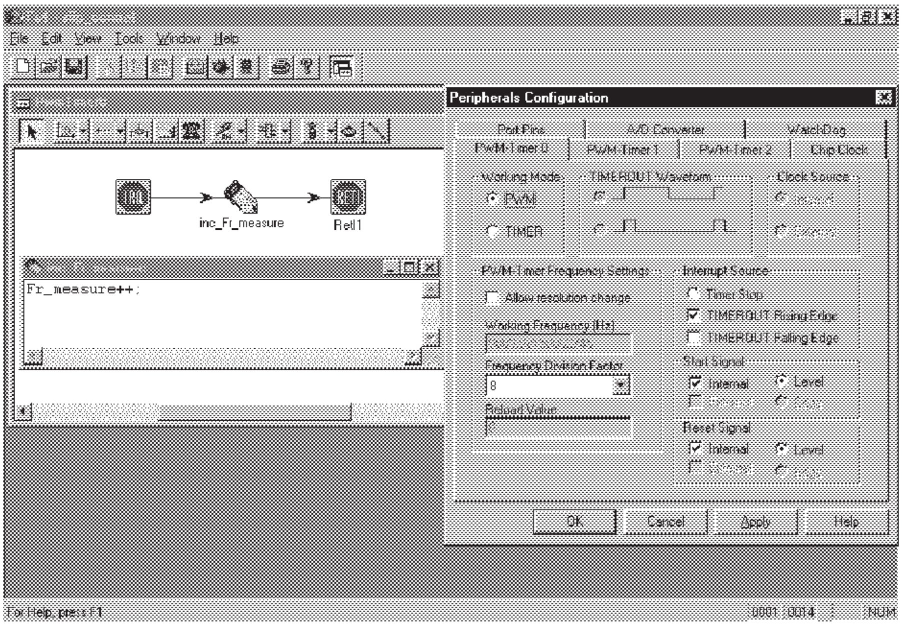
$$Trotor=Fr\_measure*102\ \mu s*8=(816*Fr\_measure)\ \mu s.$$

You have to notice that the value 102 ls is the period of PWM0, corresponding at a frequency value of 9.8 KHz, as you can see in figure 5 in the 'working frequency' box.

In order to make consistent the measure of rotor frequency with that of the stator, you have to use another variable 'Frotor' so that:

$$Frotor=(816/384)*Fr\_measure=(17/8)*Fr\_measure$$

Figure 23. PWM



4.5 ’Read_table’ folder

The block “*read_table*” is used to obtain three PWM signals; each of the three instantaneous duty-cycle values are generated addressing three pointers, (called ’*sina_phase*, *sinb_phase*, *sinc_phase*”) in the look-up-table where unitary and sampled sinewave are stored.

The voltage amplitude of the sinewave is obtained by using the multiplication and division capabilities of ST52x420, as you can see in the figures 24 and 25.

In the block “*read_table0*” the instruction “table_value=sinus[sina_phase]” allows to access the look-up-table ’sinus’ and store the value addressed from the index “*sina_phase*” in the variable “*table_value*”.

Then the subroutine ’voltage’ is called, in order to calculate the duty-cycle in accordance with the modulation index; this procedure is performed three times, for each duty-cycle value, that will be charged in the respective PWM_COUNT with the block “*PWM_COUNT_set*”.

In the block “*duty_cycle_calculator*” the module of the value read from the look-up-table is multiplied by the value “*voltage_level*”, (obtained from fuzzy block “*slip_control*”) and divided by “*level_number*”, in order to obtain the instantaneous duty-cycle. Moreover, the block “*reset_cursor*” is used to control if the values of the indexes reached the maximum, in order to reset them if that happened.

Figure 24. Read Table folder

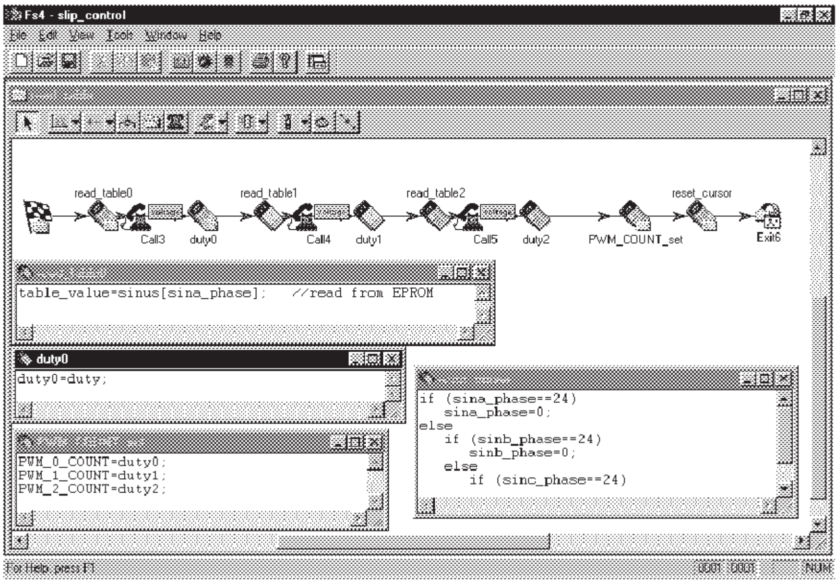
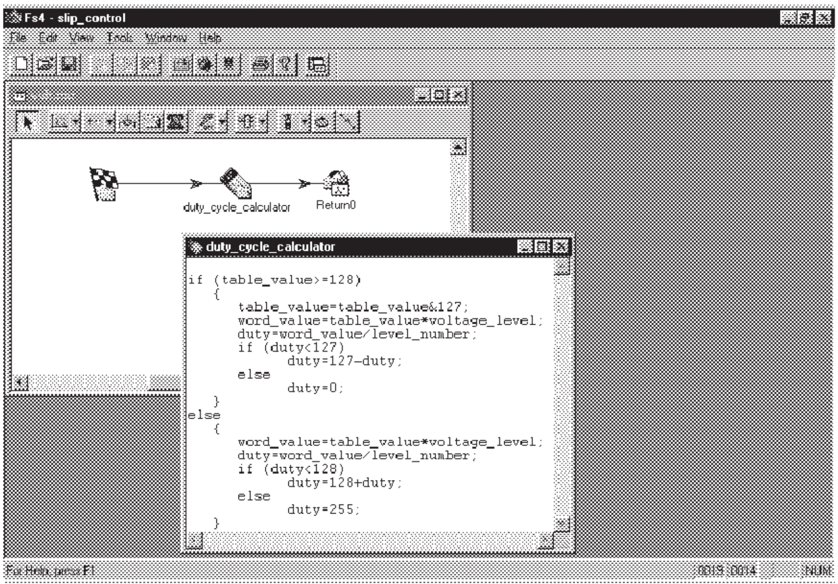


Figure 25. Voltage



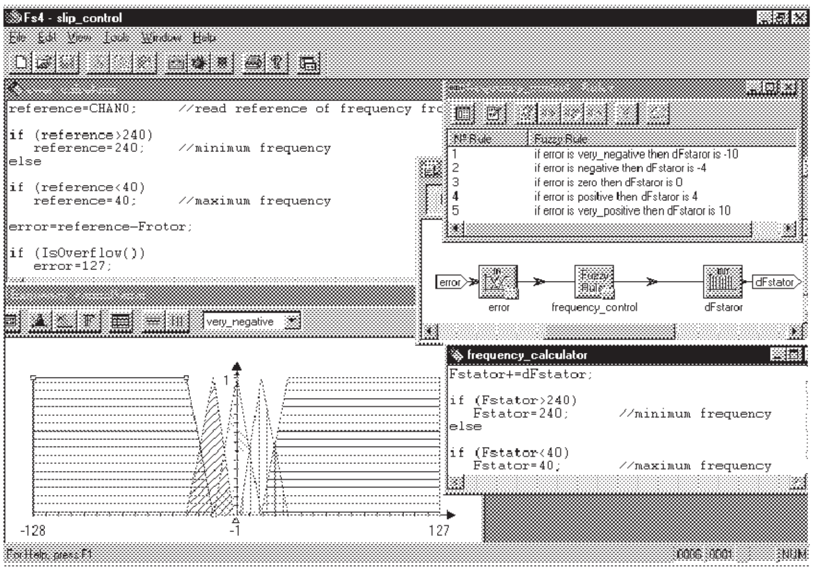
4.6 Fuzzy Controls

Two fuzzy blocks are present in this program: the first is used to control the frequency, the second to control the slip.

The block "*error_calculator*" performs the instruction 'error=reference-Frotor'; "*reference*" is the value desired for the motor speed, that can be varied with a trimmer, in the range '1/(16e-6*24*240)= 10.8Hz -- 1/(16e-6*24*40)=65.1Hz', instead "*Frotor*" is the frequency measured with the tachometer.

Before sending the "*error*" value to the fuzzy input, a control to avoid an overflow or underflow is performed. In according with the input value, the fuzzy block "*frequency_control*" produces the incremental value "dFstator", that is added (with sign), in the block "*frequency_calculator*", to the current value of the variable "*Fstator*". In this way, the speed motor is adjusted to reach the reference value.

Figure 26. Frequency control

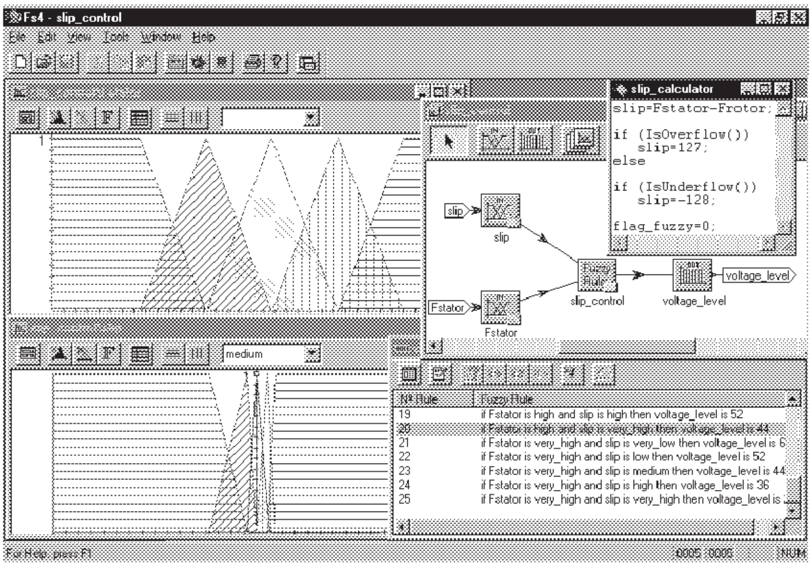


The block "*slip_calculator*" is used to calculate the slip, as 'slip=Fstator-Frotor', with a control to avoid an overflow or underflow.

According with the "*slip*" value and the statoric frequency, the fuzzy block "*slip_control*" calculates the value of the modulation index "*voltage_level*", that allows to adjust the voltage level of the sinusoidal phases.

The memberships and the fuzzy rules of this block represent the mathematical model of the motor and were obtained through experiments with different points of operation.

Figure 27. Slip calculation



APPENDIX A - Assembler code

```
; Interrupt Vector Configuration
   irq  4 External
   irq  0 AD_Converter
   irq  1 PwmTimer0
   irq  2 PwmTimer1
   irq  3 PwmTimer2

; Global MBF Definition
   mbf 0      45 195  45
   mbf 1       6 128   8
   mbf 2      45 240   0
   mbf 3       0  98  24
   mbf 4      45 105  45
   mbf 5      17 113  13
   mbf 6      24 122   6
   mbf 7       0  96  17
   mbf 8       5 140   0
   mbf 9      17 160   0
   mbf 10     15 128  15
   mbf 11      0  60  45
   mbf 12      7 135   5
   mbf 13     45 150  45
   mbf 14     13 143  17

; Tables Allocation

; ”BYTE sinus[24]” use 24 eprom locations from 63(Page:0 Offset:63) to 86(Page:0
Offset:86)
   data 0   63  0   ; sinus[0] = 0
   data 0   64  33  ; sinus[1] = 33
   data 0   65  63  ; sinus[2] = 63
   data 0   66  90  ; sinus[3] = 90
   data 0   67  110 ; sinus[4] = 110
   data 0   68  123 ; sinus[5] = 123
   data 0   69  127 ; sinus[6] = 127
   data 0   70  123 ; sinus[7] = 123
   data 0   71  110 ; sinus[8] = 110
   data 0   72  90  ; sinus[9] = 90
   data 0   73  63  ; sinus[10] = 63
   data 0   74  33  ; sinus[11] = 33
   data 0   75  128 ; sinus[12] = 128
   data 0   76  161 ; sinus[13] = 161
   data 0   77  191 ; sinus[14] = 191
   data 0   78  218 ; sinus[15] = 218
   data 0   79  238 ; sinus[16] = 238
   data 0   80  251 ; sinus[17] = 251
   data 0   81  255 ; sinus[18] = 255
   data 0   82  251 ; sinus[19] = 251
   data 0   83  238 ; sinus[20] = 238
   data 0   84  218 ; sinus[21] = 218
   data 0   85  191 ; sinus[22] = 191
   data 0   86  161 ; sinus[23] = 161

; Tables Allocation Report:
; byte used : 24
; from : 63(Page:0 Offset:63)
; to : 86(Page:0 Offset:86)

   setmem 0   87
```

```
; Store Device Configuration Parameters into Eprom

;   Default Interrupt Priority
    data 0   87  228

; Port Configuration
    data 0   90  0
    data 0   98  248
    data 0   99  243
    data 0   100 3
    data 0   101 241
    data 0   102 0

; A/D Converter Configuration
    data 0   89  58

; WatchDog Configuration
    data 0   88  12

; Pwm-Timer 0  Configuration
    data 0   91  208
    data 0   92  35
    data 0   93  0

; Pwm-Timer 1  Configuration
    data 0   94  208
    data 0   95  35

; Pwm-Timer 2  Configuration
    data 0   96  208
    data 0   97  35

    setmem 0   103
; End ***********************************

; Set Device Configuration Parameters
    pgset 0
    ldce 1   87
    ldce 2   88
    ldce 3   89
    ldce 4   90
    ldce 5   91
    ldce 6   92
    ldce 7   93
    ldce 8   94
    ldce 9   95
    ldce 10  96
    ldce 11  97
    ldce 12  98
    ldce 13  99
    ldce 14  100
    ldce 15  101
    ldce 16  102
    ldrc 0   0
    ldpr 4   0
    ldpr 6   0
    ldpr 8   0


; ** User Defined Variables **
;   NAME -> REG
```

```
; ----------------------------------------
; AD_int_counter ->  10
; Fr_measure ->  11
; Frotor ->  12
; Fstator ->  13
; dFstator ->  14
; duty ->  15
; duty0 ->  16
; duty1 ->  17
; duty2 ->  18
; error ->  19
; flag ->  20
; flag_fuzzy ->  21
; level_number ->  22
; reference ->  23
; sina_phase ->  24
; sinb_phase ->  25
; sinc_phase ->  26
; slip ->  27
; table_value ->  28
; temp ->  29
;  voltage_level ->  30
; word1 ->  31  32
; word_value ->  33  34
; *********************************


main:
; ********** Start procedures "main"

Start:

initialize:

init_var:
   ldrc 20  0
   ldrc 21  0
   ldrc 24  0
   ldrc 25  8
   ldrc 26  16
   ldrc 22  128
   ldrc 30  64
   ldrc 23  200
   ldrc 13  200
   ldrc 27  248
   ldrc 10  0
enable_int:
; IrqEnableMask
   mdgi
   ldrc 0   3
   ldcr 0   0
   megi

ADC_start:
; ADC Setting
   mdgi
   ldrc 0   63
   ldcr 3   0
   megi

three_PWM_start:
; ALL_PWM Setting
```

```
    mdgi
    ldrc 0   224
    ldcr 7   0
    ldrc 0   213
    ldcr 5   0
    ldrc 0   213
    ldcr 8   0
    ldrc 0   213
    ldcr 10  0
    ldrc 0   0
    ldcr 7   0
    megi

Exit0:
    jp   initialize_Exit

initialize_Exit:

flag_fuzzy:
    mdgi
    ldrc 0   1
    sub  0   21
    megi
    jpnz End_If_6
    jp   read_table

End_If_6:

no_operation:
    jp   flag_fuzzy

read_table:

read_table0:
    mdgi
    ldrr 0   24
    ldrc 28  63
    add  28  0
    pgset 0

Read:
    ldrc 0   28
    ldre (0) (28)
    megi

Call3:
    call voltage

duty0:
    ldrr 16  15

read_table1:
    mdgi
    ldrr 0   25
    ldrc 28  63
    add  28  0
    pgset 0

Read_1:
    ldrc 0   28
    ldre (0) (28)
    megi
```

```
Call4:
   call voltage

duty1:
   ldrr 17  15

read_table2:
   mdgi
   ldrr 0   26
   ldrc 28  63
   add  28  0
   pgset 0

Read_2:
   ldrc 0   28
   ldre (0) (28)
   megi

Call5:
   call voltage

duty2:
   ldrr 18  15

PWM_COUNT_set:
   ldpr 3   16
   ldpr 5   17
   ldpr 7   18

reset_cursor:
   mdgi
   ldrc 0   24
   sub  0   24
   megi
   jpnz No_If_9
   ldrc 24  0
   jp   End_If_9

No_If_9:
   mdgi
   ldrc 0   24
   sub  0   25
   megi
   jpnz No_If_8
   ldrc 25  0
   jp   End_If_8

No_If_8:
   mdgi
   ldrc 0   24
   sub  0   26
   megi
   jpnz End_If_7
   ldrc 26  0

End_If_7:

End_If_8:

End_If_9:

Exit6:
   jp   read_table_Exit
```

```
read_table_Exit:

error_calculator:
    ldri 23  1
    mdgi
    ldrc 0   240
    sub  0   23
    megi
    jpns No_If_11
    ldrc 23  240
    jp   End_If_11

No_If_11:
    mdgi
    ldrc 0   40
    ldrr 1   23
    sub  1   0
    megi
    jpns End_If_10
    ldrc 23  40

End_If_10:

End_If_11:
    mdgi
    ldrr 19  23
    subo 19  12
    megi
    jpnc No_If_13
    ldrc 19  255
    jp   End_If_13

No_If_13:
    jpns End_If_12
    ldrc 19  0

End_If_12:

End_If_13:

frequency_control:
; Fuzzy System : frequency_control
    mdgi

;   Init error
    ldfr 0   19

; Output Variable : dFstaror
    fuzzy
    ldp   0   7
    ldp   0   7
    fzand
    con   118
    ldp   0   5
    ldp   0   5
    fzand
    con   124
    ldp   0   10
    ldp   0   10
    fzand
    con   128
    ldp   0   14
```

```
    ldp   0   14
    fzand
    con   132
    ldp   0   9
    ldp   0   9
    fzand
    con   138
    out   14
    megi
;
; End Fuzzy System : frequency_control

frequency_calculator:
    mdgi
    addo 13  14
    megi
    mdgi
    ldrc 0   240
    sub  0   13
    megi
    jpns No_If_15
    ldrc 13  240
    jp   End_If_15

No_If_15:
     mdgi
    ldrc 0   40
    ldrr 1   13
    sub  1   0
    megi
    jpns End_If_14
    ldrc 13  40

End_If_14:

End_If_15:

slip_calculator:
    mdgi
    ldrr 27  13
    subo 27  12
    megi
    jpnc No_If_17
    ldrc 27  255
    jp   End_If_17

No_If_17:
    jpns End_If_16
    ldrc 27  0

End_If_16:

End_If_17:
    ldrc 21  0

slip_control:
;   Fuzzy System : slip_control
    mdgi

;   Init slip
    ldfr 0   27

;   Init Fstator
```

```
   ldfr 1   13

; Output Variable : voltage_level
   fuzzy
   ldp   1   11
   ldp   0   3
   fzand
   con   124
   ldp   1   11
   ldp   0   6
   fzand
   con   116
   ldp   1   11
   ldp   0   1
   fzand
   con   108
   ldp   1   11
   ldp   0   12
   fzand
   con   100
   ldp   1   11
   ldp   0   8
   fzand
   con   92
   ldp   1   4
   ldp   0   3
   fzand
   con   108
   ldp   1   4
   ldp   0   6
   fzand
   con   100
   ldp   1   4
   ldp   0   1
   fzand
   con   92
   ldp   1   4
   ldp   0   12
   fzand
   con   84
   ldp   1   4
   ldp   0   8
   fzand
   con   76
   ldp   1   13
   ldp   0   3
   fzand
   con   92
   ldp   1   13
   ldp   0   6
   fzand
   con   84
   ldp   1   13
   ldp   0   1
   fzand
   con   76
   ldp   1   13
   ldp   0   12
   fzand
   con   68
   ldp   1   13
   ldp   0   8
   fzand
```

```
   con   60
   ldp   1   0
   ldp   0   3
   fzand
   con   76
   ldp   1   0
   ldp   0   6
   fzand
   con   68
   ldp   1   0
   ldp   0   1
   fzand
   con   60
   ldp   1   0
   ldp   0   12
   fzand
   con   52
   ldp   1   0
   ldp   0   8
   fzand
   con   44
   ldp   1   2
   ldp   0   3
   fzand
   con   60
   ldp   1   2
   ldp   0   6
   fzand
   con   52
   ldp   1   2
   ldp   0   1
   fzand
   con   44
   ldp   1   2
   ldp   0   12
   fzand
   con   36
   ldp   1   2
   ldp   0   8
   fzand
   con   28
   out   30
   megi
;
; End Fuzzy System : slip_control
   jp   no_operation
;
; End procedures ”main” *****


AD_Converter:
; ** Start procedures ”AD_Converter”

set_sinus_frequency:
   mdgi
   ldrr 0   10
   sub  0   13
   megi
   jpnz End_If
   mdgi
   inc  24
   megi
   mdgi
```

```
   inc  25
   megi
   mdgi
   inc  26
   megi
   ldrc 10  255

End_If:
   mdgi
   inc  10
   megi

RetI2:
   reti
;
; End procedures ”AD_Converter” *****


PwmTimer0:
; ****** Start procedures ”PwmTimer0”

inc_Fr_measure:
   mdgi
   inc  11
   megi

RetI1:
   reti
;
; End procedures ”PwmTimer0” ******


PwmTimer1:
;****** Start procedures ”PwmTimer1”
   reti
;
; End procedures ”PwmTimer1”*******


PwmTimer2:
;****** Start procedures ”PwmTimer2”
   reti
;
; End procedures ”PwmTimer2”********


External:
; ******* Start procedures ”External”

WatchDog_0:
; WDT Setting
   wdtrfr

Jump0:
   mdgi
   ldrc 0   0
   sub  0   20
   megi
   jpnz End_If_1
   jp   enable_Tim0_int

End_If_1:
```

```
disable_Tim0_int:
; IrqEnableMask
    mdgi
    ldrc 0   3
    ldcr 0   0
    megi

Fr_calculator:
    ldrc 20  0
    ldrc 21  1
    mdgi
    ldrc 0   120
    sub  0   11
    megi
    jpns End_If_2
    ldrc 11  120
End_If_2:
    ldrc 29  8
    mdgi
    ldrc 31  17
    mult 31  11
    megi
    mdgi
    ldrr 0   31
    ldrr 1   32
    div  0   29
    ldrr 12  1
    megi

RetI0:
    reti

enable_Tim0_int:
; IrqEnableMask
    mdgi
    ldrc 0   7
    ldcr 0   0
    megi

reset_Fr_measure:
    ldrc 11  0
    ldrc 20  1
    jp   RetI0
;
; End procedures ”External” *********


voltage:
; ******* Start procedures ”voltage”

duty_cycle_calculator:
    mdgi
    ldrc 0   128
    ldrr 1   28
    sub  1   0
    megi
    jps  No_If_5
    mdgi
    ldrc 0   127
    and  28  0
    megi
    mdgi
    ldrr 33  28
```

```
    mult 33  30
    megi
    mdgi
    ldrr 0   33
    ldrr 1   34
    div  0   22
    ldrr 15  1
    megi
    mdgi
    ldrc 0   127
    ldrr 1   15
    sub  1   0
    megi
    jpns No_If_3
    mdgi
    ldrc 0   127
    sub  0   15
    ldrr 15  0
    megi
    jp   End_If_3

No_If_3:
    ldrc 15  0

End_If_3:
    jp   End_If_5

No_If_5:
    mdgi
    ldrr 33  28
    mult 33  30
    megi
    mdgi
    ldrr 0   33
    ldrr 1   34
    div  0   22
    ldrr 15  1
    megi
    mdgi
    ldrc 0   128
    ldrr 1   15
    sub  1   0
    megi
    jpns No_If_4
    mdgi
    ldrc 0   128
    add  15  0
    megi
    jp   End_If_4

No_If_4:
    ldrc 15  255

End_If_4:

End_If_5:

Return0:
    ret
;
; End procedures ”voltage” ************
```