AN1362
APPLICATION NOTE

Switched Reluctance Motor Control by ST FIVE

AUTHOR: N. ABBATE

ABSTRACT

This application note proposes a solution to drive the Switched Reluctance Motor (SRM) by using a low cost 8-bit microcontroller. This work has been developed for a vacuum cleaner application in order to reach a high rotational speed and low acoustic and electrical emissions.

Keywords: SRM, Switched Reluctance Motor, Synchronous Motor.

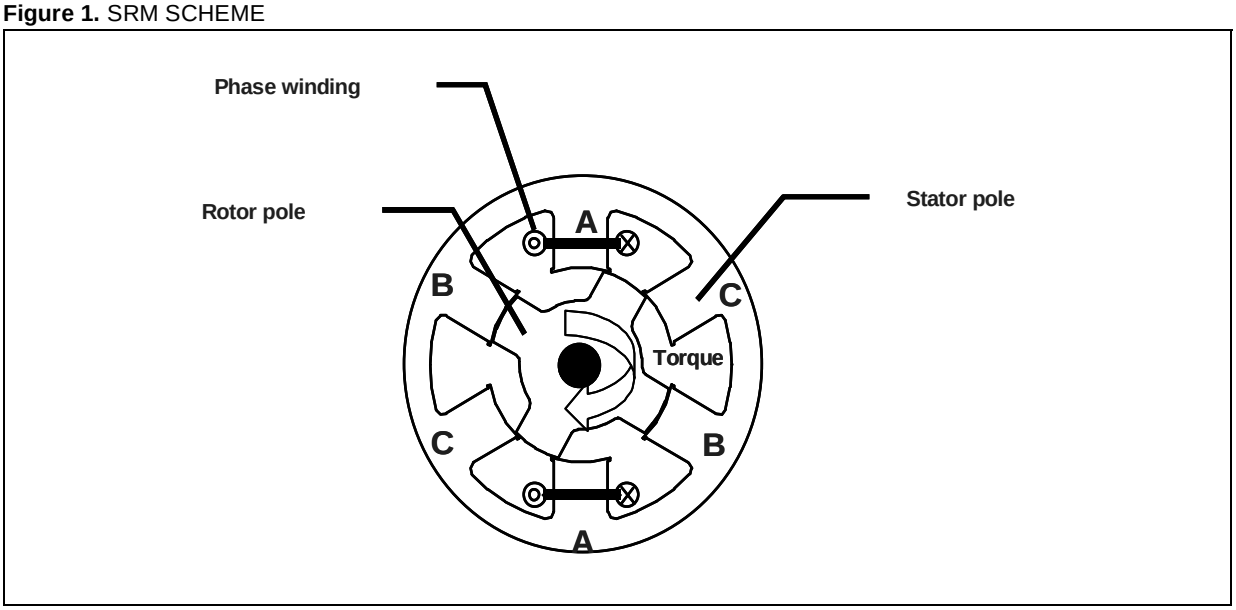
INTRODUCTION

Although SRM motors have been well known for a long time, only today's industry is paying attention to these kinds of electrical machines. The decreasing costs of microcontrollers and inverter stages, the need of compliance with EMI standards, together with the simplicity in motor assembly are driving the home-appliance market's attention to synchronous motors, such as Permanent Magnet Brushless DC and Switched Reluctance. These machines, that belong to the group of Electrically Commuted Motors, need rotor position sensors in order to know where the rotor poles are located in the space during rotor revolution. Good starting torque, wide speed range, absence of brushes and of expensive magnetic material make SRM motors very interesting electromechanical drivers for the consumer market.

SRM OVERVIEW

A SRM can be considered as a big step-motor where both the stator and the rotor present salient poles. In the SRM no permanent magnets are used therefore the magnetic flux is produced by the stator coils (Unique Flux Synchronous Machine). The rotor is composed of laminated iron sheets, which are stacked on the shaft. Two bearing-ball ends allow the rotor shaft to rotate. The stator is a typical N-phase motor and presents a pair of salient poles for each phase coil.
Fig.1 shows a transversal section of the motor. Typical pole configurations are: 4stator/2rotor poles, 6/4, 8/6, 12/8, ...

Figure 1. SRM SCHEME



Generally, the equation between stator poles Ns and rotor poles Nr is the following:

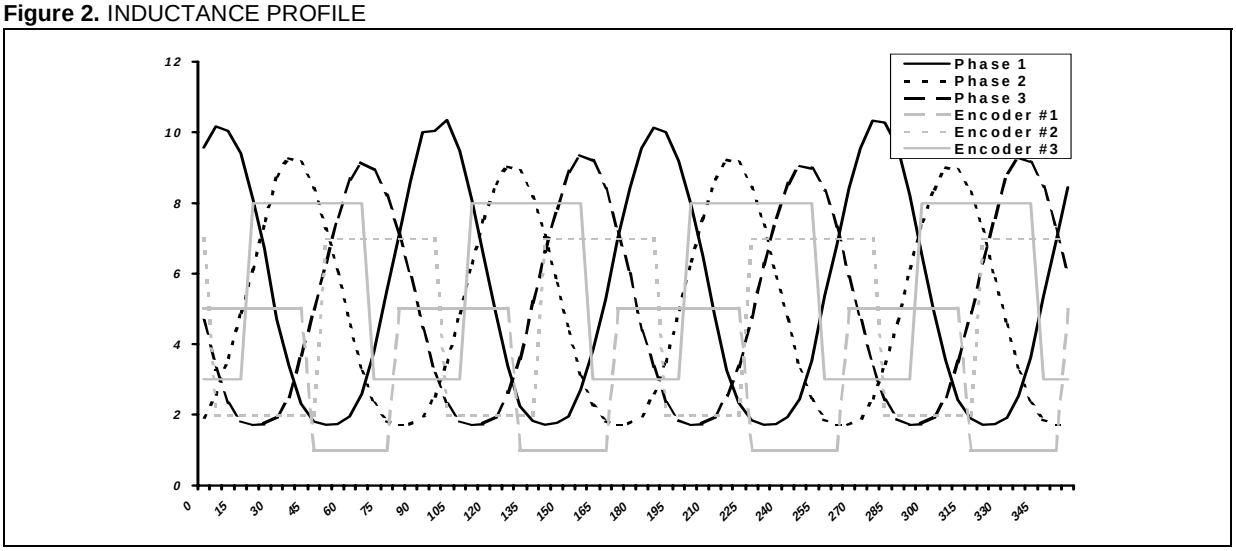
$$Nr = Ns \pm \frac{Ns}{q}$$

where *q* is the number of phases.
Due to the pole saliency, the magnetic path is different during the rotor motion since the reluctance is minimal when stator and rotor poles are aligned and maximum when poles are not aligned. Formally, the inductance can be written as follows:

$$L(\theta) = \frac{N^2}{\Re(\theta)}$$

with N number of coils and θ rotation angle of the motor shaft.
The equation above and the reluctance profile are responsible for the inductance profile shown in figure 2, which is measured on the motor used in this application: on x-axis the rotor mechanical position (in degree), on y-axis the measure of inductance (mH).

Figure 2. INDUCTANCE PROFILE

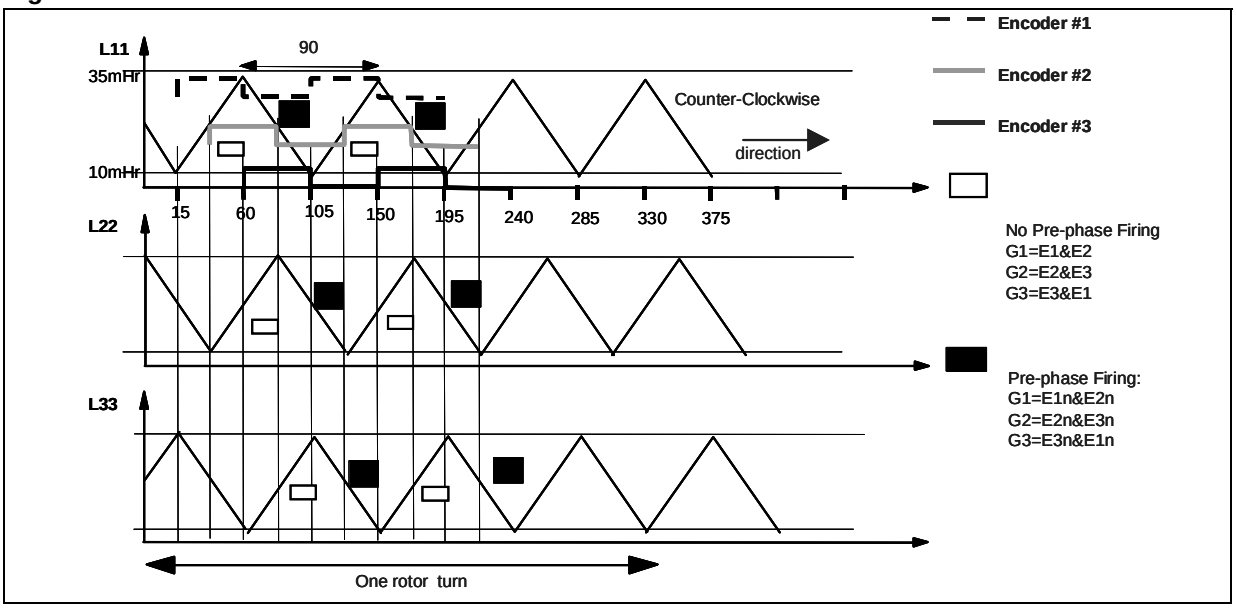


Energy and inductance considerations [1][3] lead to the following definition of the motor torque expression :

$$T_{em} = \frac{1}{2}\left(i^2 \cdot \frac{dL}{d\theta}\right)$$

From the equation above, one may argue that, in order to obtain a positive torque, the phase coil needs current only during the rising edge of the inductance profile.
The synchronization of the phase polarization with the rotor poles position measured through the Hall sensors is the main task of the electronic driver.
There are different ways to arrange the sensors: one of the simplest is to arrange the sensor position in order to obtain the necessary information about the rising/falling edge of 'L' for each phase.
Fig.3 shows a typical sensor arrangement for a 3-phase 6/4 SRM.
Since the phase coil is a big inductance with a low series resistance, the current rising time is strongly related to the L/R constant time of the circuit. At a very high rotational speed, this rising current time is comparable to duration of the L-rising edge. This does not allow the current to rise at suitable levels to deliver the expected torque. In this case, a temporal coils energizing advance (prephase injection) is introduced in the phases in correspondance with the L-rising profile beginning.

Figure 3. SENSOR PLACEMENT AND DRIVING



Generally, the introduction of the pre-phase current injection is achieved by adjusting the position sensor of few mechanical degrees forwards or backwards with respect to the rotor rotation.

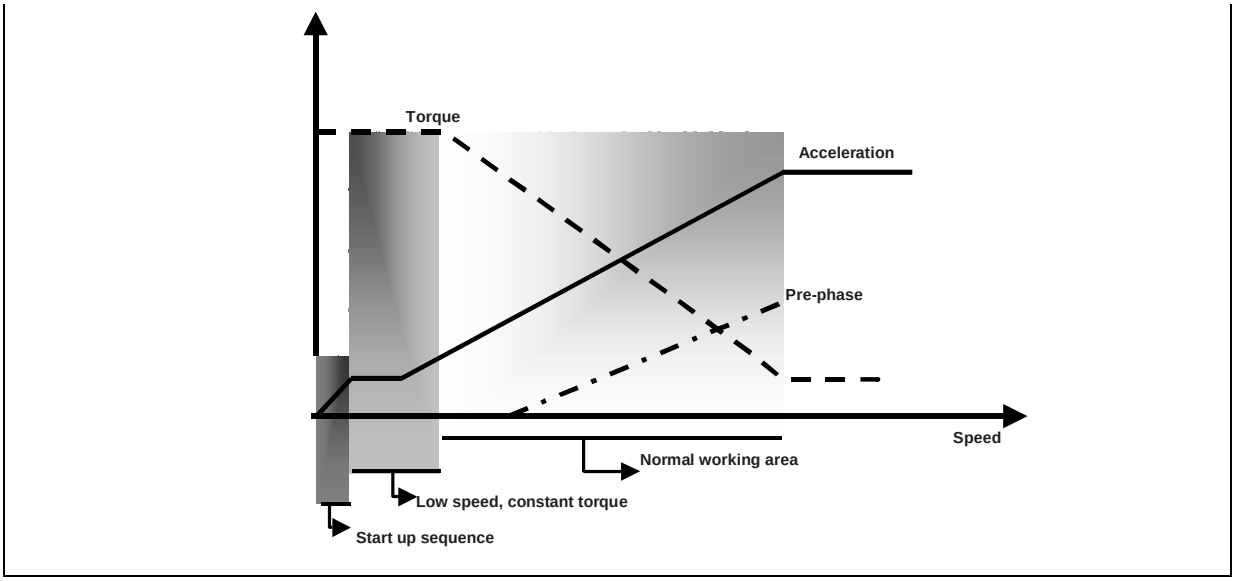
However, since sensors were permanently locked on the rotor in the motor being explained, the pre-phase injection could be applied only through a fixed value of advance. In this case, the current values are suited only for a specific speed range, in which the motor achieves the required efficiency.

For high speed values, the drawback of having fixed sensor configuration, as in the case of the motor being explained, can be overcome by adopting the strategy of extracting information from the combination of signals coming from Hall sensors to change run time the prephase value for the current injection.

This solution is enhanced further through a “current tail cut“ technique, which is described in the following.

These circumstances are in conflict with the research of techniques allowing the achievement of a very high speed with a reasonable torque. A sequence of different logical functions may modify the input encoder signal by changing the driving policy to “run time”. This method is up-graded by using a Timer to switch off the phase signal.

Figure 4. SRM MOTOR: MECHANICAL CHARACTERISTICS



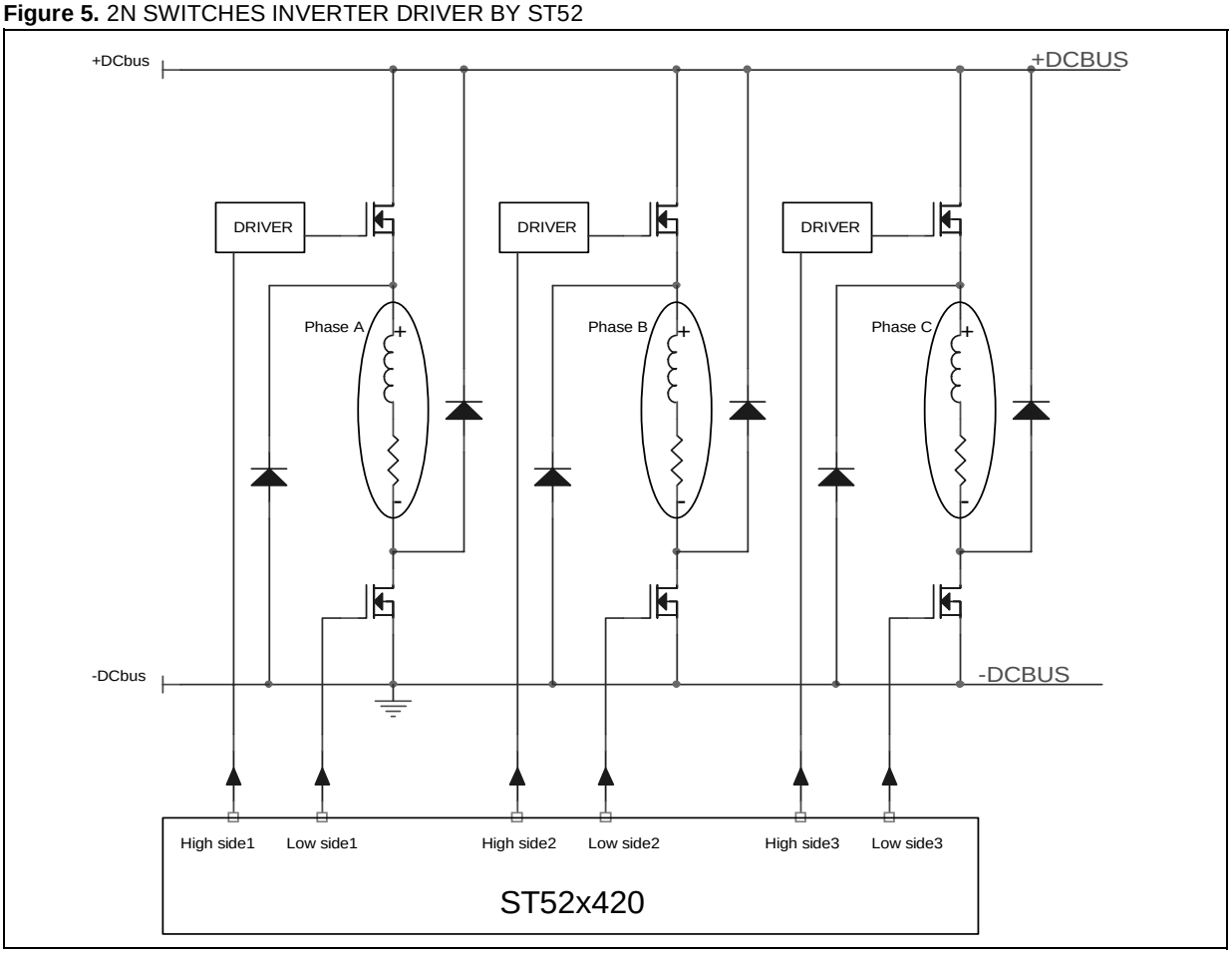
INVERTER TOPOLOGY

The SRM motor is driven by a six switch board in a typical 2N-switch topology, with N indicating the number of motor phases.

Each motor phase is connected to an asymmetric half bridge consisting of two power switches and two diodes. The complete DC voltage can be used to energize and de-energize a phase in chopping mode. When some of the switches are closed, a phase will be energized by the positive DC voltage supply. When both switches are opened, the current commutates between the switches and the relative freewheeling diodes. The voltage across the phase is now the negative DC voltage.

This inverter topology is the most used: it presents the main advantage of exploiting the full DC bus and is therefore suited for driving the motor at high speeds. The drawback disadvantage of this inverter topology is the high number of power devices: each half bridge needs two power switches and two freewheeling diodes. Three L6386 high-side/low-side drivers, not shown in Fig.5, are used to control six IGBTs (12A 600V), with six high speed free-wheeling diodes (5 A).

Figure 5. 2N SWITCHES INVERTER DRIVER BY ST52

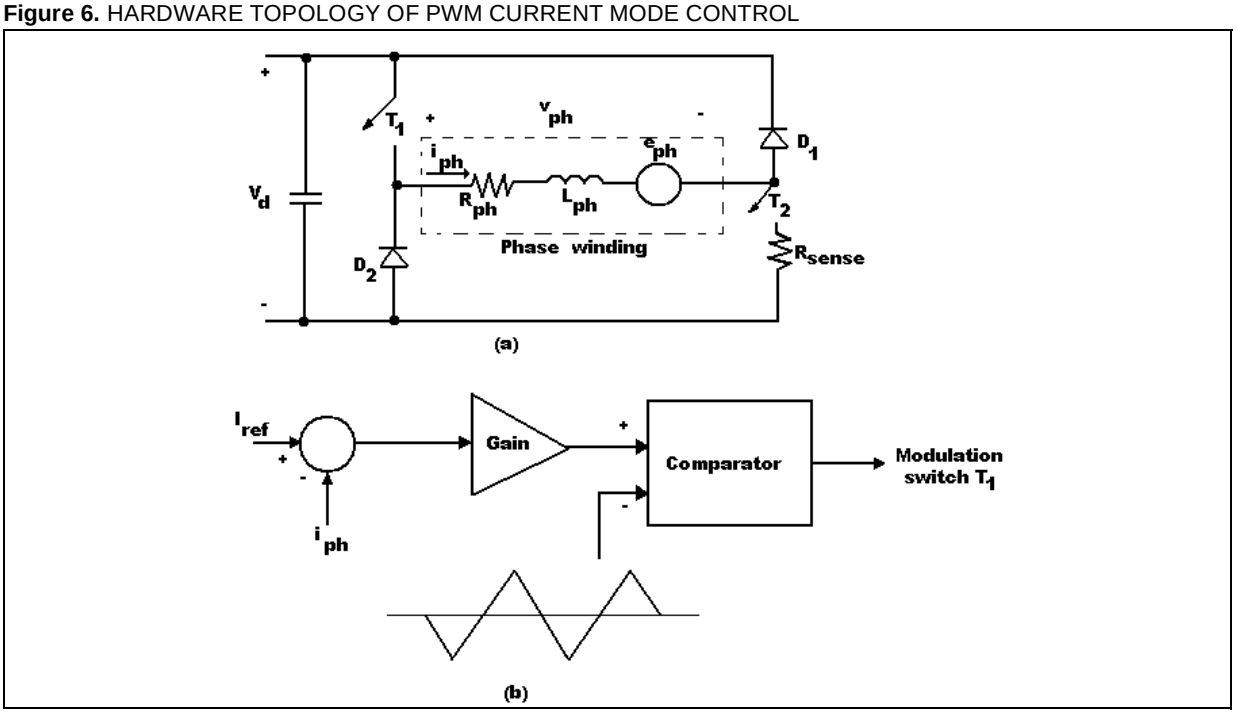


If high speed performances are not needed, a simpler and cheaper inverter topology can be used: the “N+1 switches” topology, where N indicates the number of phases.

This topology uses only one High Side leg to chop each of the three phases, a TD300 three phase driver and only one L6386, saving two IGBT’s, two Diodes and one L6386 .

The ST52x420 activates the driver by manipulating the encoder signal. A choppered PWM is used together with an hysteresis control. In Fig. 6 the encoder signal CH2 shows the chopping high side with a reference current at 2A and implements the Current Mode Control. The scheme is the following:

Figure 6. HARDWARE TOPOLOGY OF PWM CURRENT MODE CONTROL



In this control technique, the magnitude of the current flowing into windings is controlled by a control loop with a current feedback. The current in a motor phase winding is measured directly by a current/voltage converter or by a current sense resistor connected in series with the phase. The current is compared with the reference current, obtaining an error signal. The current error is compensated by the control law, and an appropriate control action is taken.

Figure 7. V AND I PHASE WAVEFORM

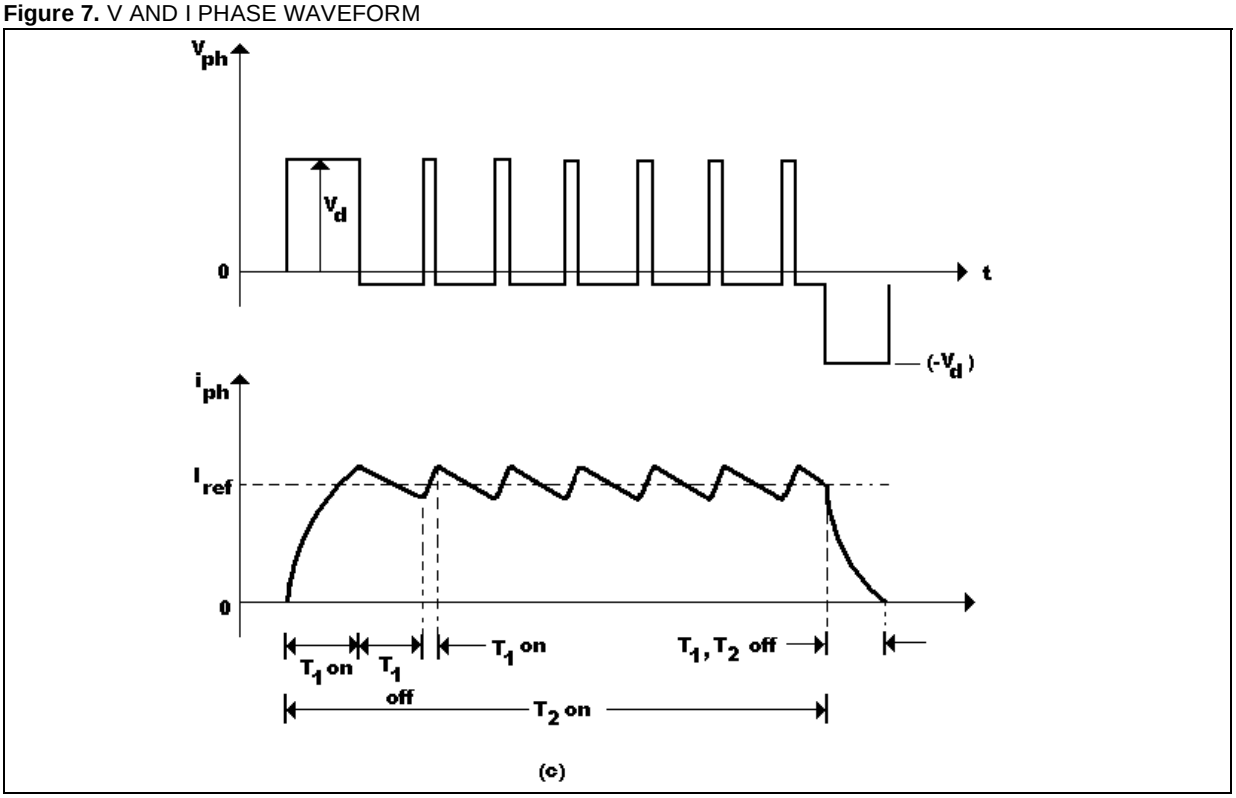
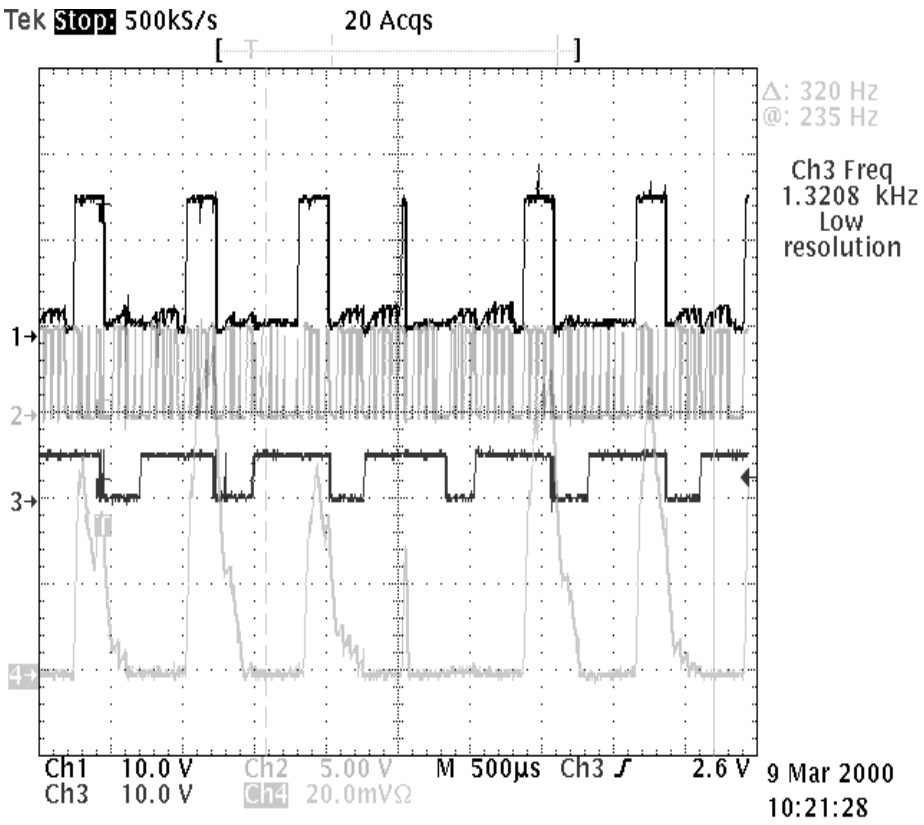


Figure 8. CURRENT MODE MODULATION AND INSERTION OF OVER-CURRENT PROTECTION



Some hardware protections are implemented on board to prevent drivers and switches from damage caused by over-current on Bus. The previous picture shows how to protect them. The current sensing resistor, which is printed on-board, enables to detect the Bus current by using one channel of the micro A/D, the same one used for the current mode modulation. In the picture, the gate signal “off” is a direct result of one over-current in a no-showed phase, but it shuts down all the drivers L6386.

DRIVING STRATEGIES

Many strategies are checked to drive the motor phases, by manipulating the encoder signal with some logic operators.

SRM control is often described in terms of “low-speed” and “high-speed” range. Low-speed operations are based on the control of the current.

As the motor speed increases, it is very difficult to control the current because of a combination of the back EMF effects and a reduced amount of time for the commutation interval.

The motor speed can often be increased by setting the turn-on and turn-off angles, so that the phase commutation begins early by producing current in the winding, while the inductance is at the decreasing phase. Besides, the extra time obtained allows the reduction of the current in the winding before the rotor reaches the negative torque region. The control of the firing angles can be carried out in different ways in relation to the type of position feedback available and the specifics of the application.

The analysis of the logic combination of the encoder signals is a correct way to drive the SR Motor. In this document, the logical AND between Encoder E1 and Encoder E2, or E2 & E3 or E1 & E3 is used to produce a set of three signals, which allows an appropriate firing sequence. At the same time, the logical function AND between E1 and the complementary E3, is another sequence representing an appropriate pre-fired control signal to drive the SR Motor at high speed. The choice of the correct logic combination function derives from the comparison between the Encoder Signal chart and the measured Inductance

(i.e. fig.2). This comparison is important in order to design some strategies, which implement the expected outputs (fig. 3), by switching from one to another.
This technique may be implemented either via firmware (if the computational time allows it), or via external logic devices in order to reach very high performances. The switch from one strategy to another, allows the jump from a starting phase to the speed-up phase and/or to the necessary pre-injection phases to reach the speed target.
These pre-injection phases are important to correct the “current tale action”.
By using a lead in the energizing phase, the current tale can be in the rising “L” zone (with dL/dt >0 and consequentially torque positive), in this way the brake effect of the SRM motor at high speed is avoided (fig.3).

SOFTWARE DESCRIPTION

The targets are: speed performances and a start-up sequence with a predetermined counter clockwise. In the starting phase a “soft start procedure” is necessary in order to guarantee a start in the right sense of rotation and to increase the Phase Current and the speed from zero to first step speed. After the start-up, the program switches to the “low-speed” procedures to reach the 20000 rpm with a Bus of 220 VAC.
The implementation used in this application, performs the driving of the SRM motor, by analyzing the encoders and by computing a logical function of these signals. This computation is carried out without an external logic, to achieve a low cost target.
A patented pending technique is used to perform a variable pre-phase injection, in order to obtain a driving signal of reduced timing. In this “Hybrid” control phase, a logical combination of the encoder is used to turn on the phase of the SR Motor, while a timer is used to turn it off. The “counter” of the Timer decreases by increasing the speed in order to obtain a firing signal, which is, in turn, synchronized with the correct sequence. This sequence is less wide than the allowed minimum time from the simple combination of encoder signals. The value of the “counter” could be chosen from an L.U.T., a standard linear function or Fuzzy Block which is aimed to extract the correct timing in relation to the speed. In this application, an L.U.T. is used to allow the Pre-phase timing.
A flow chart of the software is displayed in the following section, developed in Visual FIVE, the visual Development Tool of the ST FIVE Family micros.
The flow chart could be divided into the following steps:

1 PERIPHERAL CONFIGURATION AND MICRO INITIALIZATION

In this step, micro is initialized to perform the Hw implementation, in particular with:
·timer configuration (Timer 2 in PWM mode @20 KHz, Timer 1 in timer mode to perform the pre-phase switch off of the phase at high speed);
·the A/D configuration (current sensing);
·the initialization of the ports (analog input, digital input/output) in order to drive the L6386 and to implement the over-current protection.

2 START-UP PROCEDURE (SOFT START)

Start up procedure is essential to allow the SRM motor to start correctly, in a direction or in the opposite, and to speed up the motor from zero to 5000 rpm.
The correct sequence of phase ignition allows the counterclockwise direction. A variable Duty Cycle allows the start-up @full DCBus from 0 rpm.

3 SPEED-UP ALGORITHM. (Low-Speed).

After the start up phase, the encoder signal, obtained in digital mode, is filtered by an algorithm which performs a specified logic function, in order to express the drive sequence.

4 PRE-PHASE, HIGH SPEED ROUTINE (High-Speed)

There are lots of sequences of the manipulated encoder signals aimed to drive the motor, each of them for a specific speed range. There are other two different sequences to get a speed of 20Krpm. There are

no logic combinations of the encoder to perform both the right pre-phase firing sequence and the timing, in order to obtain a speed higher than the previous one.
The expected speed can be reached only by using a specific Timer peripheral to switch off the phase in advance. The Interrupt routine at high speed turns the phase off after that the logical function turns it on. This function is performed inside the Timer1 Interrupt, which turns off the phase by using the right element of a Table (The Visual FIVE L.U.T.) pointed by the variable “pointer” inside the PWM_0_COUNT. The Timer1 configuration will be changed after the soft-start phase, by using the same peripheral both for soft start and high-speed algorithm, saving resources for the other functions. The right value of the “pointer” is reached after measuring the encoder inside Timer_0 Interrupt routine.
In order to understand the algorithm, the flow chart is showed below, and the assembler file is found in the enclosure (appendix 2).

Figure 9. MAIN PROCEDURE

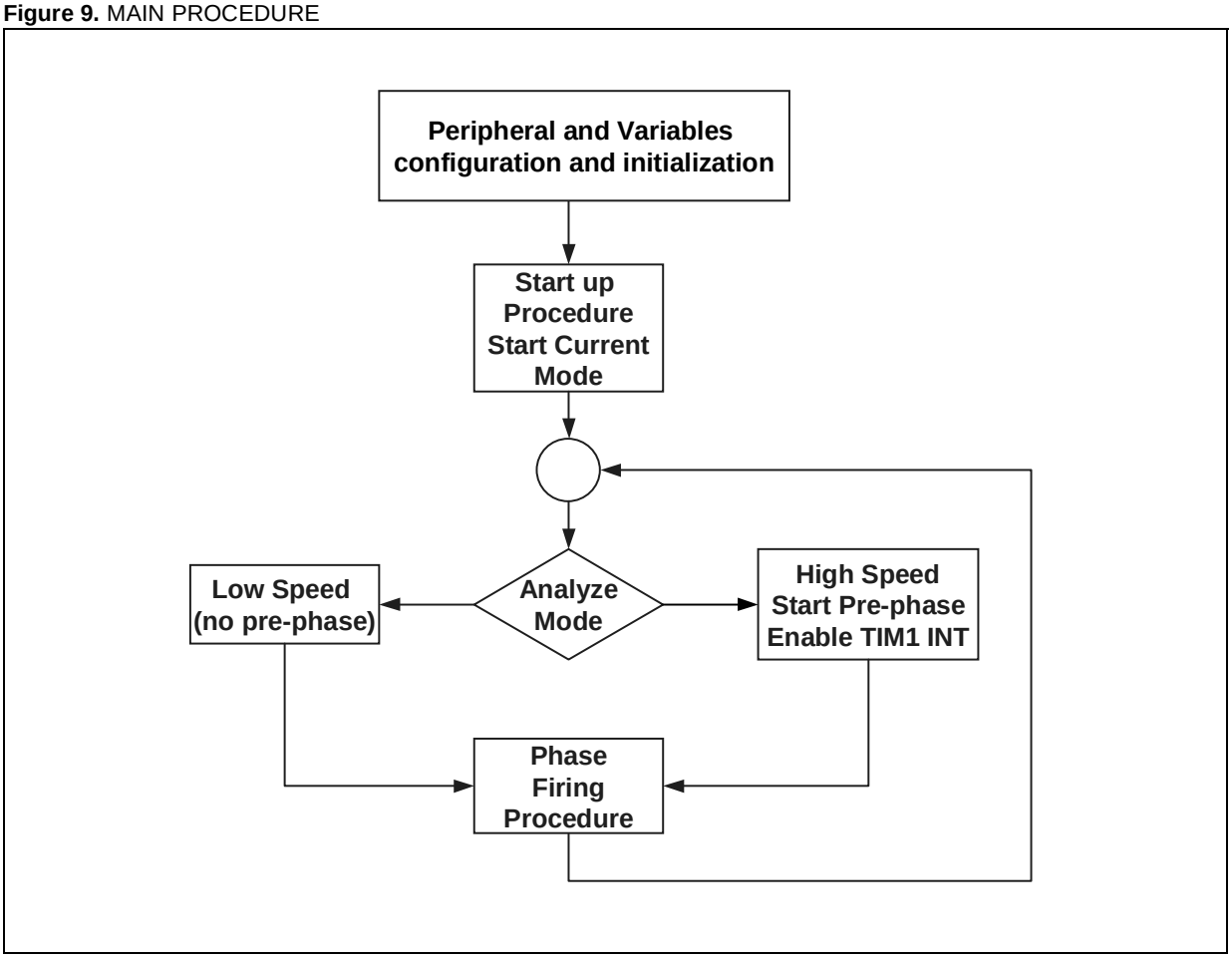


Figure 10. ANALYZE PROCEDURE

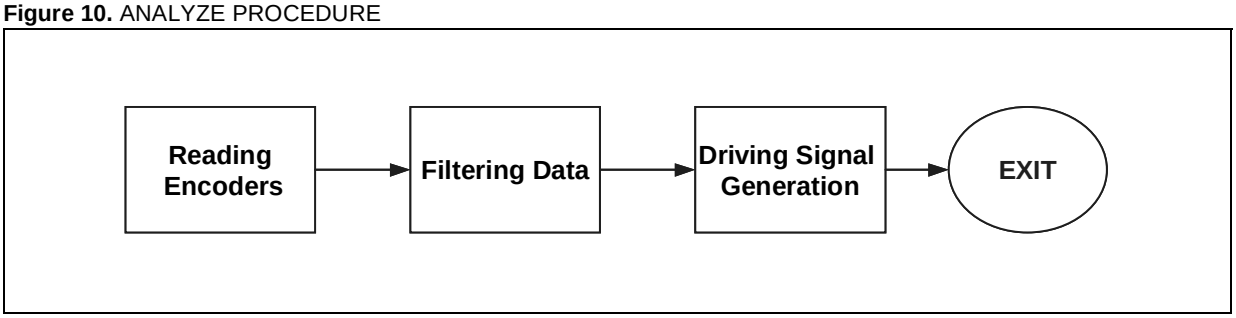


Figure 11. A/D AND TIMER 1 INTERRUPT ROUTINES

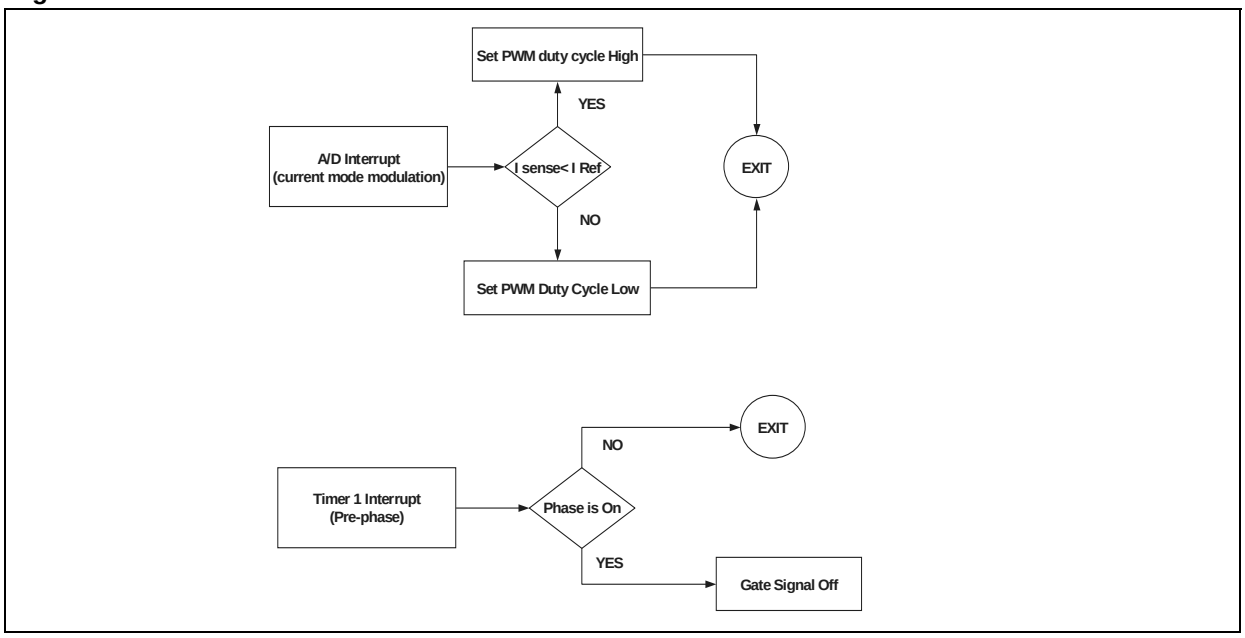


Figure 12. Visual FIVE: PHASE INJECTION ROUTINE

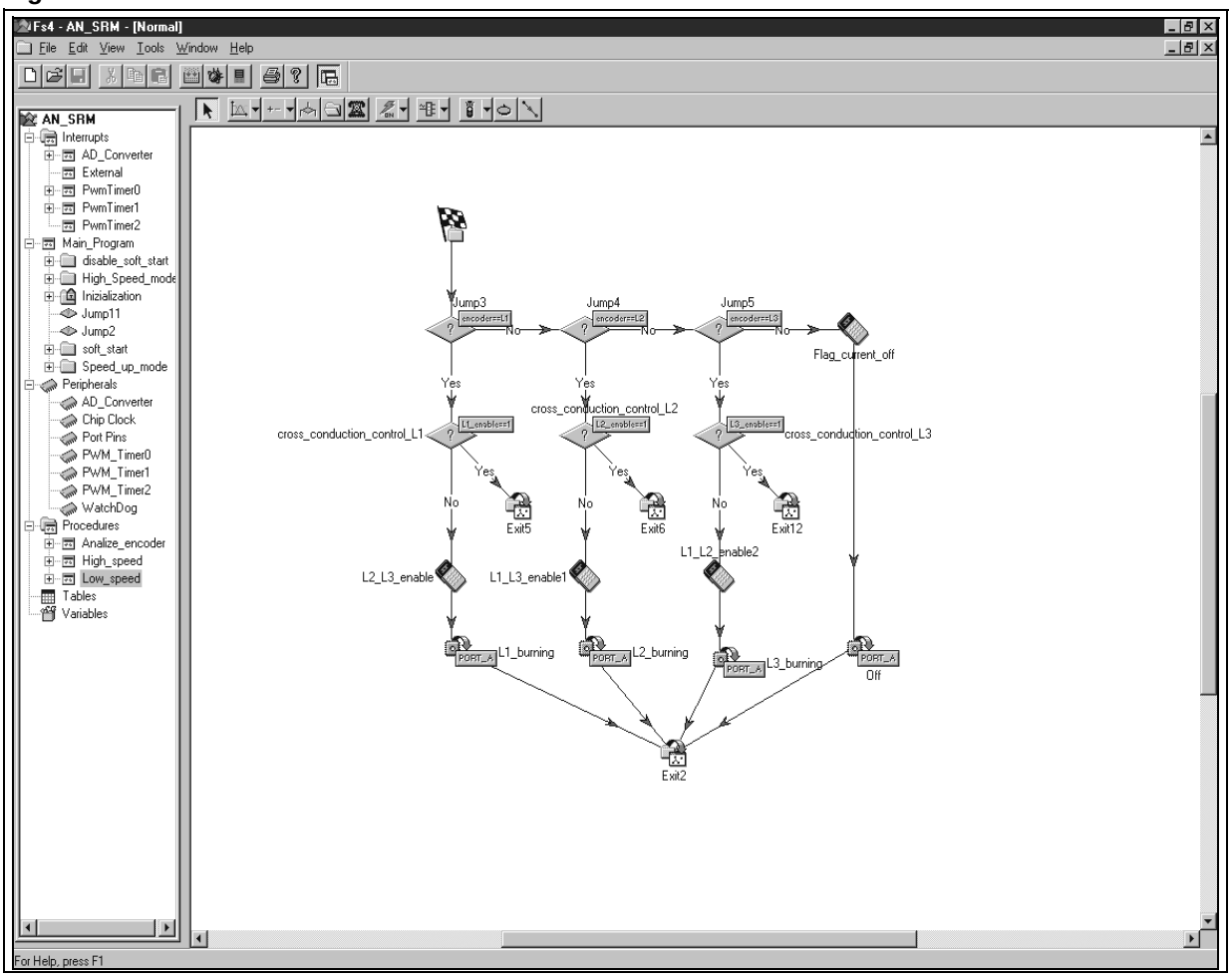
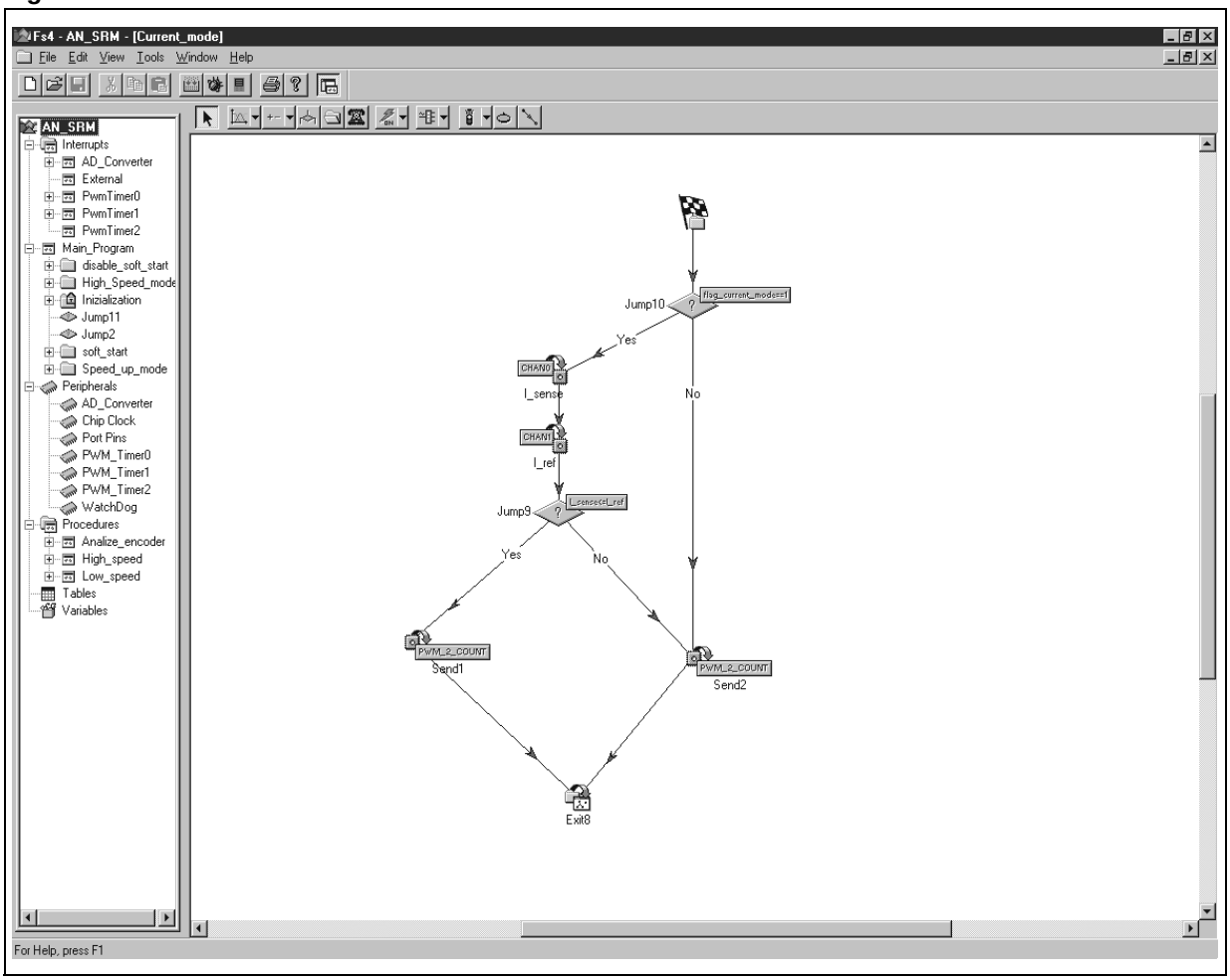


Figure 13. AD INTERRUPT ROUTINE



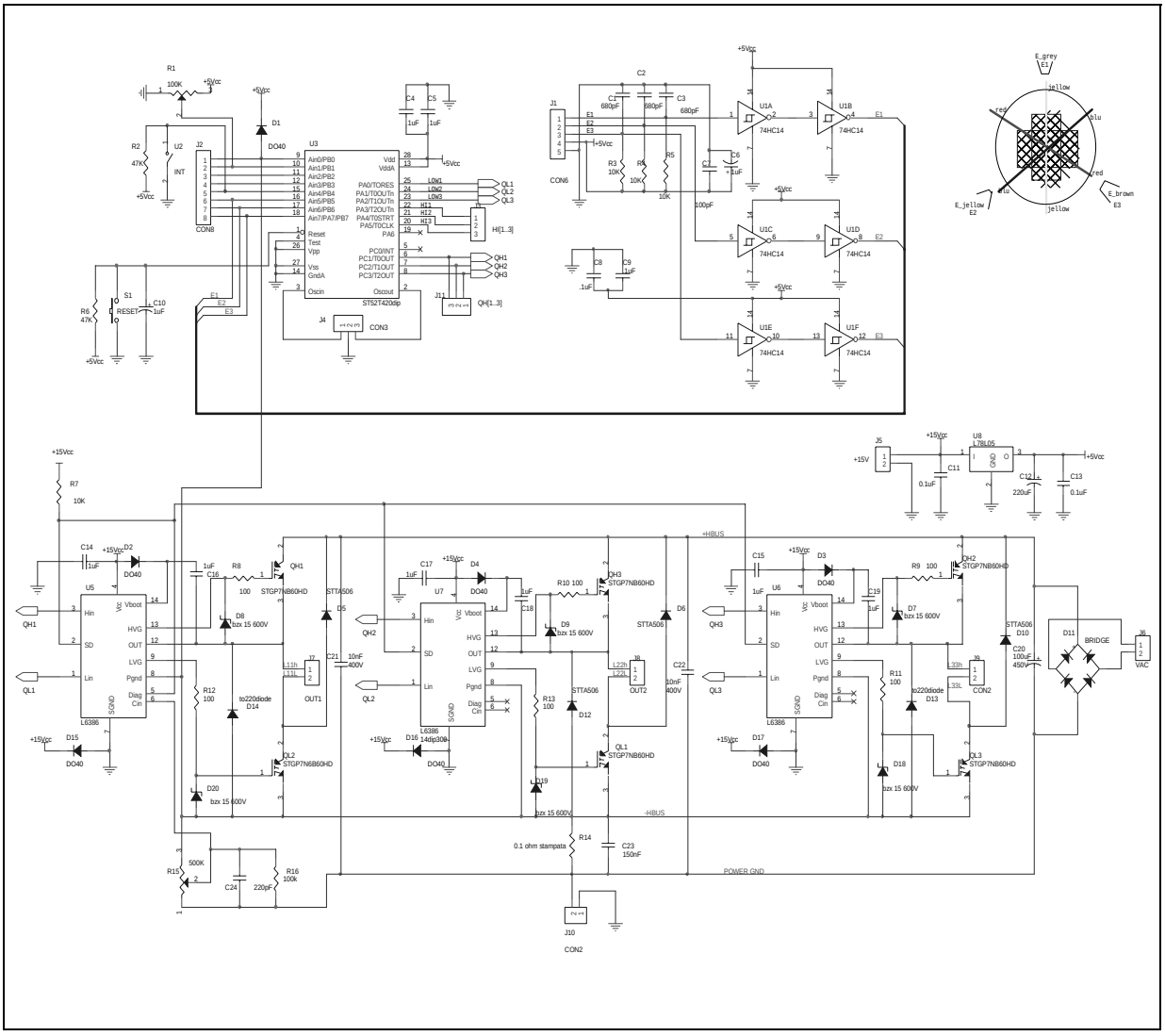
CONCLUSION

Thanks to the driving strategy adopted, the 25000rpm has been reached with a feeding of 230VAC and a BUS current of around 3A RMS by using a standard full-bridge driver and an 8 bit micro controller. The “smart” hardware performances of the ST Five micro controllers, and the efficient Visual FIVE tool have allowed the development of this application in a simple way. The intrinsic low cost features of the SR Motor, the optimum quality/price rate of the ST Five Micro controller Family and the efficient electronic topology emphasize this result.

REFERENCES

[1] Mohan, Undeland, Robbins "POWER ELECTRONICS: Converters, Applications, and Design" - John Wiley & sons
[2] ST524E420 Data Sheet, Visual FIVE - User Manual
[3] T.J.E. Miller “Switched Reluctance motors and their control" - OXFORD SCIENCE Publication 1993
[4] Vukosavic, Stefanovic "SRM Inverter Topologies: A Comparative Evaluation" - IEEE Transaction on Industry Applications, Vol.27 1991
[5] Ichinohura, Onda, Kimura, Watanabe, Yanada Guo "Analysis of Dynamic Characteristics of SRM based on SPICE" IEEE Transaction on Magnets, Vol.34 1998
[6] Becerra, Ehsani, Miller "Commutation of SR Motors" IEEE Transactions on Power Electronics, Vol.8 1993

APPENDIX A: DRIVER BOARD SCHEMATIC



APPENDIX B: ASM FIRMWARE

```
;*****************************************************************************************************************
;
; Author: N. Abbate 15/06/2001

;This Project shows how to drive a Three Phase Switched Reluctance Motor.
;It uses a PWM signal on pin PC2/T1OUT with a frequency of 20kHz and a variable duty cycle.
;On pin PB0/Ain0, the internal peripheral A/D Converter reads the Bus current.
;The Timer0 peripheral measures the encoder signal in order to compute the speed of the motor. The frequency of the timer is 610 Hz.
;Timer1 has two functions: the former supplies a predetermined timing for the Soft Start Phase. The latter turns off the control signal at high
speed. (pre-phase)
;The Analyze_encoder procedure filters the encoder signals to be put into Main Program.
;The Low_speed procedure sends the control signals to the driver in order to energize the phases at low speed.
;The High_speed procedure sends the control signal at high speed.
;The AD Converter Interrupt Routine performs the Current Mode Modulation: an Hysteresis control of the Bus current.
;The Timer 0 Interrupt Routine measures the encoder in order to obtain the speed and choices the right “pointer” for the pre-phase L.U.T.
;(Table).
;The Timer 1 Interrupt Routine is dedicated to the Soft Start and to "turn off" the driving signals during the pre-phase action.
;*****************************************************************************************************************
'

;  Prb 4
;  Prb 10

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

;  Tables Allocation

;    "BYTE Pre_phase[10]" use 10 eprom locations from 18(Page:0 Offset:18) to 27(Page:0 Offset:27)
    data  0   18  25 ;  Pre_phase[0] = 25       ; 00011001b, 0x19
    data  0   19  50 ;  Pre_phase[1] = 50       ; 00110010b, 0x32
    data  0   20  80 ;  Pre_phase[2] = 80       ; 01010000b, 0x50
    data  0   21  120;  Pre_phase[3] = 120      ; 01111000b, 0x78
    data  0   22  135;  Pre_phase[4] = 135      ; 10000111b, 0x87
    data  0   23  152;  Pre_phase[5] = 152      ; 10011000b, 0x98
    data  0   24  160;  Pre_phase[6] = 160      ; 10100000b, 0xA0
    data  0   25  180;  Pre_phase[7] = 180      ; 10110100b, 0xB4
    data  0   26  200;  Pre_phase[8] = 200      ; 11001000b, 0xC8
    data  0   27  220;  Pre_phase[9] = 220      ; 11011100b, 0xDC

;    Tables Allocation Report:
;      byte used : 10
;           from : 18(Page:0 Offset:18)
;             to : 27(Page:0 Offset:27)

    setmem 0   28

;  Store Device Configuration Parameters into Eprom

;    Default Interrupt Priority (Hi-->Lo): A/D, Timer0, Timer1, Timer2
    data  0   28  228;  RegConf_01 = 11100100

;    Pin Configuration
    data  0   31  144;  RegConf_04 = 10010000
    data  0   39  248;  RegConf_12 = 11111000
    data  0   40  255;  RegConf_13 = 11111111
    data  0   41  3  ;  RegConf_14 = 00000011
    data  0   42  243;  RegConf_15 = 11110011
    data  0   43  1  ;  RegConf_16 = 00000001

;    A/D Converter Configuration
;    ConvertionMode : Continuous
;    ChannelMode    : Multiple
;
'


ClockMode      : Divided
;    Channel       : 1
    data  0   30  58 ;  RegConf_03 = 00111010

;    WatchDog Configuration
    data  0   29  0  ;  RegConf_02 = 00000000

;    Pwm-Timer 0  Configuration
    data  0   32  88 ;  RegConf_05 = 01011000
    data  0   33  47 ;  RegConf_06 = 00101111
    data  0   34  4  ;  RegConf_07 = 00000100

;    Pwm-Timer 1  Configuration
    data  0   35  192;  RegConf_08 = 11000000
```

```
   data  0   36  44 ;  RegConf_09 = 00101100

;    Pwm-Timer 2  Configuration
   data  0   37  208;  RegConf_10 = 11010000
   data  0   38  34 ;  RegConf_11 = 00100010

   setmem 0   44
;  End ********************************************

;  Set Device Configuration Parameters
   pgset  0
   ldce  1   28  ;  RegConf_01 = 11100100
   ldce  2   29  ;  RegConf_02 = 00000000
   ldce  3   30  ;  RegConf_03 = 00111010
   ldce  4   31  ;  RegConf_04 = 10010000
   ldce  5   32  ;  RegConf_05 = 01011000
   ldce  6   33  ;  RegConf_06 = 00101111
   ldce  7   34  ;  RegConf_07 = 00000100
   ldce  8   35  ;  RegConf_08 = 11000000
   ldce  9   36  ;  RegConf_09 = 00101100
   ldce  10  37  ;  RegConf_10 = 11010000
   ldce  11  38  ;  RegConf_11 = 00100010
   ldce  12  39  ;  RegConf_12 = 11111000
   ldce  13  40  ;  RegConf_13 = 11111111
   ldce  14  41  ;  RegConf_14 = 00000011
   ldce  15  42  ;  RegConf_15 = 11110011
   ldce  16  43  ;  RegConf_16 = 00000001
   ldrc  0   0   ;  00000000b, 0x00
   ldpr  4   0
   ldpr  6   0
   ldpr  8   0


;  ********* User Defined Variables *********
;               NAME -> REG
;  ----------------------------------------
;          Flag_mode ->  10
;             I_ref ->  11
;            I_sense ->  12
;          L1_enable ->  13
;          L2_enable ->  14
;          L3_enable ->  15
;  enable_current_mode ->  16
;               enc1 ->  17
;               enc2 ->  18
;               enc3 ->  19
;         enc_speed1 ->  20
;         enc_speed2 ->  21
;         enc_speed3 ->  22
;            encoder ->  23
;      exit_soft_start ->  24
;               flag ->  25
;   flag_burning_phase ->  26
;    flag_current_mode ->  27
;             pointer ->  28
;        speed_triple ->  29
;               temp ->  30
;              temp1 ->  31
;  ****************************************


;  ****** One Time Vars Initialization ******
;  ----------------------------------------
   ldrc  10  1   ;  00000001b, 0x01
   ldrc  11  0   ;  00000000b, 0x00
   ldrc  12  0   ;  00000000b, 0x00
   ldrc  13  0   ;  00000000b, 0x00
   ldrc  14  0   ;  00000000b, 0x00
   ldrc  15  0   ;  00000000b, 0x00
   ldrc  16  0   ;  00000000b, 0x00
   ldrc  23  0   ;  00000000b, 0x00
   ldrc  24  0   ;  00000000b, 0x00
   ldrc  25  0   ;  00000000b, 0x00
   ldrc  27  0   ;  00000000b, 0x00
   ldrc  29  0   ;  00000000b, 0x00
   ldrc  30  25  ;  00011001b, 0x19
;  ****************************************


main:
;  ********** Start procedure "main"
```

```
Start:

Inizialization:
;       ++++++++++++++++++++++++++++++++
   mdgi

Config_PB_8_bit:
;  Asm Block : Config_PB_8_bit
   ldrc  30  0   ;  00000000b, 0x00
   ldcr  12  30
   ldrc  30  255 ;  11111111b, 0xFF
   ldcr  13  30
   ldrc  30  3   ;  00000011b, 0x03
   ldcr  14  30
   ldrc  30  0   ;  00000000b, 0x00
   ldcr  15  30
   ldrc  30  0   ;  00000000b, 0x00
   ldcr  16  30

init:
   ldrc  30  0   ;  00000000b, 0x00
   ldrc  2   255 ;  11111111b, 0xFF
   ldpr  3   2
   ldrc  2   255 ;  11111111b, 0xFF
   ldpr  5   2
   ldrc  2   255 ;  11111111b, 0xFF
   ldpr  7   2

IrqMask_0:

;  IrqEnableMask
;     Enable: PwmTimer1
;    Disable: AD_Converter PwmTimer0 PwmTimer2 External
   ldrc  2   8
   ldcr  0   2   ;  RegConf_00 = 00001000

IrqPriority_0:

;  IrqPriority (Hi-->Lo): PwmTimer1 AD_Converter PwmTimer0 PwmTimer2
   ldrc  2   210
   ldcr  1   2   ;  RegConf_01 = 11010010

Port_A_Off:
   ldrc  2   0   ;  00000000b, 0x00
   ldpr  0   2

PWM2_Off:
   ldrc  2   0   ;  00000000b, 0x00
   ldpr  7   2

PWM_Timer1_0:
;  PWM_1 Setting
   ldrc  2   196
   ldcr  8   2   ;  RegConf_08 = 11000100

PWM_Timer2_0:
;  PWM_2 Setting
   ldrc  2   213
   ldcr  10  2   ;  RegConf_10 = 11010101

AD_Converter_0:
;  ADC Setting
   ldrc  2   63
   ldcr  3   2   ;  RegConf_03 = 00111111

PWM_Timer0_2:
;  PWM_0 Setting
   ldrc  2   93
   ldcr  5   2   ;  RegConf_05 = 01011101

IrqMask_3:

;  IrqEnableMask
;     Enable: PwmTimer1
;    Disable: AD_Converter PwmTimer0 PwmTimer2 External
   ldrc  2   8
   ldcr  0   2   ;  RegConf_00 = 00001000

IrqReset_0:

;  Reset Interrupts
   rint  4   ;  External
   rint  0   ;  AD_Converter
   rint  1   ;  PwmTimer0
```

```
   rint  2   ;  PwmTimer1
   rint  3   ;  PwmTimer2

PWM_Timer1_1:
;  PWM_1 Setting
   ldrc  2   197
   ldcr  8   2   ;  RegConf_08 = 11000101

Exit0:
   jp    Inizialization_Exit

Inizialization_Exit:
   megi
;       ------------------------------

Soft_start_encoder:

Receive0:
;       ++++++++++++++++++++++++++++++++
   mdgi
;  ------ldri 23 10
   ldpr  1   0
   ldri  23  10
   megi
;       ------------------------------


Assembler0:
;  Asm Block : Assembler0
   mirror 23

Send8:
   ldpr  0   23

Exit6:
   jp    Soft_start_encoder_Exit

Soft_start_encoder_Exit:

Jump11:
;       ++++++++++++++++++++++++++++++++
   mdgi
   ldrc  2   0   ;  00000000b, 0x00
   sub   2   24
   jpnz  End_If_25
   megi
;       ------------------------------
   jp    Soft_start_encoder

End_If_25:
   megi
;       ------------------------------

Change_configuration_timer1:
;  Asm Block : Change_configuration_timer1
   ldrc  31  208 ;  11010000b, 0xD0
   ldcr  8   31
   ldrc  31  35  ;  00100011b, 0x23
   ldcr  9   31

soft_start_normal:

Call0:
   call  Analize_encoder

Call3:
   call  Low_speed

Exit9:
   jp    soft_start_normal_Exit

soft_start_normal_Exit:

Jump2:
;       ++++++++++++++++++++++++++++++++
   mdgi
   ldrc  2   0   ;  00000000b, 0x00
   sub   2   16
   jpnz  End_If_26
   megi
;       ------------------------------
   jp    soft_start_normal

End_If_26:
```

```
   megi
;       --------------------------------

enab_current_mode_and_pre_phase:

Enable_current_mode_pre_phase:

;  IrqEnableMask
;     Enable: AD_Converter PwmTimer2
;    Disable: PwmTimer0 PwmTimer1 External
;       ++++++++++++++++++++++++++++++++
   mdgi
   ldrc  2   18
   ldcr  0   2   ;  RegConf_00 = 00010010
   megi
;       --------------------------------

Exit10:
   jp    enab_current_mode_and_pre_phase_

enab_current_mode_and_pre_phase_:

Jump0:
;       ++++++++++++++++++++++++++++++++
   mdgi
   ldrc  2   0   ;  00000000b, 0x00
   sub   2   10
   jpnz  End_If_27
   megi
;       --------------------------------
   jp    normal_mode

End_If_27:
   megi
;       --------------------------------

High_mode:

Call5:
   call  Analize_encoder

Call4:
   call  High_speed

Exit16:
   jp    High_mode_Exit

High_mode_Exit:
   jp    Jump0

normal_mode:

Call2:
   call  Analize_encoder

Call1:
   call  Low_speed

Exit4:
   jp    normal_mode_Exit

normal_mode_Exit:
   jp    Jump0




AD_Converter:
;  ********** Start procedure "AD_Converter"

Folder4:
;       ++++++++++++++++++++++++++++++++
   mdgi

Jump10:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   27
   jpnz  End_If_11
   jp    I_sense

End_If_11:
```

```
Brake:
   ldrc  2   0   ;  00000000b, 0x00
   ldpr  7   2

IrqReset_2:

;  Reset Interrupts
   rint  0   ;  AD_Converter

Exit8:
   jp    Folder4_Exit

I_sense:
   ldri  12  1


I_ref:
   ldri  11  2


Jump9:
   ldrr  2   11
   sub   2   12
   jps   End_If_12
   jp    boost

End_If_12:
   jp    Brake

boost:
   ldrc  2   192 ;  11000000b, 0xC0
   ldpr  7   2
   jp    IrqReset_2

Folder4_Exit:
   megi
;       ------------------------------

RetI3:
   reti

PwmTimer0:
;  ********** Start procedure "PwmTimer0"

speed_encoder:
;       ++++++++++++++++++++++++++++++++
   mdgi

Receive1:
   ldri  29  12


PWM_Timer0_0:
;  PWM_0 Setting
   ldrc  2   92
   ldcr  5   2   ;  RegConf_05 = 01011100

PWM_Timer0_1:
;  PWM_0 Setting
   ldrc  2   93
   ldcr  5   2   ;  RegConf_05 = 01011101

Exit1:
   jp    speed_encoder_Exit

speed_encoder_Exit:
   megi
;       ------------------------------

Choice:
;       ++++++++++++++++++++++++++++++++
   mdgi

Jump21:
   ldrc  2   80  ;  01010000b, 0x50
   sub   2   29
   jps   End_If_3
   jp    High_Speed

End_If_3:

Low_Speed:
   ldrc  10  0   ;  00000000b, 0x00
```

```
Exit13:
   jp    Choice_Exit

High_Speed:
   ldrc  10  1   ;  00000001b, 0x01

Jump12:
   ldrc  2   10  ;  00001010b, 0x0A
   sub   2   29
   jps   End_If_4
   jp    Alu0

End_If_4:

Jump15:
   ldrc  2   20  ;  00010100b, 0x14
   sub   2   29
   jps   End_If_5
   jp    Alu2

End_If_5:

Jump16:
   ldrc  2   30  ;  00011110b, 0x1E
   sub   2   29
   jps   End_If_6
   jp    Alu3

End_If_6:

Jump19:
   ldrc  2   40  ;  00101000b, 0x28
   sub   2   29
   jps   End_If_7
   jp    Alu4

End_If_7:

Jump18:
   ldrc  2   50  ;  00110010b, 0x32
   sub   2   29
   jps   End_If_8
   jp    Alu5

End_If_8:

Jump17:
   ldrc  2   60  ;  00111100b, 0x3C
   sub   2   29
   jps   End_If_9
   jp    Alu6

End_If_9:

Jump20:
   ldrc  2   70  ;  01000110b, 0x46
   sub   2   29
   jps   End_If_10
   jp    Alu7

End_If_10:

Alu1:
   ldrc  28  6   ;  00000110b, 0x06
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read:
   ldpe  5   (2)

Exit14:
   jp    Choice_Exit

Alu7:
   ldrc  28  6   ;  00000110b, 0x06
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_1:
   ldpe  5   (2)
```

```
   jp   Exit14

Alu6:
   ldrc  28  6   ;  00000110b, 0x06
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_2:
   ldpe  5   (2)
   jp   Exit14

Alu5:
   ldrc  28  5   ;  00000101b, 0x05
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_3:
   ldpe  5   (2)
   jp   Exit14

Alu4:
   ldrc  28  4   ;  00000100b, 0x04
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_4:
   ldpe  5   (2)
   jp   Exit14

Alu3:
   ldrc  28  3   ;  00000011b, 0x03
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_5:
   ldpe  5   (2)
   jp   Exit14

Alu2:
   ldrc  28  1   ;  00000001b, 0x01
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_6:
   ldpe  5   (2)
   jp   Exit14

Alu0:
   ldrc  28  0   ;  00000000b, 0x00
   ldrr  3   28
   ldrc  2   18  ;  00010010b, 0x12
   add   2   3
   pgset  0

Read_7:
   ldpe  5   (2)
   jp   Exit14

Choice_Exit:
   megi
;       ------------------------------

RetI1:
   reti

PwmTimer1:
;  ********** Start procedure "PwmTimer1"

Jump8:
;       ++++++++++++++++++++++++++++++
   mdgi
   ldrc  2   0   ;  00000000b, 0x00
   sub   2   24
   jpnz  End_If
```

```
   megi
;       ------------------------------
   jp    timer_soft_start

End_If:
   megi
;       ------------------------------

Off_phase:
;       ++++++++++++++++++++++++++++++++
   mdgi

Send2:
   ldrc  2   0   ;  00000000b, 0x00
   ldpr  0   2

PWM_Timer2_2:
;  PWM_2 Setting
   ldrc  2   212
   ldcr  10  2   ;  RegConf_10 = 11010100

Exit17:
   jp    Off_phase_Exit

Off_phase_Exit:
   megi
;       ------------------------------

RetI2:
   reti

timer_soft_start:
;       ++++++++++++++++++++++++++++++++
   mdgi

Assembler2:
;  Asm Block : Assembler2
   inc   30

Jump14:
   ldrc  2   5   ;  00000101b, 0x05
   ldrr  3   30
   sub   3   2
   jps   End_If_1
   jp    End_soft_start1

End_If_1:

Send13:
   ldpr  7   30

Exit11:
   jp    timer_soft_start_Exit

End_soft_start1:
   ldrc  24  1   ;  00000001b, 0x01

Send14:
   ldpr  7   30

Jump13:
   ldrc  2   50  ;  00110010b, 0x32
   ldrr  3   30
   sub   3   2
   jps   End_If_2
   jp    Off_inc_PWM1

End_If_2:

Exit12:
   jp    timer_soft_start_Exit

Off_inc_PWM1:

;  IrqEnableMask
;     Enable:
;    Disable: AD_Converter PwmTimer0 PwmTimer1 PwmTimer2 External
   ldrc  2   0
   ldcr  0   2   ;  RegConf_00 = 00000000

IrqReset_3:

;  Reset Interrupts
   rint  2   ;  PwmTimer1
```

```
enable_current_mode1:
   ldrc  16  1   ;  00000001b, 0x01
   jp    Exit11

timer_soft_start_Exit:
   megi
;       ------------------------------
   jp    RetI2

PwmTimer2:
;  ********** Start procedure "PwmTimer2"

RetI4:
   reti


External:
;  ********** Start procedure "External"

RetI0:
   reti

Low_speed:
;  ********** Start procedure "Low_speed"

Normal:
;       ++++++++++++++++++++++++++++++++
   mdgi

Jump3:
   ldrc  2   160 ;  10100000b, 0xA0
   sub   2   23
   jpnz  End_If_13
   jp    cross_conduction_control_L1

End_If_13:

Jump4:
   ldrc  2   192 ;  11000000b, 0xC0
   sub   2   23
   jpnz  End_If_14
   jp    cross_conduction_control_L2

End_If_14:

Jump5:
   ldrc  2   96  ;  01100000b, 0x60
   sub   2   23
   jpnz  End_If_15
   jp    cross_conduction_control_L3

End_If_15:

Flag_current_off:
   ldrc  27  0   ;  00000000b, 0x00

Off:
   ldrc  2   0   ;  00000000b, 0x00
   ldpr  0   2

Exit2:
   jp    Normal_Exit

cross_conduction_control_L3:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   15
   jpnz  End_If_16
   jp    Exit19

End_If_16:

L1_L2_enable2:
   ldrc  13  0   ;  00000000b, 0x00
   ldrc  14  0   ;  00000000b, 0x00
   ldrc  15  1   ;  00000001b, 0x01
   ldrc  27  1   ;  00000001b, 0x01

L3_burning:
   ldrc  2   4   ;  00000100b, 0x04
   ldpr  0   2
   jp    Exit2

Exit19:
```

```
   jp    Normal_Exit

cross_conduction_control_L2:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   14
   jpnz  End_If_17
   jp    Exit18

End_If_17:

L1_L3_enable1:
   ldrc  13  0   ;  00000000b, 0x00
   ldrc  14  1   ;  00000001b, 0x01
   ldrc  15  0   ;  00000000b, 0x00
   ldrc  27  1   ;  00000001b, 0x01

L2_burning:
   ldrc  2   2   ;  00000010b, 0x02
   ldpr  0   2
   jp    Exit2

Exit18:
   jp    Normal_Exit

cross_conduction_control_L1:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   13
   jpnz  End_If_18
   jp    Exit5

End_If_18:

L2_L3_enable:
   ldrc  13  1   ;  00000001b, 0x01
   ldrc  14  0   ;  00000000b, 0x00
   ldrc  15  0   ;  00000000b, 0x00
   ldrc  27  1   ;  00000001b, 0x01

L1_burning:
   ldrc  2   1   ;  00000001b, 0x01
   ldpr  0   2
   jp    Exit2

Exit5:
   jp    Normal_Exit

Normal_Exit:
   megi
;       ------------------------------

Return2:
   ret

High_speed:
;  ********** Start procedure "High_speed"

High:
;       ++++++++++++++++++++++++++++++++
   mdgi

Jump1:
   ldrc  2   128 ;  10000000b, 0x80
   sub   2   23
   jpnz  End_If_19
   jp    cross_conduction_control_L4

End_If_19:

Jump6:
   ldrc  2   64  ;  01000000b, 0x40
   sub   2   23
   jpnz  End_If_20
   jp    cross_conduction_control_L6

End_If_20:

Jump7:
   ldrc  2   32  ;  00100000b, 0x20
   sub   2   23
   jpnz  End_If_21
   jp    cross_conduction_control_L5

End_If_21:
```

```
Flag_current_off1:
   ldrc  27  0   ;  00000000b, 0x00

Off1:
   ldrc  2   0   ;  00000000b, 0x00
   ldpr  0   2

PWM_Timer1_2:
;  PWM_1 Setting
   ldrc  2   196
   ldcr  8   2   ;  RegConf_08 = 11000100

Exit3:
   jp    High_Exit

cross_conduction_control_L5:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   15
   jpnz  End_If_22
   jp    Exit21

End_If_22:

L1_L2_enable1:
   ldrc  13  0   ;  00000000b, 0x00
   ldrc  14  0   ;  00000000b, 0x00
   ldrc  15  1   ;  00000001b, 0x01
   ldrc  27  1   ;  00000001b, 0x01

L3_burning1:
   ldrc  2   4   ;  00000100b, 0x04
   ldpr  0   2
   jp    PWM_Timer1_2

Exit21:
   jp    High_Exit

cross_conduction_control_L6:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   14
   jpnz  End_If_23
   jp    Exit20

End_If_23:

L1_L3_enable2:
   ldrc  13  0   ;  00000000b, 0x00
   ldrc  14  1   ;  00000001b, 0x01
   ldrc  15  0   ;  00000000b, 0x00
   ldrc  27  1   ;  00000001b, 0x01

L2_burning1:
   ldrc  2   2   ;  00000010b, 0x02
   ldpr  0   2
   jp    PWM_Timer1_2

Exit20:
   jp    High_Exit

cross_conduction_control_L4:
   ldrc  2   1   ;  00000001b, 0x01
   sub   2   13
   jpnz  End_If_24
   jp    Exit15

End_If_24:

L2_L3_enable1:
   ldrc  13  1   ;  00000001b, 0x01
   ldrc  14  0   ;  00000000b, 0x00
   ldrc  15  0   ;  00000000b, 0x00
   ldrc  27  1   ;  00000001b, 0x01

L1_burning1:
   ldrc  2   1   ;  00000001b, 0x01
   ldpr  0   2
   jp    PWM_Timer1_2

Exit15:
   jp    High_Exit

High_Exit:
   megi
;       -------------------------------
```

```
Return1:
   ret

Analize_encoder:
;  ********** Start procedure "Analize_encoder"

Folder2:

read_encoder:
;       +++++++++++++++++++++++++++++++++
   mdgi
;  ------ldri 23 10
   ldpr  1   0
   ldri  23  10
   megi
;       --------------------------------
Filtering:
;       +++++++++++++++++++++++++++++++++
   mdgi
   ldrc  2   224 ;  11100000b, 0xE0
   and   23  2
   megi
;       --------------------------------

Exit7:
   jp    Folder2_Exit

Folder2_Exit:

Return0:
   ret
```