Three phase current measurements using a single line resistor on the TMS320F240

Literature Number: BPRA077
Texas Instruments Europe
May 1998

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

List of Figures

Three phase current measurements using a single line resistor on the TMS320F240

ABSTRACT

Most inverter control systems require a knowledge of the phase currents. The simplest method of obtaining these currents is to measure them directly. Depending on the motor winding connections, this requires at least two sensors to be applied directly to the motor phases. Usually, these types of sensors are expensive due to their need to be isolated. There is a second method of measuring these phase currents using a simple, cheap resistor. However, under certain conditions, the measurement becomes difficult and even impossible due to hardware limitations. In this paper, a solution is described for circumventing this problem and results are given following its implementation on a Digital Signal Processor, the TMS320F240 from Texas Instruments.

1. Introduction

Most three phase motor control algorithms require that the motor phase currents be known in order to deliver high motor performance. The following system recombines the three phase currents of the motor using only one simple resistive sensor.

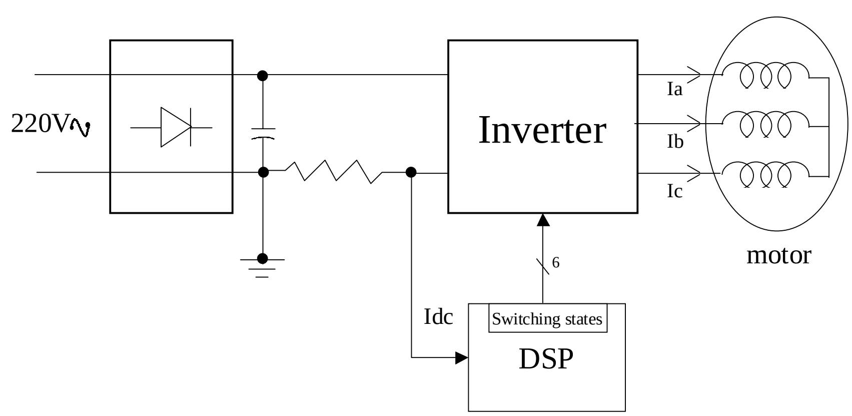

Figure 1: Schematic of a system including power, control and load

2. Measurement Process

In order to control most inverter systems it is necessary to know all the phase currents. The most basic method of obtaining these currents is to measure each of them directly but, depending on the motor winding connections, this requires at least two sensors to be applied directly to the individual motor phases. These types of sensors are usually sophisticated and expensive, as they need to be isolated. A second, but more complex, method is to measure only the DC line current and then recombine the 3 phase currents using the inverter switching states. This second method requires only a simple cheap, resistive sensor.

As the inverter's switching state is controlled by the Digital Signal Processor, it is possible to know the exact electrical route taken by the input current through the inverter. We can thus directly relate the phase currents to the line current, as shown in the following schematic diagram.

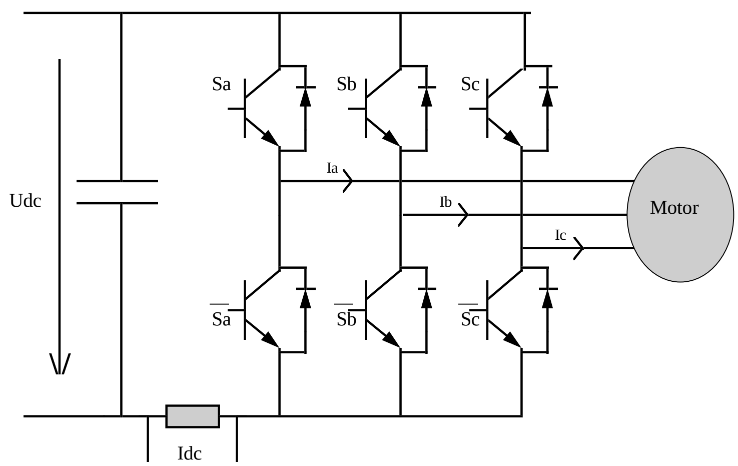

Figure 2: Schematic diagram of the inverter module

The phase currents we obtain are due to a real measurement of the current and are not the result of a simulation requiring a model of the output circuit. The estimator presented here is independent of the output load of the inverter. It may be used to control a motor but can also be part of a UPS system or any other system requiring the control of phase currents.

2.1 Measurement process

For a better understanding of the measurement process, and to represent the switching state of the inverter, we define a switching function Sa for phase A as follows: $Sa = 1$ when the upper transistor of phase A is on, and $Sa = 0$ when the lower transistor of phase A is on. Similar definitions can be made for phases B and C.

Note:

1. The explanation of the process is based on the assumption that the inverter is fed in complementary mode. In this mode, the signals $\overline{S}a, \overline{S}b, \overline{S}c$ controlling the lower transistors, are the opposite of Sa, Sb, Sc controlling the upper transistor. A similar current measurement could be applied to a non- complementary control.
2. Dead-band is the name given to the time difference between the commutations of the upper and lower transistors of one phase. These two transistors must never conduct at

the same time. The aim of the dead-band is to protect the power devices during commutation by avoiding conduction overlap which would result in a high current transient. In the notation, the dead-band is not present, the power devices are considered perfect. During implementation phase this time must be taken into account.

The stator currents can then be expressed as follows depending on the switching states:

$i_{dc} = i_a$ when $(Sa, Sb, Sc) = (1,0,0)$
$i_{dc} = -i_a$ when $(Sa, Sb, Sc) = (0,1,1)$
$i_{dc} = i_b$ when $(Sa, Sb, Sc) = (0,1,0)$
$i_{dc} = -i_b$ when $(Sa, Sb, Sc) = (1,0,1)$
$i_{dc} = i_c$ when $(Sa, Sb, Sc) = (0,0,1)$
$i_{dc} = -i_c$ when $(Sa, Sb, Sc) = (1,1,0)$
$i_{dc} = 0$ when $(Sa, Sb, Sc) = (1,1,1)$
$i_{dc} = 0$ when $(Sa, Sb, Sc) = (0,0,0)$

The following figure gives an example of the switching state:

$(Sa, Sb, Sc) = (0,0,1)$ where $i_{dc} = + i_c$.

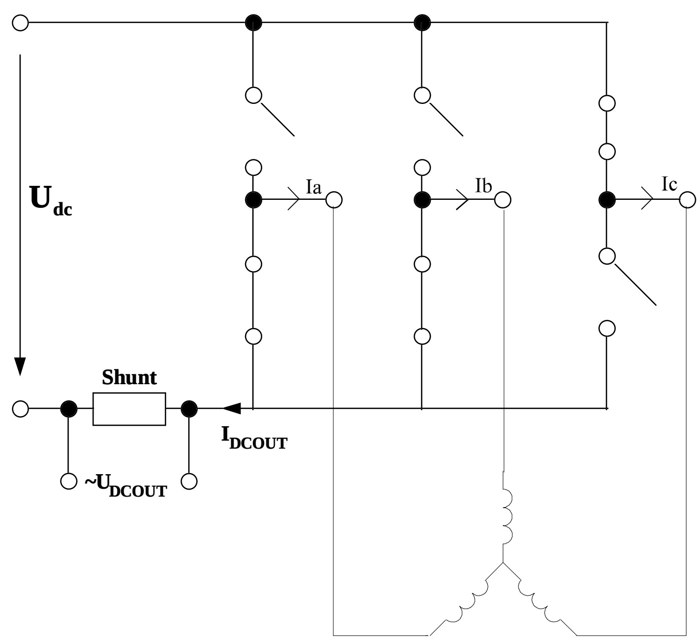

Figure 3: Inverter supplying a net of three star windings

Based on the above equations, one phase current (Ic) can be related directly to the dc line current. Therefore, all three-phase currents can be measured by looking only at the dc line. If the Pulse Width Modulation period frequency is high enough, the phase current will only vary slightly over one or two PWM periods.

2.2 Hardware considerations

Two times are defined as follows. u_1 is the time gap between the transistor commutation on the first phase and the commutation of the equivalent transistor in the next phase within the first half of the PWM period. Similarly u_2 is the time difference between the commutations on the second and third phases.

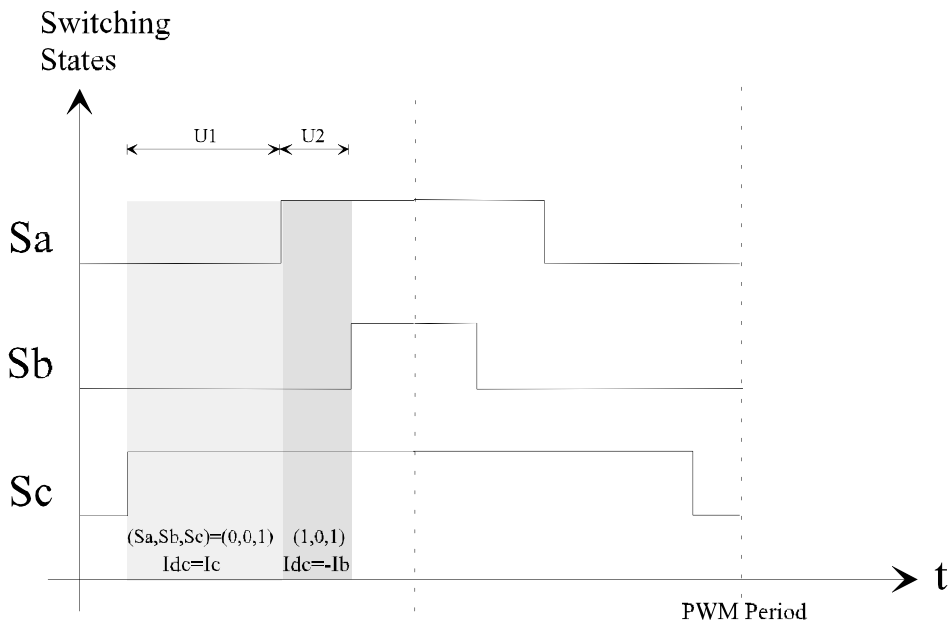

Figure 4: Signals controlling the upper transistors

In the case of symmetrical PWM generation, the first half period of the PWM starts with the state (0,0,0), followed by two states (u_1 and u_2) where at least one of the upper transistors is on, and finishes with the state (1,1,1). The second half of the PWM period consists of the same state sequence in reverse order .

1. It is not possible to make any measurement during the states (0,0,0) and (1,1,1) as no current is flowing in the dc line. A maximum of two different phase measurements can be made during any one PWM period.
2. Thus line current measurements made during times u_1 and u_2 will generate two different phases. The third current is deduced using the formula: $i_a+i_b+i_c=0$, in the case of a star or a triangle winding structure.

In the previous example, during time u_1 the inverter state is (0,0,1) and so the measured phase current is $i_c=i_{dc}$. Similarly during time u_2, the inverter state is (1,0,1) and so $i_b=-i_{dc}$. Since i_b and i_c have been calculated it follows that $i_a=-(i_b+i_c)$.

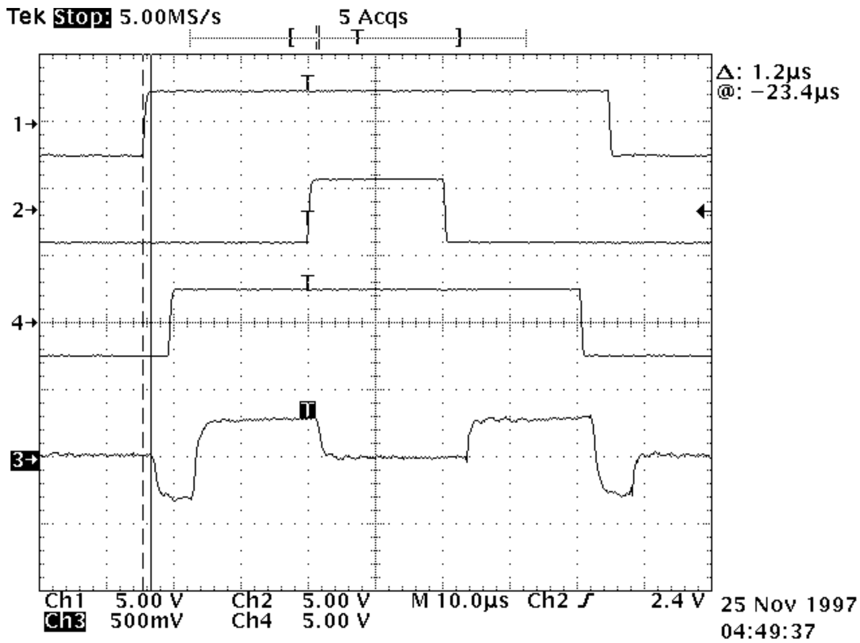

Figure 5: Plot1,2,3: PWM signals - Plot4: Idc current

2.3 Hardware limitations

Under certain conditions the periods u_1 or u_2 are very short. In this case, due to the transistor commutation times, dead-bands and response delays of the processing electronics, the actual phase current is invisible on the line current. As a result it is not possible to estimate the phase currents from the line current under these circumstances.

The method described in the next paragraph provides a solution which circumvents this hardware limitation and allows more accurate measurement of the phase currents than previous methods do. This improvement enables the motor to operate over a wider range of speeds and motor loads.

2.4 Solution to circumvent the hardware limitations

To help explain the solution of the problem, an artificial PWM pattern signal is generated. This signal is shown in the next graph and is the result of the addition of the three weighted upper transistor switching signals (PWM). The three PWM signals are indexed with 1, 2 and 4 respectively. At any one instance the switching state of the six transistors can be deducted from the PWM pattern.

The Figure 6 shows:

- Plot2-3-4: 3 symmetrical PWMs,
- Plot1: the 3 PWMs are added with addition of weight

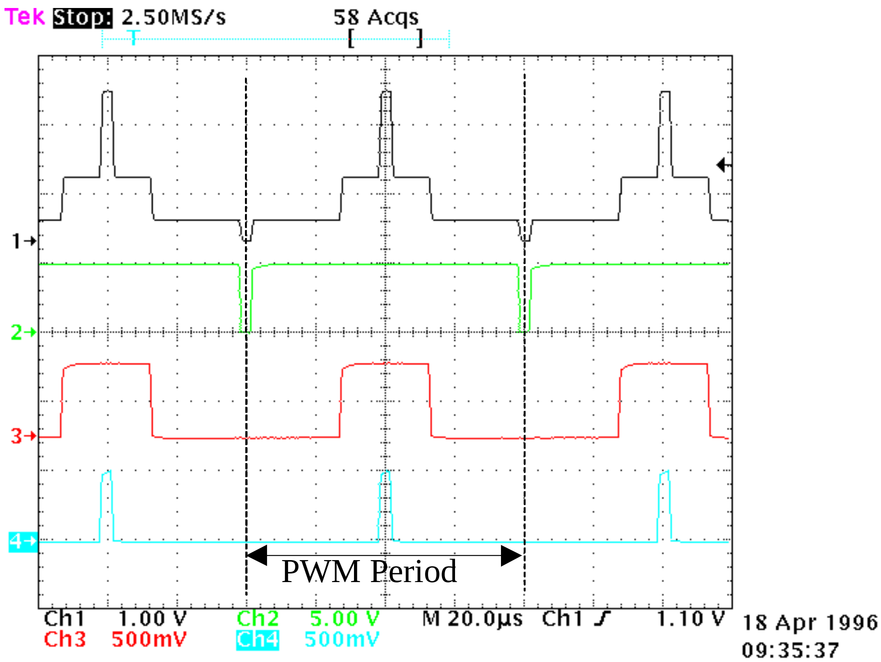

Figure 6: 3 Symmetrical PWMs with & without addition of weight

The problem is that it is not possible to measure the phase currents during a short u_1 and u_2 (in the range of few hundreds of nanoseconds to a few micro-seconds).

The Figure 7 shows:

- Plot 1: 3 PWM patterns, u_1 = 12µs, u_2 = 1.5µs
- Plot 2: Line current, only one current is detectable

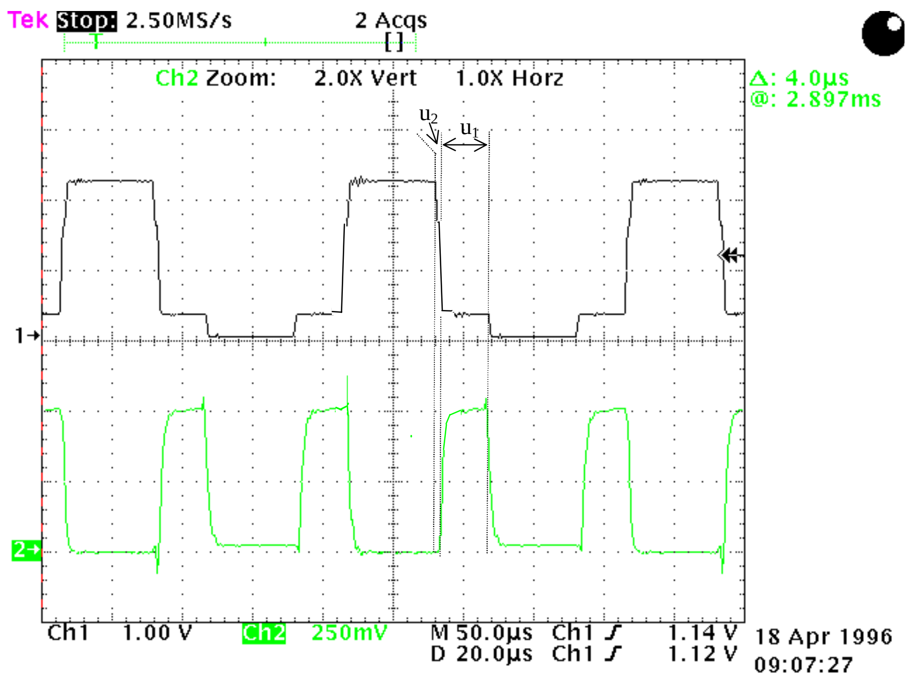

Figure 7: Line current, only one current is detectable

One of the methods to solve this problem and allow both measurements is to force the short period (here u_2) to the minimum measurement time[1] imposed by the chosen hardware. In this case u_2 is changed to $u_{2measure} = 4\mu s$.

The solution is to lengthen the required section of the pattern for one PWM period in order to make the measurement possible and compensate for it by generating shorter patterns during the PWM periods where no measurements are made.

Let us consider the example of a controller with a cycle time of 80µs. The PWM ha a carrier frequency of 12.5kHz (80µs). Therefore during one control cycle, one pattern is generated. In our example, the hardware imposes a minimum time of 4µs between two consecutive switching states in order to make an accurate measurement. The problem can be illustrated in the situation where, for a given speed and load, the control algorithm calculates, at a time t, time differences between PWM commutations of $u_1 = 12us$ and $u_2 = 1.5us$ respectively. The first time difference will allow a valid current measurement but the second one will not.

The Figure 8 shows:

- Plot 1: 3 PWM patterns, u_1 = 12µs, $u_{2measure}$ = 4µs
- Plot 2: Line current

[1] TI-patent pending

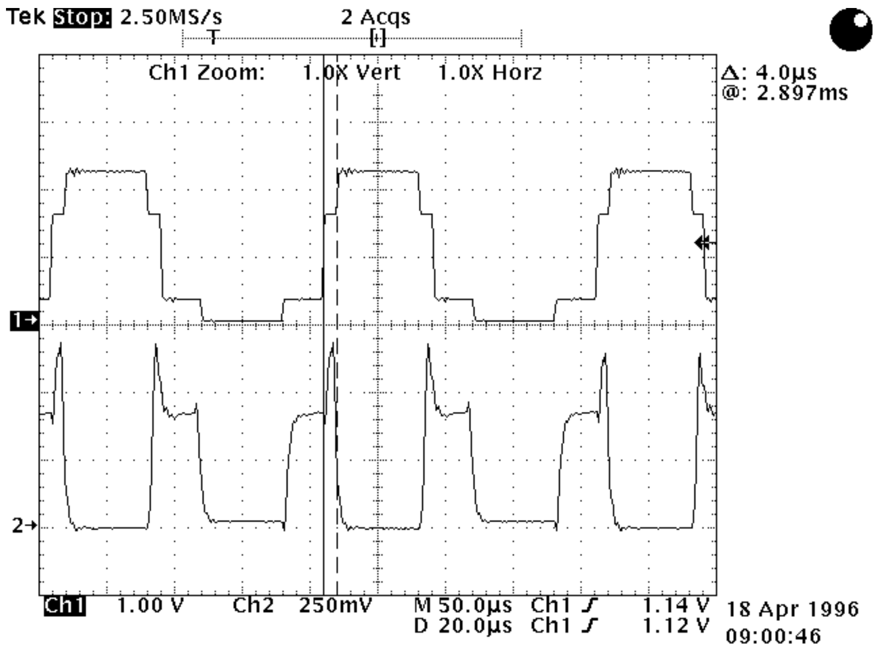

Figure 8: 2 currents detectable on the DC line

On the above plot the two phase currents are detectable and can be measured.

2.5 Enhanced algorithm

This simple modification algorithm described above may be used in most systems. However, this artificial modification of the PWMs will result in modified currents being applied to the motor, giving poor control of the stator flux. This poor control of the flux will result in more power being applied to the motor than is required thus reducing its efficiency as well as leading to more torque ripple.

For systems that often work at these limit conditions or where the best current shape for torque ripple control is required, an enhanced solution is proposed.

In most controllers the main control cycle frequency is lower than the PWM frequency. The control will then generate several identical PWM patterns for every control cycle phase current update. The enhanced algorithm overcomes the drawbacks of the simple modification method described above and can apply the theoretical phase signals calculated by the controller to the motor. It works by adapting the PWM patterns as required in order to meet any minimum periods required to make a measurement. During the measurement period, the PWM patterns signals will be adapted to correspond to the hardware's minimum time criteria. Similarly, during the remaining PWM periods when no measurements are made the PWM patterns will be compensated[2] throughout the controller cycle time to generate the correct mean phase currents in the motor.

[2] TI-patent pending

If we refer back to our example and extend the control cycle time from 80µs to 500µs (but not the PWM period), five PWM patterns of 80µs will be calculated by a single controller cycle. During the measurement period, u_2 is modified to $u_{2measure}$ = 4µs and u_1 remains equal to 12µs. The four other PWM patterns are then modified to compensate for the extra energy generated by this measurement pattern, by having a time delay $u_{2compensate}$ equivalent to:

$$4u_{2compensate} + u_{2measure} = 5u_2$$

$$u_{2compensate} = \frac{5 \times 1.5 - u_{2measure}}{4} = 875ns$$

u_1 remains the same.

The following graph illustrates this example, when (u_1, $u_{2measure}$) is generated, a peak appears on the line current, this is the desired current. During (u_1, $u_{2compensate}$) no measurement is possible as the time between two switching states is too low for the hardware used in the application.

The Figure 9 shows:

- Plot 1: PWM pattern, u_1 = 12µs, u_{2meas} = 4µs, u_{2comp} = 875ns
- Plot 2: Line current

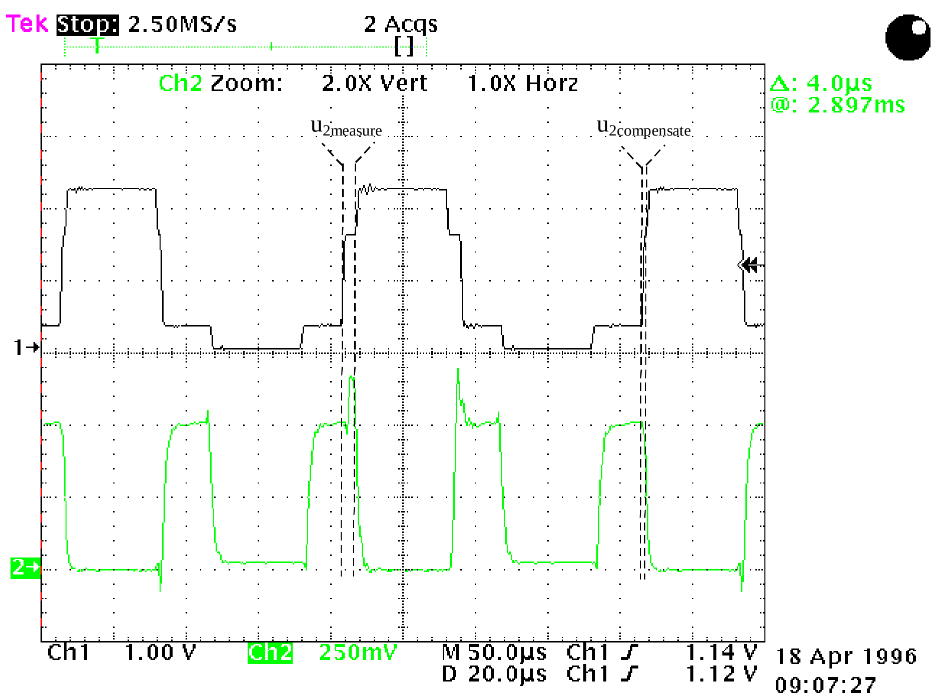

Figure 9: Phase current detectable when needed

In the above plot the two phase currents are detectable only when needed.

3. Hardware sensor

3.1 Schematic

The inverter considered in the application note has three legs. The load is modelled by an AC motor with the assumption that $Ia+Ib+Ic = 0$. The shunt is placed in the circuit so that the current going into and out of the inverter flows across it. An Operational Amplifier scales the shunt terminal voltage to fit the input voltage of the 'F240 Analogue Digital converter which has a maximum range of 0-5V.

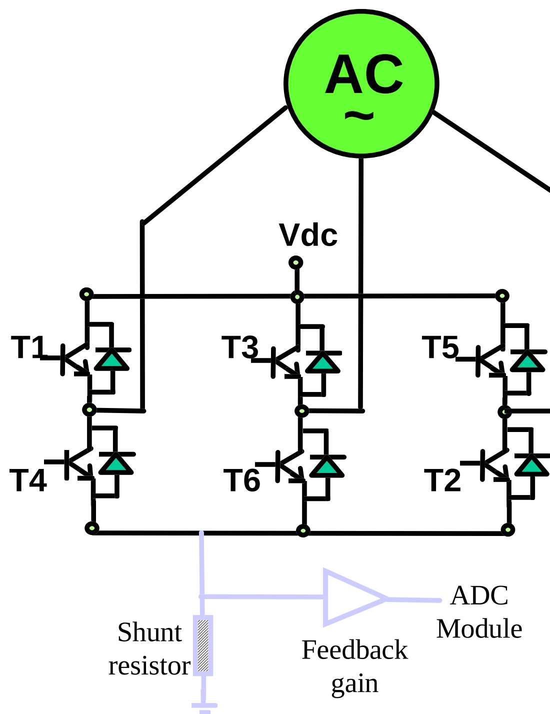

Figure 10: Three bridges inverter with Idc measurement

The additional circuitry required to sample the current from the DC line consists of:

- a shunt resistor whose value depends on two factors. A low dissipated power RI^2 and a voltage Vshunt high enough to get a reasonable ADC scaling gain. For instance, a phase current range of [-10,10] Amps with an AOP gain of 10 requires a shunt of 25 mΩ to get an AD input in the range of [0,5] Volts. This shunt will have to dissipate a maximum of 2.5 Watt.
- an operational amplifier with a bandwidth high enough to see the Idc current transitions. As an indication, the AOP bandwidth used in the example described previously to detect an Idc current lasting 4µs was 1MHz,
- An other requirement is to create an offset voltage of 2.5V for the A/D converter input in order to scale the AD input in the range 0-5V,

- If the AOP supply is 15V, a clamp diode has to be added to limit AD input voltage to 5V.

The following schematic gives an example of a basic circuit to scale the current from the shunt resistor, where:

$$ADinput = \frac{R_2}{R_1}(Vshunt) + 2.5$$

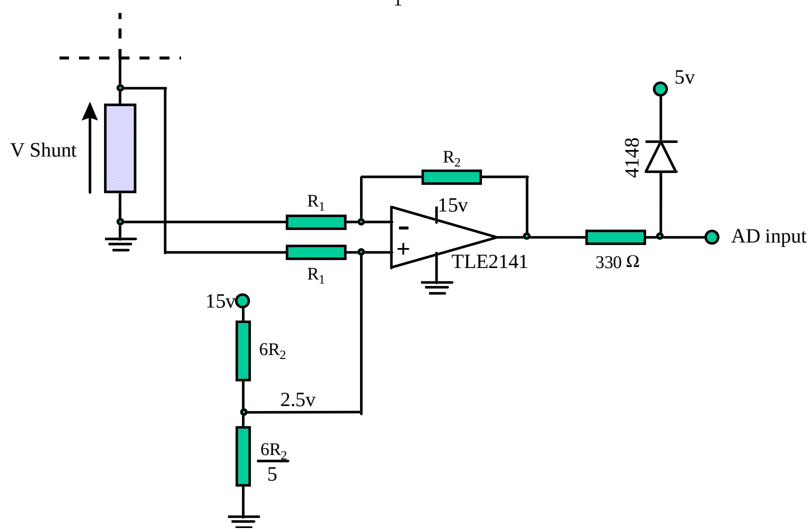

Figure 11: Basic schematic for Idc current measurement

The precision of the resistors will determine the accuracy of the conversion.

3.2 Cost comparison

Other solutions may be considered to measure the phase currents. The costs ratio of other solutions are compared below.

3.2.1 Three shunts on each inverter leg

This method consists of using three shunts to measure the currents flowing in each of the three inverter legs. The measurement is then possible at any time, except during free wheeling. This solution requires three circuits comparable to the solution described in the previous paragraph. The cost will be three times higher.

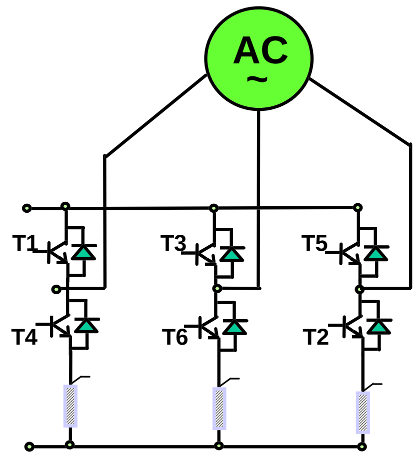

Figure 12: Current measurement using three shunts

A solution with two shunts could also be used, the third current being deduced from the two current samples.

3.2.2 Isolated phase sensors

Another solution commonly found in industry today is to sense the phase currents directly.

1. two shunts are used on the phases. This requires two isolated AOPs and twice the scaling circuit for AD input,
2. two isolated Hall Effect sensors are placed on the motor phase currents. Again, the cost of two scaling circuits must be added on top of the cost of the isolated sensors.

The ratio between the prices of the two implementations is greater than four. This ratio grows as the power increases.

4. Software Implementation

4.1 Implementation of the first solution

The software implemented on the TMS320F240 evaluation module may use any kind of control and load with the condition that $ia+ib+ic = 0$. For demonstration purposes, the effects of the algorithm are best seen when simply observing the phase currents on an open-loop system. Consequently if any disturbance occurs on the current due to the current measurement process it will not be corrected by the control. For this reason a voltage/frequency open-loop control is considered with an AC induction drive as a load. As a conclusion, an example of a closed-loop control using the sampled currents will be demonstrated.

The software ensures that all the time gaps *u1* and *u2* reach a minimum time called Mingap. Two current conversions will be done during each PWM period in order to calculate the three currents flowing in the inverter phases.

The solution uses minimal CPU time. Most of the functions required to get the phase currents from a single shunt resistor are performed asynchronously from the DSP core, as a result of an optimum use of the 'F240 Event Manager.

The implemented solution uses

- 1 timer and 6 PWM from the full compare for the control.
- 1 A/D channel.
- 1 PWM from a second timer for the synchronisation of the ADC conversions which are done automatically on the timer compare match.
- 1 interrupt on the full compare period underflow event to synchronise the control.
- All registers are reloaded synchronously with the control cycle.

To generate the PWM, the timer T1 is used and counts successively in Up and Down modes (symmetric mode). An interrupt, called PR_int, occurs at the end of every Down mode.

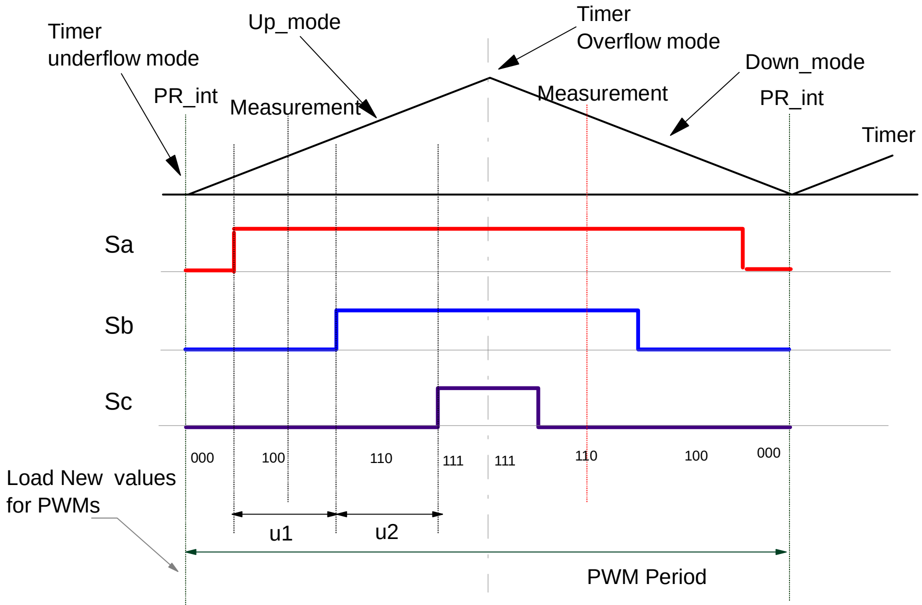

Figure 13: PWM generation related to the time in the case of a symmetric PWM

The register T1PER contains the period of the PWM.

The measurements are made using the Analogue to Digital converters. Only one channel is used in the software: Channel 2 (ADC1CHSEL = 001). It is connected to IDC channel.

The starts of conversions are synchronised with the PWMs, on the compare register match. It does not require any CPU load due to the TMS320C24x event manager.

For more flexibility the conversions are started on the PWM compare of Timer 3. Its period is the same as T1, but its time base is shifted by a time called ‘delay’ (T1CNT = T3CNT + delay). This adjustment is made to compensate for the response time between the PWM command and the moment Idc current actually switches. This software delay is implemented for ease of adaptation to any hardware and avoids the use of any glue logic. The ‘delay’ can be adjusted at start up through the user interface. The process for adapting it is to visualise T3PWM together with the DC current while modifying the variable. When they are synchronised the value of delay can be set as default value. This parameter is dependent on the hardware and once determined, it doesn’t need to be changed.

N.B. ‘Delay’ is taken into account only during the software initialisation at start-up. Consequently, each time the variable ‘delay’ is modified through the user interface, the DSP must be reset while running so that the change can be operative.

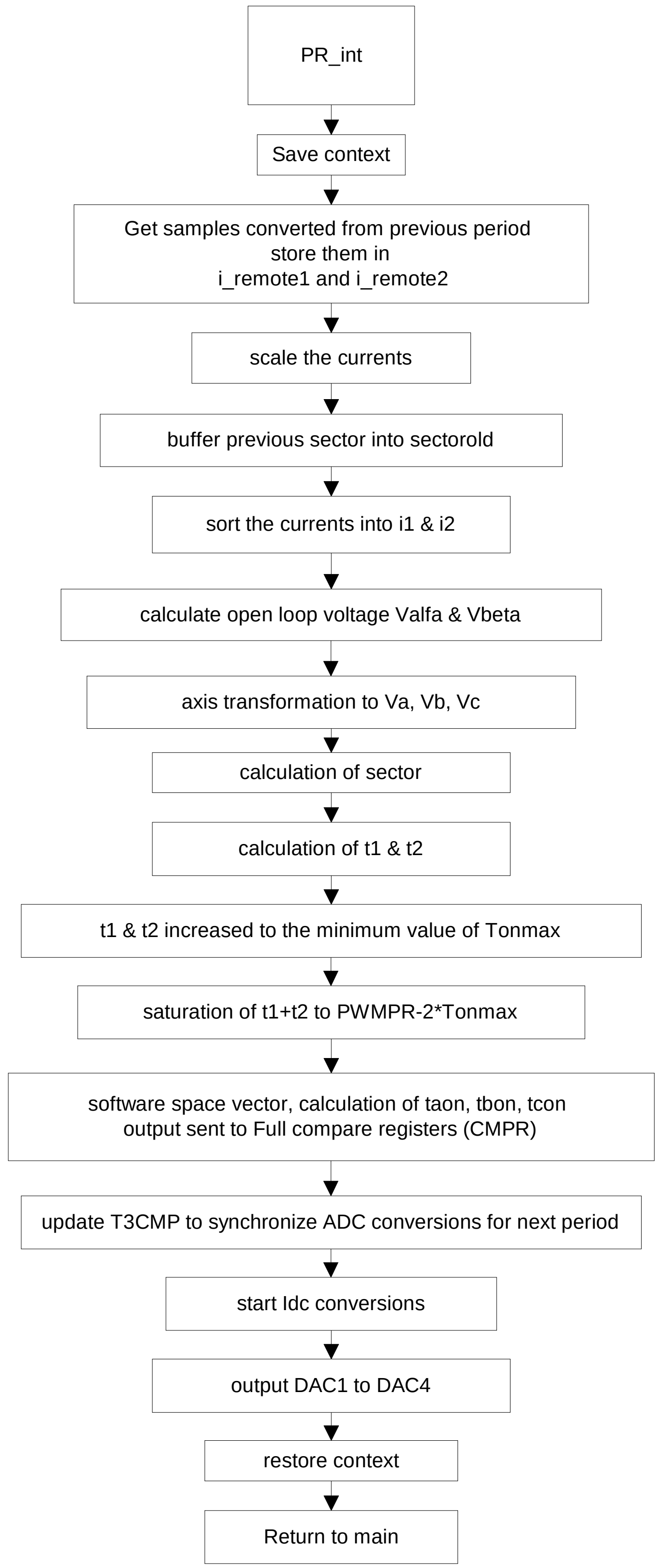
PR_int
Save context
Get samples converted from previous period
store them in
i_remote1 and i_remote2
scale the currents
buffer previous sector into sectorold
sort the currents into i1 & i2
calculate open loop voltage Valfa & Vbeta
axis transformation to Va, Vb, Vc
calculation of sector
calculation of t1 & t2
t1 & t2 increased to the minimum value of Tonmax
saturation of t1+t2 to PWMPR-2*Tonmax
software space vector, calculation of taon, tbon, tcon
output sent to Full compare registers (CMPR)
update T3CMP to synchronize ADC conversions for next period
start Idc conversions
output DAC1 to DAC4
restore context
Return to main

Figure 14: V/f control and Idc current measurement block diagram

4.2 Implementation of the advanced method

The control in this example is performed once every five PWM periods. Then only two Idc measurements are required to get the phase currents every five PWM periods. To enable the synchronisation of the control, PR_int sets a flag every five PWM periods to implement the controller. At the end of the control cycle time, PR_int calculates the measurement pattern defined by $u_{1measure}$ and $u_{2measure}$ (u1meas and u2meas in the flowcharts). The current measurement is taken during the period preceding the control.

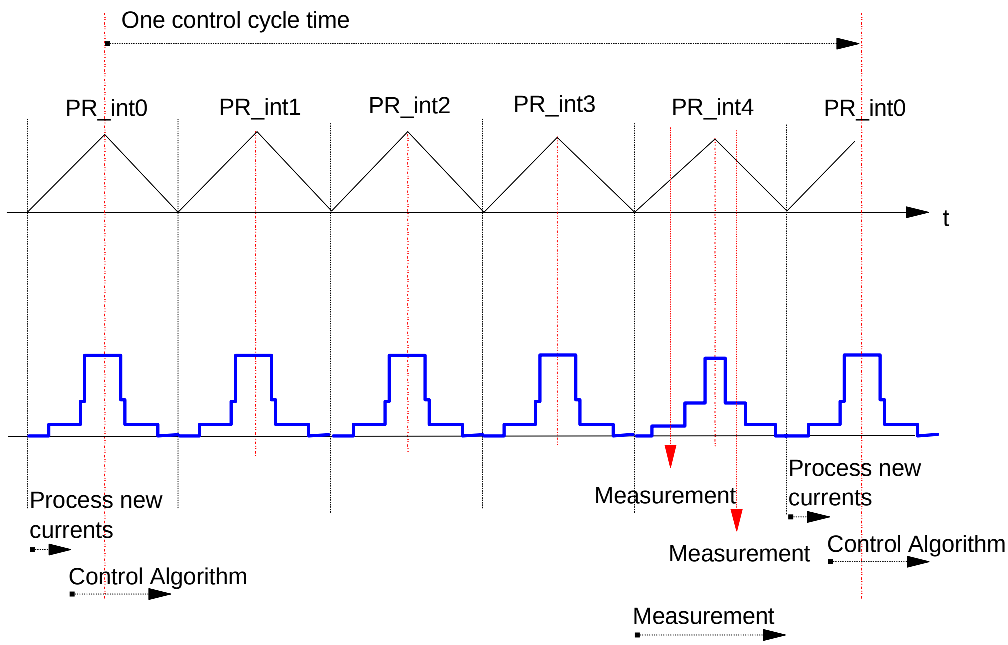

Figure 15: Synchronisation diagram of the control

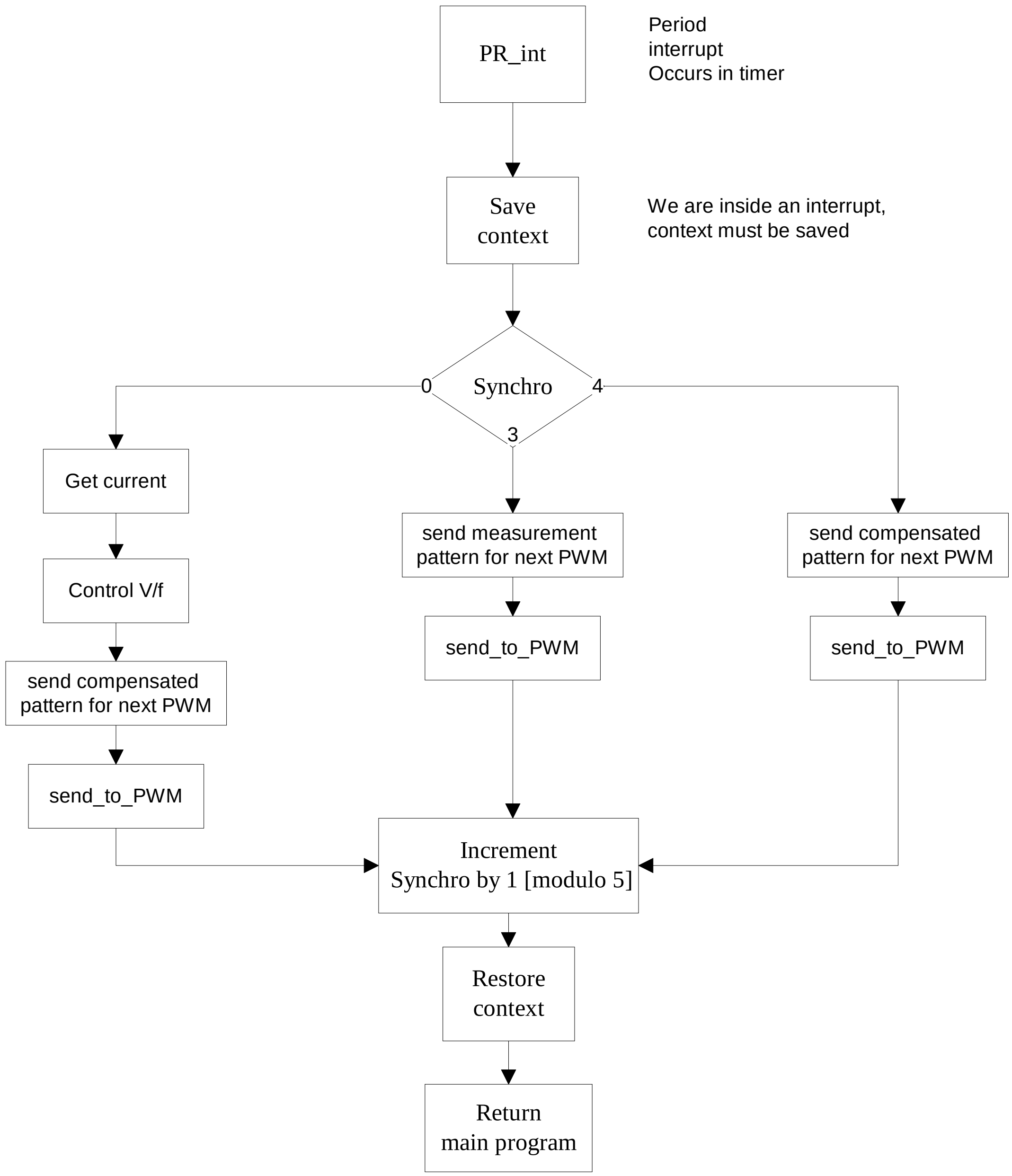

Figure 16: Flowchart of the Period interrupt: PR_int

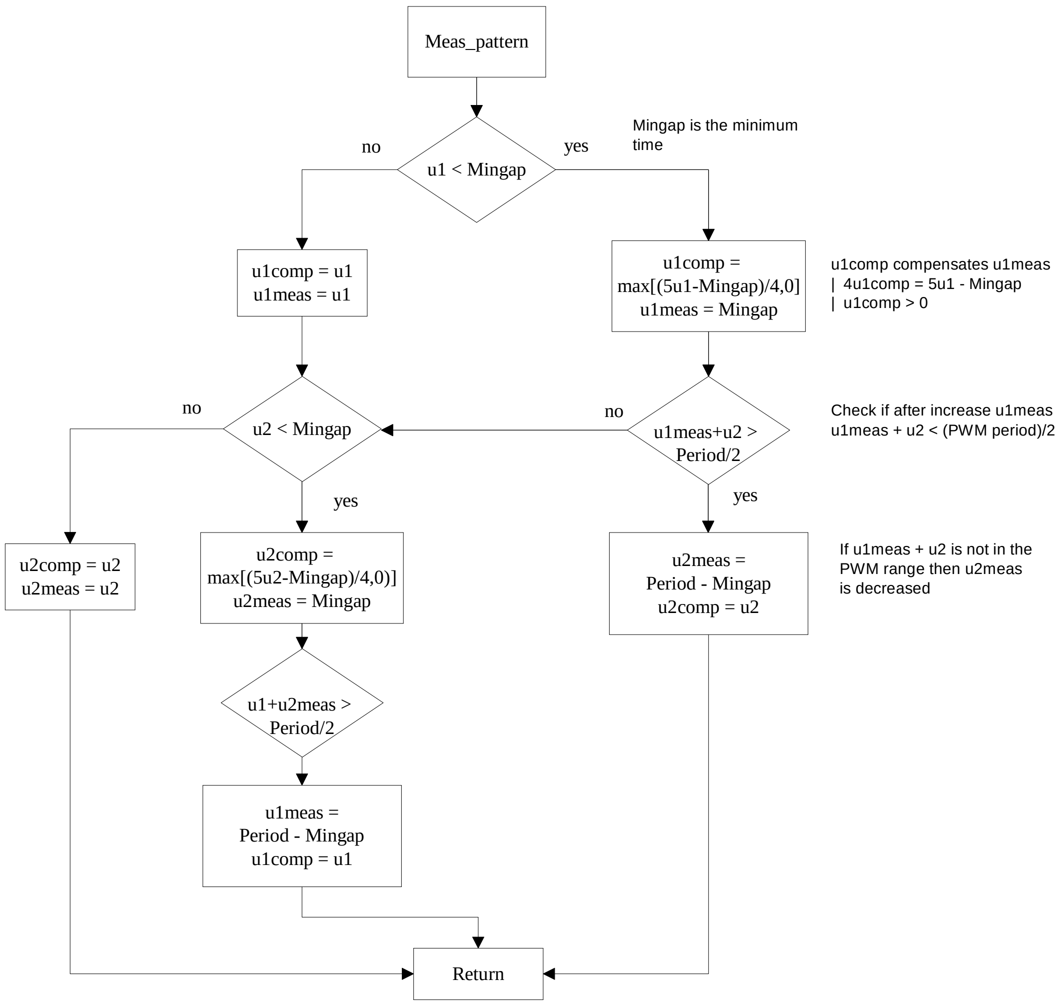

Figure 17: Meas_pattern function flowchart

5. Other solutions found in the literature

5.1 Principle

Another method found in the literature consists of generating one pattern during one control cycle time (in the example: 250µs). The line current is then continuously sampled (every 15.6µs) and sorted according to the inverter state to update a stack containing the measured phase currents. With all the samples obtained, averages are taken to give each phase current.

As the sampling is performed at fixed-time, some small patterns (less than 30µs) may not be detected. To circumvent these undetectable signals, a zero duty cycle will be used for the first PWM and the theoretical PWM will be accumulated to the next period duty cycle of the same vector. This process continues until the accumulated duty cycle exceeds 30µs.

As the samples are not synchronised to the inverter states, ensuring the line currents match the correct states requires a large minimum duty cycle (here: 30µs).

5.2 Performance comparison

Let us compare the performances of the above two methods for a 450W, two pair poles, asynchronous motor at a speed of 150rpm with an empty drum as a load and a voltage supply of 270V. This speed and load represent the worst case for our defined hardware.

The maximum ΔPWM (time gap between two switching states) we can detect, due to our hardware limitations is 2.8µs. Let us now consider that we have to generate an energy inside the motor corresponding to u_n with $n \in [1,2]$, equal to 2.8µs over 400µs. The second method described in this application report is called the ‘compensated solution’. It will be possible to measure the current during every control cycle, by generating one pattern with $u_{n\_measure}$=2.8µs and four others with $u_{n\_compensate}$=0. To keep the same ratio for both methods (they have different control cycle times), the energy corresponding to 2.8µs over 400µs is 1.75µs over 250µs. The sampling rates for these two methods for this low speed and low load condition are given below.

To acquire a sample, the standard method requires a minimum duty cycle of 30µs. To get the performance described above, the register has to accumulate (*abs(*30/1.75*)+*1*=)* 18 times the energy over 250µs. The control will then acquire a sample every 18x250=4.5ms.

In the same conditions the ‘compensated solution’ will get a sample every 400µs.

The sampling rate is then 10 times higher in our solution.

The hardware used in this case has a dead-band of 1.2µs but already some higher speed range inverters are able to switch off in less than 150ns and have drivers able to generate a dead-band of few hundreds of nanoseconds. Therefore, it is possible that existing devices will reach a ΔPWM equals to less than 500ns. The performance of the ‘compensated solution’ over the classic solution is increased by the same ratio.

All these calculations have been performed for a specific application. The above figures and ratios may be very different for another application, but in every case the results of our measurement will remain more accurate than that of the classic solution.

Advantages of the ‘compensated’ solution:

- This is a **synchronous method**, therefore all the algorithms can be used with a constant time base and this is the basis for all control algorithms
- It provides a smooth control for low speed and low load and therefore a **better efficiency**
- As the exact current sampling time is known it is necessary to take only one sample. To obtain the final measured current it is not necessary **either to calculate an average of the samples or to make a filter** to reduce the effects of wrong state

latching. Therefore, there is a saving of calculation time needed to measure the currents.

- It is possible to control a motor over a very wide range of speeds and loads with **performance 10 times higher** than usual methods.

6. Results

6.1 Hardware configuration

The results are given on a board using an inverter from International Rectifier. The inverter is made of six IGBT IRGPC40F with

- max. continuous collector current of 27A
- turn-on delay time of 25ns
- rise time of 37ns
- max. turn-off delay time of 410ns
- max. fall timer of 420ns

The driver, an IR 2130, has a dead-band of 2µs.

The minimum time during which it's possible to make the measurement is 3.5µs.

6.2 First method

This method gives good results for most of the cases. The best results are achieved with currents close to their nominal values. In the application, the maximum value for sampling current is in the range of +/-10 Amps. The following results are obtained for a phase current equal to 11% of the detectable current, the next plot is made for a ratio of 17%. These plots are obtained without any software filter, only a smoothing filter from the oscilloscope has been used to suppress the measurement noise from the probe.

The Figure 18 shows:

- Plot 1: 11% phase current calculated with control
- Plot 2: measured phase current through Hall effect sensor

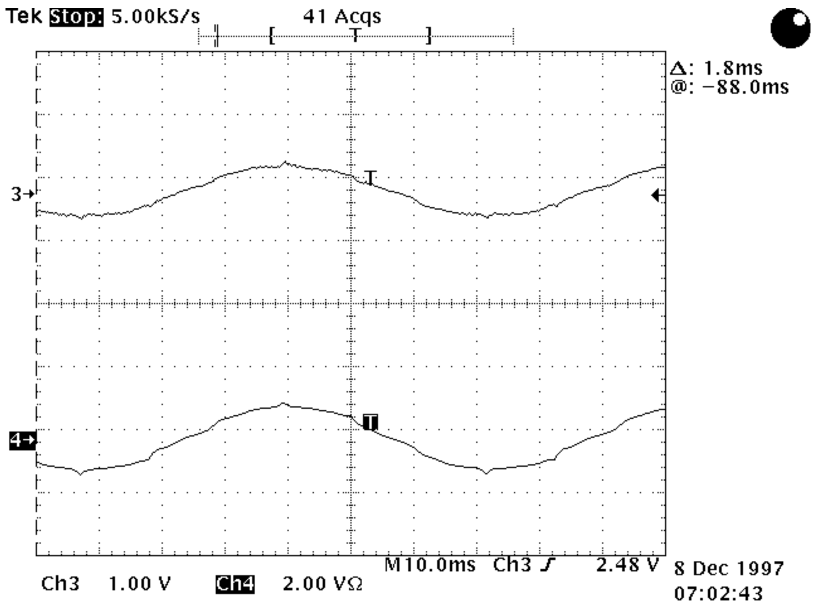

Figure 18: 11 % sensed an built current comparison

Plot1 is output through a digital to analogue converter included in the EVMF240. Comparing its maximum re-scaled value to the real phase current from the Hall Effect sensor, they are both equal. Looking more into details, some non-linearities appear on the plots. Most of them are due to the measurement process that forces some PWM patterns to minimum values as explained in chapter “Solution to circumvent the hardware limitations”. On the other hand, the spikes present on both the rebuilt current and the measured phase currents illustrate the dynamic of the process and the lack of filtering.

The Figure 19 shows:

- Plot 1: 15% phase current calculated with control 1
- Plot 2: measured phase current through Hall effect sensor

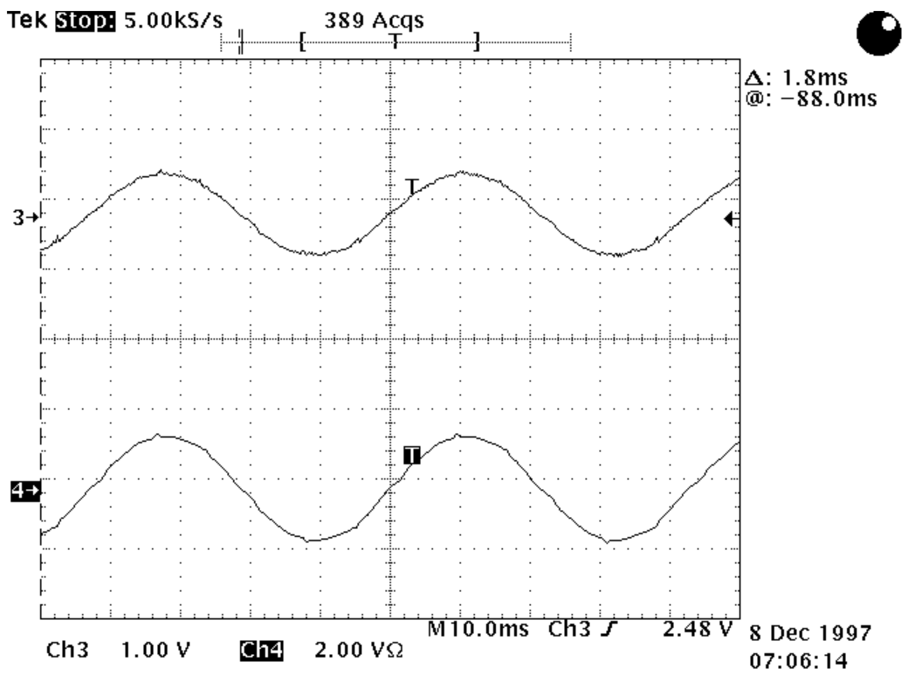

Figure 19: 15 % sensed an built current comparison

It can be observed that for a higher current the measured phase current is more sinusoidal. The ratio of current for which the current may be considered as sinusoidal varies depending on the inverter used.

If lower ratios of current detected are needed, more advanced inverters may be considered.

6.3 Second method tested

The following plot illustrates the process described in the chapter “2.5 Enhanced algorithm”. Its comparison with the previous plots from the first method shows that for the a current ratio of 14% non-linearities due to the minimum pattern imposed can hardly be detected.

The Figure 20 shows:

- Plot 1: 14% phase current calculated with control 2
- Plot 2: measured phase current through Hall effect sensor

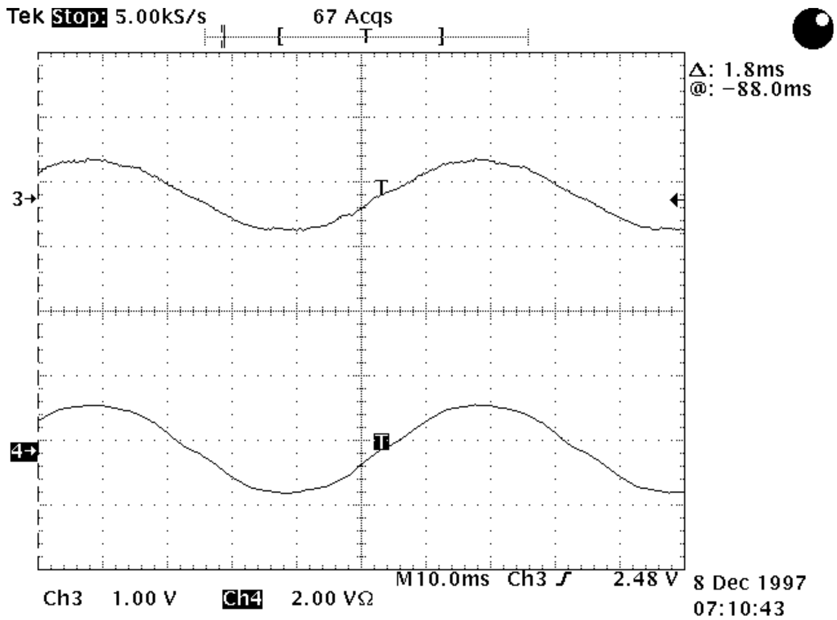

Figure 20: The perturbation of the measurement process decreased by five

One measurement is performed every five PWM. The perturbation of the measurement process is then decreased by five.

6.4 Closed loop control with the second method

The previous plots were taken with a V/f control. As the voltage is maintained constant, the perturbations are observed on the current, and therefore on the torque. If a current control is applied, it is possible to get a perfect sinusoidal current together with the 'reduced current sensor' algorithm.

The next plot has been measured with the same hardware but the AC induction motor is controlled with a Field Orientated Control Algorithm.

The Figure 21 shows:

- Plot 1: measured phase current through Hall Effect sensor
- Plot 2: calculated phase current at 300rpm (nominal speed: 1500rpm)

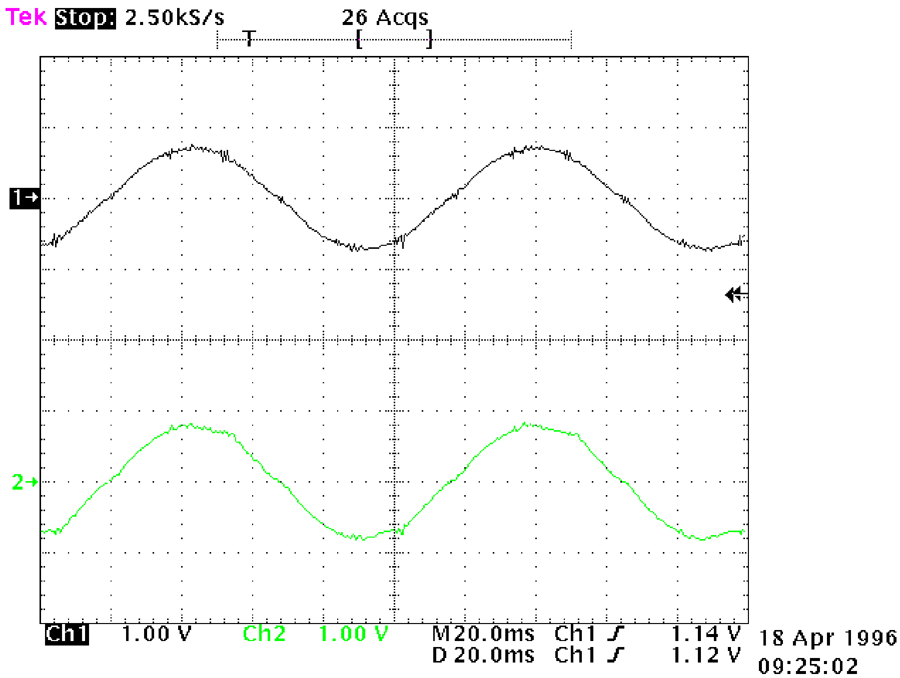

Figure 21: FOC with shunt measurement

The measured current is obtained here without any filtering or interpolation.

6.5 Speed limitation

The measurement method presented in the application note doesn't impose any constraint of speed range. The only magnitude that limits the use of the idc current measurement is the ratio between the actual current and the nominal current.

The speed range is limited only by the controller capability.

The Figure 22 shows:

- Plot 1: calculated current / 4700rpm
- Plot 2: phase current sensed 10mA<=>1A

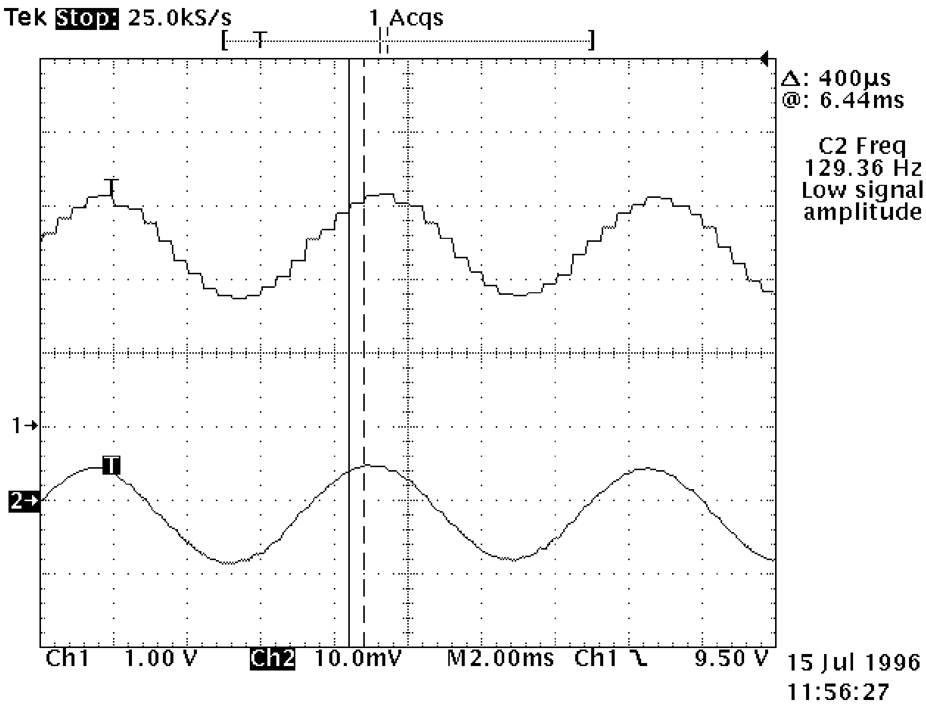

Figure 22: FOC with shunt measurement at high speed

The above plot shows the effect of the controller cycle time on the current. The current samples perfectly match the real phase current.

The Figure 23 shows:

- Plot 1: calculated current / 2.2 rpm
- Plot 2: phase current sensed 10mA<=>1A

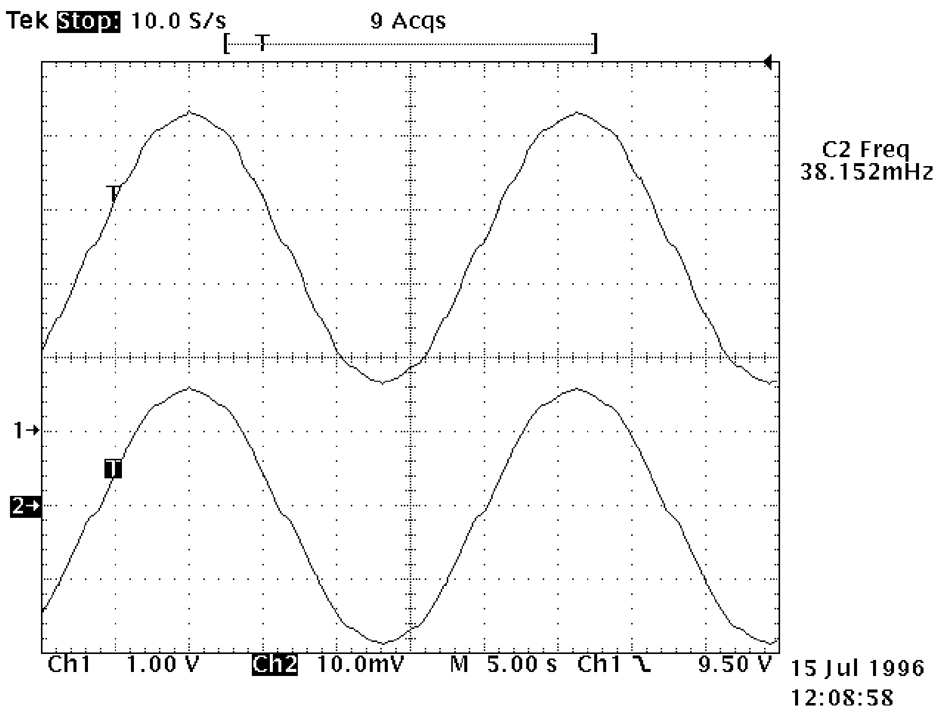

Figure 23: FOC with shunt measurement at very low speed

Low speed is also not a limitation to the current measurement as soon as the current is high enough.

7. Conclusion

The algorithm and its high performance have increased the utility of the DSP in motor control.

The performance can increase in terms of torque and speed control by using efficient control algorithms with current feedback at a similar price to that of existing lower performance solutions. This method is applicable to most synchronous and asynchronous motor drives, or, more generally, to three phase inverters. This technique is useful in the White Goods, inverter and machine tools market.

Texas Instruments has a U.S. patent pending on some of the topics described in this application note. Serial number: 08/903,110.

References

1. TMS320C24x DSP Controllers, Reference sets 1997 Texas Instruments ref. SPRU160A & SPRU161A
2. Current detection method for DC to three-phase converters using a single DC sensor, U.S. patent application 5,309,349, K.S. Kwan, 1992
3. Current estimator for a three phase inverter, U.S. Patent application 08/903,110, Texas Instruments, M. Platnic, 1997
4. A stator Flux-Oriented Voltage Source Variable-Speed Drive Based on dc Link Measurement, Xue, Xu, Habetler, Divan, 1991 IEEE
5. A Low Cost Stator Flux Oriented Voltage Source Variable Speed Drive, Xue, Xu, Habetler, Divan, University of Wisconsin-Madison, 1990 IEEE
6. DSP Solution for Permanent Magnet Synchronous Motor, Texas Instruments, Application report 1996 ref. BPRA044
7. DSP Solution for AC Induction Motor, Texas Instruments, Application report 1996, ref. BPRA043

Appendix A: Linker command file

```
/****************************************************************/
/*                        TEXAS INSTRUMENTS                     */
/****************************************************************/
/*   File Name:  link.cmd                                       */
/*   Originator: Michel Platnic                                 */
/*                                                              */
/*   Description:Link command file                              */
/*  MEMORY SPECIFICATION FOR THE EVMF240 FROM TEXAS INSTRUMENTS */
/*  Block B0 is configured as data memory (CNFD) and MP/MC- = 1 */
/*  (microprocessor mode). Note that data memory locations 6h--5Fh*/
/*  and 80h--1FFh are not configured.                           */
/*                                                              */
/*   Target:     TMS320F240, EVMF240                            */
/*   status:     Working                                        */
/*                                                              */
/*   History:    Completed on 28 November 97                    */
/****************************************************************/

MEMORY
{
    PAGE 0:
    FLASH_VEC   : origin =    0h, length =  40h
    FLASH       : origin =  040h, length =  03FC0h

    PAGE 1:
    REGS       : origin =     0h, length =    60h
    BLK_B22    : origin =    60h, length =    20h
    BLK_B0     : origin =   200h, length =   100h
    BLK_B1     : origin =   300h, length =   100h
    EXT_DATA   : origin =  8000h, length =  1000h
}

/*-------------------------------------------------------------*/
/* SECTIONS ALLOCATION                                         */
/*-------------------------------------------------------------*/
SECTIONS
{
    vectors : { } > FLASH_VEC PAGE 0  /* INTERRUPT VECTOR TABLE  */
    .text   : { } > FLASH PAGE 0      /* CODE                    */
    blockb2 : { } > BLK_B22 PAGE 1    /* CONTEXT SAVE            */
    .bss    : { } > EXT_DATA PAGE 1   /* GLOBAL VARS, STACK, HEAP*/
    .data   : { } > EXT_DATA PAGE 1   /* VARIABLES               */
    table   : { } > EXT_DATA PAGE 1   /* SINE TABLE              */
}
```

Appendix B: User interface Quick Basic program

```
REM ****************************************************************
REM *                TEXAS INSTRUMENTS                             *
REM ****************************************************************
REM *   File Name:  open_spe.bas                                   *
REM *   Originator: Michel Platnic                                 *
REM *   Description:User Interface on Quick Basic                  *
REM *                                                              *
REM *   Function list: No function, linear software               *
REM *   Target:     TMS320F240, EVMF240 with 4 DAC use             *
REM *                                                              *
REM *   History:    Completed on 28 November 97                    *
REM ****************************************************************

OPEN "COM1: 9600,N,8,1,CD0,CS0,DS0,OP0,RS,TB1,RB1" FOR OUTPUT AS #1
PRINT #1, "1"; CHR$(0); CHR$(0); : REM speed reference initialization to 0
PRINT #1, "2"; CHR$(0); CHR$(1); CHR$(2); CHR$(3); : REM DAC initialization
delay = 10
Mingap = 80
est = 0
speedref = 0
init = 0
VDC = 310
da1 = 0: da2 = 1
da3 = 18: da4 = 24

speedpu = 1500: REM base speed

DIM daout$(200)
daout$(0) = "i1"
daout$(1) = "i2"
daout$(2) = "i3"
daout$(3) = "i_remote1"
daout$(4) = "i_remote2"
daout$(5) = "i_remote3"
daout$(6) = "u1"
daout$(7) = "u2"
daout$(8) = "seno1"
daout$(9) = "coseno"
daout$(10) = "Va"
daout$(11) = "Vb"
daout$(12) = "Vc"
daout$(13) = "VDC"
daout$(14) = "taon"
daout$(15) = "tbon"
daout$(16) = "tcon"
daout$(17) = "teta"
daout$(18) = "Valfar"
daout$(19) = "Vbetar"
daout$(20) = "speedr"
daout$(21) = "X"
daout$(22) = "Y"
daout$(23) = "Z"
daout$(24) = "sector"
daout$(25) = "sectorold/synchro"

nDA = 8

1 CLS
```

```
FOR i = 0 TO nDA
COLOR 11
LOCATE (15 + i), 2: PRINT "("; : PRINT USING "##"; i; : PRINT ") "; daout$(i)
LOCATE (15 + i), 29: PRINT "("; : PRINT USING "##"; i + nDA + 1; : PRINT ") "; daout$(i + nDA + 1)
LOCATE (15 + i), 56: PRINT "("; : PRINT USING "##"; i + 2 * nDA + 2; : PRINT ") "; daout$(i + 2 * nDA
+ 2)
NEXT i
LOCATE 2, 11
COLOR 12: PRINT " Digital Control of an AC Induction Motor using V/f"
LOCATE 3, 7
COLOR 12: PRINT "Demo for 3 phase currents measurement with one shunt resistor"
PRINT
PRINT
COLOR 10: PRINT "   <1>"; : COLOR 2: PRINT " Speed_reference      ("; speedref; "rpm )"
COLOR 10: PRINT "   <2>"; : COLOR 2: PRINT " DAC_Outputs          DAC1: ("; daout$(da1); ")"
LOCATE 7, 48: PRINT "DAC2: ("; daout$(da2); ")"
PRINT "                            DAC3: ("; daout$(da3); ")"
LOCATE 8, 48: PRINT "DAC4: ("; daout$(da4); ")"
COLOR 10: PRINT "   <3>"; : COLOR 2: PRINT " Delay  (*50ns)       ("; delay; ")"
COLOR 10: PRINT "   <4>"; : COLOR 2: PRINT " Mingap (*50ns)       ("; Mingap; ")"

COLOR 10: LOCATE 12, 14: PRINT "Choice : ";
DO

a$ = INKEY$
LOOP UNTIL ((a$ <= "4") AND (a$ >= "1")) OR (a$ = "r") OR (a$ = "R")

SELECT CASE a$
CASE "1"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "Speed_Reference ("; speedref; "rpm ) : ";
    INPUT speedref$
    IF speedref$ = "" THEN 1
    speedrpu = VAL(speedref$) / speedpu
    IF (ABS(speedrpu) > 1.2) THEN speedrpu = 1.2 * SGN(speedrpu)
    IF (speedrpu >= 7.999755859#) THEN speedrpu = 7.999755859#
    IF (speedrpu <= -8) THEN speedrpu = -8
    speedrefpu = CLNG(speedrpu * 4096)
    IF (speedref < 0) THEN speedrefpu = 65536 + speedrefpu
    PRINT #1, "1"; CHR$(speedrefpu AND 255); CHR$((speedrefpu AND 65280) / 256)
    speedref = speedrpu * speedpu
    GOTO 1
CASE "2"
    REM standard decimal format
    PRINT a$; ") ";
    PRINT "DAC1, DAC2, DAC3 or DAC4 ? ";
2       dach$ = INKEY$
    IF dach$ = "" THEN 2
    IF dach$ = CHR$(13) THEN 1
    IF dach$ = "1" THEN
        PRINT "DAC1 Output ("; da1; ") : ";
        INPUT da$
        IF da$ = "" THEN 1
        da1 = VAL(da$)
    END IF
    IF dach$ = "2" THEN
        PRINT "DAC2 Output ("; da2; ") : ";
        INPUT da$
        IF da$ = "" THEN 1
        da2 = VAL(da$)
    END IF
```

```
    IF dach$ = "3" THEN
        PRINT "DAC3 Output ("; da3; ") : ";
        INPUT da$
        IF da$ = "" THEN 1
        da3 = VAL(da$)
    END IF
    IF dach$ = "4" THEN
        PRINT "DAC4 Output ("; da4; ") : ";
        INPUT da$
        IF da$ = "" THEN 1
        da4 = VAL(da$)
    END IF
    PRINT #1, "2"; CHR$(da1 AND 255); CHR$(da2 AND 255); CHR$(da3 AND 255); CHR$(da4 AND 255)
    GOTO 1
CASE "3"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "delay ("; delay; ") : ";
    INPUT delay$
    IF delay$ = "" THEN 1
    delay = VAL(delay$)
    PRINT #1, "3"; CHR$(delay AND 255); CHR$((delay AND 65280) / 256)
    GOTO 1
CASE "4"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "Mingap ("; Mingap; ") : ";
    INPUT Mingap$
    IF Mingap$ = "" THEN 1
    Mingap = VAL(Mingap$)
    PRINT #1, "4"; CHR$(Mingap AND 255); CHR$((Mingap AND 65280) / 256)
    GOTO 1
CASE ELSE
    PRINT #1, "1"; CHR$(speedrefpu AND 255); CHR$((speedrefpu AND 65280) / 256)
    PRINT #1, "2"; CHR$(da1 AND 255); CHR$(da2 AND 255); CHR$(da3 AND 255); CHR$(da4 AND 255)
    PRINT #1, "3"; CHR$(delay AND 255); CHR$((delay AND 65280) / 256)
    PRINT #1, "4"; CHR$(Mingap AND 255); CHR$((Mingap AND 65280) / 256)
    GOTO 1
END SELECT
CLOSE #1
```

Appendix C: Software program describing the first method

```
****************************************************************
*                TEXAS INSTRUMENTS                             *
****************************************************************
*   File Name:  open_spe.asm                                   *
*   Originator: Michel Platnic                                 *
*   Description:The software includes                          *
*                  -Induction motor open loop control          *
*                  -current measurement with shunt resistor    *
*                  -V/f control                                *
*                  -User Interface                             *
*                                                              *
*   Function list: No function, linear software                *
*   Target:     TMS320F240, EVMF240 if DAC use                 *
*                                                              *
*   History:    Completed on 28 November 97                    *
****************************************************************

            .include ".\c240app.h"
            .mmregs

****************************************************************
* Start                                                        *
****************************************************************
            .globl  _c_int0 ;set _c_int0 as global symbol

            .sect    "vectors"
            b       _c_int0 ;reset interrupt handler
_c_int1     b       _c_int1 ;RTI,SPI,SCI,Xint interrupt handler
            b       _PR_int ;PWM interrupt handler
_c_int3     b       _c_int3 ;
_c_int4     b       _c_int4 ;
_c_int5     b       _c_int5 ;
_c_int6     b       _c_int6 ;capture/ encoder Interrupts
            .space 16*6     ;reserve 6 words in interrupt table


****************************************************************
* Auxiliary Register used                                      *
* ar4   pointer for context save stack                         *
* ar5   used in the interruption PR_int for control calculation*
* ar6   for main program                                       *
****************************************************************


stack         .usect "blockb2",15    ;space for Status Register context save in Page 0

*** Motor ERCOLE MARELLI, Nr D 50525/s MW ***
*** Numeric formats: all 4.12 fixed point format twos complement for negative values (4 integer &
sign + 12 fractional) except otherwise specified
* - Currents: 1000h (4.12)= 1A
* - Voltages: 1000h (4.12)= 311 V
* - Angles  : [0;ffffh]  = [0;360] degrees
* - Speed   : [0;1000h] (4.12= = [0;1500] rpm
*** END Numeric formats

****************************************************************
* Look-up tables .includes                                     *
* N.B. all tables include 256 elements                         *
```

```
***************************************************************
                .sect "table"

sintab          .include        sine.tab
            ;sine wave look-up table for sine and cosine waves generation
            ;generated by the BASIC program "SINTAB.BAS"
            ;4.12 format
*** END look-up table .includes

***************************************************************
* Variables and constants initialisations                     *
***************************************************************
                .data

*** current sampling constants
Kcurrent        .word   019b5h  ;8.8 format (25.71) sampled currents
                                     normalisation constant
*** axis transformation constants
SQRT3inv        .word   093dh   ;1/SQRT(3) 4.12 format
SQRT32          .word   0ddbh   ;SQRT(3)/2 4.12 format
*** PWM modulation constants
PWMPRD          .set    0896    ;PWM Period=2*896 -> Tc=2*896*50ns=89.6us (50ns resolution)
Mingap          .word   80      ;minimum PWM duty cycle
                                ;the MAXDUTY is calculated as PWMPRD-2*Mingap
                                ;it is the maximum utilisation of the inverter
delay           .word   10      ;delay for Idc measurement
zero            .word   0h
MAX .set 736

    .bss    tmp,1           ;temporary variable (to use in ISR only !!!)
    .bss    option,1        ;virtual menu option number
    .bss    daout,1         ;address of the variable to send to the DACs
    .bss    daouttmp,1      ;value to send to the DACs
    .bss    tetaad,1        ;teta openloop variable

*** DAC displaying table starts here
    .bss    i1,1            ;phase current i1
    .bss    i2,1            ;phase current i2
    .bss    i3,1            ;phase current i3
    .bss    i_remote1,1     ;first of the 2 idc currents
    .bss    i_remote2,1     ;second of the 2 idc currents
    .bss    i_remote3,1     ;sum of the 2 idc currents negated
    .bss    u1,1            ;SVPWM T1 (see SV PWM references for details)
    .bss    u2,1            ;SVPWM T2 (see SV PWM references for details)
    .bss    seno1,1         ;generated sine wave value
    .bss    coseno,1        ;generated cosine wave value
    .bss    Va,1            ;Phase 1 voltage
    .bss    Vb,1            ;Phase 2 voltage
    .bss    Vc,1            ;Phase 3 voltage
    .bss    VDC,1           ;DC Bus Voltage
    .bss    taon,1          ;PWM commutation instant phase 1
    .bss    tbon,1          ;PWM commutation instant phase 2
    .bss    tcon,1          ;PWM commutation instant phase 3
    .bss    teta,1          ;rotor electrical position in the range [0;1000h]
                            ;4.12 format = [0;360] degrees
    .bss    Valfar,1        ;alfa-axis reference voltage
    .bss    Vbetar,1        ;beta-axis reference voltage
    .bss    speedr,1        ;speed reference
    .bss    X,1             ;SVPWM variable
    .bss    Y,1             ;SVPWM variable
    .bss    Z,1             ;SVPWM variable
    .bss    sectordisp,1    ;SVPWM sector for display
```

```
    .bss    sectorold,1     ;SVPWM sector buffer for current measurement
*** END DAC displaying table

    .bss    sector,1        ;SVPWM sector
    .bss    serialtmp,1     ;serial communication temporary variable
    .bss    da1,1           ;DAC displaying table offset for DAC1
    .bss    da2,1           ;DAC displaying table offset for DAC2
    .bss    da3,1           ;DAC displaying table offset for DAC3
    .bss    da4,1           ;DAC displaying table offset for DAC4
    .bss    VDCinv,1        ;1/VDC 4.12 format
    .bss    VDCinvTc,1      ;VDCinv*(Tc/2) (used in SVPWM)
    .bss    tetaincr,1      ;V/f open loop tetaincr    (1pu speed)
    .bss    Vamplitude,1    ;V/f open loop Vamplitude
    .bss    indice1,1       ;pointer used to access sine look-up table
    .bss    tmp1,1          ;tmp word to convert to C24
    .bss    accb,2          ;2 words to replace ACCB in C24
    .bss    acc_tmp,2       ;2 words to allow swapping of ACC in C24
    .bss    tetaref,1
*** END Variables and constants initializations


    .text                   ;link in "text section

****************************************************************
* _PR_int ISR                                                  *
* synchronisation of the control algorithm with the PWM        *
* underflow interrupt                                          *
****************************************************************
_PR_int:
    larp    ar4             ;context save
    mar     *-
    sst     #1,*-           ;status register 1
    sst     #0,*-           ;status register 0
    sach    *-              ;Accu. low saved for context save
    sacl    *-              ;Accu. high saved
    ldp     #IFRA>>7
    splk    #07FFh,IFRA     ;Clear all flags, may be change with only T1 underflow int.

    mar     *,ar5           ;used later for DACs output

****************************************************************
* Current Remote measurement - AD conversions                  *
* N.B. we will have to take only 10 bit (LSB)                  *
****************************************************************
    clrc    SXM
    ldp     #DP_PF1
    lacc    ADC_FIFO2       ;empty stack
    lacc    ADC_FIFO2       ;
    lacc    ADC_FIFO1,10    ;10.6 format
    ldp     #i_remote1
    sach    i_remote1       ;sampled current, f 4.12
    ldp     #DP_PF1
    lacc    ADC_FIFO1,10    ;10.6 format
    ldp     #i_remote1
    sach    i_remote2       ;sampled current, f 4.12
    setc    SXM

    ldp     #tbon
    bldd    tbon,#T1CMP
    bldd    tbon,#T3CMP
    ldp     #DP_PF1
    splk    #1803h,ADC_CNTL1;Channel 2 ADC2 selected for idc
```

```
                        ;ADC2 disable
                        ;start
    spm     3
    ldp     #i1
    lacl    i_remote1
    and     #3ffh
    sub     #512        ;then we have to subtract the offset (2.5V) to have
                        ;positive and negative values of the sampled current
    sacl    tmp
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    sub     #00h        ;then we subtract a DC offset (that should be zero, but it isn't
    sacl    i_remote1   ;sampled current  f 4.12

    lacl    i_remote2
    and     #3ffh
    sub     #512        ;then we have to subtract the offset (2.5V) to have
                        ;positive and negative values of the sampled current
    sacl    tmp
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    sub     #00h        ;then we subtract a DC offset (that should be zero, but it isn't
    neg                 ;second current always negative with the convention
    sacl    i_remote2   ;sampled current  f 4.12
    spm     0
    add     i_remote1
    neg
    sacl    i_remote3   ;third current calculated

*****************************************************************
* Current Remote measurement -                                  *
* determination of current measured    depending on sector      *
*****************************************************************

    lacc    sectorold
    sub     #3
    bcnd    sector123,LEQ
    sub     #3          ;sector 4,5 or 6
    bcnd    sector45,NEQ
    bldd    i_remote3,#i1   ;sector 6
    bldd    i_remote2,#i2
    b       end_remote
sector45
    bldd    i_remote2,#i1   ;sector 4 or 5
    add     #1
    bcnd    sector4,NEQ
    bldd    i_remote1,#i2   ;sector 5
    b       end_remote
sector4
    bldd    i_remote3,#i2   ;sector 4
    b       end_remote

sector123
    add     #2          ;sector 1,2 or 3
    bcnd    sector23,NEQ
    bldd    i_remote1,#i2   ;sector 1
```

```
    bldd    i_remote3,#i1
    b       end_remote
sector23
    bldd    i_remote1,#i1   ;sector 2 or 3
    sub     #1
    bcnd    sector3,NEQ
    bldd    i_remote2,#i2   ;sector 2
    b       end_remote
sector3
    bldd    i_remote3,#i2   ;sector 3

end_remote
    lacc    sector
    sacl    sectorold
*****************************************************************
* creating reference voltage for induction motor                *
*****************************************************************
    mar     *,AR5
    ldp     #tetaref
    lacc    speedr
    abs
    sacl    Vamplitude
    lt      speedr
    mpy     #126h
    pac
    sach    tetaincr,4
    lacc    tetaref
    add     tetaincr
    sacl    tetaref
    rpt     #3
    sfr
    sacl    teta
    rpt     #3
    sfr
    and     #0ffh           ;now ACC contains the pointer to access the table
    sacl    indice1         ;
    add     #sintab         ;
    sacl    tmp             ;
    lar     ar5,tmp
    nop
    nop                     ;
    mar     *,ar5
    lacl    *               ;
    nop
    sacl    seno1           ;now we have sine value

    lacl    indice1         ;the same thing for cosine ... cos(teta) = sin(teta+90°)
    add     #040h           ;90 degrees = 40h elements of the table
    and     #0ffh           ;
    sacl    indice1         ;we use the same pointer (we don't care)
    add     #sintab         ;
    sacl    tmp             ;
    lar     ar5,tmp         ;
    lacc    *               ;
    sacl    coseno          ;now we have cosine value

    lt      coseno
    mpy     Vamplitude
    pac

    sach    Valfar,4        ;format 4.12
    lt      seno1
```

```
    mpy     Vamplitude
    pac
    sach    Vbetar,4

****************************************************************
* Phase 1(=a) 2(=b) 3(=c) Voltage calculation                  *
* (alfa,beta) -> (a,b,c) axis transformation                   *
* modified exchanging alfa axis with beta axis                 *
* for a correct sector calculation in SVPWM                    *
* Va = Vbetar                                                  *
* Vb = (-Vbetar + sqrt(3) * Valfar) / 2                        *
* Vc = (-Vbetar - sqrt(3) * Valfar) / 2                        *
****************************************************************
    lt      Valfar          ;TREG0=Valfar
    mpy     SQRT32          ;PREG=Valfar*(SQRT(3)/2)
    pac                     ;ACC=PREG
    sub     Vbetar,11       ;ACC-=Vbetar*2^11
    sach    Vb,4            ;shift by 12 to reformat
    pac                     ;ACC=PREG
    neg                     ;ACC=-ACC
    sub     Vbetar,11       ;ACC-=Vbetar*2^11
    sach    Vc,4            ;shift by 12 to reformat
    lacl    Vbetar          ;ACC=Vbetar
    sacl    Va              ;Va=ACCL
*** END Phase 1(=a) 2(=b) 3(=c) Voltage calculation
****************************************************************
* SPACE VECTOR Pulse Width Modulation                          *
* (see SVPWM references)                                       *
****************************************************************
    lt      VDCinvTc
    mpy     SQRT32          ;change to dma
    pac
    sach    tmp,4           ;implement bsar 12 and sacl
    lt      tmp
    mpy     Vbetar
    pac
    sach    X,4
    lacc    X               ;ACC = Vbetar*K1
    sach    accb        ;
    sacl    accb+1          ;ACCB = Vbetar*K1
    sacl    X,1             ;X=2*Vbetar*K1
    lt      VDCinvTc
    splk    #1800h,tmp
    mpy     tmp             ;implement mpy #01800h
    pac
    sach    tmp,4           ;shift by 12 to reformat
    lt      tmp
    mpy     Valfar
    pac
    sach    tmp,4
    lacc    tmp             ;reload ACC with Valfar*K2
    add     accb+1
    add     accb,16
    sacl    Y               ;Y = K1 * Vbetar + K2 * Valfar
    sub     tmp,1
    sacl    Z               ;Z = K1 * Vbetar - K2 * Valfar
*** 60 degrees sector determination
    lacl    #0
    sacl    sector
    lacc    Va
    bcnd    Va_neg,LEQ      ;If Va<0 do not set bit 1 of sector
    lacc    sector          ;
```

```
    or      #1
    sacl    sector          ;implement opl #1,sector
Va_neg
    lacc    Vb        ;
    bcnd    Vb_neg,LEQ      ;If Vb<0 do not set bit 2 of sector
    lacc    sector          ;
    or      #2
    sacl    sector          ;implement opl #2,sector
Vb_neg
    lacc    Vc
    bcnd    Vc_neg,LEQ      ;If Vc<0 do not set bit 3 of sector
    lacc    sector          ;
    or      #4
    sacl    sector          ;implement opl #4,sector
Vc_neg
*** END 60 degrees sector   determination

*** T1 and T2 (= u1 and u2) calculation depending on the sector number
    lacl    sector          ;(see SPACE VECTOR Modulation references for details)
    sub     #1
    bcnd    no1,NEQ
    lacc    Z
    sacl    u1
    lacc    Y
    sacl    u2
    b       u1u2out
no1 lacl    sector
    sub     #2
    bcnd    no2,NEQ
    lacc    Y
    sacl    u1
    lacc    X
    neg
    sacl    u2
    b       u1u2out
no2 lacl    sector
    sub     #3
    bcnd    no3,NEQ
    lacc    Z
    neg
    sacl    u1
    lacc    X
    sacl    u2
    b       u1u2out
no3 lacl    sector
    sub     #4
    bcnd    no4,NEQ
    lacc    X
    neg
    sacl    u1
    lacc    Z
    sacl    u2
    b       u1u2out
no4 lacl    sector
    sub     #5
    bcnd    no5,NEQ
    lacc    X
    sacl    u1
    lacc    Y
    neg
    sacl    u2
    b       u1u2out
```

```
no5 lacc    Y
    neg
    sacl    u1
    lacc    Z
    neg
    sacl    u2
u1u2out
    lacc    u1              ;u1 and u2 minimum values must be Mingap
    sub     Mingap
    bcnd    u1_ok,GEQ       ;if u1>Mingap then u1_ok
    lacl    Mingap
    sacl    u1
u1_ok
    lacc    u2
    sub     Mingap
    bcnd    u2_ok,GEQ       ;if u2>Mingap then u2_ok
    lacl    Mingap
    sacl    u2
u2_ok
*** END u1 and u2 calculation

    lacc    u1              ;if u1+u2>2*Mingap we have to saturate u1 and u2
    add     u2              ;
    sacl    tmp             ;
    add     Mingap,1        ;
    sub     #PWMPRD
    bcnd    nosaturation,LT,EQ

*** u1 and u2 saturation,
    lacc    #PWMPRD,14      ;divide PERIOD-2MINGAP by (u1+u2)
    sub     Mingap,15
    sfl
    rpt     #15             ;
    subc    tmp             ;
    sacl    tmp             ;
    lt      tmp             ;calculate saturate values of u1 and u2
    mpy     u1              ;u1 (saturated)=u1*(PERIOD-2MINGAP/(u1+u2))
    pac                     ;
    sach    u1,1            ;
    mpy     u2              ;u2 (saturated)=u2*(PERIOD-2MINGAP/(u1+u2))
    pac                     ;
    sach    u2,1            ;
*** END u1 and u2 saturation

nosaturation
*** taon,tbon and tcon calculation
    lacc    #PWMPRD         ;calculate the commutation instants taon, tbon and tcon
    sub     u1              ;of the 3 PWM channels
    sub     u2              ;taon=(PWMPRD-u1-u2)/2
    sfr                     ;
    sacl    taon            ;
    add     u1              ;tbon=taon+u1
    sacl    tbon            ;
    add     u2              ;tcon=tbon+u2
    sacl    tcon            ;
*** END taon,tbon and tcon calculation

*** sector switching
    lacl    sector          ;depending on the sector number we have
    sub     #1              ;to switch the calculated taon, tbon and tcon
    bcnd    nosect1,NEQ     ;to the correct PWM channel
                            ;(see SPACE VECTOR Modulation references for details)
```

```
    bldd    tbon,#CMPR1     ;sector 1
    bldd    taon,#CMPR2
    bldd    tcon,#CMPR3
    b       dacout
nosect1
    lacl    sector
    sub     #2
    bcnd    nosect2,NEQ
    bldd    taon,#CMPR1     ;sector 2
    bldd    tcon,#CMPR2     ;
    bldd    tbon,#CMPR3     ;
    b       dacout
nosect2
    lacl    sector
    sub     #3
    bcnd    nosect3,NEQ
    bldd    taon,#CMPR1     ;sector 3
    bldd    tbon,#CMPR2     ;
    bldd    tcon,#CMPR3     ;
    b       dacout
nosect3
    lacl    sector
    sub     #4
    bcnd    nosect4,NEQ
    bldd    tcon,#CMPR1     ;sector 4
    bldd    tbon,#CMPR2     ;
    bldd    taon,#CMPR3     ;
    b       dacout
nosect4
    lacl    sector
    sub     #5
    bcnd    nosect5,NEQ
    bldd    tcon,#CMPR1     ;sector 5
    bldd    taon,#CMPR2     ;
    bldd    tbon,#CMPR3     ;
    b       dacout
nosect5
    bldd    tbon,#CMPR1     ;sector 6
    bldd    tcon,#CMPR2     ;
    bldd    taon,#CMPR3     ;
*** END sector switching
*** END * SPACE VECTOR Pulse Width Modulation

dacout
****************************************************************
* DAC output of channels 'da1' and 'da2'                       *
* Output on 12 bit Digital analog Converter                    *
* 5V equivalent to FFFh                                        *
****************************************************************
    lacc    sector,7        ;scale sector by 2^7 to have good displaying
    sacl    sectordisp      ;only for display purposes

*** DAC out channel 'da1'
    lacc    #i1             ;get the address of the first elements
    add     da1             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;on a 12 bit DAC, the number 2000h = 5 Volt
```

```
    sfr                     ;-2000h is 0 Volt
    add     #800h           ;0 is 2.5 Volt.
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC0_VAL
*** END DAC out channel 'da1'

*** DAC out channel 'da2'
    lacc    #i1             ;get the address of the first elements
    add     da2             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;we have 10 bit DAC, we want to have the number 2000h = 5 Volt
    sfr
    add     #800h           ;
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC1_VAL
*** END DAC out channel 'da2'

*** DAC out channel 'da3'
    lacc    #i1             ;get the address of the first elements
    add     da3             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;we have 10 bit DAC, we want to have the number 2000h = 5 Volt
    sfr
    add     #800h
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC2_VAL
*** END DAC out channel 'da3'

*** DAC out channel 'da4'
    lacc    #i1             ;get the address of the first elements
    add     da4             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;we have 10 bit DAC, we want to have the number 2000h = 5 Volt
    sfr
    add     #800h
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC3_VAL
*** END DAC out channel 'da4'

    OUT     tmp,DAC_VAL     ;start conversion


*** Context restore
    larp    ar4
    mar     *+
    lacl    *+              ;Accu. restored for context restore
    add     *+,16
```

```
    lst     #0,*+
    lst     #1,*+
*** End Context restore
    clrc    INTM
    ret

*** END _PR_int ISR


_c_int0:
****************************************************************
* Board general settings                                       *
****************************************************************
    clrc    CNF
    setc    SXM
    clrc    XF

****************************************************************
* Function to disable the watchdog timer                       *
****************************************************************
    ldp     #DP_PF1
    splk    #006Fh, WD_CNTL
    splk    #05555h, WD_KEY
    splk    #0AAAAh, WD_KEY
    splk    #006Fh, WD_CNTL

****************************************************************
* Function to initialise the Event Manager                     *
* GPTimer 1 => Full PWM                                        *
* Enable Timer 1==0 interrupt on INT2 and CAP1 on INT4         *
* Capture 1 reads tacho input                                  *
* All other pins are IO                                        *
****************************************************************
    ; Set up SYSCLK and PLL for C24 EVM with 10MHz External Clk
    ldp     #DP_PF1
    splk    #00000010b,CKCR0    ; PLL disabled
                                ; LPM0
                                ; ACLK enabled
                                ; SYSCLK 5MHz
    splk    #10110001b,CKCR1    ; 10MHz clk in for ACLK
                                ; Do not divide PLL
                                ; PLL ratio x2
    splk    #10000011b,CKCR0    ; PLL enabled
                                ; LPM0
                                ; ACLK enabled
                                ; SYSCLK 10MHz  PLL x2

    ; Set up CLKOUT to be SYSCLK
    splk    #40C0h,SYSCR

    ; Clear all reset variables
    lacc    SYSSR
    and     #69FFh
    sacl    SYSSR

    ; Set up zero wait states for external memory
    lacc    #0004h
    sacl    *
    out     *,WSGR

    ; Clear All EV Registers
    zac
```

```
    ldp     #DP_EV
    sacl    GPTCON
    sacl    T1CNT
    sacl    T1CMP
    sacl    T1PER
    sacl    T1CON
    sacl    T2CNT
    sacl    T2CMP
    sacl    T2PER
    sacl    T2CON
    sacl    T3CNT
    sacl    T3CMP
    sacl    T3PER
    sacl    T3CON
    sacl    COMCON
    sacl    ACTR
    sacl    SACTR
    sacl    DBTCON
    sacl    CMPR1
    sacl    CMPR2
    sacl    CMPR3
    sacl    SCMPR1
    sacl    SCMPR2
    sacl    SCMPR3
    sacl    CAPCON
    sacl    CAPFIFO
    sacl    FIFO1
    sacl    FIFO2
    sacl    FIFO3
    sacl    FIFO4

***  T1 is time base for PWMs
***  T3 starts conversions, T3 + delay = T1

    ;Initialise PWM    ; No software dead-band
    splk    #666h,ACTR  ; Bits 15-12 not used, no space vector
                        ; PWM compare actions
                        ; PWM6/PWM5 - Active Low/Active High
                        ; PWM4/PWM3 - Active Low/Active High
                        ; PWM2/PWM1 - Active Low/Active High
    splk    #100,CMPR1
    splk    #200,CMPR2
    splk    #300,CMPR3
    splk    #0207h,COMCON; FIRST enable PWM operation
                        ; Reload Full Compare when T1CNT=0
                        ; Disable Space Vector
                        ; Reload Full Compare Action when T1CNT=0
                        ; Enable Full Compare Outputs
                        ; Disable Simple Compare Outputs
                        ; Full Compare Units in PWM Mode
    splk    #8207h,COMCON; THEN enable Compare operation

    splk    #PWMPRD,T1PER; Set T1 period
    splk    #PWMPRD/2,T1CMP; Set T1 compare
    ldp     #delay
    bldd    delay,#T1CNT; configure counter register
    LDP     #DP_EV
    splk    #0A802h,T1CON; Ignore Emulation suspend
                        ; Cont Up/Down Mode
                        ; x/1 prescalar
                        ; Use own TENABLE
                        ; Disable Timer,enable later
```

```
                        ; Internal Clock Source
                        ; Reload Compare Register Immediately
                        ; Enable Timer Compare operation


*****************************************************************
* current remote measurement                                    *
* T3 starts the AD conversions                                  *
*****************************************************************
    ldp     #DP_EV
    splk    #PWMPRD,T3PER    ; configure period register
    splk    #PWMPRD/2,T3CMP  ; Set T3 compare
    splk    #0000,T3CNT
    splk    #0A88Ah,T3CON    ; configure
                             ; use TENABLE of T1CON
    splk    #1822h,GPTCON    ; bit 11-12: Start conversion on T3 compare match
    splk    #1862h,GPTCON    ; bit 11-12: Start conversion on T3 compare match
                             ; Enable compare outputs
                             ; T1 and T3 are Active high

    ; Enable Timer 1 and Timer 3
    lacc    T1CON
    or      #40h
    sacl    T1CON


    splk    #1802h,ADC_CNTL1; Channel 2, ADC1 selected for idc

*****************************************************************
* Part dedicated to the Hardware board used                     *
* PWM Channel enable for Driver                                 *
* 74HC541 chip enable connected to IOPC3 of Digital i/o         *
*****************************************************************
    ; Configure IO\function MUXing of pins
    ldp     #DP_PF2           ; Enable Power Security Function
    splk    #280Fh,OPCRA      ; Ports A/B all IO except ADCs, T1PWM and T3PWM
    splk    #00F9h,OPCRB      ; Port C as non IO function    except IOPC2&3
    splk    #0FF08h,PCDATDIR  ; bit IOPC3
*** END: PWM enable

*****************************************************************
* Initialize ar4 as the stack for context save                  *
* space reserved: DARAM B2 60h-80h (page 0)                     *
*****************************************************************
    lar     ar4,#79h


*****************************************************************
* A/D initialization                                            *
*****************************************************************
    ldp     #DP_PF1
    splk    #0403h,ADC_CNTL2  ; prescaler set for a 10MHz oscillator
                              ; enable conversion start by EV
*** END A/D initialization


*****************************************************************
* Variables initialization                                      *
*****************************************************************
    ldp     #speedr
    lacc    #500h
    sacl    speedr
```

```
    zac
    sacl    tetaref
    sacl    indice1
    sacl    Va
    sacl    Vb
    sacl    Vc
    splk    #0,da1             ;default i1
    splk    #1,da2             ;default i2
    splk    #18,da1            ;default Valfar
    splk    #24,da1            ;default sector

    spm     0                  ;no shift after multiplication
    setc    OVM
    setc    SXM                ;sign extension
*** END Variables initialization

****************************************************************
* VDC initialization                                           *
****************************************************************
    splk    #1000h,VDC         ; The DC voltage is 310V
                               ; Vdc in 4.12 with a Vbase=310V
    splk    #1000h,VDCinv      ; 1/Vdc
    splk    #380h,VDCinvTc     ; Tc/Vdc/2 or PWMPRD/VDC rescaled by 4.12

****************************************************************
* Serial communication initialization                          *
****************************************************************
    ldp     #DP_PF1
    splk    #00010111b,SCICCR ;one stop bit, no parity, 8bits
    splk    #0013h,SCICTL1    ;enable RX, TX, clk
    splk    #0000h,SCICTL2    ;disable SCI interrupts
    splk    #0000h,SCIHBAUD   ;MSB |
    splk    #0082h,SCILBAUD   ;LSB |9600 Baud for sysclk 10MHz
    splk    #0022h,SCIPC2     ;I/O setting
    splk    #0033h,SCICTL1    ;end initialization

****************************************************************
* Enable Interrupts                                            *
****************************************************************
    ; Clear EV IFR and IMR regs
    ldp     #DP_EV
    splk    #07FFh,IFRA
    splk    #00FFh,IFRB
    splk    #000Fh,IFRC

    ; Enable T1 Underflow Int
    splk    #0200h,IMRA
    splk    #0000h,IMRB
    splk    #0000h,IMRC

    ;Set IMR for INT2 and INT4 and clear any Flags
    ;INT2 (PWM interrupt) is used for motor control synchronization
        ;INT4 () is used for capture 3
    ldp     #0h
    lacc    #0FFh
    sacl    IFR
    lacc    #0000010b
    sacl    IMR

    ldp     #i1                ;set the right control variable page
    clrc    INTM               ;enable all interrupts, now we may serve
                               ;interrupts
```

```
*** END Enable Interrupts

****************************************************************
* Virtual Menu                                                 *
****************************************************************
menu
    clrc    XF                ;default mode (will be saved as context)
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available ?
    bcnd    menu,ntc          ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0ffh             ;only 8 bits !!!
    ldp     #option           ;if yes, get it and store it in option
    sacl    option            ;now in option we have the option number
                              ;of the virtual menu
    sub     #031h             ;is it option 1 ?
    bcnd    notone,neq        ;if not branch to notone

*****************************
* Option 1): Speed reference
*****************************
navail11
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail11,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp         ;if yes, get it and store it in serialtmp
navail12
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail12,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8        ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp         ;add ACC with lower byte
    sacl    speedr            ;store it
    b       menu              ;return to the main polling cycle
*** END Option 1): speed reference

notone
    lacc    option
    sub     #032h             ;is it option 2 ?
    bcnd    nottwo,neq        ;if not branch to nottwo

*****************************
* Option 2): DAC update
*****************************
navail21
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail21,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #da1
    sacl    da1               ;if yes, get it and store it in da1
navail22
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail22,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
```

```
    and     #0FFh              ;take the 8 LSB
    ldp     #da1
    sacl    da2                ;if yes, get it and store it in da2
navail23
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail23,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #da1
    sacl    da3                ;if yes, get it and store it in da3
navail24
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail24,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #da1
    sacl    da4                ;if yes, get it and store it in da4
    b       menu               ;return to the main polling cycle
*** END Option 2): DAC update

nottwo
    lacc    option
    sub     #033h              ;is it option 2 ?
    bcnd    notthree,neq       ;if not branch to nottwo

*****************************
* Option 3): delay
*****************************
navail31
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail31,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail32
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
    bcnd    navail32,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8         ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp          ;add ACC with lower byte
    sacl    delay              ;store it
    b       menu               ;return to the main polling cycle
*** END Option 3): delay

notthree
    lacc    option
    sub     #034h              ;is it option 2 ?
    bcnd    notfour,neq        ;if not branch to nottwo

*****************************
* Option 4): Mingap
*****************************
navail41
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail41,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
```

```
    and     #0FFh           ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp       ;if yes, get it and store it in serialtmp
navail42
    ldp     #DP_PF1
    bit     SCIRXST,BIT6    ;8 MSB available ?
    bcnd    navail42,ntc    ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8      ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp       ;add ACC with lower byte
    sacl    Mingap          ;store it
    b       menu            ;return to the main polling cycle
*** END Option 4): Mingap

notfour
    b    menu
```

Appendix D: Software program describing the second method

```
****************************************************************
*                TEXAS INSTRUMENTS                             *
****************************************************************
*   File Name:  open_spe.asm                                   *
*   Originator: Michel Platnic                                 *
*   Description:The software includes                          *
*                  -Induction motor open loop control          *
*                  -current measurement with shunt resistor    *
*                    2 current samples taken every 5 PWM period*
*                  -V/f control                                *
*                  -User Interface                             *
*                                                              *
*   Function list: -PR_int                                     *
*                  -control_Vf                                 *
*                  -meas_pattern                               *
*                  -get_current                                *
*                  -send_to_PWM                                *
*                                                              *
*   Target:     TMS320F240, EVMF240 if DAC use                 *
*   status:     Working                                        *
*                                                              *
*   History:    Completed on 28 November 97                    *
****************************************************************

            .include ".\c240app.h"
            .mmregs

****************************************************************
* Start                                                        *
****************************************************************
            .globl  _c_int0 ;set _c_int0 as global symbol

            .sect    "vectors"
            b       _c_int0 ;reset interrupt handler
_c_int1     b       _c_int1 ;RTI,SPI,SCI,Xint interrupt handler
            b       _PR_int ;PWM interrupt handler
_c_int3     b       _c_int3 ;
_c_int4     b       _c_int4 ;
_c_int5     b       _c_int5 ;
_c_int6     b       _c_int6 ;capture/ encoder Interrupts
            .space 16*6     ;reserve 6 words in interrupt table


****************************************************************
* Auxiliary Register used                                      *
* ar4   pointer for context save stack                         *
* ar5   used in the interruption PR_int for control calculation*
* ar6   for main program                                       *
****************************************************************


stack         .usect "blockb2",15    ;space for Status Register context save in Page 0

*** Motor ERCOLE MARELLI, Nr D 50525/s MW ***
*** Numeric formats: all 4.12 fixed point format twos complement for negative values (4 integer &
sign + 12 fractional) except otherwise specified
* - Currents: 1000h (4.12)= 1A
* - Voltages: 1000h (4.12)= 311 V
* - Angles  : [0;ffffh]  = [0;360] degrees
```

```
* - Speed   : [0;1000h] (4.12= = [0;1500] rpm
*** END Numeric formats
```

```
****************************************************************
* Look-up tables .includes                                     *
* N.B. all tables include 256 elements                         *
****************************************************************
                .sect "table"

sintab          .include        sine.tab
            ;sine wave look-up table for sine and cosine waves generation
            ;4.12 format
*** END look-up table .includes

****************************************************************
* Variables and constants initializations                      *
****************************************************************
                .data

*** current sampling constants

Kcurrent        .word   019b5h  ;8.8 format (25.71) sampled currents
               normalization constant
*** axis transformation constants
SQRT3inv        .word   093dh   ;1/SQRT(3) 4.12 format
SQRT32          .word   0ddbh   ;SQRT(3)/2 4.12 format
*** PWM modulation constants
PWMPRD          .set    0896    ;PWM Period=2*896 -> Tc=2*896*50ns=89.6us (50ns resolution)
Mingap          .word   80      ;minimum PWM duty cycle
                                ;the MAXDUTY is calculated as PWMPRD-2*Mingap
                                ;it is the maximum utilization of the inverter
delay           .word   10      ;delay for Idc measurement
zero            .word   0h
MAX .set 736

    .bss    tmp,1           ;temporary variable (to use in ISR only !!!)
    .bss    option,1        ;virtual menu option number
    .bss    daout,1         ;address of the variable to send to the DACs
    .bss    daouttmp,1      ;value to send to the DACs
    .bss    tetaad,1        ;teta openloop variable

*** DAC displaying table starts here
    .bss    i1,1            ;phase current i1
    .bss    i2,1            ;phase current i2
    .bss    i3,1            ;phase current i3
    .bss    i_remote1,1     ;first of the 2 idc currents
    .bss    i_remote2,1     ;second of the 2 idc currents
    .bss    i_remote3,1     ;sum of the 2 idc currents negated
    .bss    u1,1            ;SVPWM T1 (see SV PWM references for details)
    .bss    u2,1            ;SVPWM T2 (see SV PWM references for details)
    .bss    seno1,1         ;generated sine wave value
    .bss    coseno,1        ;generated cosine wave value
    .bss    Va,1            ;Phase 1 voltage
    .bss    Vb,1            ;Phase 2 voltage
    .bss    Vc,1            ;Phase 3 voltage
    .bss    VDC,1           ;DC Bus Voltage
    .bss    taon,1          ;PWM commutation instant phase 1
    .bss    tbon,1          ;PWM commutation instant phase 2
    .bss    tcon,1          ;PWM commutation instant phase 3
    .bss    teta,1          ;rotor electrical position in the range [0;1000h]
                            ;4.12 format = [0;360] degrees
    .bss    Valfar,1        ;alfa-axis reference voltage
    .bss    Vbetar,1        ;beta-axis reference voltage
    .bss    speedr,1        ;speed reference
    .bss    X,1             ;SVPWM variable
```

```
    .bss    Y,1             ;SVPWM variable
    .bss    Z,1             ;SVPWM variable
    .bss    sectordisp,1    ;SVPWM sector for diplay
    .bss    synchrodisp,1   ;Synchronization of PWM, shifted for display
*** END DAC displaying table
```

```
    .bss    sector,1        ;SVPWM sector
    .bss    synchro,1       ;Synchronization signal
    .bss    serialtmp,1     ;serial communication temporary variable
    .bss    u1_meas,1       ;u1 calculated for measurement
    .bss    u2_meas,1       ;u2 calculated for measurement
    .bss    u1_comp,1       ;u1 calculated to compensate the measurement
    .bss    u2_comp,1       ;u2 calculated to compensate the measurement
    .bss    da1,1           ;DAC displaying table offset for DAC1
    .bss    da2,1           ;DAC displaying table offset for DAC2
    .bss    da3,1           ;DAC displaying table offset for DAC3
    .bss    da4,1           ;DAC displaying table offset for DAC4
    .bss    VDCinv,1        ;1/VDC 4.12 format
    .bss    VDCinvTc,1      ;VDCinv*(Tc/2) (used in SVPWM)
    .bss    tetaincr,1      ;V/f open loop tetaincr    (1pu speed)
    .bss    Vamplitude,1    ;V/f open loop Vamplitude
    .bss    indice1,1       ;pointer used to access sine look-up table
    .bss    tmp1,1          ;tmp word to convert to C24
    .bss    accb,2          ;2 words to replace ACCB in C24
    .bss    acc_tmp,2       ;2 words to allow swapping of ACC in C24
    .bss    tetaref,1
*** END Variables and constants initializations


    .text                   ;link in "text section

****************************************************************
* _PR_int ISR                                                  *
* synchronization of the control algorithm with the PWM        *
* underflow interrupt                                          *
****************************************************************
_PR_int
    larp    ar4             ;context save
    mar     *-
    sst     #1,*-           ;status register 1
    sst     #0,*-           ;status register 0
    sach    *-              ;Accu. low saved for context save
    sacl    *-              ;Accu. high saved
    ldp     #IFRA>>7
    splk    #07FFh,IFRA     ;Clear all flags, may be change with only T1 underflow int.

    mar     *,ar5           ;used later for DACs output

    ldp     #i1
    lacc    synchro
    bcnd    synchro0,EQ
    lacc    synchro
    sub     #3
    bcnd    synchro3,EQ
    sub     #1
    bcnd    synchro4,EQ
    b       synchro_incr

synchro0
    call    get_current     ;from previous period
    call    control_Vf      ;start control
    call    meas_pattern    ;calculate u1 and u2 for measurement
    bldd    u1_comp,#u1     ;send new compensated PWM pattern for next period
    bldd    u2_comp,#u2     ;send new compensated PWM pattern for next period
    call    send_to_PWM
    b       synchro_incr

synchro3
```

```
    bldd    u1_meas,#u1     ;send measurement PWM pattern for next period
    bldd    u2_meas,#u2     ;send measurement PWM pattern for next period
    call    send_to_PWM
    b       synchro_incr

synchro4
    bldd    u1_comp,#u1     ;send compensated PWM pattern for next period
    bldd    u2_comp,#u2     ;send compensated PWM pattern for next period
    call    send_to_PWM
    ldp     #i1
    zac                     ;one control every 5 PWM
    sacl    synchro
    b       context

synchro_incr
    ldp     #i1
    lacc    synchro,9       ;shift by 9 for better display
    sacl    synchrodisp     ;variable for visualization on DAC
    lacc    synchro
    add     #1
    sacl    synchro

context
*** Context restore
    larp    ar4
    mar     *+
    lacl    *+              ;Accu. restored for context restore
    add     *+,16
    lst     #0,*+
    lst     #1,*+
*** End Context restore
    clrc    INTM
    ret

*** END _PR_int ISR


****************************************************************
* Get the current from A/D                                     *
*   Input var: A/D FIFO                                        *
*   Output var: i1, i2 the phase current                       *
* Current Remote measurement - AD conversions                  *
* N.B. we will have to take only 10 bit (LSB)                  *
****************************************************************
get_current
    clrc    SXM
    ldp     #DP_PF1
    lacc    ADC_FIFO1,10    ;10.6 format
    ldp     #i_remote1
    sach    i_remote1       ;sampled current, f 4.12
    ldp     #DP_PF1
    lacc    ADC_FIFO1,10    ;10.6 format
    ldp     #i_remote1
    sach    i_remote2       ;sampled current, f 4.12
    setc    SXM

    spm     3
    ldp     #i1
    lacl    i_remote1
    add     #00h            ;then we subtract a DC offset (that should be
       zero, but it isn't
    and     #3ffh
```

```
    sub     #512            ;then we have to subtract the offset (2.5V) to
       have
                            ;positive and negative values of the sampled
       current
    sacl    tmp
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    sacl    i_remote1       ;sampled current  f 4.12

    lacl    i_remote2
    add     #00h            ;then we subtract a DC offset (that should be zero, but it isn't
    and     #3ffh
    sub     #512            ;then we have to subtract the offset (2.5V) to have
                            ;positive and negative values of the sampled current
    sacl    tmp
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    neg                     ;second current always negative with the convention
    sacl    i_remote2       ;sampled current  f 4.12
    spm     0
    add     i_remote1
    neg
    sacl    i_remote3       ;third current calculated

****************************************************************
* Current Remote measurement -                                 *
* determination of current measured depending on sector        *
****************************************************************

    lacc    sector
    sub     #3
    bcnd    sector123,LEQ
    sub     #3              ;sector 4,5 or 6
    bcnd    sector45,NEQ
    bldd    i_remote3,#i1   ;sector 6
    bldd    i_remote2,#i2
    b       end_remote
sector45
    bldd    i_remote2,#i1   ;sector 4 or 5
    add     #1
    bcnd    sector4,NEQ
    bldd    i_remote1,#i2   ;sector 5
    b       end_remote
sector4
    bldd    i_remote3,#i2   ;sector 4
    b       end_remote

sector123
    add     #2              ;sector 1,2 or 3
    bcnd    sector23,NEQ
    bldd    i_remote1,#i2   ;sector 1
    bldd    i_remote3,#i1
    b       end_remote
sector23
    bldd    i_remote1,#i1   ;sector 2 or 3
    sub     #1
```

```
    bcnd    sector3,NEQ
    bldd    i_remote2,#i2   ;sector 2
    b       end_remote
sector3
    bldd    i_remote3,#i2      ;sector 3

end_remote

    ret

*** end function get_current
```

```
****************************************************************
* function control_Vf provides a Vf control                    *
*       input: i1, i2, speedr                                  *
*       output: u1, u2                                         *
* creating reference voltage for induction motor               *
****************************************************************
control_Vf
    mar     *,AR5
    ldp     #i1
    lacc    speedr
    abs
    sacl    Vamplitude
    lt      speedr
    mpy     #5beh           ;tetainc calculated for 5PWM
    pac
    sach    tetaincr,4
    lacc    tetaref
    add     tetaincr
    sacl    tetaref
    rpt     #3
    sfr
    sacl    teta
    rpt     #3
    sfr
    and     #0ffh           ;now ACC contains the pointer to access the table
    sacl    indice1         ;
    add     #sintab         ;
    sacl    tmp             ;
    lar     ar5,tmp
    nop
    nop                     ;
    mar     *,ar5
    lacl    *               ;
    nop
    sacl    seno1           ;now we have sine value

    lacl    indice1         ;the same thing for cosine ... cos(teta) = sin(teta+90°)
    add     #040h           ;90 degrees = 40h elements of the table
    and     #0ffh           ;
    sacl    indice1         ;we use the same pointer (we don't care)
    add     #sintab         ;
    sacl    tmp             ;
    lar     ar5,tmp         ;
    lacc    *               ;
    sacl    coseno          ;now we have cosine value

    lt      coseno
    mpy     Vamplitude
    pac

    sach    Valfar,4        ;format 4.12
    lt      seno1
    mpy     Vamplitude
    pac
    sach    Vbetar,4

****************************************************************
* Phase 1(=a) 2(=b) 3(=c) Voltage calculation                  *
* (alfa,beta) -> (a,b,c) axis transformation                   *
* modified exchanging alfa axis with beta axis                 *
* for a correct sector calculation in SVPWM                    *
* Va = Vbetar                                                  *
```

```
* Vb = (-Vbetar + sqrt(3) * Valfar) / 2                         *
* Vc = (-Vbetar - sqrt(3) * Valfar) / 2                         *
*****************************************************************
    lt      Valfar          ;TREG0=Valfar
    mpy     SQRT32          ;PREG=Valfar*(SQRT(3)/2)
    pac                     ;ACC=PREG
    sub     Vbetar,11       ;ACC-=Vbetar*2^11
    sach    Vb,4            ;shift by 12 to reformat
    pac                     ;ACC=PREG
    neg                     ;ACC=-ACC
    sub     Vbetar,11       ;ACC-=Vbetar*2^11
    sach    Vc,4            ;shift by 12 to reformat
    lacl    Vbetar          ;ACC=Vbetar
    sacl    Va              ;Va=ACCL
*** END Phase 1(=a) 2(=b) 3(=c) Voltage calculation

*****************************************************************
* SPACE VECTOR Pulse Width Modulation                           *
* (see SVPWM references)                                        *
*****************************************************************
    lt      VDCinvTc
    mpy     SQRT32          ;change to dma
    pac
    sach    tmp,4           ;implement bsar 12 and sacl
    lt      tmp
    mpy     Vbetar
    pac
    sach    X,4
    lacc    X               ;ACC = Vbetar*K1
    sach    accb
    sacl    accb+1          ;ACCB = Vbetar*K1
    sacl    X,1             ;X=2*Vbetar*K1
    lt      VDCinvTc
    splk    #1800h,tmp
    mpy     tmp             ;implement mpy #01800h
    pac
    sach    tmp,4           ;shift by 12 to reformat
    lt      tmp
    mpy     Valfar
    pac
    sach    tmp,4
    lacc    tmp             ;reload ACC with Valfar*K2
    add     accb+1
    add     accb,16
    sacl    Y               ;Y = K1 * Vbetar + K2 * Valfar
    sub     tmp,1
    sacl    Z               ;Z = K1 * Vbetar - K2 * Valfar
*** 60 degrees sector determination
    lacl    #0
    sacl    sector
    lacc    Va
    bcnd    Va_neg,LEQ      ;If Va<0 do not set bit 1 of sector
    lacc    sector          ;
    or      #1
    sacl    sector          ;implement opl #1,sector
Va_neg
    lacc    Vb        ;
    bcnd    Vb_neg,LEQ      ;If Vb<0 do not set bit 2 of sector
    lacc    sector          ;
    or      #2
    sacl    sector          ;implement opl #2,sector
Vb_neg
```

```
    lacc    Vc
    bcnd    Vc_neg,LEQ      ;If Vc<0 do not set bit 3 of sector
    lacc    sector          ;
    or      #4
    sacl    sector          ;implement opl #4,sector
Vc_neg
*** END 60 degrees sector   determination

*** T1 and T2 (= u1 and u2) calculation depending on the sector number
    lacl    sector          ;(see SPACE VECTOR Modulation references for details)
    sub     #1
    bcnd    no1,NEQ
    lacc    Z
    sacl    u1
    lacc    Y
    sacl    u2
    b       u1u2out
no1 lacl    sector
    sub     #2
    bcnd    no2,NEQ
    lacc    Y
    sacl    u1
    lacc    X
    neg
    sacl    u2
    b       u1u2out
no2 lacl    sector
    sub     #3
    bcnd    no3,NEQ
    lacc    Z
    neg
    sacl    u1
    lacc    X
    sacl    u2
    b       u1u2out
no3 lacl    sector
    sub     #4
    bcnd    no4,NEQ
    lacc    X
    neg
    sacl    u1
    lacc    Z
    sacl    u2
    b       u1u2out
no4 lacl    sector
    sub     #5
    bcnd    no5,NEQ
    lacc    X
    sacl    u1
    lacc    Y
    neg
    sacl    u2
    b       u1u2out
no5 lacc    Y
    neg
    sacl    u1
    lacc    Z
    neg
    sacl    u2
u1u2out

*** END u1 and u2 calculation
```

```
    ret
*** END function control_Vf


*****************************************************************
* Function: meas_pattern                                        *
*           calculate the measurement patterns                  *
*   Input: u1, u2                                               *
*   Output: u1_meas, u2_meas, u1_comp, u2_comp                  *
*****************************************************************
meas_pattern:
    lacc    u1              ;u1 and u2 minimum values must be Mingap
    sacl    u1_meas
    sacl    u1_comp
    sub     Mingap
    bcnd    u1_ok,GEQ       ;if u1>Mingap then u1_ok
    add     u1,2
    sfr
    sfr
    sacl    u1_comp         ;u1_comp =1/4(5*u1-Mingap)
    bcnd    u1_me,GEQ
    splk    #0,u1_comp      ;negative values not accepted
u1_me
    lacl    Mingap
    sacl    u1_meas
u1_ok
    lacc    u2
    sacl    u2_meas
    sacl    u2_comp
    sub     Mingap
    bcnd    u2_ok,GEQ       ;if u2>Mingap then u2_ok
    add     u2,2
    sfr
    sfr
    sacl    u2_comp         ;u2_comp =1/4(5*u2-Mingap)
    bcnd    u2_me,GEQ
    splk    #0,u2_comp      ;negative values not accepted
u2_me
    lacl    Mingap
    sacl    u2_meas
u2_ok
    ret
*** END function meas_pattern


*****************************************************************
* Function: Saturate u1 and u2,                                 *
*            Send them to PWM                                   *
*   Input: u1, u2, da1, da2, da3, da4                           *
*   Output: none                                                *
* SPACE VECTOR Pulse Width Modulation                           *
*****************************************************************
send_to_PWM:
    ldp     #i1
    lacc    u1              ;if u1+u2>2*Mingap we have to saturate u1 and u2
    add     u2              ;
    sacl    tmp             ;
    add     Mingap,1        ;
    sub     #PWMPRD
    bcnd    nosaturation,LT,EQ

*** u1 and u2 saturation,
```

```
    lacc    #PWMPRD,14      ;divide PERIOD-2MINGAP by (u1+u2)
    sub     Mingap,15
    sfl
    rpt     #15             ;
    subc    tmp             ;
    sacl    tmp             ;
    lt      tmp             ;calculate saturate values of u1 and u2
    mpy     u1              ;u1 (saturated)=u1*(PERIOD-2MINGAP/(u1+u2))
    pac                     ;
    sach    u1,1            ;
    mpy     u2              ;u2 (saturated)=u2*(PERIOD-2MINGAP/(u1+u2))
    pac                     ;
    sach    u2,1            ;
*** END u1 and u2 saturation

nosaturation
*** taon,tbon and tcon calculation
    lacc    #PWMPRD         ;calculate the commutation instants taon, tbon and tcon
    sub     u1              ;of the 3 PWM channels
    sub     u2              ;taon=(PWMPRD-u1-u2)/2
    sfr                     ;
    sacl    taon            ;
    add     u1              ;tbon=taon+u1
    sacl    tbon            ;
    add     u2              ;tcon=tbon+u2
    sacl    tcon            ;
*** END taon,tbon and tcon calculation

*** ADC synchronization
    bldd    tbon,#T1CMP
    bldd    tbon,#T3CMP     ;Event Manager synchronization for start
                            ;of AD conversion
*** End ADC synchronization

*** sector switching
    lacl    sector          ;depending on the sector number we have
    sub     #1              ;to switch the calculated taon, tbon and tcon
    bcnd    nosect1,NEQ     ;to the correct PWM channel
                            ;(see SPACE VECTOR Modulation references for details)
    bldd    tbon,#CMPR1     ;sector 1
    bldd    taon,#CMPR2
    bldd    tcon,#CMPR3
    b       dacout
nosect1
    lacl    sector
    sub     #2
    bcnd    nosect2,NEQ
    bldd    taon,#CMPR1     ;sector 2
    bldd    tcon,#CMPR2     ;
    bldd    tbon,#CMPR3     ;
    b       dacout
nosect2
    lacl    sector
    sub     #3
    bcnd    nosect3,NEQ
    bldd    taon,#CMPR1     ;sector 3
    bldd    tbon,#CMPR2     ;
    bldd    tcon,#CMPR3     ;
    b       dacout
nosect3
    lacl    sector
    sub     #4
```

```
    bcnd    nosect4,NEQ
    bldd    tcon,#CMPR1     ;sector 4
    bldd    tbon,#CMPR2     ;
    bldd    taon,#CMPR3     ;
    b       dacout
nosect4
    lacl    sector
    sub     #5
    bcnd    nosect5,NEQ
    bldd    tcon,#CMPR1     ;sector 5
    bldd    taon,#CMPR2     ;
    bldd    tbon,#CMPR3     ;
    b       dacout
nosect5
    bldd    tbon,#CMPR1     ;sector 6
    bldd    tcon,#CMPR2     ;
    bldd    taon,#CMPR3     ;
*** END sector switching
*** END * SPACE VECTOR Pulse Width Modulation
```

```
dacout
****************************************************************
* DAC output of channels 'da1' and 'da2'                       *
* Output on 12 bit Digital analog Converter                    *
* 5V equivalent to FFFh                                        *
****************************************************************
    lacc    sector,9        ;scale sector by 2^7 to have good displaying
    sacl    sectordisp      ;only for display purposes

*** DAC out channel 'da1'
    lacc    #i1             ;get the address of the first elements
    add     da1             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;on a 12 bit DAC, the number 2000h = 5 Volt
    sfr                     ;-2000h is 0 Volt
    add     #800h           ;0 is 2.5 Volt.
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC0_VAL
*** END DAC out channel 'da1'

*** DAC out channel 'da2'
    lacc    #i1             ;get the address of the first elements
    add     da2             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;we have 10 bit DAC, we want to have the number 2000h = 5 Volt
    sfr
    add     #800h           ;
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC1_VAL
*** END DAC out channel 'da2'

*** DAC out channel 'da3'
    lacc    #i1             ;get the address of the first elements
    add     da3             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;we have 10 bit DAC, we want to have the number 2000h = 5 Volt
    sfr
    add     #800h
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC2_VAL
*** END DAC out channel 'da3'

*** DAC out channel 'da4'
    lacc    #i1             ;get the address of the first elements
    add     da4             ;add the selected output variable offset 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the variable to send to DAC1
    lar     ar5,daout       ;store it in AR5
```

```
    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to adapt the numeric format to
the DAC resolution
    sfr                     ;we have 10 bit DAC, we want to have the number 2000h = 5 Volt
    sfr
    add     #800h
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC3_VAL
*** END DAC out channel 'da4'

    OUT     tmp,DAC_VAL     ;start convertion

    ret
*** END function send_to_PWM


_c_int0:
*****************************************************************
* Board general settings                                        *
*****************************************************************
    clrc    CNF
    setc    SXM
    clrc    XF

*****************************************************************
* Function to disable the watchdog timer                        *
*****************************************************************
    ldp     #DP_PF1
    splk    #006Fh, WD_CNTL
    splk    #05555h, WD_KEY
    splk    #0AAAAh, WD_KEY
    splk    #006Fh, WD_CNTL

*****************************************************************
* Function to initialise the Event Manager                      *
* GPTimer 1 => Full PWM                                         *
* Enable Timer 1==0 interrupt on INT2 and CAP1 on INT4          *
* Capture 1 reads tacho input                                   *
* All other pins are IO                                         *
*****************************************************************
    ; Set up SYSCLK and PLL for C24 EVM with 10MHz External Clk
    ldp     #DP_PF1
    splk    #00000010b,CKCR0    ; PLL disabled
                                ; LPM0
                                ; ACLK enabled
                                ; SYSCLK 5MHz
    splk    #10110001b,CKCR1    ; 10MHz clk in for ACLK
                                ; Do not divide PLL
                                ; PLL ratio x2
    splk    #10000011b,CKCR0    ; PLL enabled
                                ; LPM0
                                ; ACLK enabled
                                ; SYSCLK 10MHz  PLL x2

    ; Set up CLKOUT to be SYSCLK
    splk    #40C0h,SYSCR

    ; Clear all reset variables
    lacc    SYSSR
    and     #69FFh
    sacl    SYSSR
```

```
    ; Set up zero wait states for external memory
    lacc    #0004h
    sacl    *
    out     *,WSGR

    ; Clear All EV Registers
    zac
    ldp     #DP_EV
    sacl    GPTCON
    sacl    T1CNT
    sacl    T1CMP
    sacl    T1PER
    sacl    T1CON
    sacl    T2CNT
    sacl    T2CMP
    sacl    T2PER
    sacl    T2CON
    sacl    T3CNT
    sacl    T3CMP
    sacl    T3PER
    sacl    T3CON
    sacl    COMCON
    sacl    ACTR
    sacl    SACTR
    sacl    DBTCON
    sacl    CMPR1
    sacl    CMPR2
    sacl    CMPR3
    sacl    SCMPR1
    sacl    SCMPR2
    sacl    SCMPR3
    sacl    CAPCON
    sacl    CAPFIFO
    sacl    FIFO1
    sacl    FIFO2
    sacl    FIFO3
    sacl    FIFO4

***  T1 is time base for PWMs
***  T3 starts conversions, T3 + delay = T1

    ;Initialise PWM    ; No software dead-band
    splk    #666h,ACTR  ; Bits 15-12 not used, no space vector
                        ; PWM compare actions
                        ; PWM6/PWM5 - Active Low/Active High
                        ; PWM4/PWM3 - Active Low/Active High
                        ; PWM2/PWM1 - Active Low/Active High
    splk    #100,CMPR1
    splk    #200,CMPR2
    splk    #300,CMPR3
    splk    #0207h,COMCON; FIRST enable PWM operation
                        ; Reload Full Compare when T1CNT=0
                        ; Disable Space Vector
                        ; Reload Full Compare Action when T1CNT=0
                        ; Enable Full Compare Outputs
                        ; Disable Simple Compare Outputs
                        ; Full Compare Units in PWM Mode
    splk    #8207h,COMCON; THEN enable Compare operation

    splk    #PWMPRD,T1PER; Set T1 period
    splk    #PWMPRD/2,T1CMP; Set T1 compare
```

```
ldp     #delay
bldd    delay,#T1CNT; configure counter register
LDP     #DP_EV
splk    #0A802h,T1CON; Ignore Emulation suspend
                    ; Cont Up/Down Mode
                    ; x/1 prescalar
                    ; Use own TENABLE
                    ; Disable Timer,enable later
                    ; Internal Clock Source
                    ; Reload Compare Register Immediately
                    ; Enable Timer Compare operation
```

```
****************************************************************
* current remote measurement                                   *
* T3 starts the AD conversions                                 *
****************************************************************
    ldp     #DP_EV
    splk    #PWMPRD,T3PER    ; configure period register
    splk    #PWMPRD/2,T3CMP  ; Set T3 compare
    splk    #0000,T3CNT
    splk    #0A882h,T3CON    ; configure
                             ; use TENABLE of T1CON
    splk    #1822h,GPTCON    ; bit 11-12: Start conversion on T3 compare match
    splk    #1862h,GPTCON    ; bit 11-12: Start conversion on T3 compare match
                             ; Enable compare outputs
                             ; T1 and T3 are Active high

    ; Enable Timer 1 and Timer 3
    lacc    T1CON
    or      #40h
    sacl    T1CON

****************************************************************
* Part dedicated to the Hardware board used                    *
* PWM Channel enable for Driver                                *
* 74HC541 chip enable connected to IOPC3 of Digital i/o        *
****************************************************************
    ; Configure IO\function MUXing of pins
    ldp     #DP_PF2           ; Enable Power Security Function
    splk    #280Fh,OPCRA      ; Ports A/B all IO except ADCs, T1PWM and T3PWM
    splk    #00F9h,OPCRB      ; Port C as non IO function    except IOPC2&3
    splk    #0FF08h,PCDATDIR  ; bit IOPC3
*** END: PWM enable

****************************************************************
* Initialize ar4 as the stack for context save                 *
* space reserved: DARAM B2 60h-80h (page 0)                    *
****************************************************************
    lar     ar4,#79h

****************************************************************
* A/D initialization                                           *
****************************************************************
    ldp     #DP_PF1
    splk    #1802h,ADC_CNTL1  ; ADC1
    splk    #0403h,ADC_CNTL2  ; prescaler set for a 10MHz oscillator
                              ; disable conversion start by EV
;    splk    #1c02h,ADC_CNTL1  ; ADC1
*** END A/D initialization

****************************************************************
* Variables initialization                                     *
****************************************************************
    ldp     #speedr
    lacc    #500h
    sacl    speedr
    zac
    sacl    synchro
    sacl    tetaref
    sacl    indice1
    sacl    Va
    sacl    Vb
    sacl    Vc
    splk    #0,da1            ;default i1
```

```
    splk    #1,da2            ;default i2
    splk    #18,da1           ;default Valfar
    splk    #24,da1           ;default sector

    spm     0                 ;no shift after multiplication
    setc    OVM
    setc    SXM               ;sign extension
*** END Variables initialization

****************************************************************
* VDC initialization                                           *
****************************************************************
    splk    #1000h,VDC        ; The DC voltage is 310V
                              ; Vdc in 4.12 with a Vbase=310V
    splk    #1000h,VDCinv     ; 1/Vdc
    splk    #380h,VDCinvTc    ; Tc/Vdc/2 or PWMPRD/VDC rescaled by 4.12

****************************************************************
* Serial communication initialization                          *
****************************************************************
    ldp     #DP_PF1
    splk    #00010111b,SCICCR ;one stop bit, no parity, 8bits
    splk    #0013h,SCICTL1    ;enable RX, TX, clk
    splk    #0000h,SCICTL2    ;disable SCI interrupts
    splk    #0000h,SCIHBAUD   ;MSB |
    splk    #0082h,SCILBAUD   ;LSB |9600 Baud for sysclk 10MHz
    splk    #0022h,SCIPC2     ;I/O setting
    splk    #0033h,SCICTL1    ;end initialization

****************************************************************
* Enable Interrupts                                            *
****************************************************************
    ; Clear EV IFR and IMR regs
    ldp     #DP_EV
    splk    #07FFh,IFRA
    splk    #00FFh,IFRB
    splk    #000Fh,IFRC

    ; Enable T1 Underflow Int
    splk    #0200h,IMRA
    splk    #0000h,IMRB
    splk    #0000h,IMRC

    ;Set IMR for INT2 and INT4 and clear any Flags
    ;INT2 (PWM interrupt) is used for motor control synchronization
        ;INT4 () is used for capture 3
    ldp     #0h
    lacc    #0FFh
    sacl    IFR
    lacc    #0000010b
    sacl    IMR

    ldp     #i1               ;set the right control variable page
    clrc    INTM              ;enable all interrupts, now we may serve
                              ;interrupts
*** END Enable Interrupts


****************************************************************
* Virtual Menu                                                 *
****************************************************************
menu
```

```
    clrc    XF                ;default mode (will be saved as context)
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available ?
    bcnd    menu,ntc          ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0ffh             ;only 8 bits !!!
    ldp     #option           ;if yes, get it and store it in option
    sacl    option            ;now in option we have the option number
                              ;of the virtual menu
    sub     #031h             ;is it option 1 ?
    bcnd    notone,neq        ;if not branch to notone

*****************************
* Option 1): Speed reference
*****************************
navail11
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail11,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp         ;if yes, get it and store it in serialtmp
navail12
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail12,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8        ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp         ;add ACC with lower byte
    sacl    speedr            ;store it
    b       menu              ;return to the main polling cycle
*** END Option 1): speed reference

notone
    lacc    option
    sub     #032h             ;is it option 2 ?
    bcnd    nottwo,neq        ;if not branch to nottwo

*****************************
* Option 2): DAC update
*****************************
navail21
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail21,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #da1
    sacl    da1               ;if yes, get it and store it in da1
navail22
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail22,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #da1
    sacl    da2               ;if yes, get it and store it in da2
navail23
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail23,ntc      ;if not repeat the cycle (polling)
```

```
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #da1
    sacl    da3                ;if yes, get it and store it in da3
navail24
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail24,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #da1
    sacl    da4                ;if yes, get it and store it in da4
    b       menu               ;return to the main polling cycle
*** END Option 2): DAC update

nottwo
    lacc    option
    sub     #033h              ;is it option 2 ?
    bcnd    notthree,neq       ;if not branch to nottwo

*****************************
* Option 3): delay
*****************************
navail31
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail31,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail32
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
    bcnd    navail32,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8         ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp          ;add ACC with lower byte
    sacl    delay              ;store it
    b       menu               ;return to the main polling cycle
*** END Option 3): delay

notthree
    lacc    option
    sub     #034h              ;is it option 2 ?
    bcnd    notfour,neq        ;if not branch to nottwo

*****************************
* Option 4): Mingap
*****************************
navail41
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail41,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail42
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
    bcnd    navail42,ntc       ;if not repeat the cycle (polling)
```

```
    lacc    SCIRXBUF,8        ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp         ;add ACC with lower byte
    sacl    Mingap            ;store it
    b       menu              ;return to the main polling cycle
*** END Option 4): Mingap

notfour
    b    menu
```