Never stop thinking

XE166 Family

AP16165

XE166 Sensorless Fieldoriented Control

Application Note

V 1.0, 2009-07

Microcontrollers

Edition 2009-07

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XE164
Revision History: V 1.0 2009-07

Previous Version(s):

Page	Subjects (major changes since last revision)
–	This is the first release …

Trademarks

We Listen to Your Comments
Is there any information in this document that you feel is wrong, unclear or missing? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

1 Target Market and Motivation

The Field Oriented Control (FOC) method described in this document is aimed at the target market for cost sensitive drives, such as fans, pumps, compressors and geared motors.

One driving factor for a sensorless FOC on BLDC or PMSM is decreasing the system cost: sensors can generate problems in manufacturing and reliability, they require special wiring harnesses and connectors which increase the overall system cost.

The other driving factor is the significant improvement of energy efficiency of a motor. With FOC you can bring the efficiency up to 95%. This has a big impact on power consumption, motor dynamics, heat dissipation and noise. As an example fans and pumps in a lot of applications must operate very silently, otherwise the produced solid-borne sound distributes in water or air pipes within the whole building.

Usually this leads to concepts with sinusoidal currents in the motor coils. The zero-crossing technique for position detecion of BLDC motors does not work, because block-type currents generate audible noise.

Therefore the discussed FOC method solves the two obstacles at once:

- no need for a hall sensor or any other sensor type
- sinusoidal current shape for very efficient and quiet operation.

Figure 1 Target applications for Sensorless FOC

The limitation of the described methods is the single shunt current measurement which cannot guarantee a steady control under all circumstances. Thus the method is not suitable for high dynamic loads and positioning systems. Of course the algorithm can be adapted to 3-phase-shunt measurement which does not have this limitation.

2 Theory of Sensorless Field Oriented Control

This chapter is intended to give an overview about the theory of the sensorless Field Oriented Control (FOC) of permanent magnet syncronuous motors (PMSM). The control scheme which has been implemented at XC878 and XE164 is based on this theory. The implementation details are described in **Chapter 3**.

Before describing the FOC theory in detail, we begin some background information on motor control.

2.1 DC Motor Control

The DC motor consists of a permanent magnet which builds the stator and a coil mounted at the rotor. A mechanical commutator is needed in order to feed the current to the coils. This commutator will cause the current to be oriented according to the field of the permanent magnets at the stator.

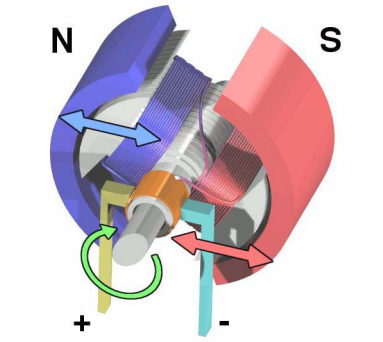


Figure 2 DC Motor

The DC Motor is controlled by the voltage at the rotor coils and the current is proportional to the torque of the motor. The speed of the motor depends on the voltage and the torque.

The nominal speed (ω_{nom}) and torque (m_{nom}) are defined in the following equations:

$$\omega_{nom} = \frac{v - i \cdot R}{K_c} \quad (1)$$

$$m_{nom} = K_c \cdot i \quad (2)$$

The time constants of speed and current control are very different. A control cascade structure for speed and current control can be utilized.

2.1.1 Open Loop Voltage Control

In an open loop voltage control, there is just a reference voltage which will cause the power inverter to generate a given voltage at the motor. The mechanical load influences the speed and the current of the DC motor.

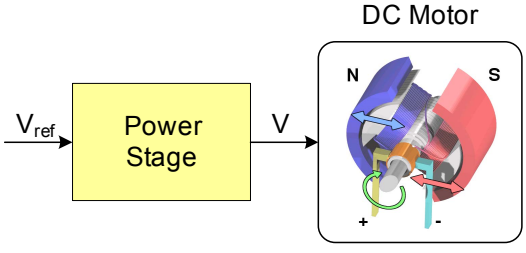


Figure 3 Open Loop Voltage Control

In steady state, the torque can be described as follows:

$$m_{stat} = K_c \cdot \frac{v - K_c \omega_{stat}}{R} \quad (3)$$

2.1.2 Closed Loop Torque Control

The torque control loop requires the measurement of the motor current with a current sensor; e.g. a shunt resistor. In general, the torque is proportional to the current.

$$m_m = K_c \cdot i \quad (4)$$

As a result, a PI controller can be used for current control. The actual current, measured by the current sensor will be controlled to converge to the reference current. This can be done only by changing the reference Voltage (V_{ref}) of the power stage. Please see following figure for details:

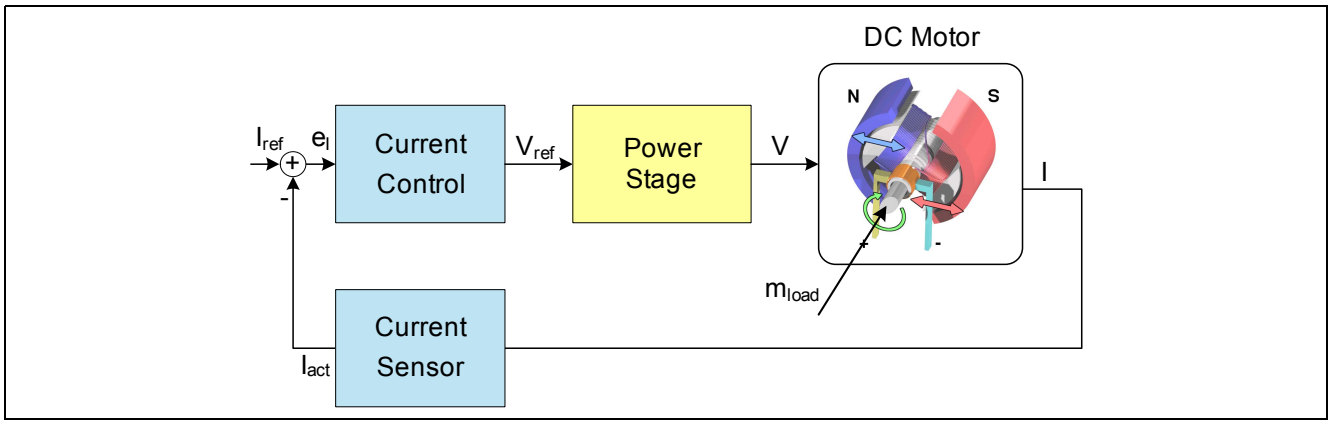


Figure 4 Closed Loop Torque Control

2.1.3 Closed Loop Speed and Torque Control

For a speed controlled DC motor, a cascaded control structure can be used, because the speed change requirement is much slower than the one for the current. The speed control requires a speed sensor; e.g. a tachometer and the current control requires a current sensor. The output of the speed control is the reference current for the current control. Usually a PI controller is used for speed and current control.

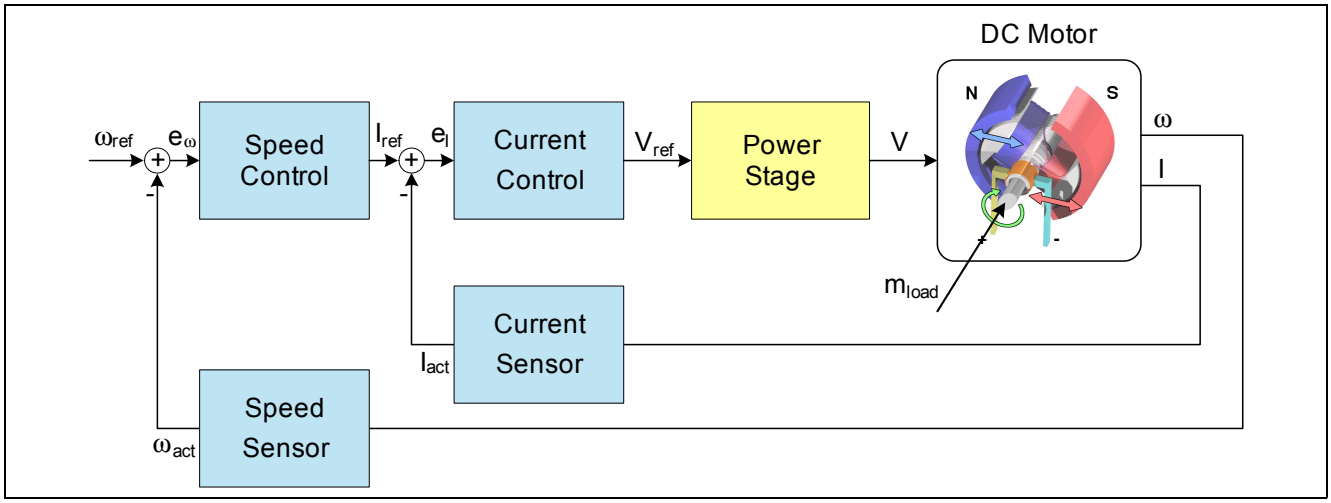


Figure 5 Cascaded Speed and Current Control

2.1.4 List of Equations

The mechanical and electrical equations of the DC motor are valid for the Permanent Magnet Synchronous Motor (PMSM) as well. In the following, a set of equations is listed, which will be used later in this application note.

Voltage Equation:

$$L \cdot \frac{di}{dt} + R \cdot i = v - v_{bemf} \tag{5}$$

Induced Voltage:

$$v_{bemf} = K_c \cdot \Psi \cdot \omega \tag{6}$$

Mechanical Friction:

$$m_{Friction} = r_{Friction} \cdot \omega \tag{7}$$

Mechanical Equation:

$$J \cdot \frac{d\omega}{dt} = m_m - m_{Load} - m_{Friction} \tag{8}$$

Angle:

$$\varphi = \int \omega dt \tag{9}$$

Speed in revolutions per minute (rpm):

$$n = 60 \cdot \frac{\omega}{2\pi} \tag{10}$$

2.2 Space Vector Modulation

In this section, the modulation of a three leg voltage source inverter is described. This type of inverter contains six MOSFETs or IGBTs which act as switches. The switches connected to the positive supply rail are called high side switches (hs) and the switches connected to the negative rail of the power supply are called low side switches (ls).

In the following figure, a block diagram of a three leg voltage source inverter can be seen:

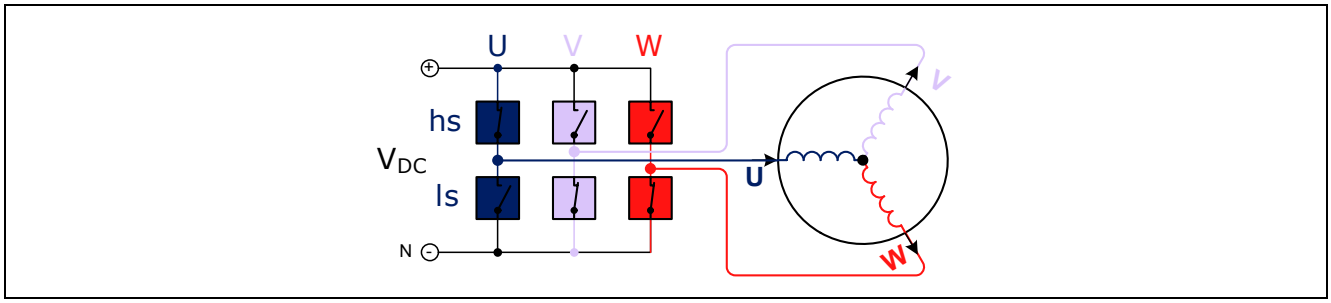


Figure 6 Three Leg Voltage Source Inverter

By switching the high side and low side switches on and off, there are eight states possible, which do not cause cross currents inside the leg itself, but allow a current flowing to and back from the motor.

State	U_{hs}	U_{ls}	V_{hs}	V_{ls}	W_{hs}	W_{ls}
000 (passive)	OFF	**ON**	OFF	**ON**	OFF	**ON**
100 (active)	**ON**	OFF	OFF	**ON**	OFF	**ON**
110 (active)	**ON**	OFF	**ON**	OFF	OFF	**ON**
010 (active)	OFF	**ON**	**ON**	OFF	OFF	**ON**
011 (active)	OFF	**ON**	**ON**	OFF	**ON**	OFF
001 (active)	OFF	**ON**	OFF	**ON**	**ON**	OFF
101 (active)	**ON**	OFF	OFF	**ON**	**ON**	OFF
111 (passive)	**ON**	OFF	**ON**	OFF	**ON**	OFF

The states can be seen as vectors in the space vector diagram of the coordinates U, V and W. There are six active and two passive vectors. The space vector diagram of a U-V-W system contains six sectors (A, B, C, D, E, F).

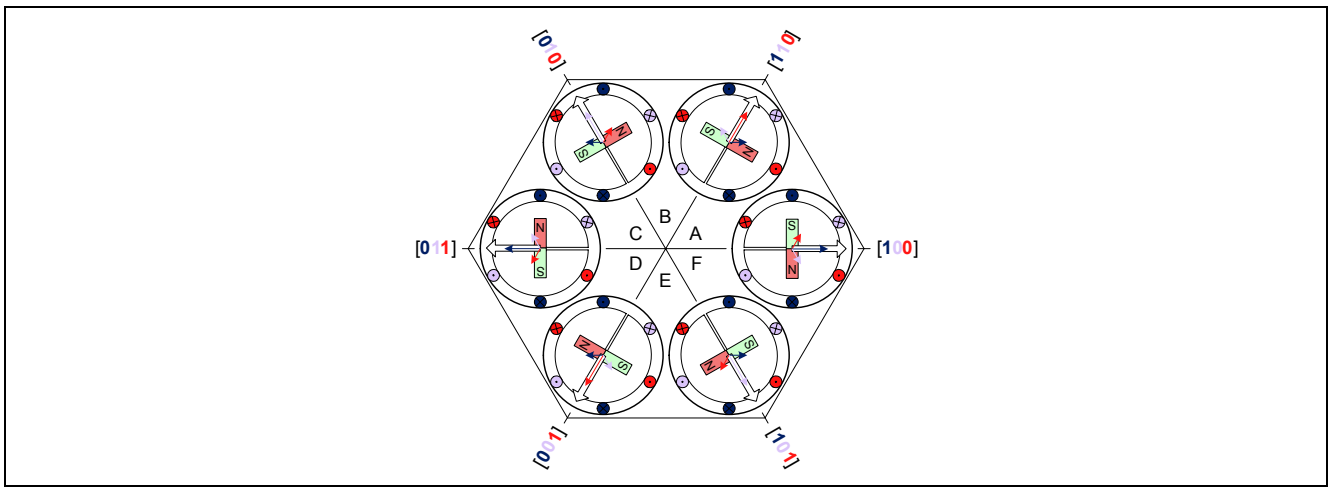


Figure 7 Space Vector Diagram of a U-V-W System

The magnetic field (flux) of a three phase PMSM can be represented in a U-V-W space vector diagram. In **Figure 7**, the flux is simplified by a magnetic dipole. Whereas the stator coils are fixed in their position, the flux rotates dependent on the sector of the hexagon.

In this case, the voltages at the three coils correspond to the values of U, V and W. The resulting vector corresponds to the flux of the stator coils.

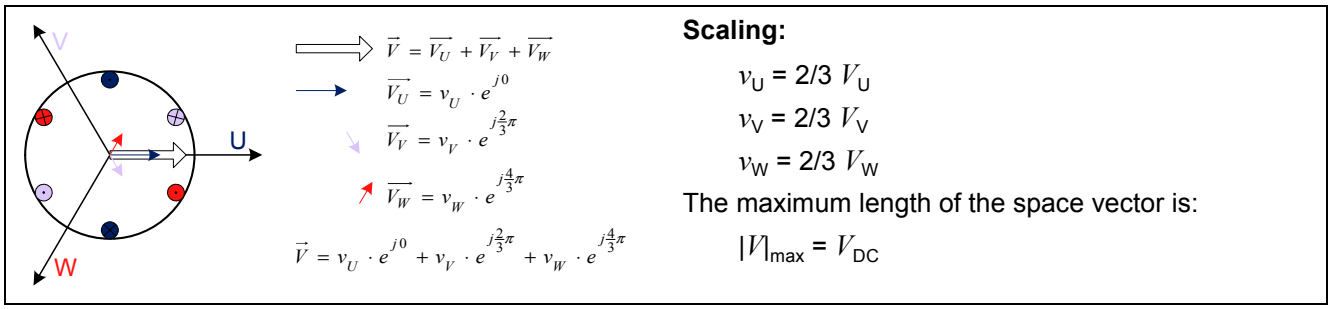


Figure 8 Space Vectors in Complex Numbers

2.2.1 Reference Vector

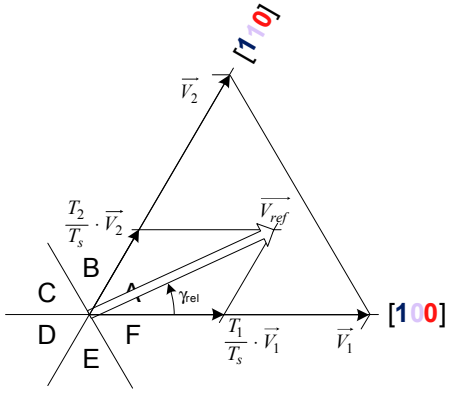
The reference vector V_{ref} represents the resulting magnetic stator field (flux). It is defined by the following equation:

$$\overrightarrow{V_{ref}} = V_{ref} \cdot e^{j\varphi} \tag{11}$$

It is rotating in space with the speed ω:

$$\omega = 2\pi \cdot f \tag{12}$$

The reference vector can be approximated by two active vectors (e.g. V1 and V2) and one zero vector (V0). The plane is dissected into six sectors and the angle φ is transformed into the relative angle γ_{rel}.



Sampling Period T_s:

$$T_s = T_1 + T_2 + T_0 \tag{13}$$

Volt-Second Balancing:

$$\overrightarrow{V_{ref}} \cdot T_s = \overrightarrow{V_1} \cdot T_1 + \overrightarrow{V_2} \cdot T_2 + \overrightarrow{V_0} \cdot T_0 \tag{14}$$

Figure 9 Reference Vector Approximation

Taking into account the scaling of the reference vector and expressing the complex numbers in polar coordinates, the following equations are valid for the PWM on-time of a space vector modulation:

$$T_1 = \frac{\sqrt{3} \cdot T_s \cdot V_{ref}}{V_{DC}} \sin\left(\frac{\pi}{3} - \gamma_{rel}\right) \tag{15}$$

$$T_2 = \frac{\sqrt{3} \cdot T_s \cdot V_{ref}}{V_{DC}} \sin(\gamma_{rel}) \tag{16}$$

$$T_0 = T_s - T_1 - T_2 \tag{17}$$

The design of the switching sequence depends on the application requirements. Usually the number of switchings is minimized, in order to reduce switching losses to a minimum.

The following figure shows a seven segment switching sequence:

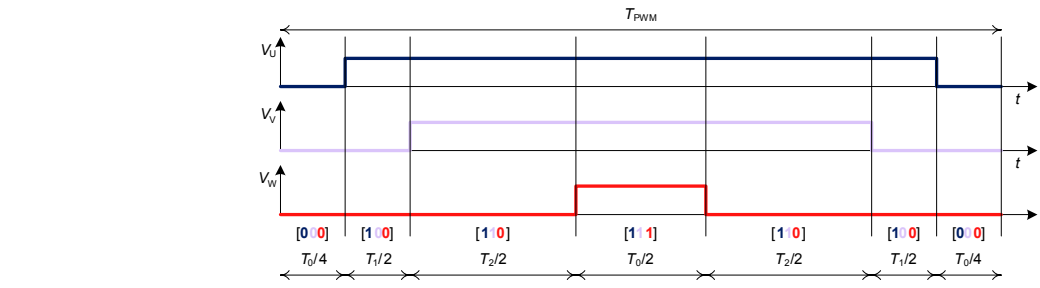


Figure 10 PWM Pattern of a Seven Segment Switching Sequence

The output voltage of each leg to neutral is shown in **Figure 11**.

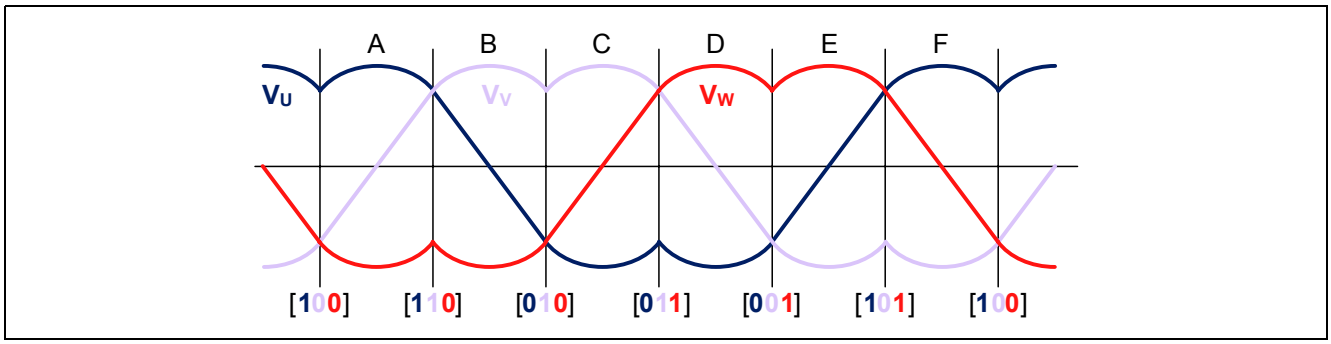


Figure 11 Output Voltage of Space Vector Modulation

The voltages V_{UV}, V_{UW} and V_{VW} at the motor are sinusoidal.

2.3 Phase Current Measurement

For many motor control schemes, the phase currents are required as input values. A cost efficient method requires only one shunt in the DC_{link}. Two phase currents can be reconstructed from the DC_{link} current (I_{DClink}) during one PWM period (T_{PWM}). The third phase current can be calculated by $I_U + I_V + I_W = 0$, but is redundant for the control.

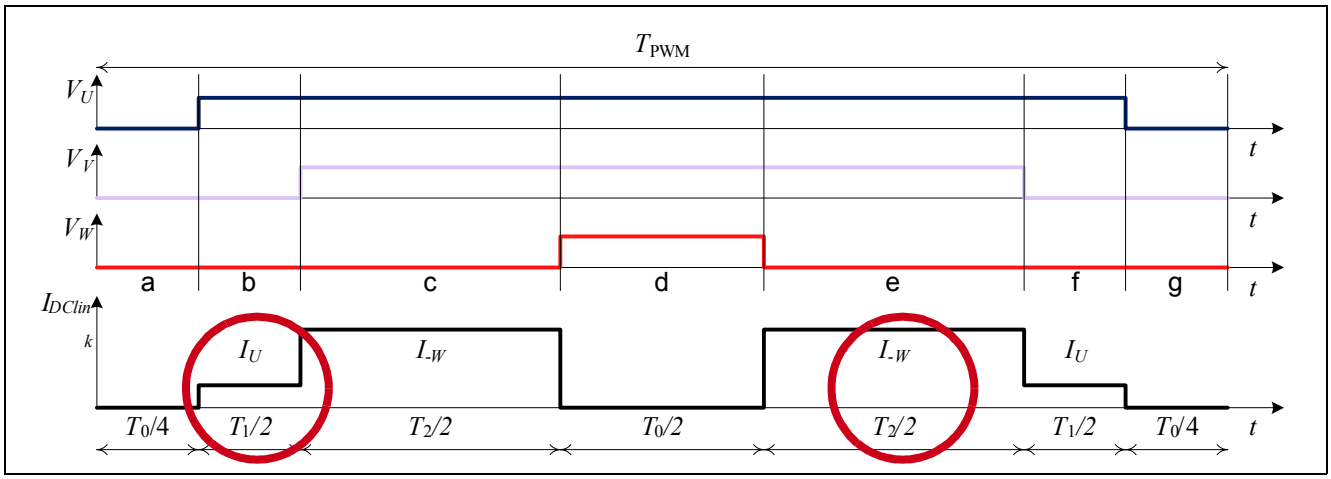


Figure 12 Phase Current Measurement within the PWM Pattern

In order to realize this method, the ADC trigger points must be adjusted according to the PWM pattern. Two different currents can be measured at PWM time T1 and T2. Depending on the actual sector, the measured currents have a different meaning. The following table shows these combinations and the calculation for phase current I_A and I_B.

Sector	A	B	C	D	E	F
T_1	U	V	V	W	W	U
T_2	-W	-W	-U	-U	-V	-V
$I_A = I_U$	U	(-W) - V	-(-U)	-(-U)	(-V) - W	U
$I_B = I_V$	(-W) - U	V	V	(-U) - W	-(-V)	-(-V)

2.3.1 Limitations of Phase Current Reconstruction

When T_1 or T_2 equals 0, only one phase current can be reconstructed. To avoid this situation, the PWM-times T_1 and T_2 have to be limited to a minimum value. This causes a ripple in the phase-phase voltage and the phase current as well. A very fast ADC is required in order to maximize the performance of phase current reconstruction.

The following figure shows the output voltage and phase voltage of a T_1-T_2-limited space vector modulation.

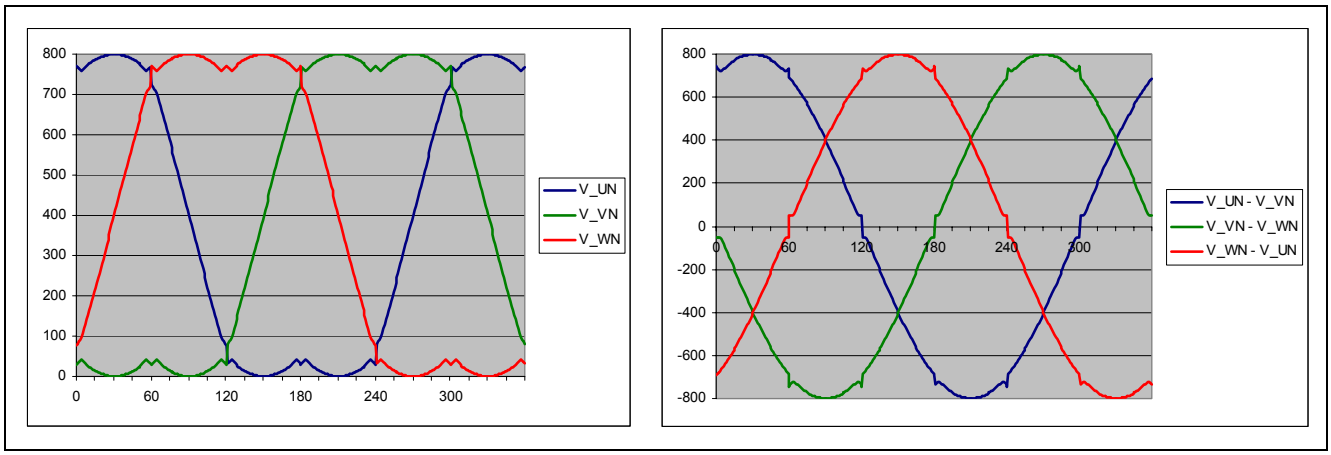


Figure 13 Distortion in Output and Motor Voltage

Although the output signal is slightly distorted, the most cost efficient method of phase current measuring is the reconstruction from the DC_{link} current via a single shunt. A very fast ADC is required to optimize the system performance. A direct trigger from the PWM unit to the ADC reduces CPU load significantly.

2.4 FOC Calculations

FOC is a method to generate a three phase sinusoidal signal, which can easily be controlled in frequency and amplitude in order to minimize the current and therefore maximize the efficiency. The essential idea is the transform of three phase signals into two rotor-fix signals and vice versa. In the rotor-fix reference frame, the currents are stationary and easy to control. The inverse vector rotation causes the contolled voltages to rotate.

2.4.1 Stationary and Rotating Reference Frames

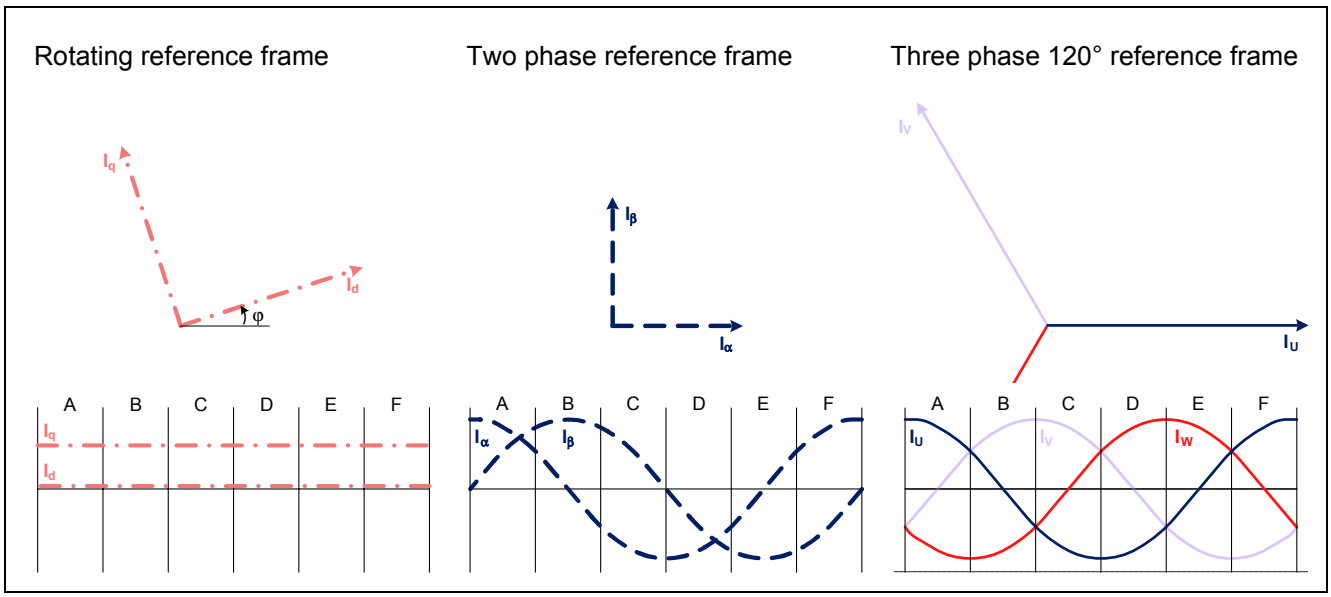


Figure 14 Stationary and Rotating Reference Frames

The transform from the three phase system to the two phase system is called the Clarke transform. The one from Two phase system to the rotating system is called the Park transform.

These transfroms are expressed in the following equations.

Clarke Transform

$$i_\alpha = i_U \tag{18}$$

$$i_\beta = \frac{1}{\sqrt{3}} \cdot (i_U + 2i_V) \tag{19}$$

$$i_U + i_V + i_W = 0 \tag{20}$$

Park Transform

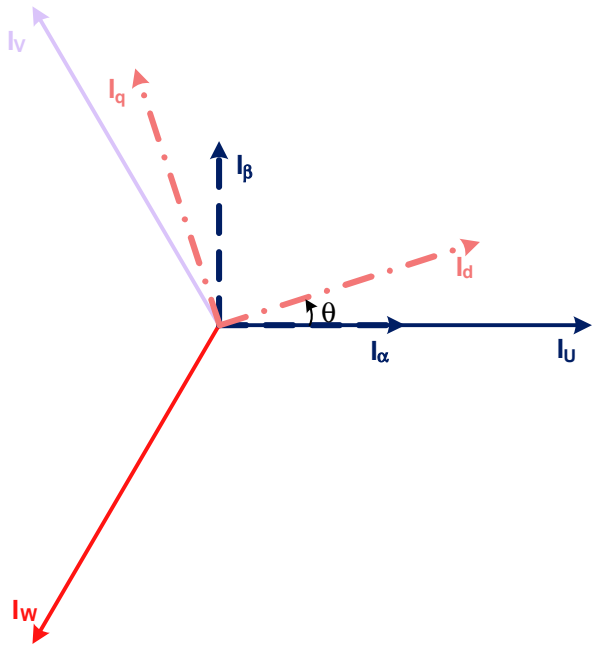
$$i_d = i_\alpha \cos\varphi + i_\beta \sin\varphi \tag{21}$$

$$i_q = -i_\alpha \sin\varphi + i_\beta \cos\varphi \tag{22}$$

Inverse Park Transform

$$i_\alpha = i_d \cos\varphi - i_q \sin\varphi \quad (23)$$

$$i_\beta = i_d \sin\varphi + i_q \cos\varphi \quad (24)$$



Summary Transformations

- Clarke transformation is used to remove the redundancy of the 3 phase system
- Park transform provides a rotating d-q-reference frame
- The currents Id and Iq are stationary and easy to control
- The inverse Park transform makes the controlled voltages rotating

Figure 15 Summary of Transformations

2.4.2 Sensorless FOC

The costs for an encoder or other position sensor can be saved, when the exact position of the rotor is not required. The motor's back EMF is utilized in order to calculate the rotation angle and rotor position. The back EMF is calculated in the flux estimator, which is based on the voltage model of the system in the two phase reference frame. A single shunt is enough to reconstruct the phase currents.

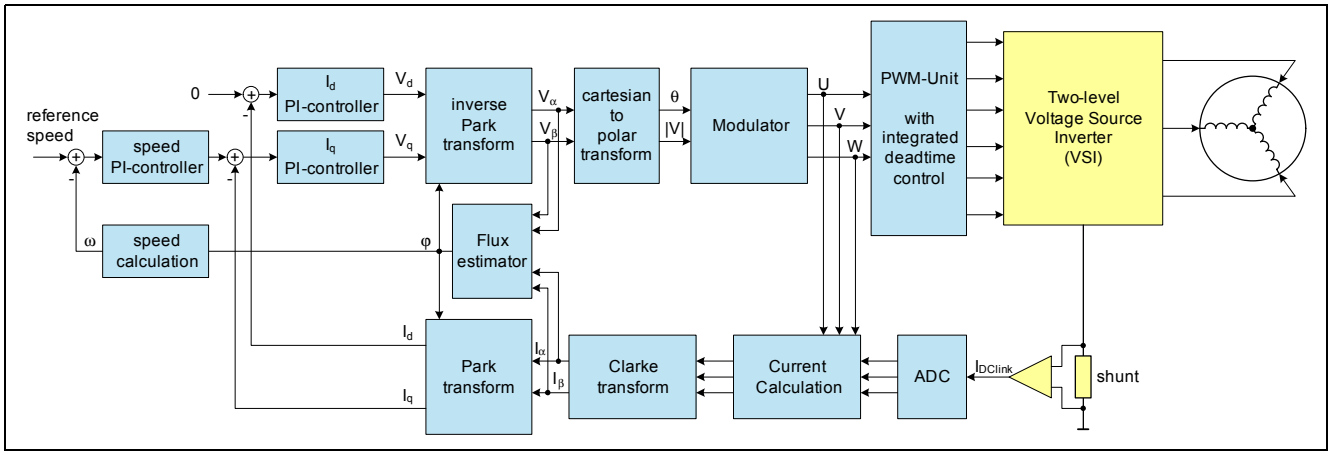
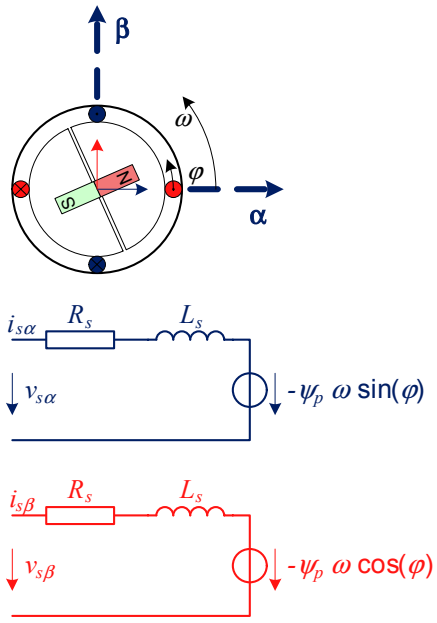


Figure 16 Block Diagram of Sensorless Field Oriented Control

Figure 16 shows the block diagram of the sensorless FOC algorithm. The flux estimator's input signals are taken from the orthogonal two phase stator system with the index α and β. The output signal represents the rotor angle.

The motor signals in the two phase stator system (α-β-system) are equivalent to the three phase system. As a result, an ideal two phase motor can be assumed in the α-β-system. Then there are only two equations to calculate. The following figure shows the voltage equations of the motor in the α-β-system.



Flux of stator ψ_s:

$$\psi_{s\alpha} = L \cdot i_{s\alpha} + \psi_{p\alpha} = L \cdot i_{s\alpha} + \psi_p \cos(\varphi) \quad (25)$$

$$\psi_{s\beta} = L \cdot i_{s\beta} + \psi_{p\beta} = L \cdot i_{s\beta} + \psi_p \sin(\varphi) \quad (26)$$

Stator Voltage Equation:

$$v_{s\alpha} = R \cdot i_{s\alpha} + \frac{d\psi_{s\alpha}}{dt} = R \cdot i_{s\alpha} + L \cdot \frac{di_{s\alpha}}{dt} - \psi_p \frac{d\varphi}{dt} \sin(\varphi) \quad (27)$$

$$v_{s\beta} = R \cdot i_{s\beta} + \frac{d\psi_{s\beta}}{dt} = R \cdot i_{s\beta} + L \cdot \frac{di_{s\beta}}{dt} + \psi_p \frac{d\varphi}{dt} \cos(\varphi) \quad (28)$$

Figure 17 Voltage Equations of Ideal 2-phase Motor

The flux of the stator (**Equation (25)**, **Equation (26)**) contain the mutual inductance $L \cdot \vec{i}_s$ and the flux of the rotor with the permanent magnet $\vec{\psi}_p$.

The stator voltage equations (**Equation (27)**, **Equation (28)**) contains the resistance of the coil $R \cdot \vec{i}_s$ and the derivative of the flux of the stator (Faraday's Law).

Integrating the voltage equations, we calculate the stator flux as follows:

$$\psi_{s\alpha} = \int (v_{s\alpha} - R \cdot i_{s\alpha}) dt \quad (29)$$

$$\psi_{s\beta} = \int (v_{s\beta} - R \cdot i_{s\beta}) dt \quad (30)$$

The flux of the rotor and hereby the orientation (angle φ) of the permanent magnet of the rotor is calculated by insertion of **Equation (29)** in **Equation (25)** and **Equation (30)** in **Equation (26)**.

$$\psi_{p\alpha} = \psi_{s\alpha} - L \cdot i_{s\alpha} = \int (v_{s\alpha} - R \cdot i_{s\alpha}) dt - L \cdot i_{s\alpha} \quad (31)$$

$$\psi_{p\beta} = \psi_{s\beta} - L \cdot i_{s\beta} = \int (v_{s\beta} - R \cdot i_{s\beta}) dt - L \cdot i_{s\beta} \quad (32)$$

$$\varphi = \operatorname{atan}\left(\frac{\psi_{p\beta}}{\psi_{p\alpha}}\right) \quad (33)$$

Finally the position of the rotor can be calculated by knowing the resistance R and inductance L of the motor. The stator voltage v_s is derived from the algorithm, and the current i_s needs to be measured in real-time.

3 Implementation of a Sensorless FOC using the XE164

Field Oriented Control or vector control is a math technique for controlling brushless dc and ac induction motors that reduces motor size, cost and power consumption and is a cousin of flux vector control.

This Chapter describes the implementation of a sensorless Field Oriented Control using the Infineon XE164 microcontroller. The XE164 Series is a new 100-pin device family of the popular C166 microcontroller architecture. Based on the enhanced C166S V2 architecture, it outperforms existing 16-bit solutions.

The 3 PWM units, several timer and two very fast and powerful ADC's make the XE164 a perfect fit for a variety of motor control applications.

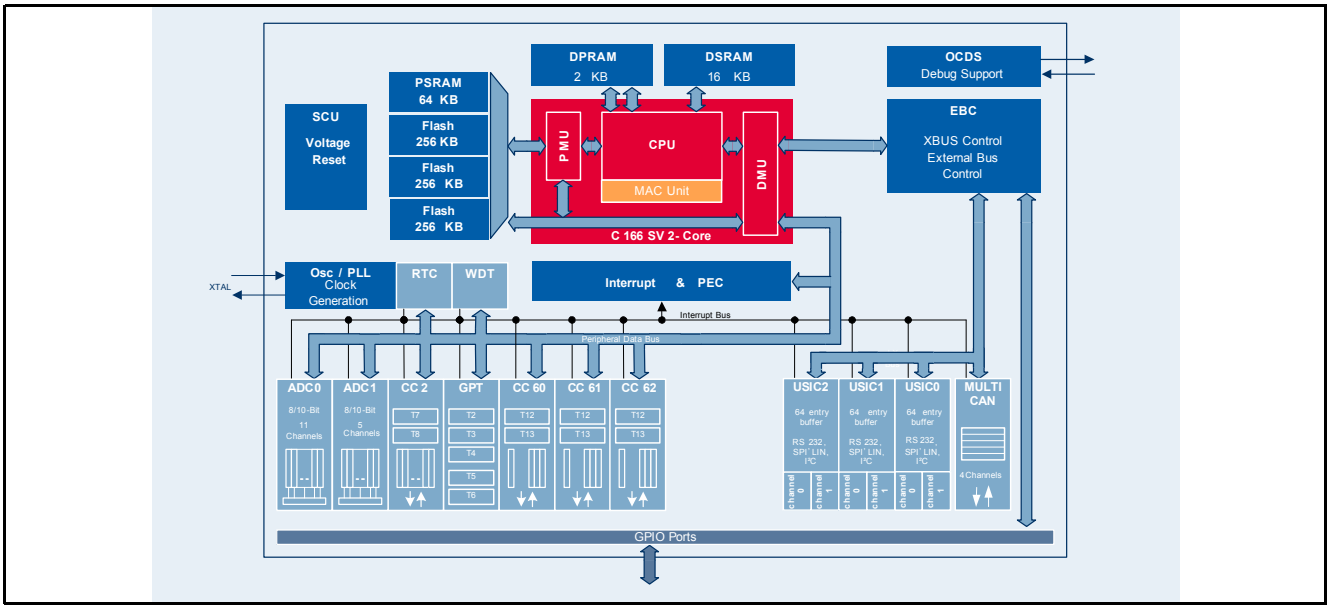


Figure 18 Block Diagram XE164

This application note makes reference to Infineon's FOC Drive Application Kit (ordering code: KIT_AK_FOCDRIVE_V1). This kit comes with the following hardware:

- DriveCard XC878
- DriveCard XE164
- Low Voltage Inverter (23V-55V, 7.5A)
- PMSM motor (15W)
- DriveMonitor USB stick
- Power supply (24V)

Figure 19 FOC Drive Application Kit

The sections which follow describe the FOC implementation on the 16-bit device XE164. The project setup is available for the Tasking classic toolchains.The project is DAvE compliant; i.e. the peripheral settings can be changed or added using DAvE code generator and recompiled afterwards.

A MicroSoft Excel sheet can be used for scaling and motor parameter adaptation. The Excel sheet generates the contents for a header file.

The DriveMonitor is used for real-time monitorying of those application variables that are of interest. The DriveMonitor also offers a control interface to the application. Data is exchanged via the CAN bus.

*Note: The descriptions in the sections which follow are intended for the advanced user; i.e. some basic knowledge of the Infineon tool-chain is a prerequisite. Several application notes are available to aid understanding (see **Chapter 6**), and 'hands-on' training is also offered. Less advanced users are recommended to begin with the DaveDrive auto-code generator.*

3.1 System Overview

The Field Oriented Control Application Software uses the Drive Card XE164.

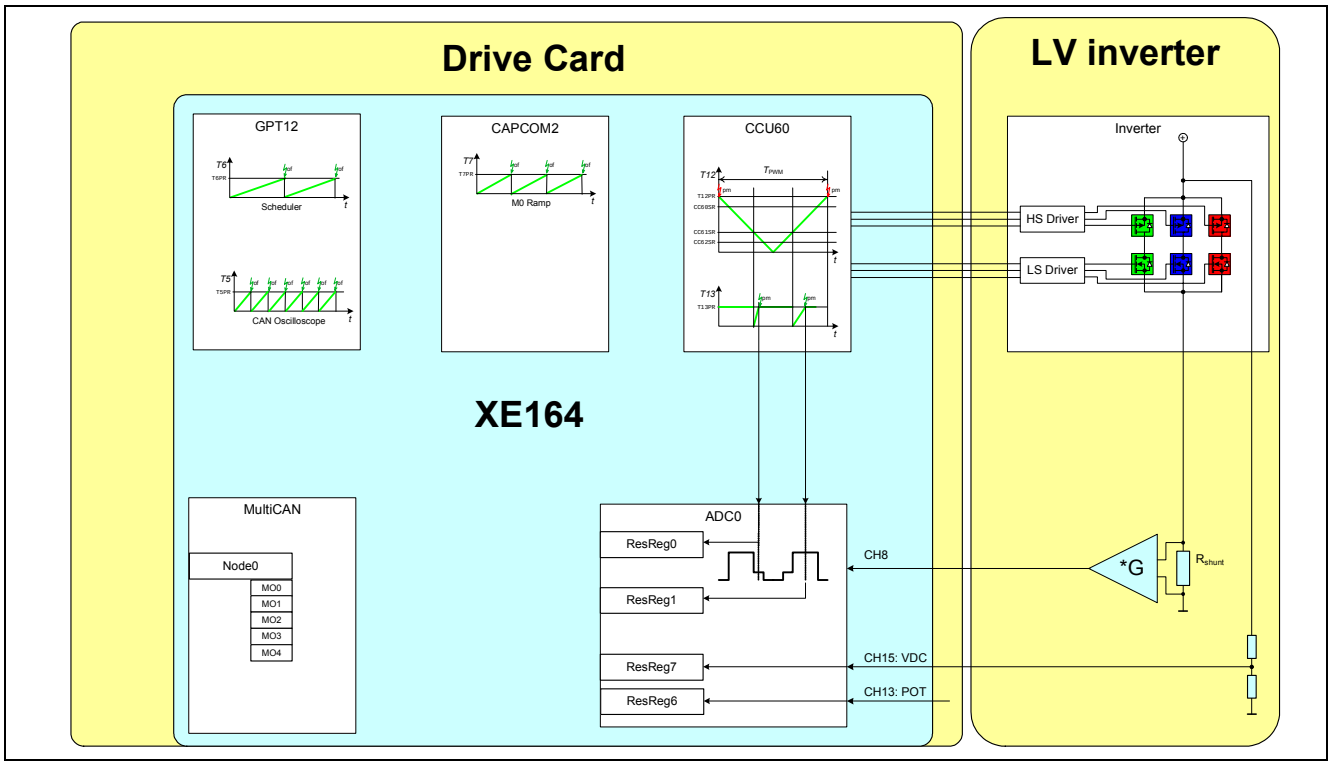


Figure 20 System Block

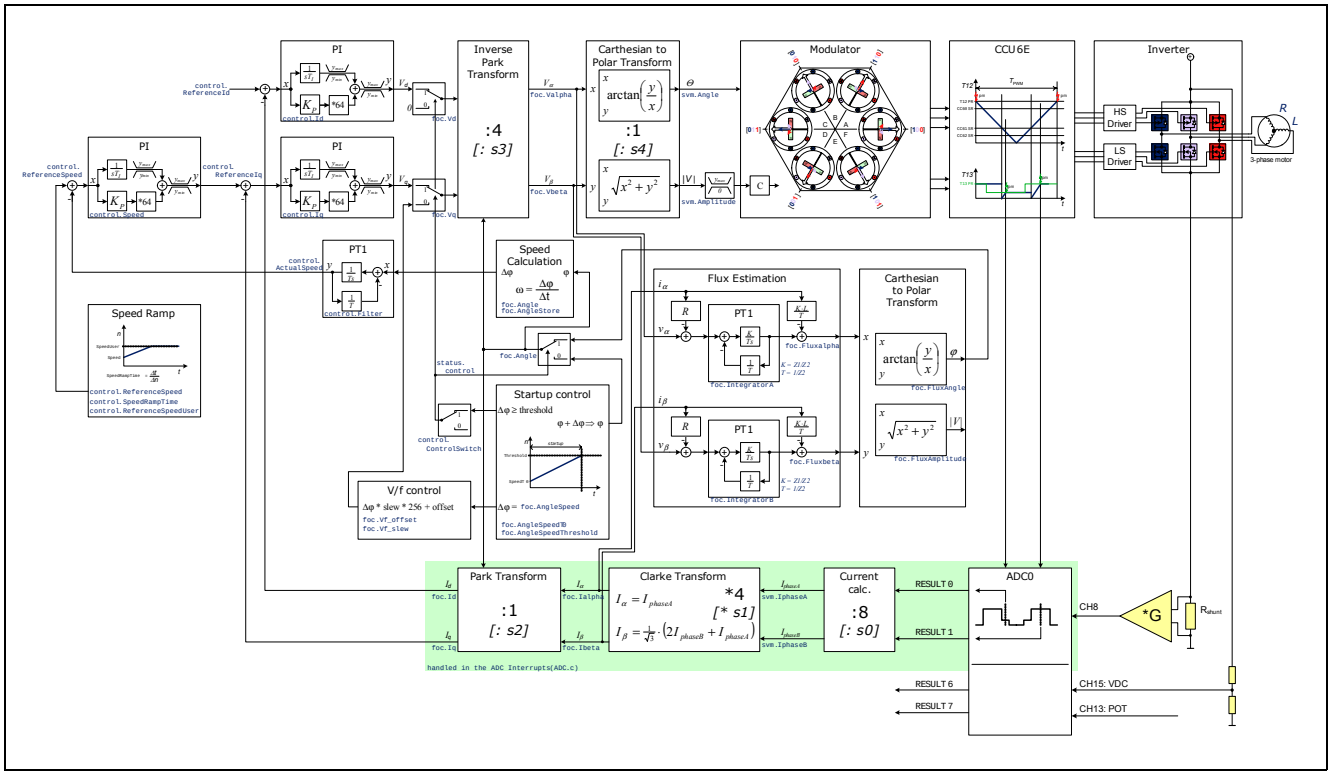


Figure 21 XE166 Sensorless FOC Block Diagram

3.2 GPT12 Timer 5 - Oscilloscope Data

Timer 5 of the GPT module send the Oscilloscope data every 0.5 millisecond via an interrupt. One message object is used to send the complete data. The structure is shown in **Chapter 4.1.9**. The Timer 5 interrupt uses the lowest priority in the system. The transfer rate of the data can be adapted to the application needs.

3.3 GPT12 Timer 6 - Statemachine and Scheduler

The GPT Timer 6 is configured as a Timer and generates a interrupt every millisecond, which executes the scheduler. The scheduler can be divided into a CAN handler and a statemachine part which handles the whole FOC application Software. For the FOC implementation a statemachines for Motor 0 is integrated. The scheduler handles the CAN commands and switches or reset's Statemachine states. Some Statemachine states are triggered from the CAN handler part of the Scheduler, while some are triggered/switched by other events or mechanisms. **Figure 22** show the structure of the scheduler. The **Figure 23** show the Statemachine for a single Motor application.

After handling a new CAN command, information is sent to the drive monitor, such as the motor states and some control values such as Vdc, the temperature on the inverter.

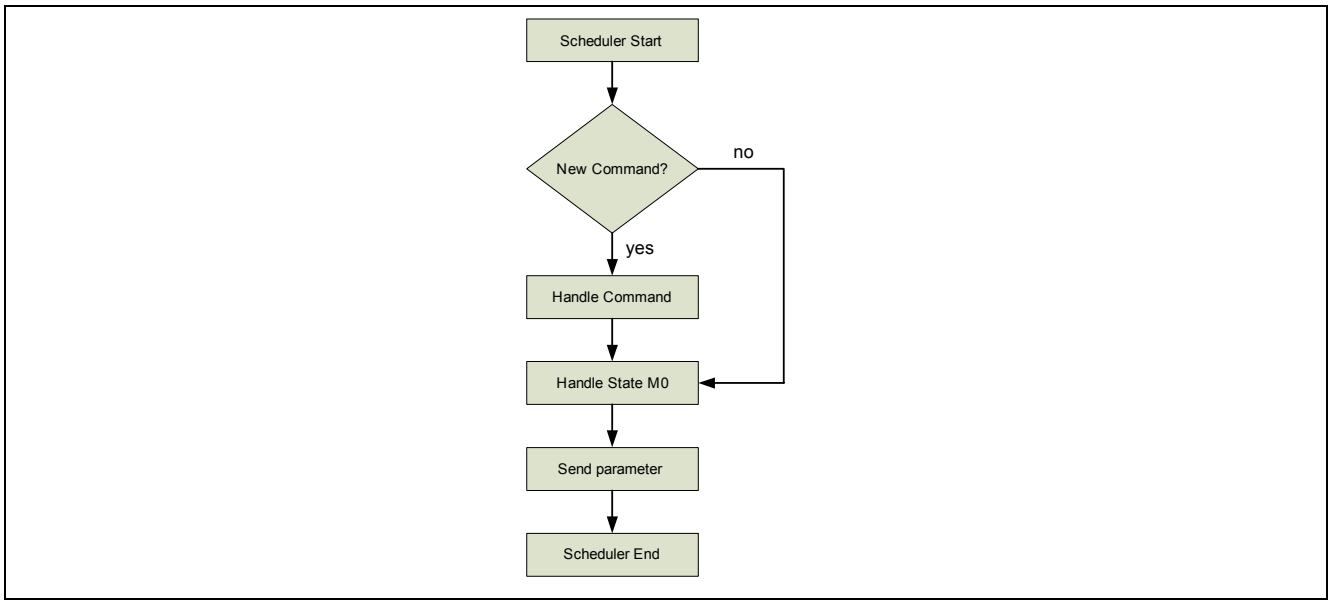


Figure 22 Scheduler Sensorless FOC

The FOC Software handles 8 different states for the Motor in a seperate global 16 bit word (STATE_M0). The dedicated states are listed for Motor 0 in the register view below.

STATE_M0 **Reset value: 0001_H**

Short Description

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-			**M0 States**			
-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw

The bits of the variable are described in a table that follows.

Field	Bits	Type	Description
M0 States	[0:3]	rw	**Statemachine States** 0001 STATEIDLE_M0(Default) 0010 STATESTARTUP_M0 0011 STATESTOP_M0 0100 STATEFOC_M0 0101 STATERAMPDN_M0 0101 STATEEMCY_M0 0111 STATEBOOTST_M0 1000 STATEPOTI_M0 ... ... *Note: Some states are triggered by the Drive Monitor, while other states are entered automatically and can not be influenced by the Communication Software.*

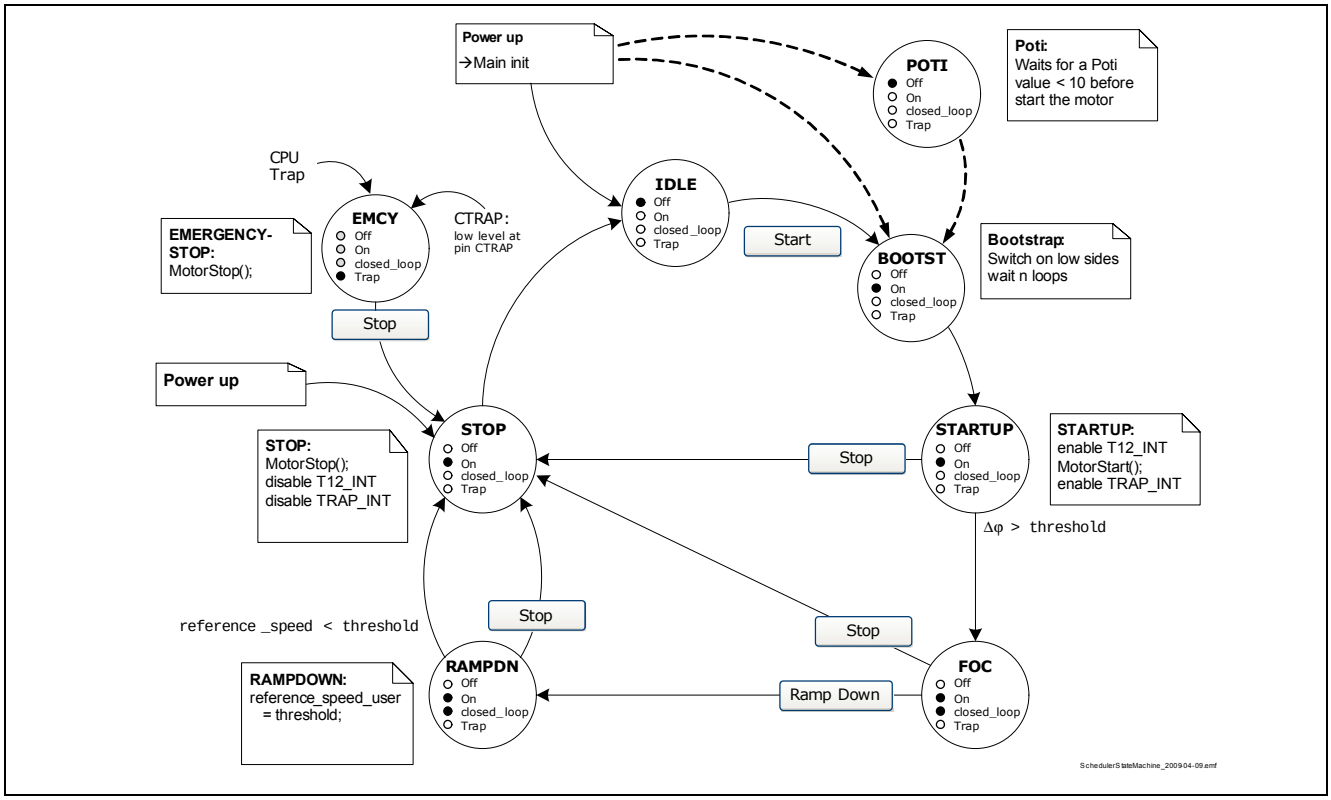


Figure 23 Statemachine for Single Motor FOC

On power-up the Statemachine normally enters the IDLE state. In IDLE state the Motor is stopped by Software and the CCU6 interrupts are disabled. To start the Motor, the BOOTST state is entered. BOOTST can also be entered directly after power-up, instead of IDLE. Whether IDLE or BOOTST mode is entered on power-up is defined by the DEFAULT_STATE setting in the define.h file.

```
#define DEFAULT_STATE        STATEBOOTST_M0
```

In the Bootsrap state the low side transistors will be switched on for the dedicated number of the ramp-up timer interrupts.

```
#define BOOTSTRAP_M0        2
```

After this the STARTUP state is entered, which means the Motor will start to ramp-up. The FOC state is entered after a selected speed of the Motor when the sensor less FOC algorithm starts to handle the Motor. A CTRAP can cause an Emergency Stop as well a CPU TRAP. For more information about the Emergency handling refer to **Chapter 3.3.4**.

3.3.1 IDLE State

IDLE is the default state entered on system power-up. In this state the CPU waits for new commands and for a trigger to start the motor. Status flags will be reset and the values for the CAPCOM2 ramp timer 7 can be changed. To change the timer value, DAvE generated functions are used.

```
case STATEIDLE_M0:

      status0.on = 0;
      status0.control = 0;
      status0.off = 1;

      CC2_vStopTmr(CC2_TIMER_7);
      CC2_vLoadTmr(CC2_TIMER_7,control0.SpeedRampSlew);
      CC2_T7REL = control0.SpeedRampSlew;
      CC2_vStartTmr(CC2_TIMER_7);
      break;
```

3.3.2 STOP State

The STOP state can be entered with a command from the DriveMonitor, or after the ramp-down state is finished. The STOP function disables the ENABLE signal for the driver IC, and also sets the status flags and FOC variables to their defaults. The CAPCOM60 Timer 12 is stopped and the interrupt nodes for Timer 12 and the Emergency interrup are disabled.

3.3.3 Ramp-down State

The Ramp-down state can be entered by a command direct from the DriveMonitor. The user reference speed is set to the REFSPEED_STARTUP_M0. Once the reference speed matches the user reference speed, the state machine automatically enters the STOP state.

The following shows a code extract from the scheduler function:

```
case STATERAMPDN_M0:
      //This ramps down the Motor 0 -> the switch to idle will be
      done on the CCU2 T7 interrupt
      status0.on = 0;
      control0.ReferenceSpeedUser = REFSPEED_STARTUP_M0;
   break;
```

3.3.4 Emergency Stop State

The Emergency State can be entered from two different event triggers:

- CCU60/CTRAP - this switches off the power devices if the trap input becomes active.
- CPU TRAPS - Trap functions are activated in response to special conditions that occur during the execution of instructions.

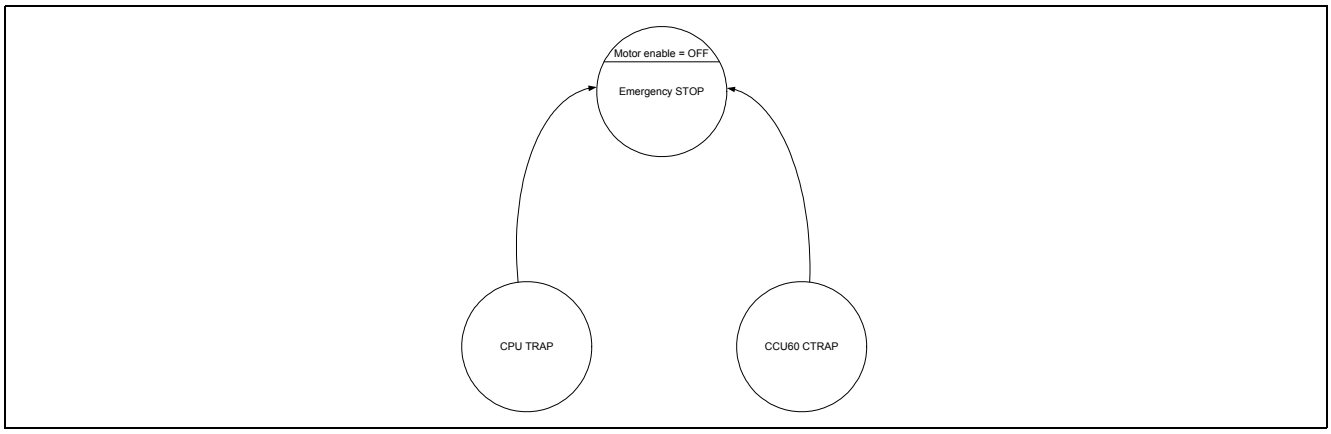


Figure 24 Emergency Stop

There are nine hardware trap functions for the XE164 divided into 2 classes. Class A traps are:

- System Request 0 (SR0)
- Stack Overflow
- Stack Underflow trap
- Software Break.

The second Class B traps are:

- System Request 1 (SR1)
- Undefined Opcode
- Memory Access Error
- Protection Fault
- Illegal Word Operand Access.

All this TRAP call the same Emergency_Stop_M0() function.

An Emergency State can only be left with a STOP command from the Drive Monitor.

3.4 CAPCOM2 Timer 7 - Motor Ramp function

The CAPCOM2 Timer 7 is base for all ramp-up and ramp-down functions of the sensorless FOC software. The speed ramp is dependent on three Excel parameters in the define.h file.

```
REF_RAMPUP_T7_M0
T7_PRESCALER
RAMP_STEP_M0
```

The REF_RAMPUP_T7_M0 defines the Timer ticks and T7_PRESCALER selects a pre scaler for Timer 7. The Ramp Step is also defined. This influences the slope of the ramp behaviour.

The Timer 7 interrupt handles the following functions:

- Bootstrapping time
- Ramp-up in V/f
- Ramp-up in closed loop FOC

– Ramp in case of user reference speed changes during run time
– Ramp-down in closed loop FOC

3.4.1 Bootstrapping

A MOSFET/IGBT is a voltage controlled device which, in theory, will not have any gate current. This makes it possible to utilize the charge inside the capacitor for control purposes. However, eventually the capacitor will lose its charge (due to parasitic gate current), so this scheme is only used where there is a steady pulse present. This is because the pulsing action allows for the capacitor to discharge (at least partially if not completely). Most control schemes that use a bootstrap capacitor force the high side driver (N-MOSFET) off for a minimum time to allow for the capacitor to refill. This means that the duty cycle will always need to be less than 100% to accommodate for the parasitic discharge unless the leakage is accommodated for in another manner.

The LV inverter in use also needs a re-fill of the bootstrap capacitors. This is achieved in the Bootstrap state by using the CCU60 to set the compare values for all 3 channels to the Timer 12 period value +1. This will switch on the low side MOSFET for a dedicated time. The actual time is calculated from the number of Capcom2 Timer 7 overruns that is stored in the variable BOOTSTRAP_M0, in the define.h file.

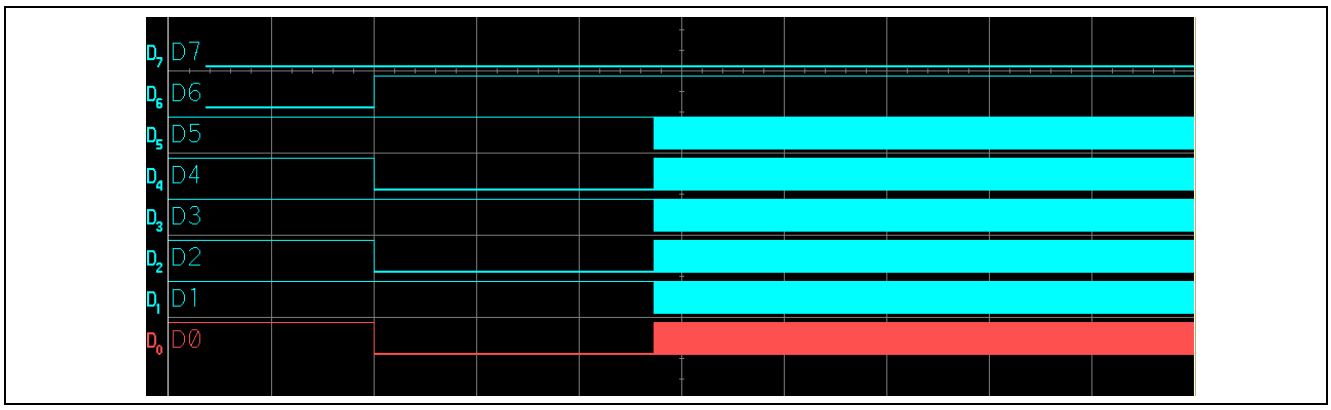


Figure 25 Bootstrap of the LV inverter (D6 Enable Signal; D0,D2,D4 low side; D1,D3,D5 high side)

3.4.2 Motor Ramp-up

The startup mechanism to turn the motor is quite complex and can be divided into two parts. First the Motor will start to turn by a V/f to a dedicated user defined speed. From this point on the Field Oriented Control will handle the Motor.

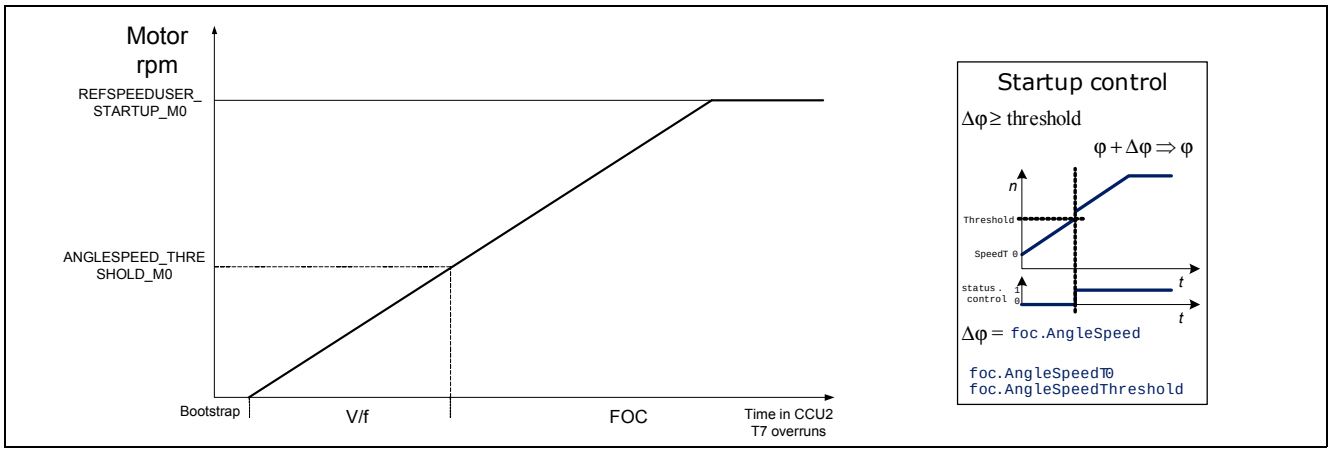


Figure 26 Motor Ramp-Up

A #define in the define.h file is used to switch selection between the V/f control and FOC.

```
#define CONTROL_SWITCH_M0  1
```

If this control switch is set to 0 the motor ramps-up to the angle speed threshold in the V/f control. The motor will run with that speed without switching to the FOC algorithm. This method can be use to check the calculated angle of the motor with the angle used in the V/f control.

3.4.3 Running state

After the motor ramps up to the defined threshold speed, the user can change the User reference speed with the DriveMonitor. In case the define SPEED_POTI is active in the define.h file, the Poti of the DriveCard is used to change the speed of the Motor.

3.4.4 Motor RAMP-down

After the scheduler receives the command for ramp-down (**Chapter 3.3.3**) , the reference speed will be ramped-down to the REFSPEED_STARTUP_M0. Once the speed ramp-down is complete, the scheduler state is automatically changed to the STOP state.

The following shows a code extract from the CAMCOM2 Timer 7 interrupt function:

```
//*****************************************************************************
// controlling the speed of the Motor 0 if switched to control mode
//*****************************************************************************
if (STATE_M0 == STATERAMPDN_M0)
{
if(control0.ReferenceSpeedUser > control0.ReferenceSpeed) control0.ReferenceSpeed += RAMP_STEP_M0;
if(control0.ReferenceSpeedUser < control0.ReferenceSpeed) control0.ReferenceSpeed -= RAMP_STEP_M0;
if(control0.ReferenceSpeedUser == control0.ReferenceSpeed)
          {
          STATE_M0 = STATESTOP_M0;
          }
}
```

3.5 CAPCOM60 Timer 12 and Timer 13

The PWM is generated with the CCU60 timer unit. Both timers T12 and T13 are used. T12 generates the signals for the Space Vector Modulation (SVM). T13 generates the trigger events for the ADC0. T13 runs in single shot mode and starts automatically on one defined compare match of one of the 3 compare channels. The generated trigger starts the conversion of the ADC0 channel which generates an interrupt after the channel is converted. Details of the ADC0 handling can be found in **Chapter 3.6.1**.

The passive level selection of the CCU60 compare output signals fits to the Infineon 3 phase gate driver 6ED003L06-F. To avoid spikes and dangerous transitions at startup the CCU60 has a special feature which is used for the initialization and for the Start/Stop/Emergency states. Therefore the CCU60's trap state is forced by software. This is done at the very beginning of CC6_vInit() by setting the flag TRPF (note: DAvE does not set this flag). By this it is ensured that the output signals always show the defined passive level during initialization until the flag TRPF is cleared.

There are two application specific constants which have to be defined before compilation in defines.h. They will be initialized accordingly:

```
T12PERIODE_M0
DEADTIME_M0
```

The T12 period needs to be setup according to the PWM frequency. The deadtime counter needs to be setup according to the Hardware setup. DAvE settings will be overwritten in this instance.

3.5.1 CAPCOM60 Timer 12 interrupt

The FOC algorithm can be explained by **Figure 21**. The shown building blocks are exactly implemented, mainly in the file FOC_Functions.c and FOC_Functions.h for inline functions. The FOC calculation is divided into two parts. The first part is located in the ADC0 Result register interrupt and executes the following calculations:

- ADC result register control
- current calculation and phase current extraction
- clark transformation

The second, larger part is located in the Timer 12 period match of the Interrupt Service Routine, and executes the following calculations:

- flux estimation
- park transformation
- speed calculation and speed control
- current control
- inverse park transformation
- cartesian to polar transformation
- space vector modulation

The Timer 12 one match of the Interrupt Service triggers the shadow transfer for the compare channels.

3.6 ADC0 Result Handling

The current implementation uses several channels of the ADC0. The ADC1 module is reserved for future implementations. Nevertheless a customer specific setup of the free and used ADC channels can be easily created within the DAvE project.

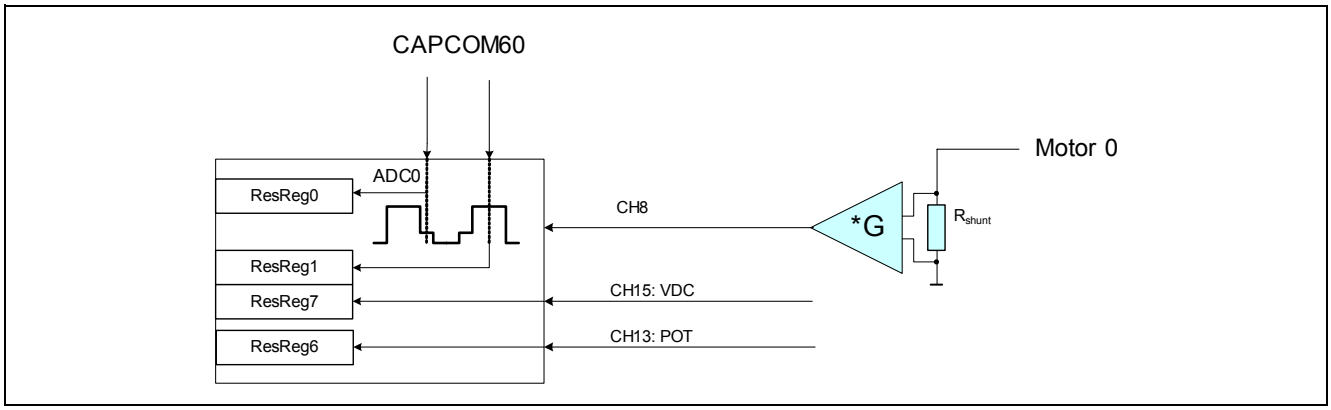


Figure 27 ADC0 Structure for Single Motor FOC

3.6.1 DC Link Shunt Measurements

The sensorless Field Oriented Control works with a single Shunt measurement. It is possible to economize one current sensor when the current measurement is not performed in the phases but in the dc-link. To effect this phase current acquisition through the dc link current it is necessary to know the actual switching pattern.

In this way, during one PWM, two times, two different phase currents can be measured through a dc-link current measurement. The third phase current can be calculated by the equation iU+iV+iW=0. (see **Chapter 2.3** for details).

In order to get a maximum time between the current conversions with the ADC, the measurement will be started on the edge of the phase with the middle “on time” (**Figure 28**).

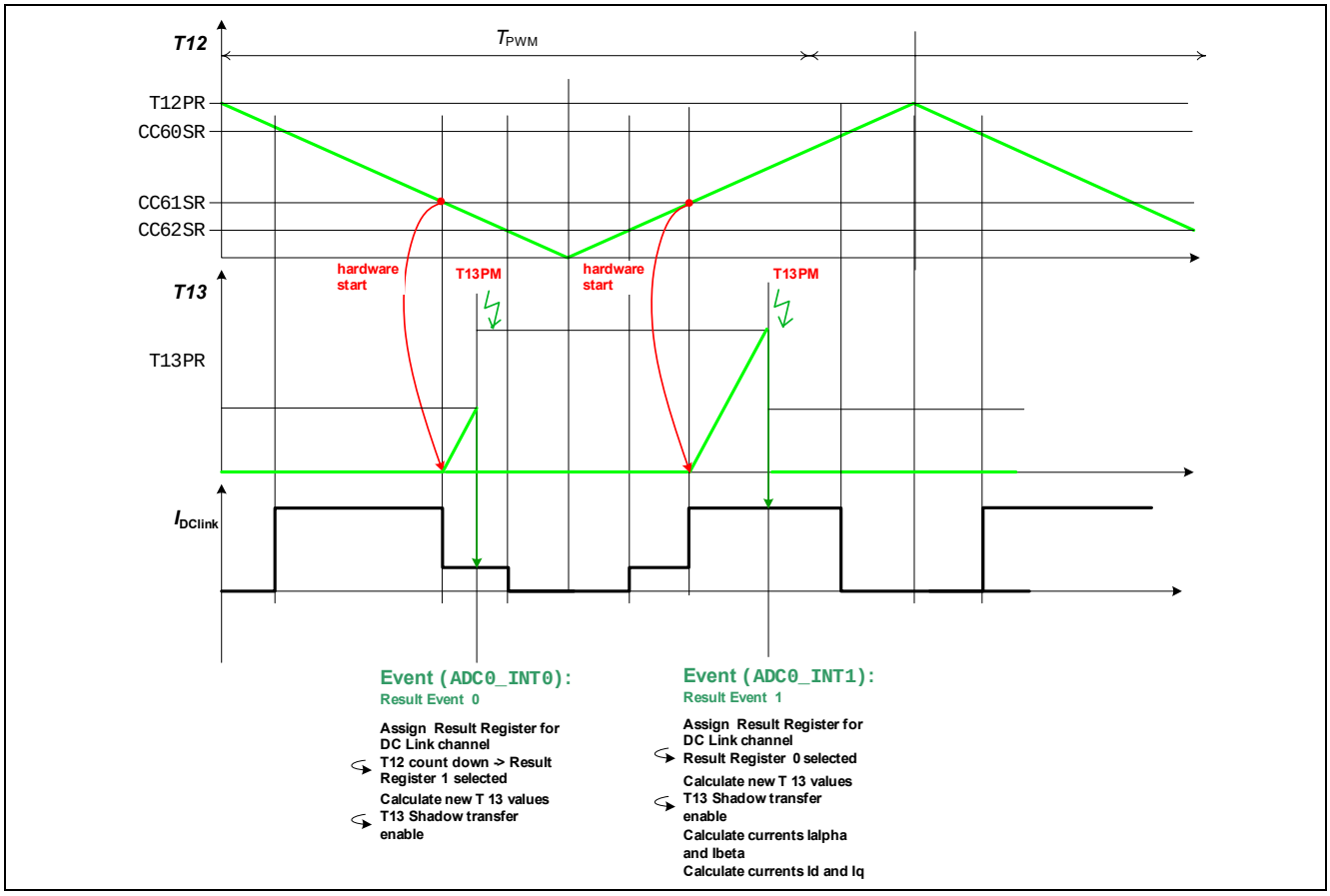


Figure 28 ADC Measurement

3.6.2 VDC and Poti

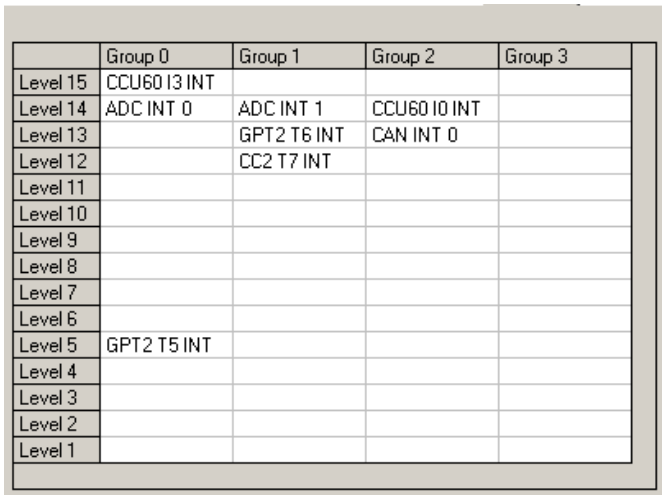
The two additional values are connected to ADC0 CH13 and ADC0 CH15. Both channels will be converted with the parallel source which is basically a channel scan. The conversion request tigger is set in software in the GPT Timer 6 interrupt. Both results are stored into global variables.

If the #define SPEED_POTI is set in the defines.h file, the poti on the XE164 Drive Card will be used to control the speed of the Motor. The control function is located in the scheduler.

3.7 Interrupt Level

The XE164 provides 96 separate interrupt nodes assignable to 16 priority levels, with 8 sub-levels (group priority) on each level.

The used interrupt level in this project are selected carefully. With the help of DAvE these levels can be adapted to application needs in case new peripherals like a USIC channel is added into the project.



	Group 0	Group 1	Group 2	Group 3
Level 15	CCU60 I3 INT			
Level 14	ADC INT 0	ADC INT 1	CCU60 I0 INT	
Level 13		GPT2 T6 INT	CAN INT 0	
Level 12		CC2 T7 INT		
Level 11				
Level 10				
Level 9				
Level 8				
Level 7				
Level 6				
Level 5	GPT2 T5 INT			
Level 4				
Level 3				
Level 2				
Level 1				

Figure 29 Interrupt Setup

3.8 Variable Definition and Administration

The variables used for the FOC algorithm and control of the whole software are separated into 4 structs located in the Controls_FOC.h file. These structs are:

- tSVM svm0;
- tFOC foc0;
- tStatus status0;
- tControl control0;

Each struct has associated variables. These structs are preloaded in the Main.c file variable declaration with the define values coming from the define.h file. These variables can be changed and monitored as required, via the DriveMonitor.

3.9 DAvE Support

The project is based on DAvE with the DIP file for XE16xx_Series V2.0. Setting changes can be made there, such as the GPT timer values to change the scheduler frequency for example, as well as adding new peripherals such as a USIC for example.

4 Communication Interface

Communication is via the Drive Monitor Software for the PC and the Drive Monitor USB Stick. Download to the XE164 is via the JTAG connection. Motor communication, such as Start and Stop commands, is via MultiCAN. The XE164 uses the MultiCAN Node 0 running at 500kBaud, together with several Message objects, for this communication.

For more information about the Drive Monitor please refer to the AppNote: Drive Monitor, Hardware Description (ap0807120_DriveMonitor.pdf).[1]

4.1 MultiCAN Interface

In the current Implementation, five CAN Message objects are used.

Message Object 10 is configured as the receive object, forwarding commands from the Drive Monitor Software via a CAN receive interrupt, to the Scheduler. CAN message objects MO1 through to MO4 are used for data transfer from the XE164 to the Drive Monitor Software.

4.1.1 Program Flow XE164

The program flow of the target microcontroller is an embedded realtime code which runs in several interrupt service routines. There is GPT Timer 6 providing a system tick of 1ms which calls a scheduler. The scheduler statemachine controls the application and reacts on host commands.

The CAN controller is configured to three interrupt vectors for receive, transmit and error handling. The CAN Message Objects have following IDs:

- ID5 - receive object - MO 10: SET/GET command and Buttons
- ID55- receive object - MO 2; unused
- ID7 - transmit object - MO1; slow data for status flags and display field
- ID77- transmit object - MO 3; fast data for oscilloscope and progress bar
- ID57- transmit object - MO 4; responds to GET command

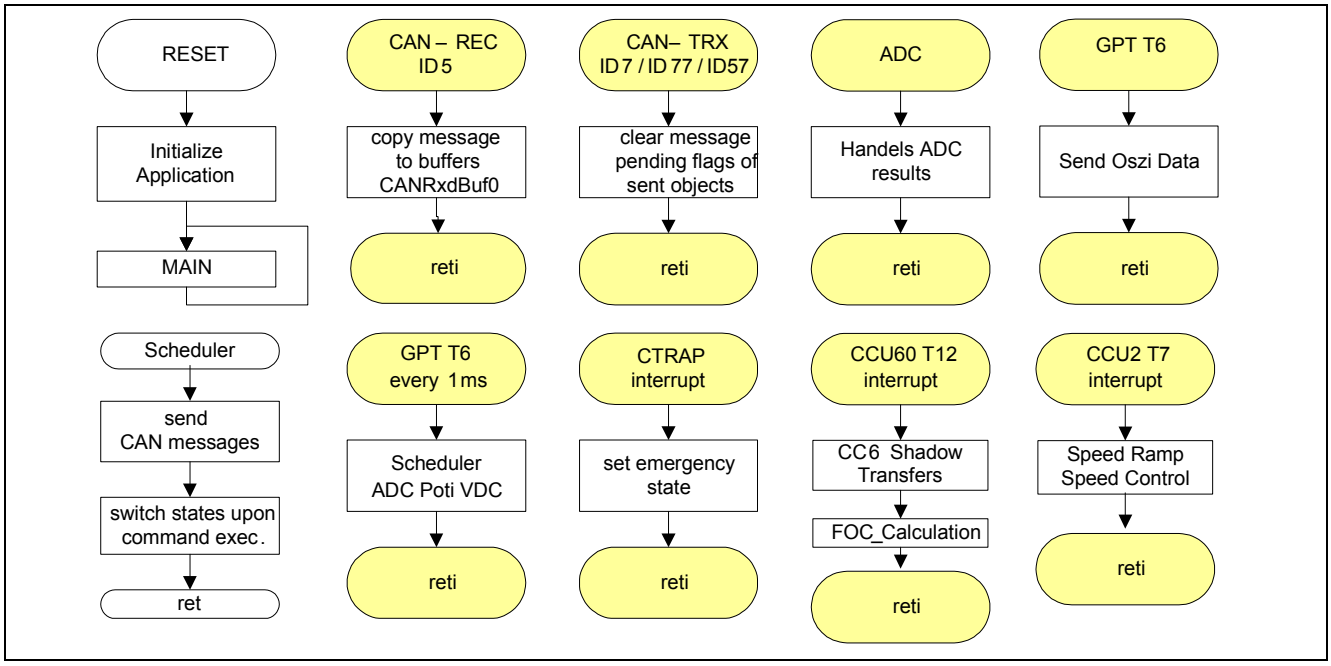


Figure 30 Program Flow

4.1.2 Command Structure

The command structure is as per the document ap0807120_DriveMonitor.pdf.[1]

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
commands from DriveMonitor to target								
adrL	adrH	valL	valH	0	0	0	CMD	5

4.1.3 Receiving a CAN Message and Command Execution

On receipt of a CAN message object (M0 10 with ID5) the receive interrupt is vectorized. The interrupt service routine scopes the received data into a global array named CANRxdBuf0.

The following shows a code extract of the CAN Receive interrupt service routine:

```
// The CAN controller has stored a new message into this object.
                       // USER CODE BEGIN (SRN0_OBJ10,4)
                      CANRxdBuf0[0]= CAN_MODATA10LL;
                      CANRxdBuf0[1]= CAN_MODATA10LH;
                      CANRxdBuf0[2]= CAN_MODATA10HL;
                      CANRxdBuf0[3]= CAN_MODATA10HH;
                      // USER CODE END
```

At every system timer tick the scheduler statemachine is executed. If a new command is received, the corresponding command is executed and the CANRxdBuf0 is cleared. The CAN receive interrupt and the GPT Timer 6 for scheduler use the same interrupt level. This mechanism ensures that an incoming command has to be served first before a new command is accepted.

4.1.4 Transmitting a CAN Message

There are three transmit message objects configured for sending data from the target to the host. The transmit messages have to be initialized. The transmit buffers CANTrxBuf0/1 are used to load the data to the message objects. The load, release and transmit request functions are generated functions from DAvE.

The following show a example code of the transmit function for message object 3:

```
_inline void CAN_Transmit_MO3(void)
{
// MO3 is shown in Oscilloscope
       CAN_vLoadData(3, (ubyte*) CANTrxBuf1 );
       CAN_vReleaseObj(3);
       CAN_vTransmit(3);
}
```

4.1.5 Command SET

With the SET command any variable of the FOC structs can be updated. Therefore it is necessary to have the address information of the respective variables. Address and data information are read from the CANRxdBuf0 and the contents of the address is written with the new value.

The following shows a code extract of the scheduler function:

```
case CMDSET:
        // byte #    0    1    2    3    4    5    6    7
        // CMD byte adrL adrH valL valH xxxx xxxx cmdL cmdH
        // CMD int  adr       val       x         CMDSET
        // int  #    0         1         2         3
                    adr = (unsigned int*) CANRxdBuf0[0];
                    value = CANRxdBuf0[1];
                    *adr = value;
                    // dont change state
                    break;
```

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
commands from DriveMonitor to target								
adrL	adrH	valL	valH	0	0	cmdL	cmdH	5

4.1.6 Command GET

With the GET command the host can read the contents of any variable. The host has to send the address information first before the target responds with the data on transmit message object MO 4.

The following shows a code extract of the scheduler function:

```
case CMDGET:
        // byte #    0    1    2    3    4    5    6    7
        // CMD byte adrL adrH xxxx xxxx xxxx xxxx cmdL cmdH
        // CMD int  adr       xxx       xxx       CMDGET
        // int  #    0         1         2         3
                    adr = (unsigned int*) CANRxdBuf0[0];

                    // Transmit MO 4 on request
                    tmp[0] = (unsigned int) adr;
                    tmp[1] = *adr;
                    CAN_vLoadData(4, (ubyte*) tmp );
                    CAN_vReleaseObj(4);
                    CAN_vTransmit(4);
                    // dont change state
                    break;
```

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
from DriveMonitor to target								
adrL	adrH	0	0	0	0	cmdL	cmdH	5

4.1.7 Button Start, Stop, Ramp Down

On pressing the Start, Stop or Ramp Down buttons, the scheduler state variable is updated and will be entered.

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
from DriveMonitor to target								
0	0	0	0	0	0	cmdL	cmdH	5

4.1.8 Scope Buttons

Realtime monitoring of fast changing data is one of the key features of the DriveMonitor. Therefore the CAN message object MO 3 - ID 77 is sent with every GPT timer 5 tick to the host. With this message three 16bit values can be monitored on the three beams of the virtual scope. With the scope buttons the addresses of the three monitoring variables can be changed.

The following shows a code extract of the scheduler function:

```
case CMDSETBEAMS:
//Button Values:  SCOPEgreen | SCOPEpink | SCOPEyellow | CMDSETSCOPE:
                ScopePointer[0] = CANRxdBuf0[0];
                ScopePointer[1] = CANRxdBuf0[1];
                ScopePointer[2] = CANRxdBuf0[2];
                break;
```

4.1.9 Oscilloscope and Progress Bar

With every GPT Timer 5 over flow the data for the soft oscilloscope and the progress bar are sent to the host. These data have the highest transmit rate. The user has to ensure that the realtime conditions are met. This can be tuned with the GPT timer 5 overflow rate. The fast data use ID77 with MO 3.

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
from DriveMonitor to target								
greenL	greenH	pinkL	pinkH	yellowL	yellowH	0	0	5

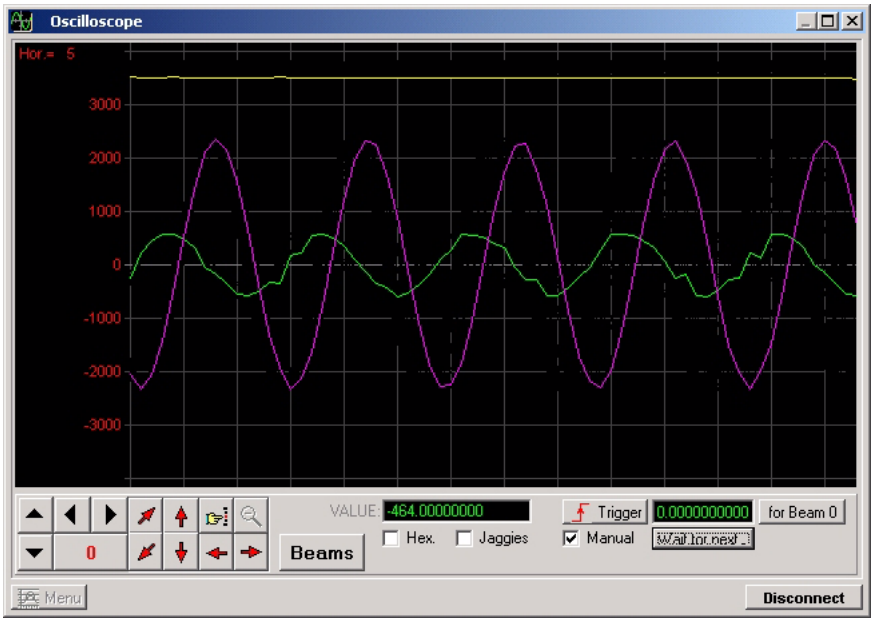


Figure 31 Drive Monitor Oscilloscope

4.1.10 Status Flags and Display Fields

For slower changing data, it is sufficient to send the data over a greater number of timer ticks. Such data would be the status flags and display fields for example. The transmit rate is defined by a variable which counts a defined number of system timer ticks. The slow data use ID7 with MO 1.

The following shows a code extract scheduler function:

```
//send parameter via CAN
if (++CANMO1Count == 50)
{
       CANTrxBuf0[0] = (unsigned int)status0.off |
                       ((unsigned int)(status0.on)<<1) |
                       ((unsigned int)(status0.control)<<2) |
                       ((unsigned int)(status0.ramp)<<3) |
                       ((unsigned int)(status0.ctrap)<<4) ;
       CANTrxBuf0[1] = svm0.Amplitude;
       CANTrxBuf0[2] = control0.ActualSpeed;
       CANTrxBuf0[3] = ADC_VDC;    //VDC
       CAN_Transmit_MO1();

       CANMO1Count=0;
}
```

5 How to Use the Setup

5.1 Excel Sheet and defines.h

Parameterization of the application is via the header file defines.h. For ease of use, an Excel sheet (see **Figure 32**) is used to generate this files contents. The algorithm itself is left untouched.

The Excel sheet has two pages: the input page *Input Parameter* and the output page *defines.h*.

The *Input Parameters* can be filled into the yellow marked fields. These are the most relevant parameters. TThe structure is as follows:

- *Motor Parameter* - this is the resistance and inductance of the motor windings. The input value refers to one phase, i.e. phase-to-neutral and not phase-to-phase.
- *Startup Parameter* - these are the values which are used in the startup block.
- *Inverter Parameter* - in case customized inverter is used, it can be adapted here.
- *Current Measurement* - in case the current amplification or the shunt is changed it has to be changed here.
- *Speed and Current Controller* - here the integral and proportional factor for the PI-controllers can be adjusted.
- *Flux Integration Scaling* - here the PT1 filter for the flux estimator can be adjusted.
- *Speed Filter* - here the PT1 filter for the speed filter can be adjusted
- *Speed Ramp* - here the speed ramp time can be adjusted. In case the Speed Ramp time in the startup parameters is not fast enough, the incremental steps can be increased.

The parameters influence each other, e.g. a very low inductance value L will scale to a very low value L' which is used for the flux estimator. In this case the input parameter K in the *Flux Integrator Scaling* should be increased.

If the Excel sheet shows *red marked* cells the scaling paramaters have to be changed accordingly.

XE164 sensorless FOC
Parameter Calculation
v1.0.00

Motor Parameter	MAXON EC32-flat	
R	6,85	ohm
L	3865	µH
Pole Pairs	4	
Startup Parameter		
Startup Speed T0	0	rpm
Startup Speed Threshold	800	rpm
Startup V/f offset ($f = 0$)	1	V
Startup V/f slew rate	0,307692308	V/Hz
Reference Speed User	2000	rpm
Speed Ramp	5000	rpm/s
Inverter Parameter		
DC link voltage	24	V
dead time	1	µs
switch delay (max)	0,7	µs
PWM frequency	15000	Hz
Bootstrap precharge time	1,5	ms
ENABLE pin logic level 1: high active -1: low active	1	
Autostart enable	1	
Use poti for speed	1	
Current Measurement		
R_shunt	0,02	ohm
R_IN	1	kOhm
R_feedback	33	kOhm
maximum Voltage at ADC	5	V

Speed PI Controller		
Ki	0,01	(< 0,5)
Kp	10	(< 64)
Current PI Controller		
scale	(<= 0,1)	(<= 0,1)

Scale Factors for Drive Monitor		
Startup		
Startup V/f offset ($f = 0$)	0,000411	V
Startup V/f slew rate	0,000028	V/Hz
Speed		
speed	3,433228	rpm
Voltages		
V(d,q)	0,000411	V
V(α,β)	0,001645	V
V(svm amplitude)	0,011871	V
Currents		
I(d,q)	0,000449	A
I(α,β)	0,000449	A
I(u,v,w)	0,001795	A
PI Controller		
Speed kp	0,001953	
Speed ki	0,000031	
Current kp	0,001953	
Current ki	0,000031	

Figure 32 Excel sheet input page

After changing the Input Parameter, the contents of output page has to be copied to the file defines.h, the C-project has to be re-complied and downloaded to the target. The Excel file also creates the scaling factors for the DriveMonitor. For instance the speed scaling changes when the number of pole pairs changes. These values have to be adapted inside the DriveMonitor configuration (CAN control editor).

5.2 Drive Monitor

The DriveMonitor is a hardware tool which bridges USB to CAN. With this it is possible to monitor some interesting data of the target on a host PC in real-time. For more information please refer to the DriveMonitor application note in [1].

5.2.1 Configuration

The configuration of the DriveMonitor setup can be found in see the DriveMonitor application note in [1].

5.2.2 Monitoring the Startup Behavior

Different motors have different startup behavior. The starting up of a synchronous motor is always quite tricky. The first revolutions have to be done in an open loop mode before the FOC algorithm can take over. There are some values which mainly influence the startup behaviour. These values can be found in the Excel sheet under *Startup Parameter* and will be discussed below.

After setting the correct R and L values, compilation and downloading the DriveMonitor can be started and used in Setting 1 | Startup. After connecting to the target, the button *Scope Start* should be pressed and then the button *Start*. The motor should start turning and after a few seconds the *Stop* button can be pressed. The scope shows in this setting two interesting signals:

- pink: `Angle` from the Startup Control block which is the given angle following the V/f ramp depending on the slew rate
- yellow: `FluxAngle` from the Carthesian to Polar Transform block, which is the estimated (calculated) angle based on the current measurement.

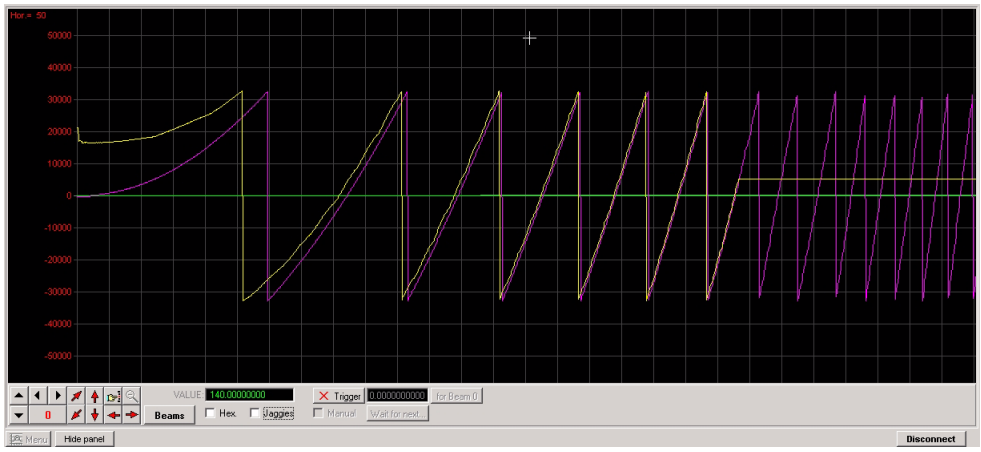
Example Fan:

After a few revolutions the estimated angle (yellow) and the given angle (pink) are quite similar. The FOC takes over at the speed threshold point.

DriveMonitor:
Speed ref

defines.h:
`AngleSpeedThreshold`

Figure 33 Startup Behaviour of a Fan before Optimization



Example Fan:

After optimization of *Vf_offset* and *Vf_slew* the startup behavior is much better (less ripples on the estimated angle).

Figure 34 Startup Behaviour of a Fan after Optimization

The point where the FOC takes over from the V/f control is of special interest. The V/f algorithm increases the speed continuously until the threshold value is reached (Excel: `Startup Speed Threshold`, defines.h: `AngleSpeedThreshold`). At this point the current and speed controllers cannot deliver the correct output as they start here. The algorithm can only prepare the controllers accordingly.

Therefore the *current controller*'s integral part is preset with current amplitude (`Vq`) based on the V/f calculation otherwise the amplitude would drop immediately. This is accomplished in CC2.c Timer interrupt function by pre-loading `control0.Iq.Yn`:

```
//*************************************************************************************
        // ramp up to motor 0

//*************************************************************************************
if (STATE_M0 == STATESTARTUP_M0)
{
if( (control0.ReferenceSpeedUser>0?foc0.AngleSpeed:-foc0.AngleSpeed) <
foc0.AngleSpeedThreshold )
        foc0.AngleSpeed += (control0.ReferenceSpeedUser>0?RAMP_STEP_M0:-RAMP_STEP_M0);
                else
                {
                   if(control0.ControlSwitch)
                   {
                                // preload parameters
                                control0.Iq.Yn = ((long)foc0.Vq)<<16;
                                // preload Iq-current controller integral value to last V/f-amplitude value
                                //*******************
                                status0.on = 0;
                                status0.control = 1;
                                svm0.SVMTMIN = SVMTMIN_M0;
                                STATE_M0 = STATEFOC_M0;
                   }

                }
        }
```

The *speed controller* must adjust at runtime. Therefore it is advisable that the speed controller is started with an input step, i.e. a difference between the `ANGLESPEED_THRESHOLD_M0` and the `REFSPEED_STARTUP_M0`. In DriveMonitor the value `Speed Ramp` can be adjusted by *Speed_ref*, i.e. the user can play with this value.

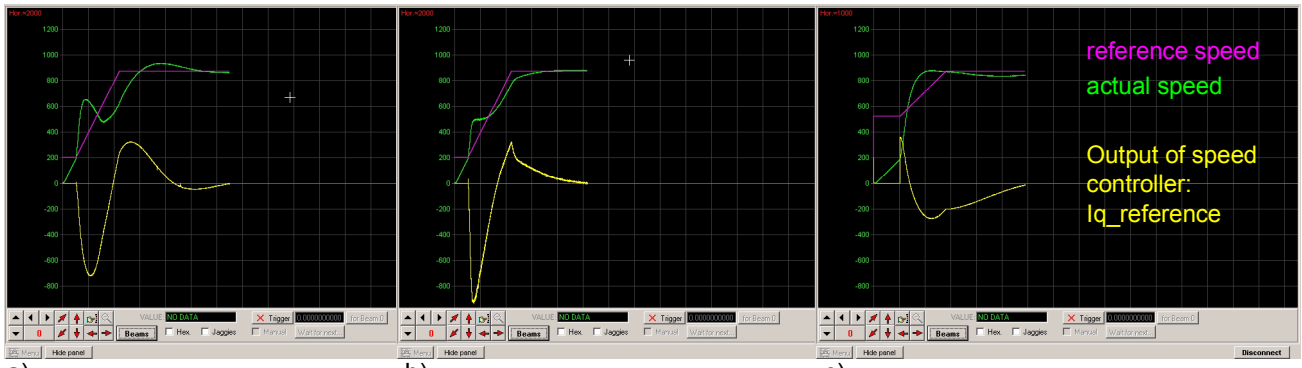


Figure 35 Switching point from V/f to FOC with different *Speed_ref* settings

In **Figure 35** different threshold conditions can be observed:

a) REFSPEED_STARTUP_M0= ANGLESPEED_THRESHOLD_M0
b) REFSPEED_STARTUP_M0>ANGLESPEED_THRESHOLD_M0
c) REFSPEED_STARTUP_M0>>ANGLESPEED_THRESHOLD_M0.

5.2.3 Monitoring the Runtime Behavior

The Runtime behavior can be influenced by the PI parameter settings of the speed and controllers. Here the DriveMonitor offers the window Setting 2 | Control. Here the *ki* and *kp* parameters can be easily adjusted. For monitoring the *Scope Speed* or *Scope I_uvq* can be used.

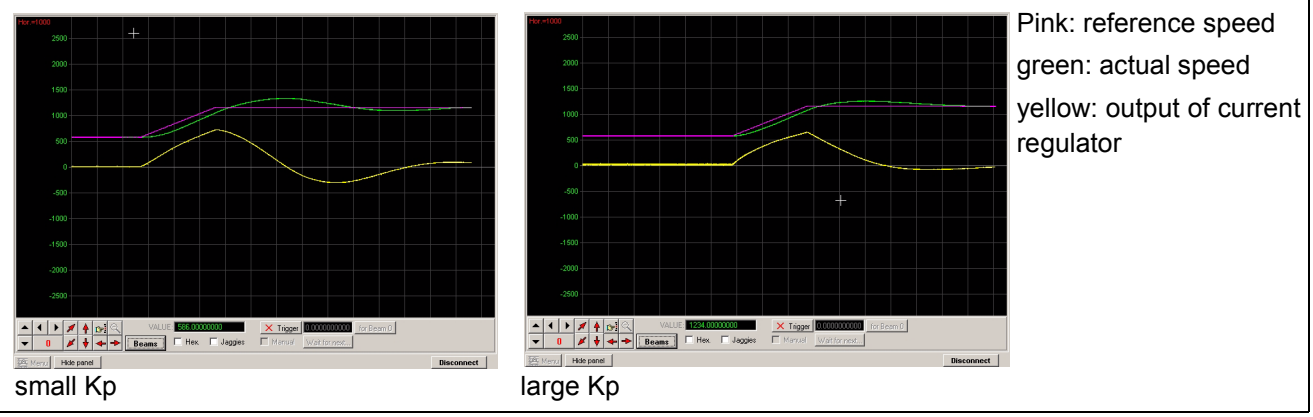


Figure 36 Speed controller adjustment, reaction on a reference speed step

5.2.4 Special tricks - Field Weakening

The nominal speed of a motor is reached when the phase amplitude is at its maximum value under zero load. This is because the BEMF (Back-Electro-Magnetic-Force) generated voltage increases with the speed and acts against the phase voltage. Therefore the speed saturates. In the FOC algorithm there are two components:

- the torque building current I_q and
- the field building current I_d.

The latter one I_d is controlled to zero because a PMSM motor generates the field by its magnet. This field component can be easily controlled to a negative value which weakens the magnetic field. With this trick the motor must turn faster than its nominal rated speed.

DriveMonitor allows the field weakening easily in the Setting 2 | Control. The third Group Entry shows *Id Ref*. A negative value here will let the motor turn faster.

A few more interesting observations can be made in this mode:

- the overall current consumption will increase - watch the power supply current,
- the torque will decrease in field weakening - this is obvious as the resulting stator flux vector is no more longer rectangular to the rotor flux vector.

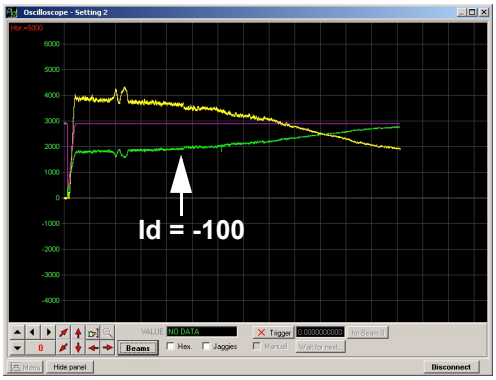


Figure 37 Field Weakening: With negative field building component *Id* the speed increases above the nominal speed.

6 References

[1] Application note - AP0807120_DriveMonitor

[2] Application note - AP0808810_XC878DriveCard

[3] Application note - AP9000110_LVinverter

[4] Application note - AP1616010_DriveCard_XE164

[5] Application note - AP0805910_Sensorless_FOC

[6] Application note - AP08090_XC878_Sensorless_FOC

[7] Link to free toolchain program -- http://www.infineon.com/Freetools

[8] Link to FOC Drive page - www.infineon.com/FOCDRIVE

Note: Please check the Infineon website (www.infineon.com) for the latest Version of these documents.

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG