MOTOROLA
Semiconductor Application Note

Order by AN1931/D
(Motorola Order Number)
Rev. 0.1, 8/01

3-Phase PM Synchronous Motor Vector Control using DSP56F80x

Design of Motor Control Application Based on Motorola Software Development Kit

Libor Prokop, Pavel Grasblum

Contents

1. Introduction of Application Benefit

This Application Note describes a design of a 3-phase Permanent Magnet (PM) synchronous motor drive. It is based on Motorola's DSP56F80x dedicated motor control device. The software design takes advantage of the SDK (Software Development Kit) developed by Motorola.

The concept of the application is that of a speed closed loop PM synchronous drive using a Vector Control technique. It serves as an example of a PM synchronous motor control design using a Motorola DSP with SDK support. It also illustrates the usage of dedicated motor control libraries that are included in the SDK.

This Application Note includes the basic motor theory, system design concept, hardware implementation and software design including the PC Master visualization tool.

2. Motorola DSP Advantages and Features

The Motorola DSP56F80x family members are well suited for digital motor control, combining the DSP's calculation capability with the MCU's controller features on a single chip. These DSPs offer many dedicated peripherals like Pulse Width Modulation (PWM) module, Analog-to-Digital Converter (ADC), Timers, communication peripherals (SCI, SPI, CAN), on-board Flash and RAM.

MOTOROLA

The typical member of the family, the DSP56F805, provides the following peripheral blocks:

- Two Pulse Width Modulator modules (PWMA & PWMB), each with six PWM outputs, three Current Sense inputs, and four Fault inputs, fault tolerant design with deadtime insertion, supports both Center- and Edge- aligned modes
- Twelve bit Analog to Digital Convertors (ADCs), supporting two simultaneous conversions with dual 4-pin multiplexed inputs; ADC can be synchronized by PWM modules
- Two Quadrature Decoders (Quad Dec0 & Quad Dec1), each with four inputs, or two additional Quad Timers A & B
- Two dedicated General Purpose Quad Timers totalling 6 pins: Timer C with 2 pins and Timer D with 4 pins
- CAN 2.0 A/B Module with 2-pin ports used to transmit and receive
- Two Serial Communication Interfaces (SCI0 & SCI1), each with two pins, or four additional GPIO lines
- Serial Peripheral Interface (SPI), with configurable 4-pin port, or four additional GPIO lines
- Computer Operating Properly (COP) timer
- Two dedicated external interrupt pins
- Fourteen dedicated General Purpose I/O (GPIO) pins, 18 multiplexed GPIO pins
- External reset pin for hardware reset
- JTAG/On-Chip Emulation (OnCE)
- Software-programmable, Phase Lock Loop-based frequency synthesizer for the DSP core clock

Table 2-1. Memory Configuration

	DSP56F801	DSP56F803	DSP56F805	DSP56F807
Program Flash	8188 x 16-bit	32252 x 16-bit	32252 x 16-bit	61436 x 16-bit
Data Flash	2K x 16-bit	4K x 16-bit	4K x 16-bit	8K x 16-bit
Program RAM	1K x 16-bit	512 x 16-bit	512 x 16-bit	2K x 16-bit
Data RAM	1K x 16-bit	2K x 16-bit	2K x 16-bit	4K x 16-bit
Boot Flash	2K x 16-bit	2K x 16-bit	2K x16-bit	2K x 16-bit

3. Target Motor Theory

The PM synchronous motor is a rotating electric machine where the stator is a classic three phase stator like that of an induction motor and the rotor has surface-mounted permanent magnets (see **Figure 3-1**).

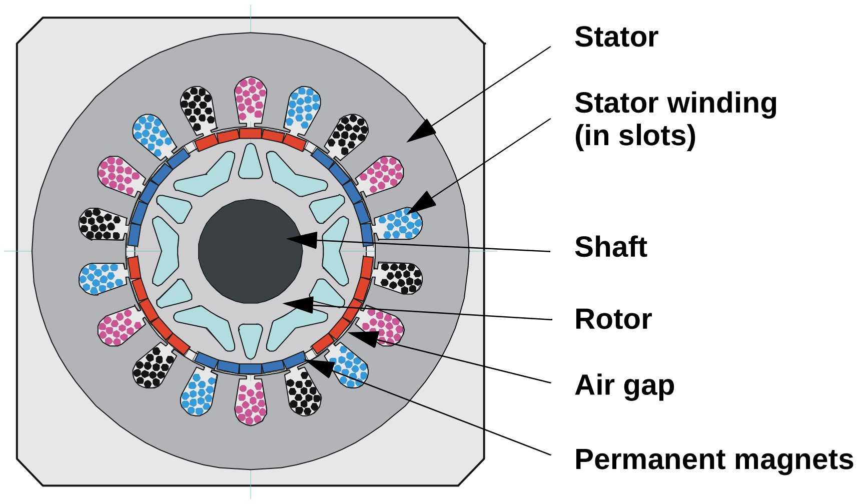


Figure 3-1. PM Synchronous Motor - Cross Section

In this respect the PM synchronous motor is equivalent to an induction motor, where the air gap magnetic field is produced by a permanent magnet. This means that the rotor magnetic field is constant. PM synchronous motors provide a set of advantages for designing modern motion control systems. The use of a permanent magnet to generate substantial air gap magnetic flux makes it possible to design highly efficient PM motors.

For a description of the PM synchronous motor, the symmetrical three-phase smooth-air-gap machine with sinusoidally distributed windings is considered. Then the voltage equations of stator in the instantaneous form can be expressed as:

$$u_{SA} = R_S i_{SA} + \frac{d}{dt}\Psi_{SA} \qquad \text{(EQ 3-1.)}$$

$$u_{SB} = R_S i_{SB} + \frac{d}{dt}\Psi_{SB} \qquad \text{(EQ 3-2.)}$$

$$u_{SC} = R_S i_{SC} + \frac{d}{dt}\Psi_{SC} \qquad \text{(EQ 3-3.)}$$

where u_{SA}, u_{SB} and u_{SC} are the instantaneous values of stator voltages, i_{SA}, i_{SB} and i_{SC} are the instantaneous values of stator currents and ψ_{SA}, ψ_{SB}, ψ_{SC} are instantaneous values of stator flux linkages in phase SA, SB and SC.

Due large number of equations in the instantaneous form the equations **(EQ 3-1.)**, **(EQ 3-2.)** and **(EQ 3-3.)** it is more practical to rewrite the instantaneous equations using two axis theory (Clark transformation). Then the PM synchronous motor can be expressed as:

$$u_{S\alpha} = R_S i_{S\alpha} + \frac{d}{dt}\Psi_{S\alpha} \qquad \text{(EQ 3-4.)}$$

$$u_{S\beta} = R_S i_{S\beta} + \frac{d}{dt}\Psi_{S\beta} \qquad \text{(EQ 3-5.)}$$

$$\Psi_{S\alpha} = L_S i_{S\alpha} + \Psi_M \cos(\Theta_r) \qquad \text{(EQ 3-6.)}$$

$$\Psi_{S\beta} = L_S i_{S\beta} + \Psi_M \sin(\Theta_r) \qquad \text{(EQ 3-7.)}$$

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p(\Psi_{S\alpha} i_{S\beta} - \Psi_{S\beta} i_{S\alpha}) - T_L\right] \qquad \text{(EQ 3-8.)}$$

where:		
	α,β	- Stator orthogonal coordinate system
	$u_{S\alpha,\beta}$	- Stator voltages
	$i_{S\alpha,\beta}$	- Stator currents
	$\Psi_{S\alpha,\beta}$	- Stator magnetic fluxes
	Ψ_M	- Rotor magnetic flux
	R_S	- Stator phase resistance
	L_S	- Stator phase inductance
	ω / ω_F	- Electrical rotor speed / fields speed
	p	- Number of poles per phase
	J	- Inertia
	T_L	- Load torque
	Θ_r	- rotor position in α,β coordinate system

The equations **(EQ 3-4.)** ... **(EQ 3-8.)** represent model of PM synchronous motor in the stationary frame α, β fixed to the stator. The main idea of the vector control is to decompose the vectors into a magnetic field generating part and a torque generating part. In order to do it we need to set up the rotary coordinate system connected to the rotor magnetic field. This coordinate system is generally called “d,q-coordinate system” (Park transformation). Thus the equations **(EQ 3-4.)** ... **(EQ 3-8.)** can be rewritten as:

$$u_{Sd} = R_S i_{Sd} + \frac{d}{dt}\Psi_{Sd} - \omega_F \Psi_{Sq} \qquad \text{(EQ 3-9.)}$$

$$u_{Sq} = R_S i_{Sq} + \frac{d}{dt}\Psi_{Sq} + \omega_F \Psi_{Sd} \qquad \text{(EQ 3-10.)}$$

$$\Psi_{Sd} = L_S i_{Sd} + \Psi_M \qquad \text{(EQ 3-11.)}$$

$$\Psi_{Sq} = L_S i_{Sq} \qquad \text{(EQ 3-12.)}$$

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p(\Psi_{Sd} i_{Sq} - \Psi_{Sq} i_{Sd}) - T_L\right] \qquad \text{(EQ 3-13.)}$$

By considering that below base speed $i_{sd}=0$ the equation **(EQ 3-13.)** can be reduced to following form:

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p(\Psi_M i_{Sq}) - T_L\right] \qquad \text{(EQ 3-14.)}$$

From the equation **(EQ 3-14.)** it can be seen that the torque is dependent and can be directly controlled by the current i_{sq} only.

4. System Concept

4.1 System Specification

The system is designed to drive a 3-phase PM synchronous motor. The application meets the following performance specifications:

- Vector control of PM motor using the Quadrature encoder as a position sensor
- Targeted for DSP56F80xEVM
- Running on 3-phase PM synchronous Motor Control Development Platform at variable line voltage 110 - 230V AC
- Control technique incorporates
  - vector control with speed closed loop and field weakening
  - rotation in both directions
  - motoring and generator mode
  - start from any motor position with rotor alignment
  - minimal speed **50** rpm
  - maximal speed **3000** rpm at input power line 230V AC
  - maximal speed **1500** rpm at input power line 115V AC
- Manual Interface (Start/Stop switch, Up/Down push button control, Led indication)
- PCMaster Interface (motor start/stop, speed set-up)
- Power Stage Identification
- Overvoltage, Undervoltage, Overcurrent, Position sensing and Overheating Fault protection

The introduced PM synchronous drive is designed to power the high voltage PM synchronous motor with a quadrature encoder. It has the following specifications:

Motor Characteristics:	Motor Type	6 poles, three phase, star connected, BLDC motor
	Speed Range:	2500 rpm (at 310V)
	Max. Electrical Power:	150 W
	Phase Voltage:	3*220V
	Phase Current	0.55A
Drive Characteristics:	Speed Range	< 2500 rpm
	Input Voltage:	310V DC
	Max DC Bus Voltage	380 V
	Control Algorithm	Speed Closed Loop Control
	Optoisolation	Required

Figure 4-2. High Voltage Hardware Set Specifications

4.2 Vector Control Drive Concept

For the drive a standard system concept is chosen (see **Figure 4-3**). The system incorporates the following hardware parts:

- three-phase PM synchronous motor high voltage development platform
- feedback sensors: position (quadrature encoder), DC-Bus voltage, phase currents, DC-Bus overcurrent detection, DC-Bus overcurrent detection, temperature
- DSP56F803EVM / DSP56F805EVM / DSP56F807EVM

The drive can be basically controlled in two different ways (or operational modes)

In the Manual operational mode the required speed is set by Start/Stop switch and Up and Down push buttons.

In the PC Master operational mode, the required speed and Start/Stop switch are set by PC Master.

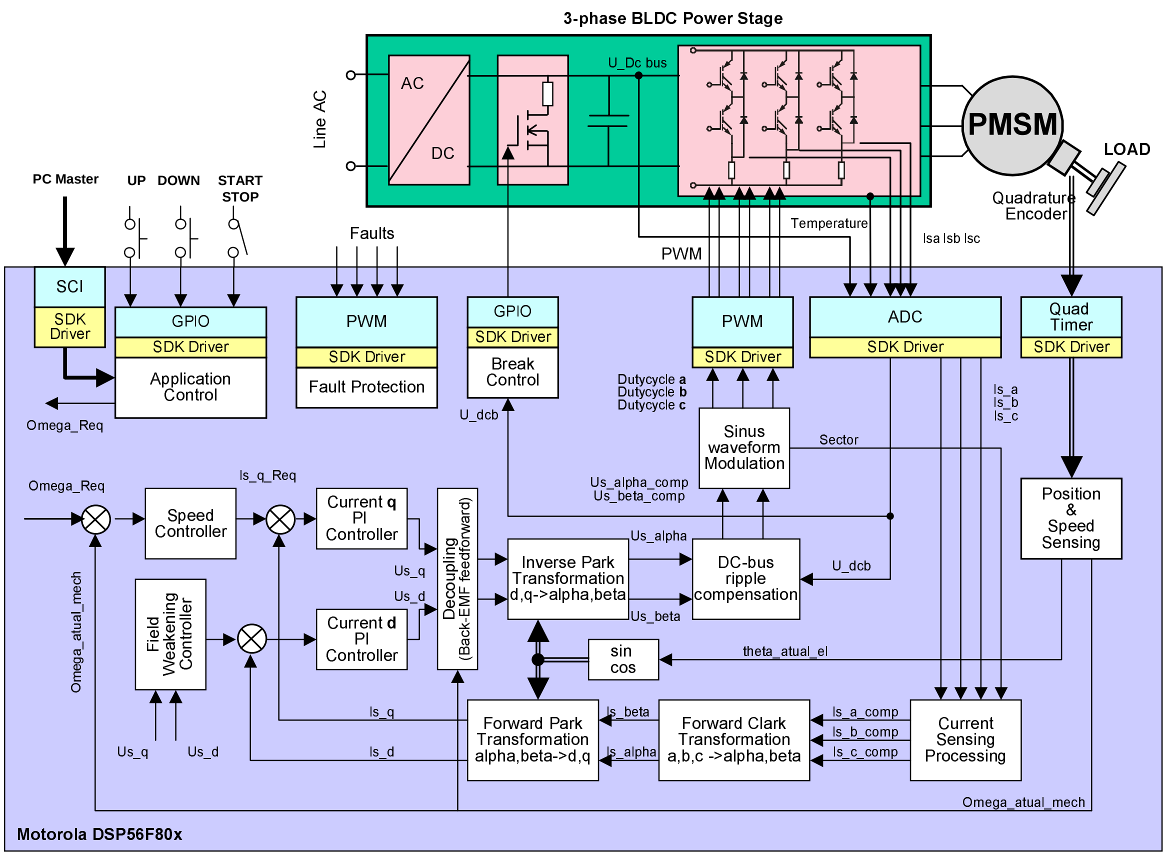


Figure 4-3. Drive Concept

The **control process** is as follows:

When the Start command is accepted (using Start/Stop Switch or PC Master command) the required speed is calculated according to Up/Down push buttons or PC Master commands (in the case of PC Master control). The required speed goes through an acceleration/deceleration ramp and a reference command is put to the speed controller. The comparison between the actual speed command and the measured speed generates a speed error. Based on the error, the speed controller generates a current *Is_qReq* which corresponds to torque. A second part of stator current *Is_dReq* which correspond to flux is given by Field Weakening Controller. Simultaneously the stator currents Is_a, Is_b and Is_c are measured and transformed from instantaneous values to the stationary reference frame α, β and consecutively to the rotary reference frame d, q (Park - Clark transformation). Based on the errors between required and actual currents in the rotary reference frame, the current controllers generate output voltages *Us_q* and *Us_d* (in the rotary reference frame d, q). The voltages *Us_q* and *Us_d* are transformed back to the stationary reference frame α, β and after DC-bus ripple elimination are recalculated to the three phase voltage system which is applied on the motor.

Except for the main control loop, the DC Bus voltage, DC Bus current and power stage temperature are measured during the control process. They are used for overvoltage, undervoltage, overcurrent and overheating protection of the drive. The undervoltage and overheating protection is performed by software while the overcurrent and overvoltage fault signal utilizes a Fault input of the DSP.

If any of the above mentioned faults occurs, the motor control PWM outputs are disabled in order to protect the drive and the fault state of the system is displayed.

Also, ahardware error is detected if the wrong power stage is used. Each power stage contains a simple module generating logic sequence unique for that type of power stage. During the chip initialization this sequence is read and evaluated according to the decoding table. If the correct power stage is identified, the program can continue. In case of wrong hardware the program stays in the infinite loop displaying the fault conditions.

4.3 Control Technique

4.3.1 Vector Control

Vector Control is an elegant control method of controllong the PM synchronous motor where field oriented theory is used to control space vectors of magnetic flux, current and voltage. It is possible to set up the coordinate system to decompose the vectors into a magnetic field generating part and a torque generating part. Then the structure of the motor controller (Vector Control controller) is almost the same as for a separately excited DC motor which simplifies the control of PM synchronous motor. This Vector Control technique was developed in the past especially to achieve similar good dynamic performance of PM synchronous motors.

As we explained in **Section 4.2, Vector Control Drive Concept**, we chose a widely used speed control with inner current closed loop where the rotor flux is controlled by a field weakening controller.

In this method we need to decompose the field generating part and the torque generating part of the stator current to be able to separately control the magnetic flux and the torque. In order to do so we need to set up the rotary coordinate system connected to the rotor magnetic field. This coordinate system is generally called “d,q-coordinate system”. Very high CPU performance is needed to perform the transformation between rotary to stationary coordinate systems. Therefore the Motorola DSP’s 56F80x is very suited for vector control algorithm. All trasformations which are needed for vector control will be described in the next section.

4.3.2 Vector Control Transformations

The whole trick of transforming the PM synchronous motor into a DC motor is based on points of view. As shown in **Section 4.3.1**, we need a coordinate transformation to do this trick.

The following transformations are involved in Vector Control:

- transformations from 3-phase to 2-phase system (Clarke transformation)
- rotation of orthogonal system
  - α,β to d,q (Park transformation)
  - d,q to α,β (Inverse Park transformation)

4.3.2.1. Clarke Transformation

Figure 4-4. shows how the three-phase system is transformed into a two-phase system.

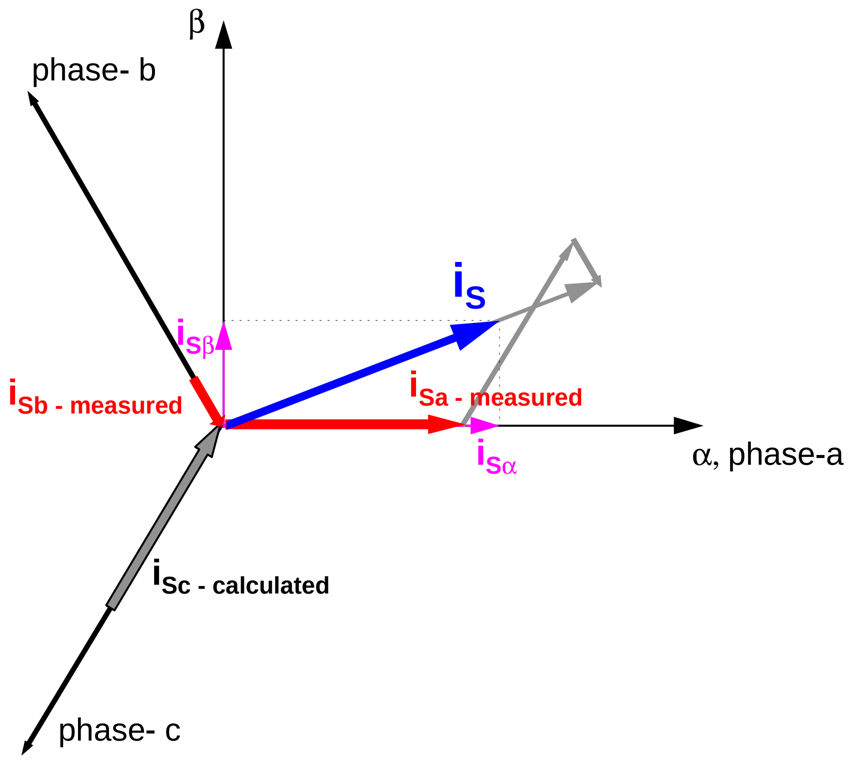


Figure 4-4. Clarke Transformation

Let us transfer the graphical representation into mathematical language:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = K \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \text{(EQ 4-1.)}$$

In most cases the three-phase system is symmetrical. It means that the sum of the phase quantities is always zero.

$$\alpha = K\left(a - \frac{1}{2}b - \frac{1}{2}c\right) = \left| \ a + b + c = 0 \ \right| = K\frac{3}{2}a \qquad \text{(EQ 4-2.)}$$

The constant “K” can be freely chosen. It is apparent that a good choice would be to equalize the α-quantity and a-phase quantity. Then:

$$\alpha = a \Rightarrow K = \frac{2}{3} \tag{EQ 4-3.}$$

Now we can fully define the Park-Clarke transformation:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\frac{2}{3} & -\frac{1}{3} & -\frac{1}{3}\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} = |\ a+b+c=0\ | = \begin{bmatrix}1 & 0 & 0\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} \tag{EQ 4-4.}$$

4.3.2.2. Transformation from α,β to d,q Coordinates and Backward

The whole Vector Control is performed in the d,q-coordinate system in order to make the control of synchronous PM motors elegant and easy **(see 4.3.1 Vector Control)**.

Of course this involves the transformation in both directions (the control action has to be transformed back to the motor side).

First we must to establish the d,q-coordinate system:

$$\Psi_M = \sqrt{\Psi_{M\alpha} + \Psi_{M\beta}} \tag{EQ 4-5.}$$

$$\sin\vartheta_{Field} = \frac{\Psi_{M\beta}}{\Psi_{Md}} \qquad \cos\vartheta_{Field} = \frac{\Psi_{M\alpha}}{\Psi_{Md}} \tag{EQ 4-6.}$$

Then the transformation from α,β to d,q coordinates is:

$$\begin{bmatrix}d\\ q\end{bmatrix} = \begin{bmatrix}\cos\vartheta_{Field} & \sin\vartheta_{Field}\\ -\sin\vartheta_{Field} & \cos\vartheta_{Field}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \tag{EQ 4-7.}$$

Figure 4-5. illustrates this transformation.

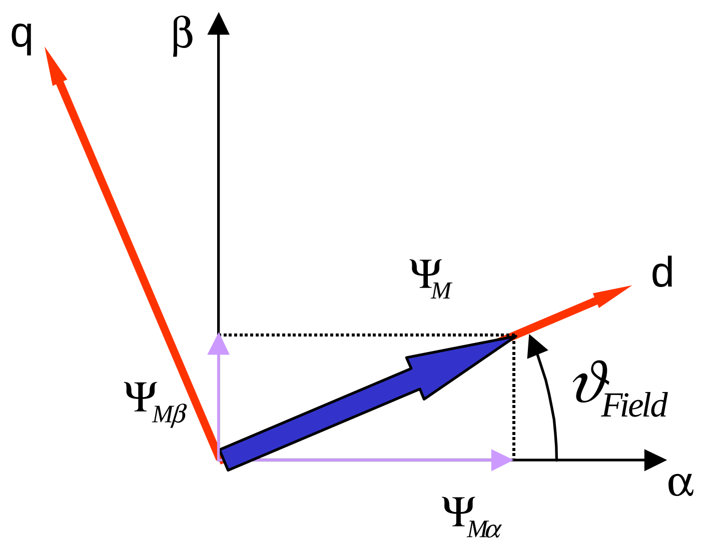


Figure 4-5. Establishing of the d,q-coordinate System (Park transformation)

The backward (Inverse Park) transformation (from d,q to α,β) is:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\cos\vartheta_{Field} & -\sin\vartheta_{Field}\\ \sin\vartheta_{Field} & \cos\vartheta_{Field}\end{bmatrix}\begin{bmatrix}d\\ q\end{bmatrix} \qquad \text{(EQ 4-8.)}$$

4.3.3 Position and speed sensing

All members of Motorola DSP family 56F80x except 56F801 have a quadrature decoder. This peripheral is commonly used for position and speed sensing. The quadrature decoder position counter counts up/down each edge of Phase A and Phase B signals according to their order. Each revolution the position counter is cleared by an Index pulse (see **Section 4-6**).

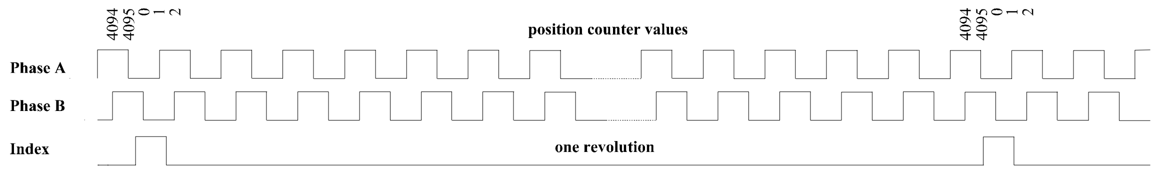


Figure 4-6. Quadrature Encoder Signals

It means that the zero position is put together with index pulse. But the vector control requires the zero position there in where the rotor is aligned to d axis (see **Section 4.3.2.2.**). Therefore using a quadrature decoder to decode the encoder’s signal requires a calculation of an offset which aligns the position counter of the quadrature decoder with aligned rotor position (zero position). To avoid to calculation of rotor position offset ,the quadrature decoder is not used in this application. Thus the quadrature decoder is available for another purpose and the presented application is able to run on the DSP56F801 which does not have a quadrature decoder.

In addition to the quadrature decoder the input signals (Phase A, Phase B and Index) are connected to quad timer module A. The quad timer module consists of four quadrature timers. Due to the wide variability of quad timer modules it is possible to use this module to decode quadrature encoder signals and to sense position and speed. A configuration of the quad timer module is shown in **Figure 4-7**.

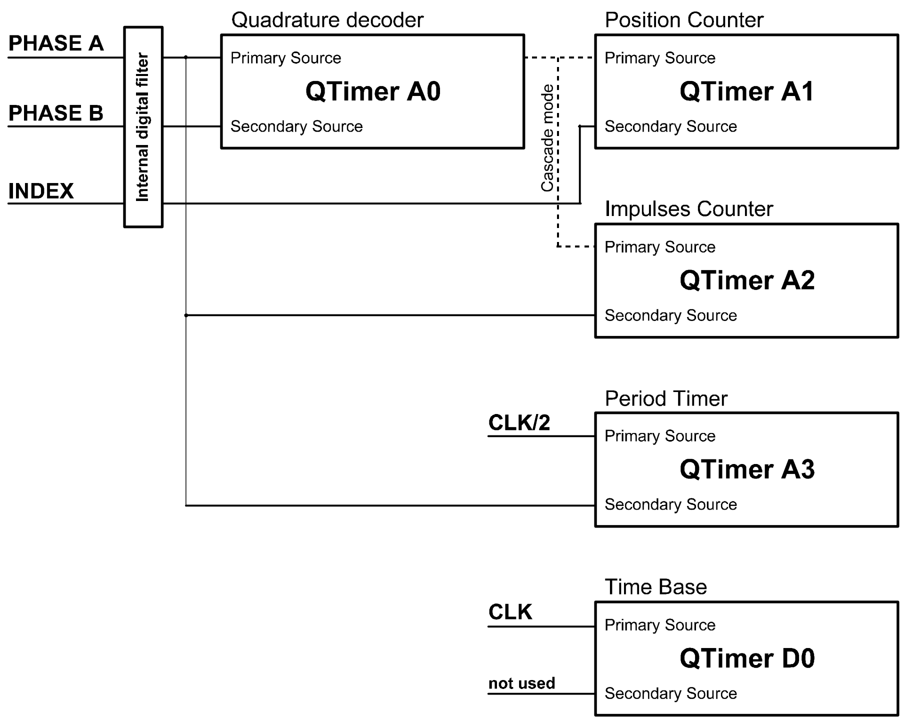


Figure 4-7. Quad Timer Module A Configuration

4.3.3.1. Position Sensing

The position and speed sensing algorithm uses all timers in module A and one other timer as time base. The timers A0 and A1 are used for position sensing. The timer A0 permits connection of three input signals to a quadrature timer even if the quadrature timer has only two inputs (primary and secondary). The timer A0 is set to count in the quadrature mode, count to zero and then reinitialize. That set timer decodes quadrature signals only. Timer A1 is connected to timer A0 in the cascade mode. In this mode the information about counting up/down is connected internally to timer A1. Thus the secondary input of the timer A1 is free to be used for index pulse counting. The counter A1 is set to count to +/- ((4*number of pulses per revolution) - 1) and reinitialize after compare. A value of the timer A1 corresponds to rotor position. A position of index pulse is sensed to avoid the loss of some pulses under the influence of noise during extended motor operation which can result in incorrect rotor position sensing. If some pulses are lost a different position of the index pulses is detected and position sensing error is signalized. If the checking of index pulse is not required the timer A1 can be removed and the timer A0 is set as position counter A1. The resulting value of timer A1 is scaled to range <-1; 1) which corresponds to <-π; π) (see **Figure 4-8**).

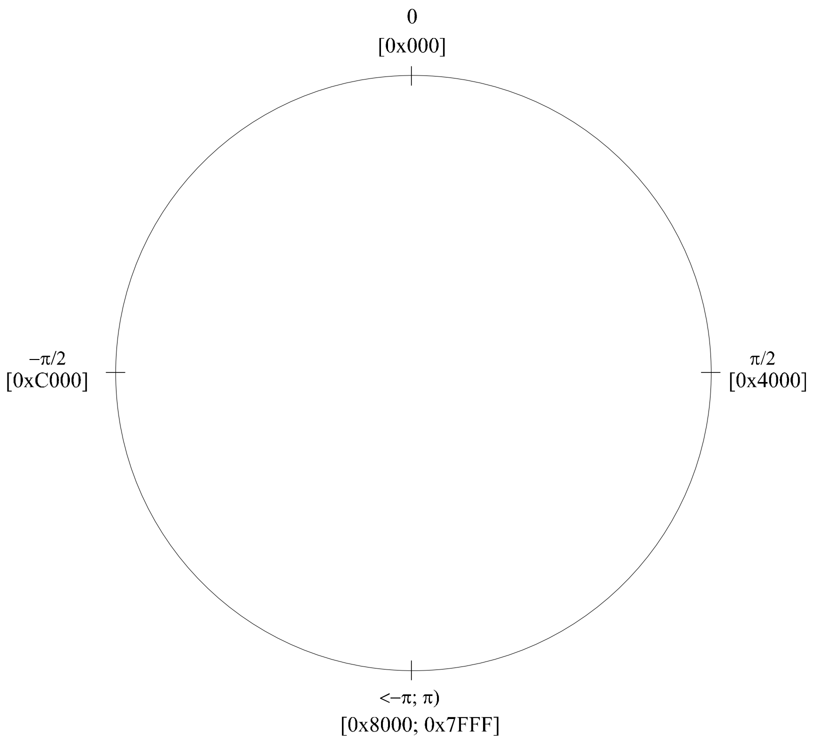


Figure 4-8. Position Scaling

4.3.3.2. Speed Sensing

There are two common ways of measuring speed. The first method measures time between two following edges of quadrature encoder and the second method measures the position difference per constant period. The first method is used at low speed. At the moment when the measured period is very short, the speed calculation algorithm switces to the second method.

The proposed algorithm combines both above mentioned methods. The algorithm measures simultaneously the number of quadrature encoder pulses per constant period and their accurate time period. Then the speed can be expressed as:

$$speed = \frac{k \cdot N}{T} \qquad \text{(EQ 4-9.)}$$

where

speed	calculated speed
k	scaling constant
N	number of pulses per constant period
T	accurate period of *N* pulses

The algorithm requires two timers for counting of pulses and their period and another timer as time base (see **Figure 4-7**). The timer A2 counts the pulses of the quadrature encoder and the timer A3 counts a system clock divided by 2. The values in both timers can be captured by each edge of the Phase A signal. The time base is provided by timer D0 which is set to call speed processing algorithm each 900 µs. The speed processing algorithm works in the following way:

At first the new captured values of both timers are read. The number of pulses difference and their accurate period is calculated from actual and previous values. Then the new values are saved for the next period and the capture register is enabled. From this time the first edge of Phase A signal captures the values of both timers (A2, A3) and the capture register is disabled. This process is repeated each calling of the speed processing algorithm (see **Figure 4-9**).

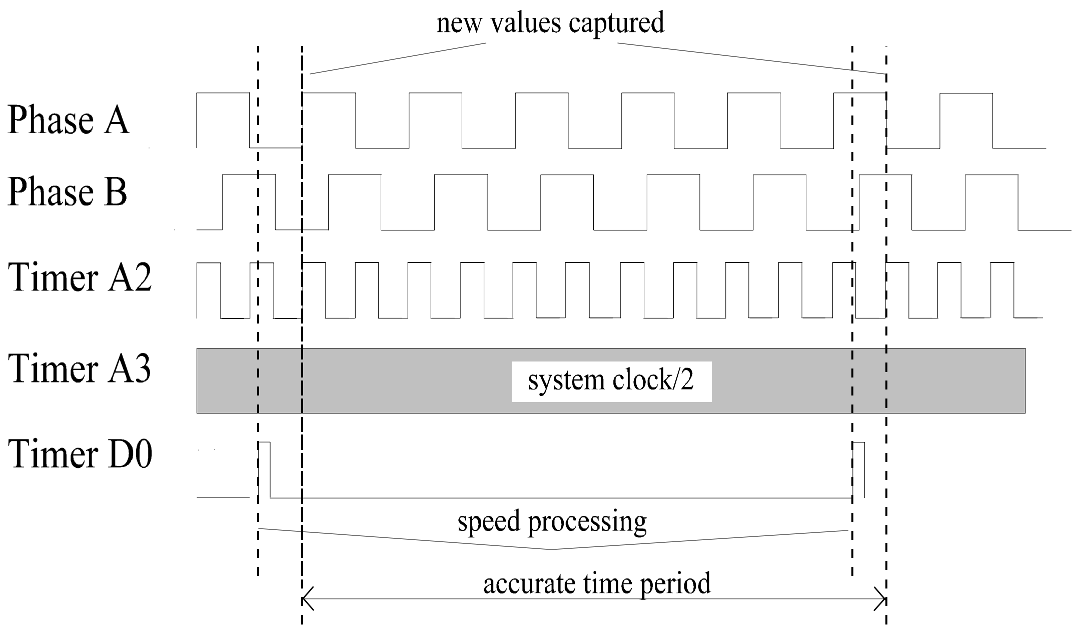


Figure 4-9. Speed Processing

4.3.3.2.1 Minimal and Maximal Speed Calculation

The minimal speed is given by following equation:

$$v_{min} = \frac{60}{4NT_{calc}} \tag{EQ 4-10.}$$

where:

v_{min}	minimal obtainable speed [rpm]
N	number of pulses per revolution [-]
T_{calc}	period of speed measuring (calculation period) [s]

In that application the quadrature encoder has 1024 pulses per revolution and calculation period 900 µs was chosen on the basis of a motor mechanical constant. Thus the equation **(EQ 4-10.)** gives the minimal speed 16.3 rpm.

The maximal speed can be expressed as:

$$v_{max} = \frac{60}{4NT_{clkT2}} \qquad \text{(EQ 4-11.)}$$

where:

v_{max}	maximal obtainable speed [rpm]
N	number of pulses per revolution [-]
T_{clkT2}	period of input clock to timer A2 [s]

After substitution in equation **(EQ 4-11.)** for N and T_{clkT2} (timer A2 input clock = system clock 36 MHz/2) we get the maximal speed 263672 rpm. As can be seen the presented algorithm is able to measure speed within wide speed range. Because such high speed is not practical, the maximum speed can be reduced to a required range by the constant k in equation **(EQ 4-9.)**. The constant k can be calculated as:

$$k = \frac{60}{4NT_{clkT2}v_{max}} \qquad \text{(EQ 4-12.)}$$

where:

k	scaling constant in the equation **(EQ 4-9.)**
v_{max}	maximal required speed [rpm]
N	number of pulses per revolution [-]
T_{clkT2}	period of input clock to timer A2 [s]

In the presented application the maximal measurable speed is limited to 6000 rpm.

Notes: To ensure a correct speed calculation, the input clock of the timer A2 must be chosen so that the calculation period of speed processing (in our case 900 µs) is represented in the timer A2 as a value lower than 0x7FFFH ($900.10^{-6}/T_{clkT2}$<=0x7FFFH).

4.3.4 Current Sensing

Phase currents are measured by a shunt resistor in a each phase. A voltage drop on the shunt resistor is amplified by a operational amplifier and shifted up by 1.65V. The resultant voltage is converted by a A/D converter (see **Figure 4-10.** and **Figure 4-11.**).

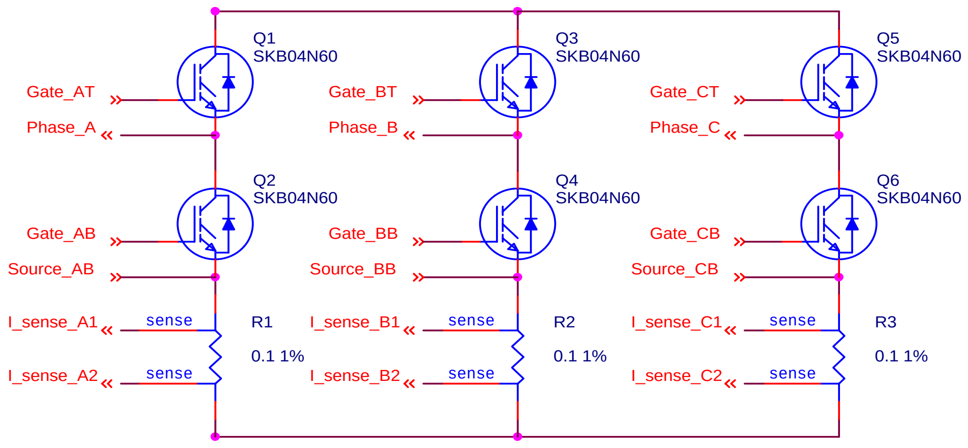


Figure 4-10. Current Shunt Resistors

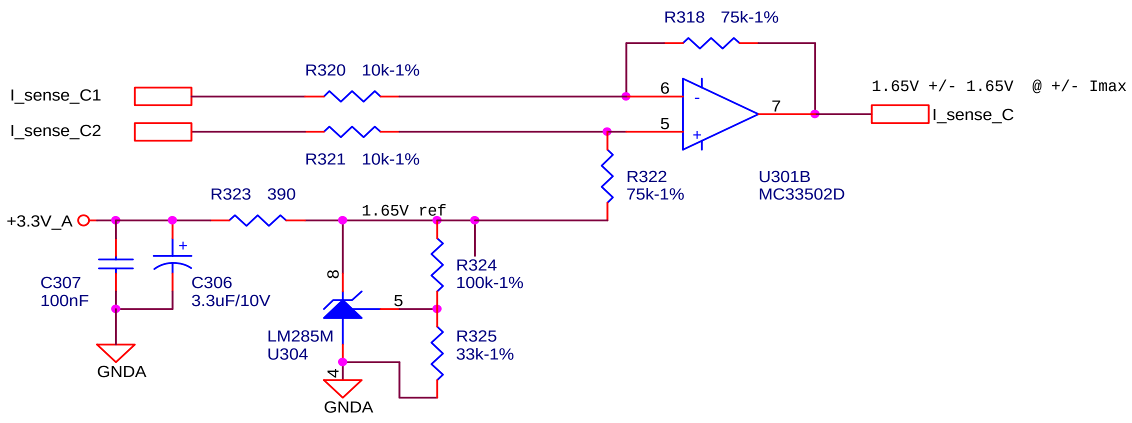


Figure 4-11. Current Amplifier

As can be seen in **Figure 4-10.**, the currents cannot be measured at any moment. For example, the current flows through phase A (and shunt resistor R1) only if the transistor Q2 is switched on. Correspondingly the current in phase B can be measured if the transistor Q4 is switched on and current in phase C can be measured if the transistor Q6 is switched on. In order to get a moment of current sensing a voltage shapes analysis must be done.

The voltage shapes of two different PWM periods are shown in **Figure 4-12**. The voltage shapes correspond to center aligned PWM sinewave modulation. As can be seen, the best moment of current sampling is in the middle of the PWM period where the all bottom transistors are switched on. Regardless all three currents cannot be measured at any voltage shape. The PWM period II in **Figure 4-12** shows a moment when the bottom transistor of phase A is on for a very short time. If the on time is shorter than a critical time the current can not be correctly measured. The critical time is given by hardware configuration (transistor commutation times, response delays of the processing electronics, etc.). Therefore, only two currents are measured only and third current is calculated from equation:

$$0 = i_A + i_B + i_C \tag{EQ 4-13.}$$

$$0 = i_A + i_B + i_C \tag{EQ 4-14.}$$

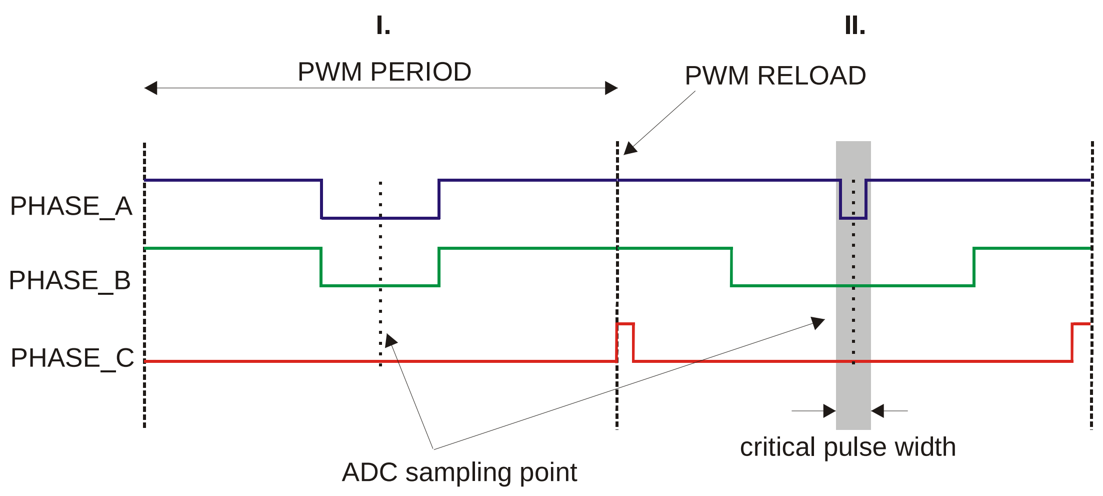


Figure 4-12. The Voltage Shapes of Two Different PWM Periods

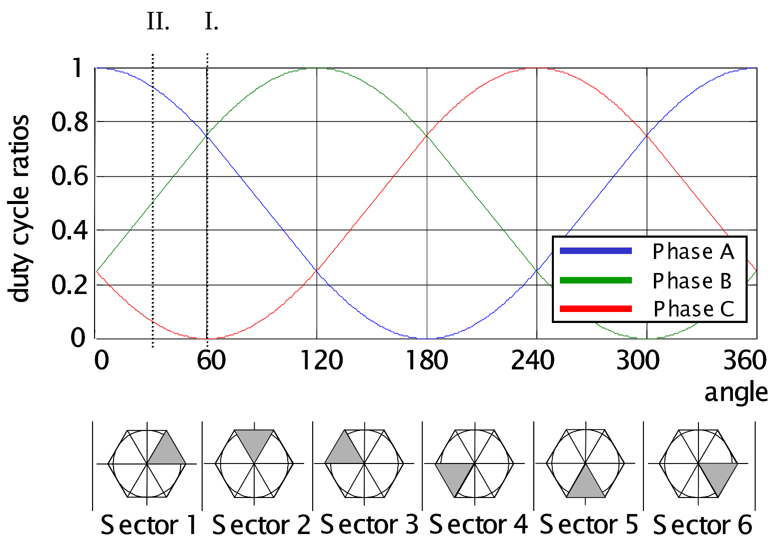


Figure 4-13. 3-phase Sinewave Voltages and Corresponding Sector Value

Now we have to decide which current is calculated. The simplest technique is to calculate the current of the most positive voltage phase. For example, phase A generates the most positive voltage within section 0 - 60°, phase B within section 60° - 120°, etc. (see **Figure 4-13**).

In our case the output voltages are divided into six sectors (see **Figure 4-13**). Then the current calculation is done according to the actual sector value:

for sector 1, 6:

$$i_A = -i_B - i_C \qquad \text{(EQ 4-15.)}$$

for sector 2, 3:

$$i_B = -i_A - i_C \qquad \text{(EQ 4-16.)}$$

for sector 4, 5:

$$i_C = -i_B - i_A \qquad \text{(EQ 4-17.)}$$

Notes: The sector value is used for current calculation only and has no other meaning at the sinewave modulation. But if we use any type of space vector modulation we can get the sector value as part of space vector calculation.

4.3.5 Voltage Sensing

The DC-Bus voltage sensor is represented by a simple voltage divider. The DC-Bus voltage does not change rapidly. It is nearly constant with the ripple given by the power supply structure. If a bridge rectifier is used for rectification of the AC line voltage, the ripple frequency is twice the AC line frequency. If the power stage is designed correctly, The ripple amplitude should not exceed 10% of the nominal DC-Bus value.

The measured DC-Bus voltage needs to be filtered in order to eliminate noise. One of the easiest (and fastest) techniques is the 1st order filter, that calculates the average filtered value recursively from the last two samples and coefficient C:

$$u_{DCBusFilt}(n+1) = (Cu_{DCBusFilt}(n+1) - Cu_{DCBusFilt}(n)) - u_{DCBusFilt}(n) \qquad \text{(EQ 4-18.)}$$

In order to speed up the initialization of the voltage sensing (filter has exponential dependency with constant of 1/N samples), the moving average filter that calculates the average value from the last N samples can be used for initialization:

$$u_{DCBusFilt} = \sum_{n=1}^{-N} u_{DCBus}(n) \qquad \text{(EQ 4-19.)}$$

4.3.6 Power Module Temperature Sensing

The measured power module temperature is used for thermal protection The hardware realization is shown in **Figure 4-14**. The circuit consists of four diodes connected in series, a bias resistor, and a noise suppression capacitor. The four diodes have a combined temperature coefficient of 8.8 mV/ºC. The resulting signal, Temp_sense, is fed back to an A/D input where software can be used to set safe operating limits. In the presented application the temperature in degrees Celsius is calculated according to conversion equation:

$$\text{temp} = \frac{\text{Temp\_sense - b}}{\text{a}} \qquad \text{(EQ 4-20.)}$$

where:

temp - power module temperature in Celsius deg.

Temp_sense - voltage drop on the diodes which is measured by ADC [V]

a - diodes dependent conversion constant (a = -0.0073738)

b - diodes dependent conversion constant (b = 2.4596)

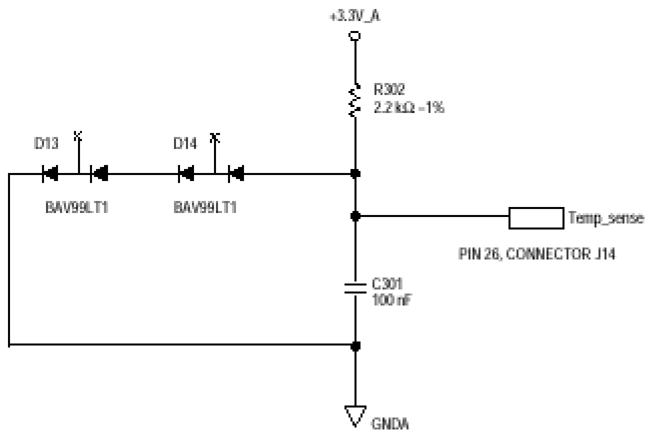


Figure 4-14. Temperature Sensing

4.3.7 RUN/STOP Switch and Button Control

The RUN/STOP switch is connected to a GPIO pin in the case of the 56F805/7EVM. The state of RUN/STOP switch can be read directly from GPIO Data Register. In the case of the 56F803EVM there are not that many free GPIO pins so the switch is connected to ADC input AN7. Therefore the switch status is obtained with AD Convertor. The switch logical status is obtained by comparison of measured value with threshold value.

Also the buttons are usually connected to GPIO pins. But in the presented application the buttons are connected to IQRA and IRQB interrupts. A reason for this solution is to be able to run same code on all EVM boards (56F803/5/7EVM). Since the 56F803 has no free GPIO pins for user buttons the application uses buttons connected to IRQA/B pins. The EVM boards do not solve the button contact bounces, which may occur during pushing and releasing of button. Therefore this issue has to be solved by software.

The IRQA and IRQB interrupts are maskable interrupts connected directly to the DSP's core. The IRQA and IRQB lines are internally synchronized with the processor's internal clock and can be programmed as level sensitive or edge sensitive. The IRQA and IRQB interrupts have no interrupt flag; therefore this flag is replaced by a software flag. The following algorithm is used to check the state of the IRQ line. The algorithm is described for one interrupt.

The IRQ interrupt is set to be negative level sensitive. When the button is pressed, the logical level 0 is applied on the IRQ line and the interrupt occurs (see **Figure 4-15**). To avoid multiple calls of ISR due to contact bounces, the ISR disables the IRQ interrupt, sets the debounce counter to predefined value and sets the variable *buttonStatus* to 1. The variable *buttonStatus* represents the interrupt flag. Using the DSP56F80x's software timer, the *ButtonProcessing* function is periodically called (see **Figure 4-15**). The function *ButtonProcessing* decrements the debounce counter and if the counter is zero the IRQ interrupt is again enabled. The button press is checked by *ButtonEdge* function **Figure 4-16**. When variable *buttonStatus* is set the *ButtonEdge* function returns 1 and clears *buttonStatus*. When the variable *buttonStatus* is not set, the *ButtonEdge* function returns 0.

According to the *ButtonProcessing* calling period, the value of the debounce counter should be set close to 180 ms. This value is sufficient to prevent multiple IRQ ISR calls due to contact bounces.

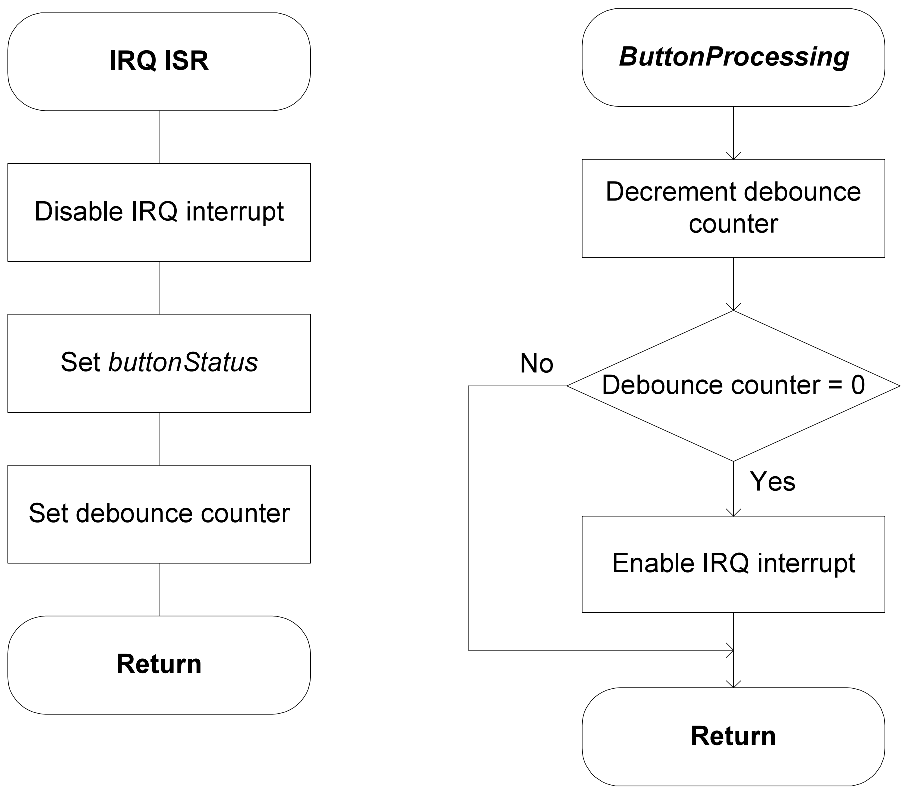


Figure 4-15. Button Control - IRQ ISR and ButtonProcessing

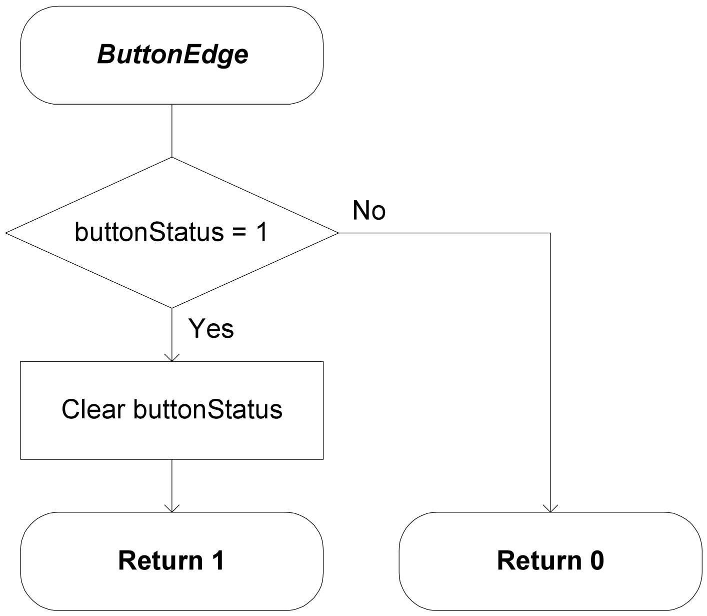


Figure 4-16. Button Control - ButtonEdge

4.3.8 RotorAlignment

After reset, the rotor position is unknown, because a quadrature encoder does not give absolute position till INDEX pulse comes. As can be seen in **Figure 4-5.** the rotor position has to be aligned with d axis of the d,q-coordinate system before a motor begins running. The alignment algorithm is shown in **Figure 4-18.** At first the position is set to zero independently of the actual rotor position. Then the I_d current is set to alignment current. Now the rotor is aligned to the required position. After rotor stabilization the encoder is reset in order to give zero position after that the I_d current is set back to zero and alignment is finished. The alignment is executed during the first transition from STOP to RUN state of RUN/STOP switch only.

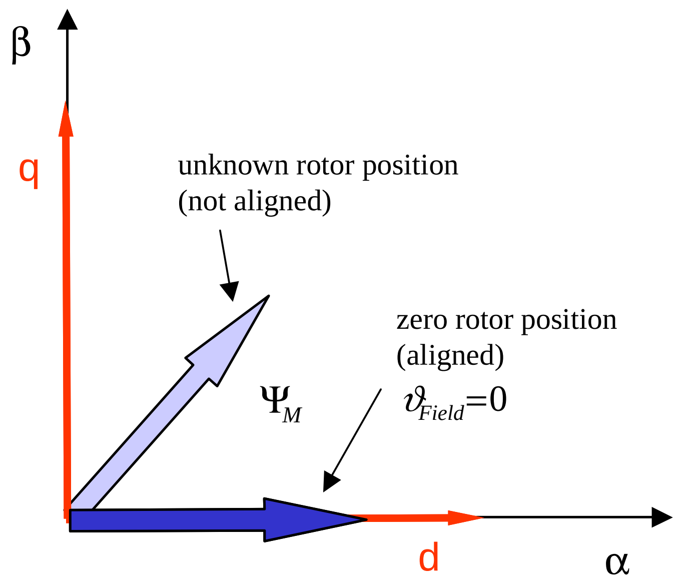


Figure 4-17. Rotor Alignment

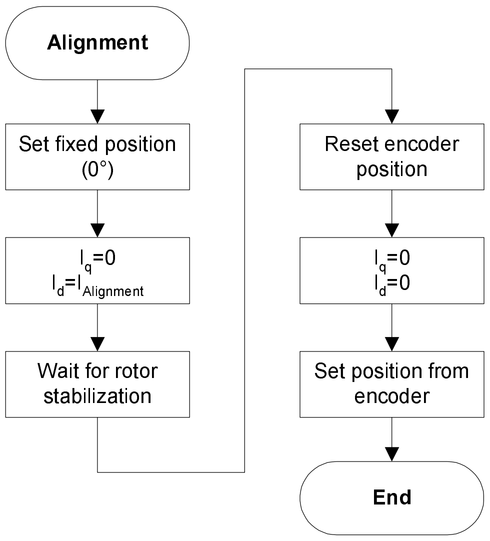

Alignment
Set fixed position (0°)
$I_q=0$ $I_d=I_{Alignment}$
Wait for rotor stabilization
Reset encoder position
$I_q=0$ $I_d=0$
Set position from encoder
End

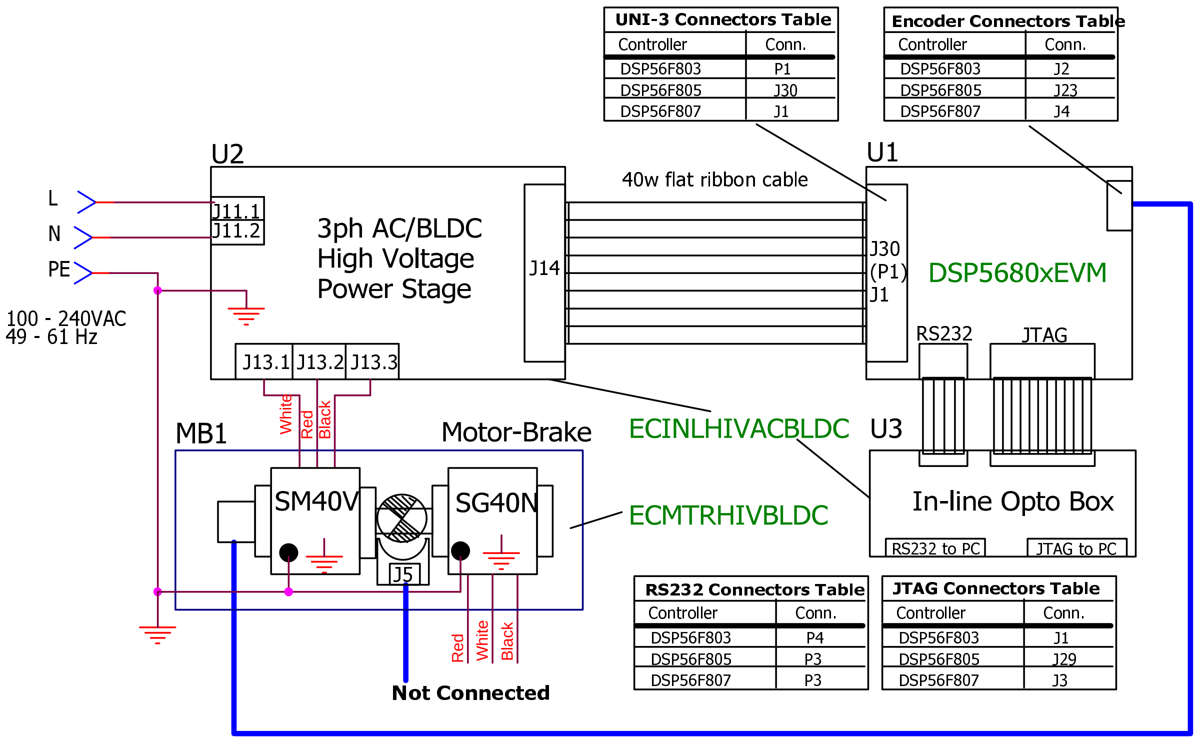
Figure 4-18. Rotor Alignment Flow Chart

5. Hardware Design

5.1 System Concept

The motor control system is designed to drive a 3-phase PM synchronous motor in a speed-closed loop.

The application can run on Motorola motor control DSPs using the DSP EVM Board DSP56F803/5/7, the Motorola’s 3-Phase AC/BLDC high voltage power stage and the BLDC high voltage motor with a quadrature encoder and an integrated brake. All parts are an integral part of Motorola’s embedded motion control development tools. The hardware setup is shown in **Figure 5-19**.



UNI-3 Connectors Table

Controller	Conn.
DSP56F803	P1
DSP56F805	J30
DSP56F807	J1

Encoder Connectors Table

Controller	Conn.
DSP56F803	J2
DSP56F805	J23
DSP56F807	J4

RS232 Connectors Table

Controller	Conn.
DSP56F803	P4
DSP56F805	P3
DSP56F807	P3

JTAG Connectors Table

Controller	Conn.
DSP56F803	J1
DSP56F805	J29
DSP56F807	J3

Figure 5-19. High Voltage Hardware System Configuration

All the system parts are supplied and documented according the following references:

- **U1 -** Controller Board for DSP56F803:
  - supplied as: DSP56803EVM
  - described in: **DSP56F805EVMUM/D DSP Evaluation Module Hardware User’s Manual**
- or **U1 -** Controller Board for DSP56F805:
  - supplied as: DSP56803EVM
  - described in: **DSP56F805EVMUM/D DSP Evaluation Module Hardware User’s Manual**
- or **U1 -** Controller Board for DSP56F807:
  - supplied as: DSP56807EVM
  - described in: **DSP56F803EVMUM/D DSP Evaluation Module Hardware User’s Manual**
- **U2 -** 3-phase AC/BLDC High Voltage Power Stage

- — supplied in kit with In-Line Optoisolation Box as: ECINLHIVACBLDC
- — described in: **MEMC3BLDCPSUM/D - 3-phase Brushless DC High Voltage Power Stage**

- **U3** - In-Line Optoisolation Box
  - — supplied in kit with 3-phase AC/BLDC High Voltage Power Stage as: ECINLHIVACBLDC or solo as ECOPTINL
  - — described in: **MEMCILOBUM/D - In-Line Optoisolation Box**

Warning: It is strongly recommended to use In-line Optoisolation Box during the development time to avoid any damage to the development equipment.

- **MB1** Motor-Brake SM40V + SG40N
  - — supplied as: ECMTRHIVBLDC

Notes: The application SW is targeted for PM Synchronous motor with sinewave Back-EMF shape. In this particular demo application the BLDC motor is used instead. This is due to the availability of the BLDC motor - Brake SM40V+SG40N supplied as ECMTRHIVBLDC. Although the Back-EMF shape of this motor is not ideally sinewave, it can be controlled by the application SW. The drive parameters will be the best with a PMSM motor with exactly sinewave Back-EMF shape.

A detailed description of individual boards can be found in the comprehensive User's Manuals belonging to each dedicated board or at *http://mot-sps.com/motor/devtools/index.html*. The user's manual incorporates the schematic of the board, description of individual function blocks and bill of materials. The individual boards can be ordered from Motorola as a standard products.

6. Software Design

This section describes the design of the software blocks of the drive. The software is described in the following terms:

- Main Software Flow Chart
- Data Flow
- State Diagram

For more information on the control technique used, refer to **Section 4.3**.

6.1 Main Software Flow Chart

The main software flow chart incorporates the Main routine entered from Reset and interrupt states. The Main routine includes the initialization of the DSP and the main loop shown in **Figure 6-20**, **Figure 6-21**, **Figure 6-22**.

The SW consist of processes: Application Control, PM Synchronous Motor (PMSM) Control, Analog sensing, Position and Speed Measurement, Fault Control, Brake Control.

The Application Control process is the highest SW level which precedes settings for other SW levels. The Input of this level is Run/Stop switch, Up/Down buttons for manual control and PC Master (via the registers **(see 6.2 Data Flow)**). This process is handled by Application Control Processing called from Main (see **Figure 6-20**).

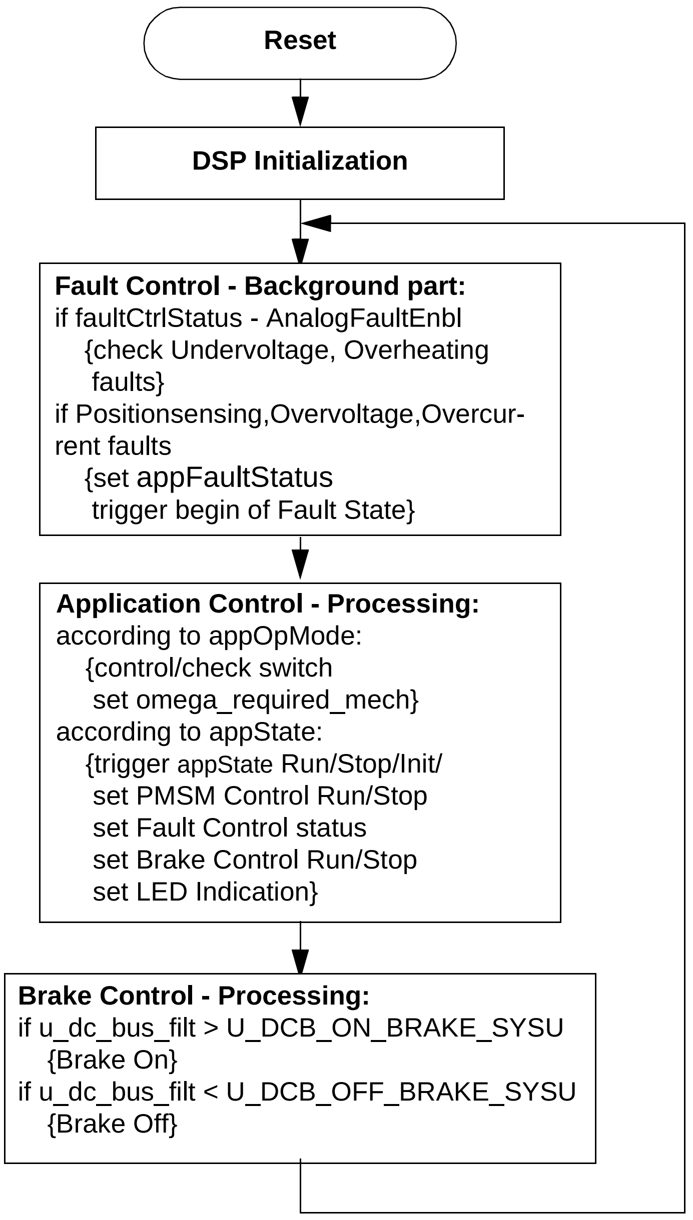

Reset
DSP Initialization
Fault Control - Background part:
if faultCtrlStatus - AnalogFaultEnbl
{check Undervoltage, Overheating faults}
if Positionsensing,Overvoltage,Overcurrent faults
{set appFaultStatus
trigger begin of Fault State}
Application Control - Processing:
according to appOpMode:
{control/check switch
set omega_required_mech}
according to appState:
{trigger appState Run/Stop/Init/
set PMSM Control Run/Stop
set Fault Control status
set Brake Control Run/Stop
set LED Indication}
Brake Control - Processing:
if u_dc_bus_filt > U_DCB_ON_BRAKE_SYSU
{Brake On}
if u_dc_bus_filt < U_DCB_OFF_BRAKE_SYSU
{Brake Off}

Figure 6-20. SW Flow Chart - General Overview I

The PMSM (PM Synchronous Motor) Control process provides most of motor control functionality. It is divided into Current Processing and speed processing parts. The current processing part is called from ADC Complete Interrupt (see **Figure 6-21**) once per 2 PWM reloads with the period 125µs (it can also be set to each PWM reload - 62.5µs, but the PC Master recorder pcmasterdrvRecorder() needs to be removed from the code). The Speed Processing part is called from Quadrature Timer D0 Interrupt (see **Figure 6-22**) with the period PER_TMR_POS_SPEED_US (900µs). The advantage of dividing the current and the speed control process is that current control can be executed with low period of calls, because the execution of the speed control is not that prioritized.

The Analog sensing process handles sensing, filtering and correction of analog variables (phase currents, temperature, dc bus voltage). It is provided by Analog Sensing Processing (see **Figure 6-21**) and also by Analog Sensing ADC Phase Set. Analog Sensing ADC Phase Set is split from Analog Sensing Processing because it sets ADC according to the svmSector variable calculated in after PMSM Control-Current Processing.

Position and Speed Measurement processes are provided by HW Timer modules and the functions giving the actual speed and position (see **Figure 4.3.3**).

The LED Indication process is provided by LED Indication Processing. It is called from Quadrature Timer D0 Interrupt, which provides the time base for the LED flashing.

The Fault Control process is split into Background part and PWM Fault ISR part. The Background part (see **Figure 6-20**) checks the Overheating, Undervoltage and Positionsensing faults. The PWM Fault ISR part (see **Figure 6-21**) takes care of Overvoltage and Overcurrent faults which causes the PWM A Fault interrupt.

Brake Control process is dedicated for the brake transistor control, which maintains DC Bus voltage level. It is called from Main (see **Figure 6-20**).

The Up/Down Button and Switch Control processes are subprocesses of Application Control. They are described in **Section 4.3.7**.

The Up/Down Button processes are split into Button Processing Interrupt part called from Quadrature Timer D0 Interrupt (see **Figure 6-22**), Button Processing BackGround part (inside of Analog Sensing), Interrupt Up Button and Interrupt Down Button (see **Figure 6-21**).

The Switch process is split into Switch Filter Processing called from Quadrature Timer D0 Interrupt (see **Figure 6-22**) and Switch Get State called from Application Control processing which handles manual switch control and Switch Get State PC Master which handles switch control in case of PC Master application operating mode.

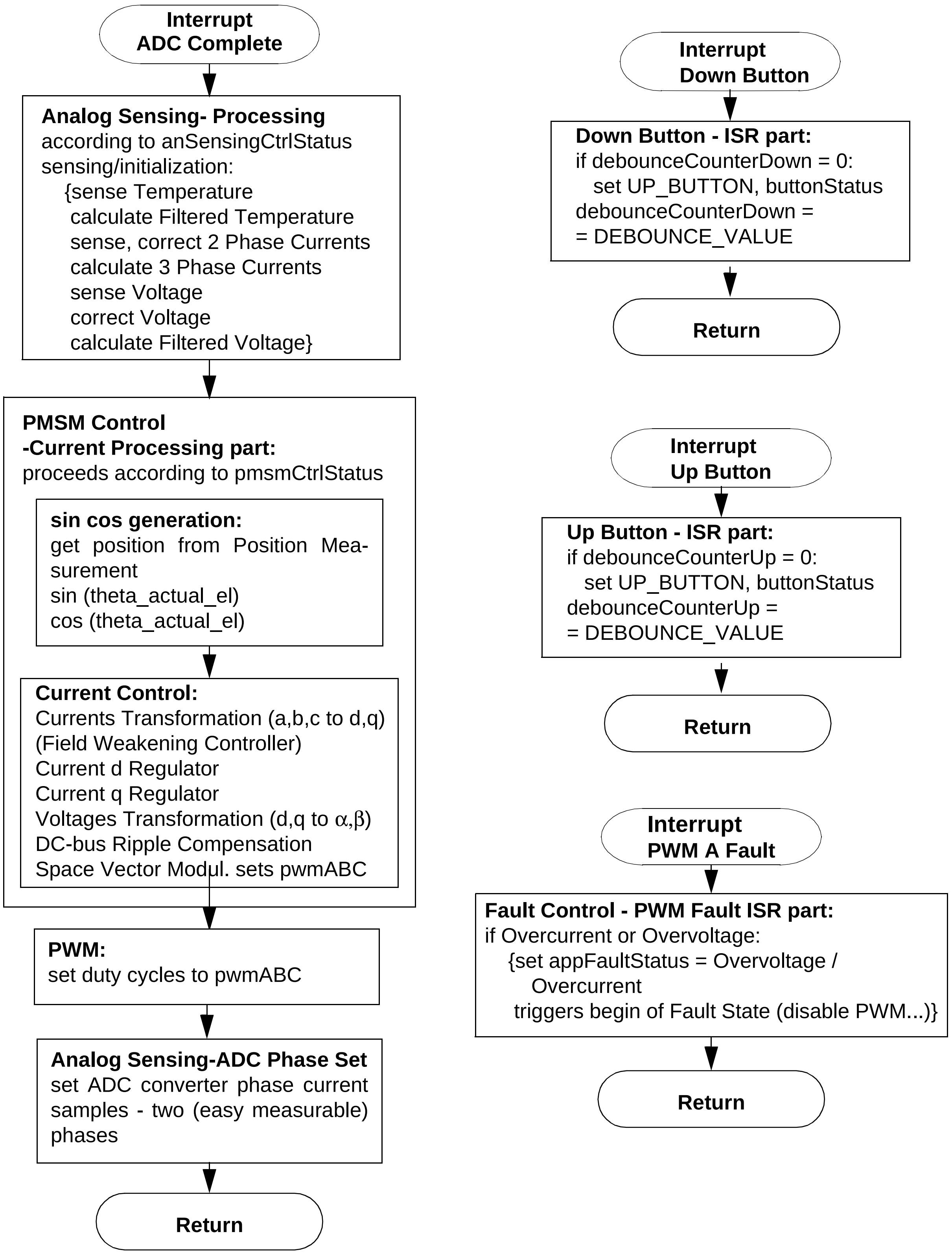

Interrupt
ADC Complete
Analog Sensing- Processing
according to anSensingCtrlStatus
sensing/initialization:
{sense Temperature
calculate Filtered Temperature
sense, correct 2 Phase Currents
calculate 3 Phase Currents
sense Voltage
correct Voltage
calculate Filtered Voltage}
PMSM Control
-Current Processing part:
proceeds according to pmsmCtrlStatus
sin cos generation:
get position from Position Measurement
sin (theta_actual_el)
cos (theta_actual_el)
Current Control:
Currents Transformation (a,b,c to d,q)
(Field Weakening Controller)
Current d Regulator
Current q Regulator
Voltages Transformation (d,q to α,β)
DC-bus Ripple Compensation
Space Vector Modul. sets pwmABC
PWM:
set duty cycles to pwmABC
Analog Sensing-ADC Phase Set
set ADC converter phase current samples - two (easy measurable) phases
Return
Interrupt
Down Button
Down Button - ISR part:
if debounceCounterDown = 0:
set UP_BUTTON, buttonStatus
debounceCounterDown =
= DEBOUNCE_VALUE
Return
Interrupt
Up Button
Up Button - ISR part:
if debounceCounterUp = 0:
set UP_BUTTON, buttonStatus
debounceCounterUp =
= DEBOUNCE_VALUE
Return
Interrupt
PWM A Fault
Fault Control - PWM Fault ISR part:
if Overcurrent or Overvoltage:
{set appFaultStatus = Overvoltage / Overcurrent
triggers begin of Fault State (disable PWM...)}
Return

Figure 6-21. SW Flow Chart - General Overview II

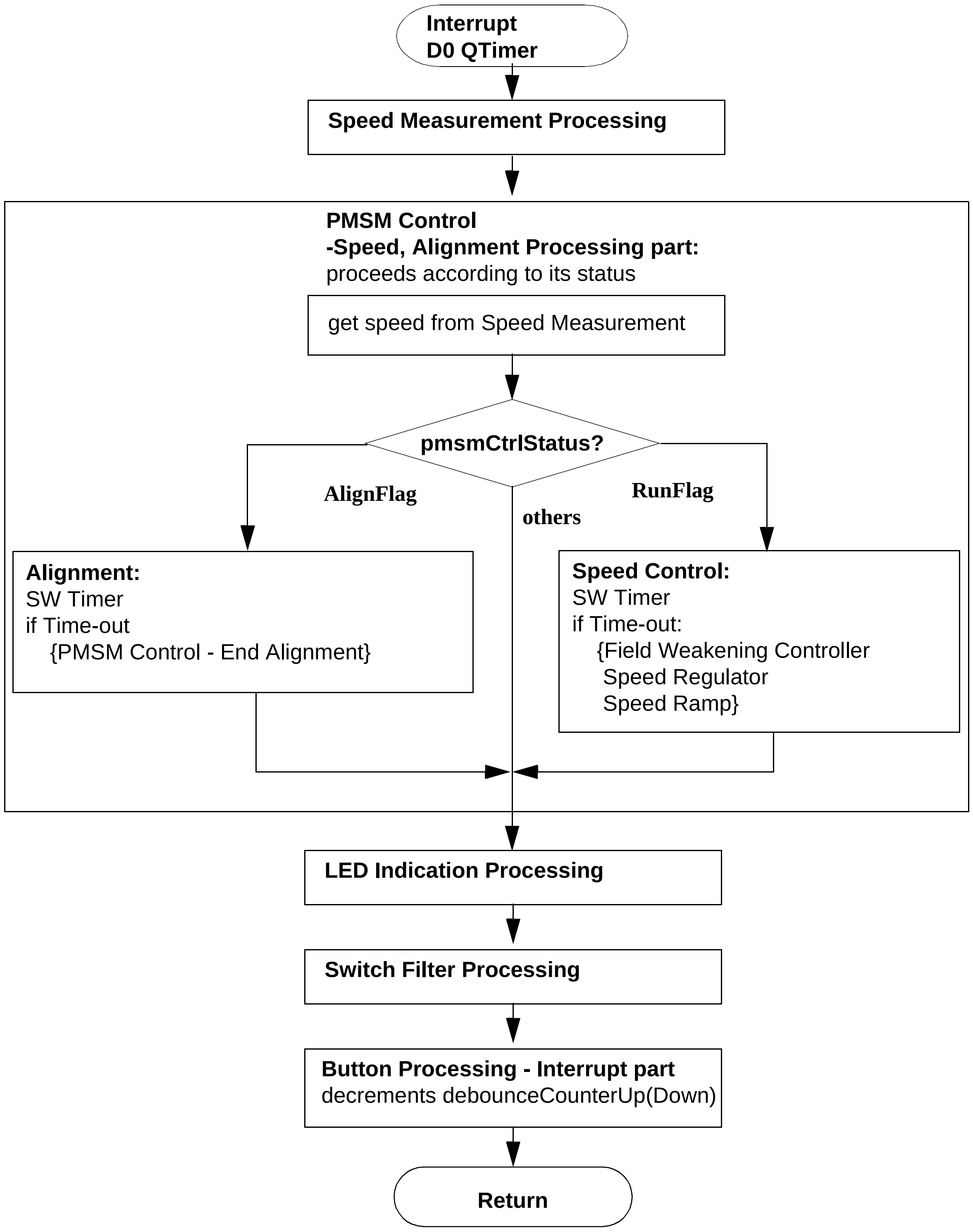

Interrupt
D0 QTimer
Speed Measurement Processing
PMSM Control
-Speed, Alignment Processing part:
proceeds according to its status
get speed from Speed Measurement
pmsmCtrlStatus?
AlignFlag
others
RunFlag
Alignment:
SW Timer
if Time-out
{PMSM Control - End Alignment}
Speed Control:
SW Timer
if Time-out:
{Field Weakening Controller
Speed Regulator
Speed Ramp}
LED Indication Processing
Switch Filter Processing
Button Processing - Interrupt part
decrements debounceCounterUp(Down)
Return

Figure 6-22. SW Flow Chart - General Overview III

6.2 Data Flow

The PM Synchronous motor Vector Control drive control algorithm is described with the data flow shown in **Figure 6-23**. and **Figure 6-24**. The variables and constants description should be clear from their naming, but are also listed in **Section 8.1**.

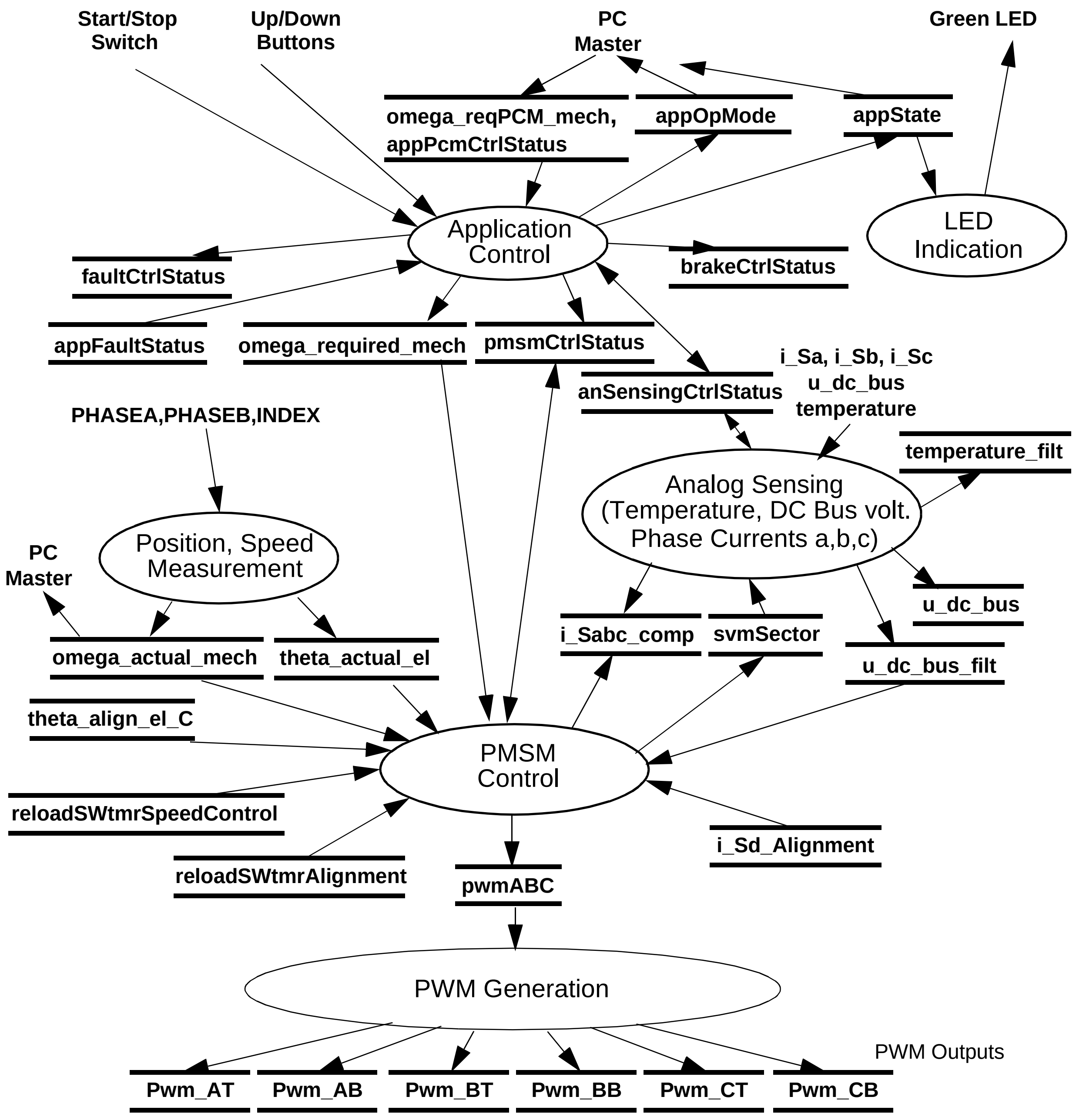


Figure 6-23. Data Flow - Part 1

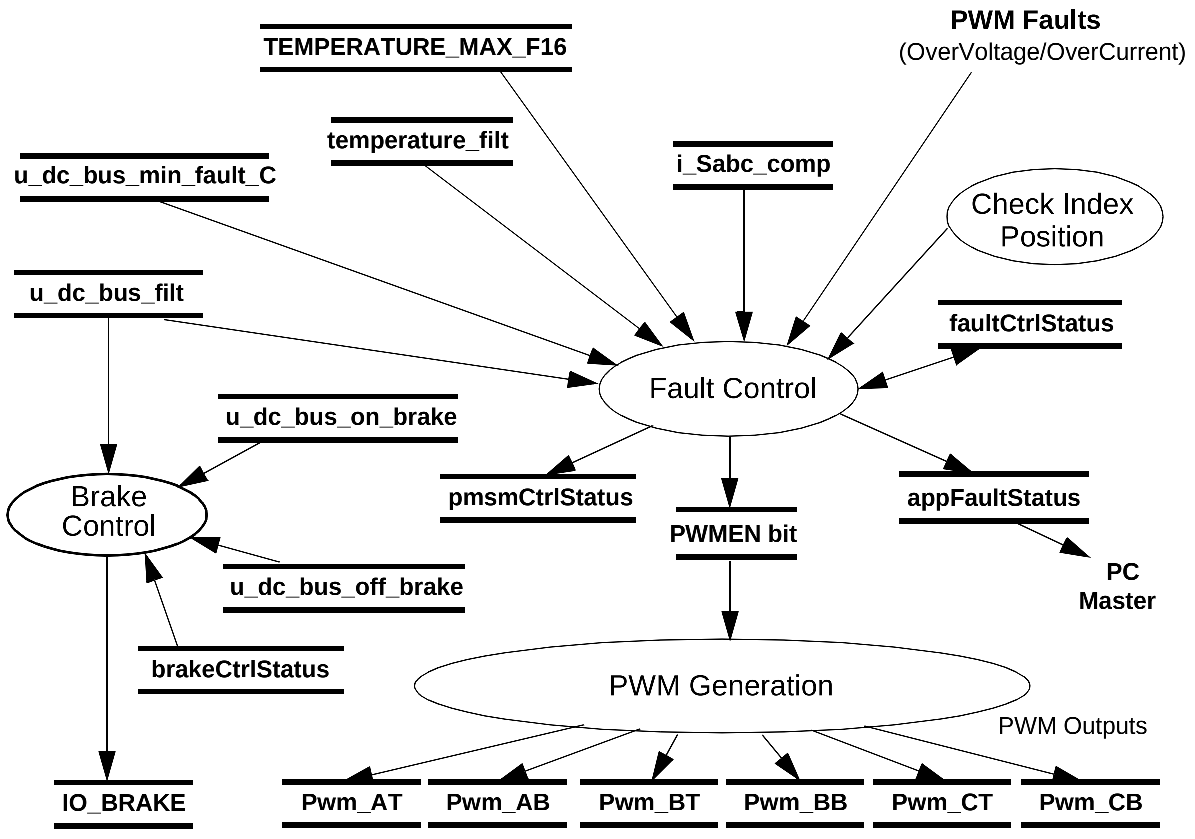


Figure 6-24. Data Flow - Part 2

The data flows consists of processes described in following sections.

6.2.1 Application Control Process

The Application Control process is the highest SW level which precedes settings for other SW levels.

The process state is determined by the variable **appState.**

The application can be controlled as follows:

- Manually
- From PC Master

The Manual/PC Master application operating mode is determined by setting of **appOpMode.**

For Manual control the input of this process is RUN/STOP switch and UP/DOWN buttons.

The PC Master communicates via **omega_reqPCM_mech** which is the required angular speed from PC Master, **appPcmCtrlStatus** which consists of flags **StartStopCtrl** for Start/Stop, **RequestCtrl** for changing of application operating mode **appOpMode** to Manual or PC Master Control. The **appFaultStatus**

The other processes are controlled by setting of **pmsmCtrlStatus, omega_required_mech, appPcmCtrlStatus, brakeCtrlStatus, faultCtrlStatus,**

6.2.2 LED Indication Process

This process controls the LED flashing according to **appState**.

6.2.3 Analog Sensing Process

The Analog sensing process handles sensing, filtering and correction of analog variables (phase currents, temperature, DC Bus voltage).

6.2.4 Position and Speed Measurement Process

The Position and Speed Measurement process gives mechanical angular speed **omega_actual_mech** and electrical position **theta_actual_el**.

6.2.5 PMSM (PM Synchronous Motor) Control Process

The PMSM (PM Synchronous Motor) Control process provides most of motor control functionality.

The **Figure 6-25** shows the data flow inside of the process PMSM Control. It shows essential subprocesses of the process. They are Sine, Cosine Transformations, Current Control, Speed, Alignment Control and Field Weakening.

The Sine and Cosine Transformations generates **sinCos_theta_el** with the components **sine**, **cosine** according to electrical position **theta_actual_el**. It is provided in the look-up table.

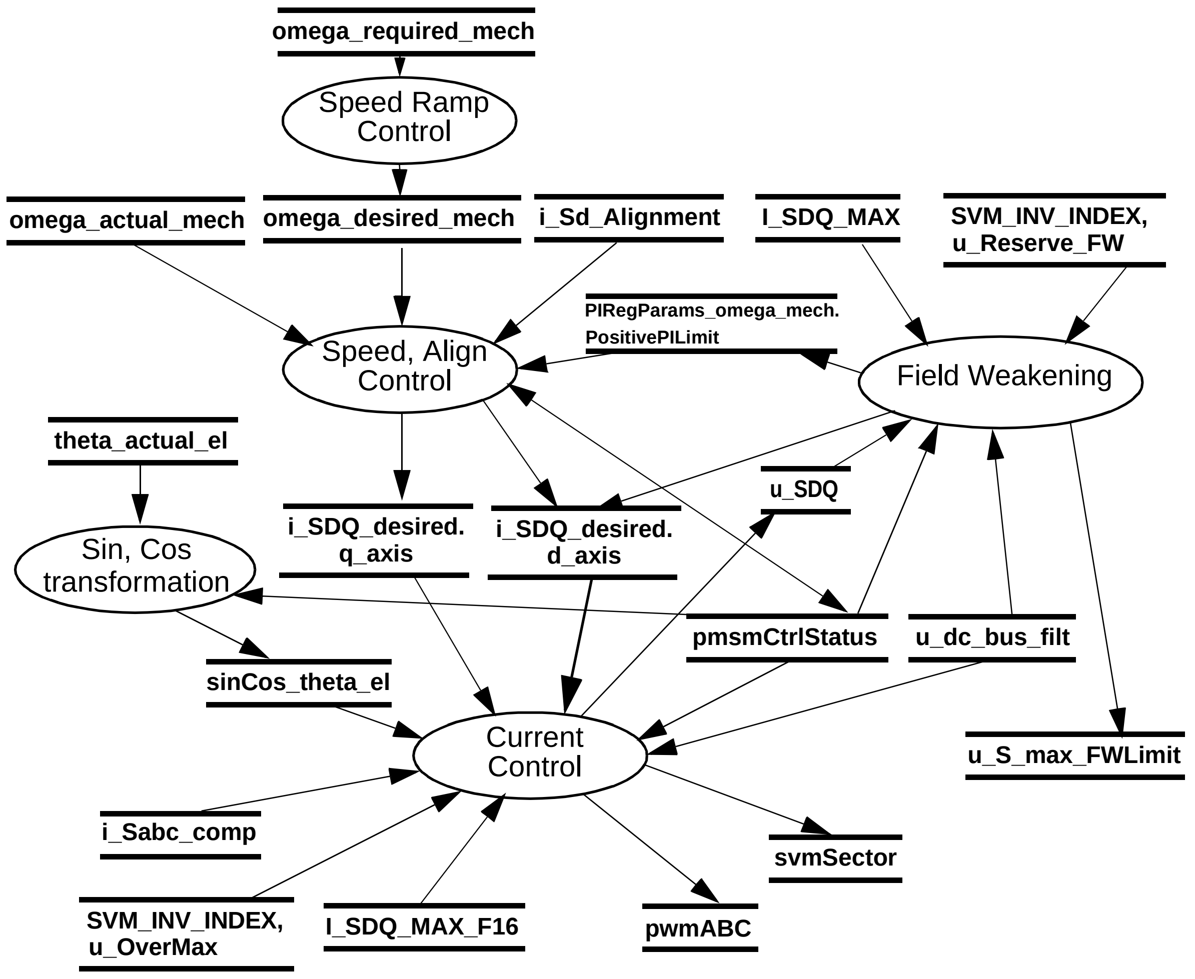


Figure 6-25. Data Flow - PMSM Control

6.2.5.1. Current Control Process

The data flow inside of the process Current Control is shown in detail in **Figure 6-26**. The measured phase currents **i_Sabc_comp** are transformed to **i_SDQ_lin** using **sinCos_theta_el** **(see 4.3.2 Vector Control Transformations)**. Both d and q components are regulated in independent PI regulators to **i_SDQ_desired** values. The outputs of the regulators are **u_SDQ_lin**.

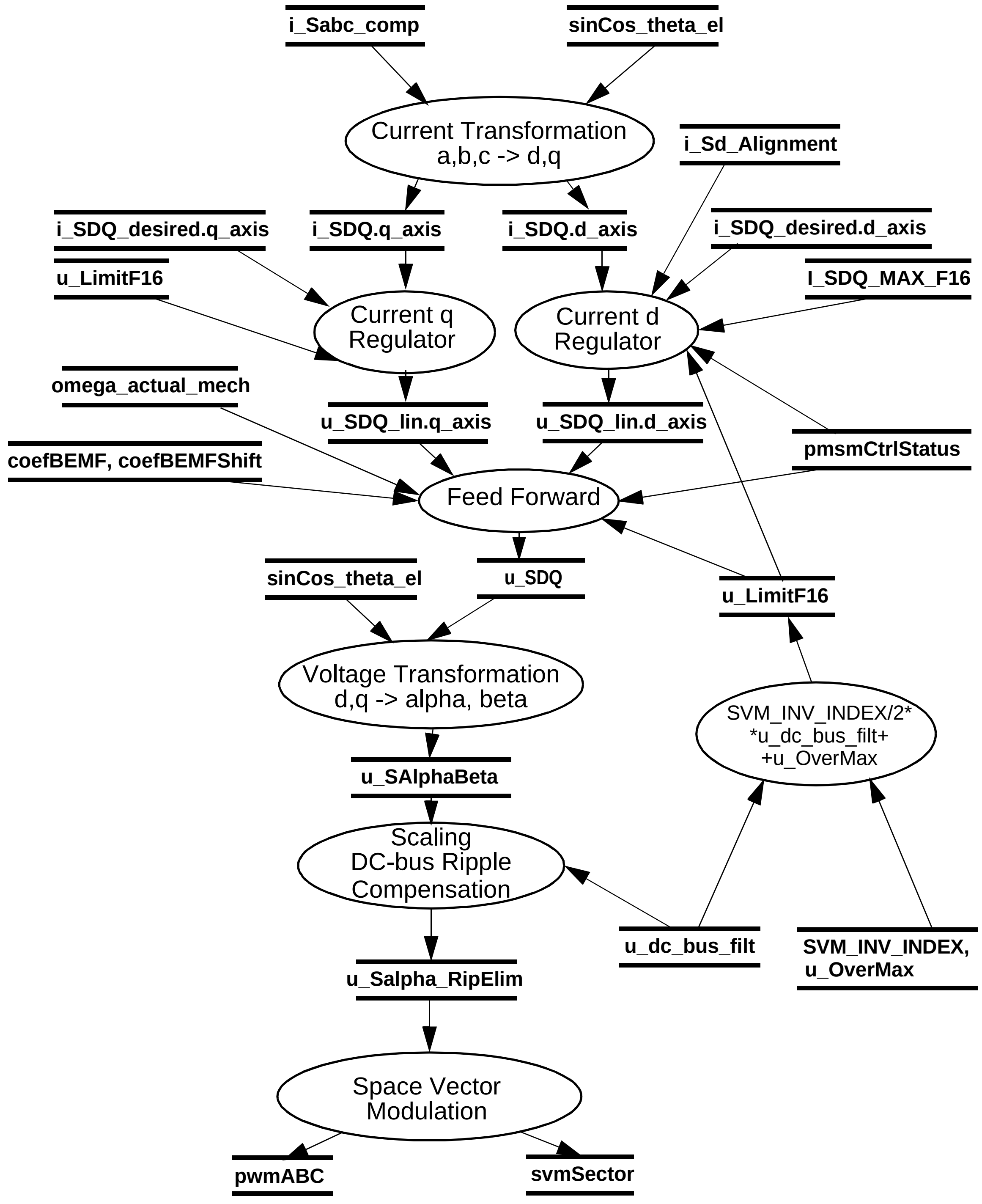


Figure 6-26. Data Flow - PMSM Control - Current Control

The Feed Forward process provides following calculation:

u_SDQ.q_axis = coefBEMF*omega_actual_mech*2^coefBEMFShft + u_SDQ_lin.q_axis

u_SDQ.d_axis = u_SDQ_lin.d_axis

The **u_SDQ** voltages are transformed to **u_SAlphaBeta** (see Section 4.3.2) by Voltage Transformation process. The Scaling DC-bus Ripple Compensation block scales **u_SAlphaBeta** according **u_dc_bus_filt** to **u_Salpha_RipElim** (described in the SDK doc for svmElimDcBusRip function). The Space Vector modulation process generates duty cycle **pwmABC** and **svmSector** according to **u_Salpha_RipElim**.

u_LimitF16 is a voltage limit for current controllers. The **u_OverMax** constant is used to increase the limitation of **u_SDQ** voltages over maximum SVM_INV_INDEX/2*u_dc_bus_filt determined by DC Bus voltage and space vector modulation. Although the **pwmABC** will be limited by Space Vector Modulation process functions, the reserve is used for Field Weakening controller dynamics (in the stable state the **u_SDQ** voltages vector will not exceed **u_S_max_FWLimit** (see Section 6.2.5.4.)).

6.2.5.2. Speed Ramp

The process generates angular speed **omega_desired_mech** from angular speed **omega_required_mech** l with a linear ramp. The speed ramp is implemented in order not to saturate speed regulator during acceleration.

6.2.5.3. Speed, Alignment Control Process

The process controls **i_SDQ_desired.q_axis** current according to PMSM Control Process Status.

In case of alignment status, it sets **i_SDQ_desired.d_axis** to **i_Sd_Alignment** and **i_SDQ_desired.q_axis** to 0.

In case of run status, it controls the **omega_actual_mech** speed to **omega_desired_mech** by calculation of PI regulator with **i_SDQ_desired.q_axis** output.

6.2.5.4. Field Weakening Process

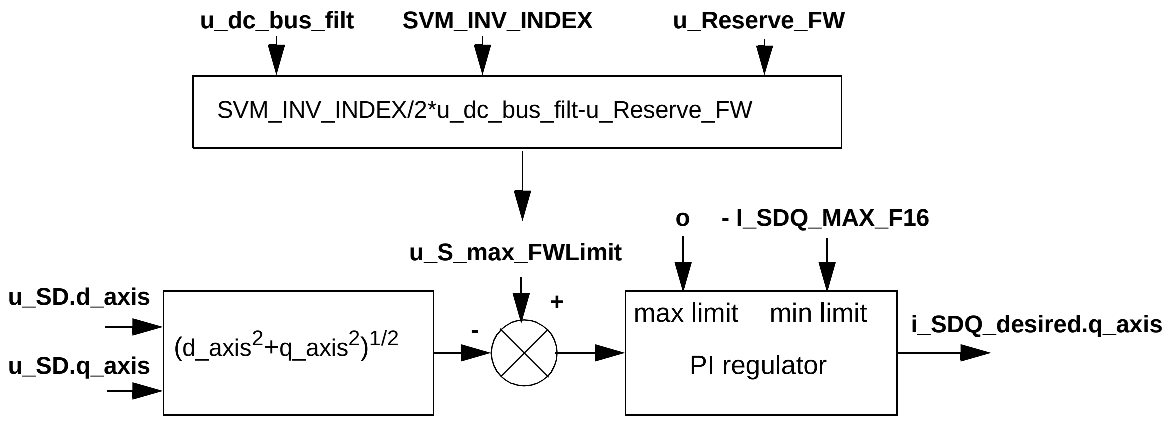


Figure 6-27. Field Weakening Controller

The Field Weakening process provides control of **i_SDQ_desired.q_axis** in order to get higher motor speed by field weakening technique. The control algorithm is shown in Figure 6-27. The **u_S_max_FWLimit** is computed from **u_dc_bus_filt**. The **u_Reserve_FW** is subtracted in order to have some voltage reserve to the maximum SVM_INV_INDEX/2*u_dc_bus_filt determined by DC Bus voltage and space vector modulation. The reserve is used for Field Weakening controller dynamics (in the stable state the **u_SDQ** voltages vector will not exceed **u_S_max_FWLimit** (see Section 6.2.5.4.).

This process also provides voltage limitation $i\_SDQ\_desired.d\_axis^2 + i\_SDQ\_desired.q\_axis^2 < (I\_SDQ\_MAX\_F16)^2$ by setting of:

PIRegParams_omega_mech.PositivePILimit = $(\mathbf{I\_SDQ\_MAX\_F16}^2 - \mathbf{i\_SDQ\_desired.d\_axis}^2)^{1/2}$

PIRegParams_omega_mech.NegativePILimit = $-(\mathbf{I\_SDQ\_MAX\_F16}^2 - \mathbf{i\_SDQ\_desired.d\_axis}^2)^{1/2}$

6.2.6 Brake Control Process

The brake control process maintains DC Bus voltage level via **IO_BRAKE** driver (which controls the brake switch). The voltage comparison levels are **u_dc_bus_on_brake** which is initialized with **U_DCB_ON_BRAKE_MAINS230_F16** or **(U_DCB_ON_BRAKE_MAINS115_F16)** constant according to mains voltage and **u_dc_bus_off_brake** initialized with **U_DCB_OFF_BRAKE_MAINS230_F16 (U_DCB_OFF_BRAKE_MAINS115_F16)**.

6.2.7 PWM Generation Process

The PWM generation process controls the generation of PWM signals, driving the 3-phase invertor.

The input is **pwmABC**, with 3 PWM components scaled to the range <0,1> of type Frac16. The scaling (according to PWM module setting) and the PWM module control (on the DSP) is provided by PWM driver.

6.2.8 Fault Control Process

The Fault Control process checks the Overheating, Undervoltage Overvoltage, Overcurrent and Positionsensing faults.

Overheating and Undervoltage are checked by comparisons **temperature_filt < TEMPERATURE_MAX_F16** and **u_dc_bus_filt < u_dc_bus_min_fault_C**, where **u_dc_bus_min_fault_C** is initialized with **U_DCB_MIN_FAULT_MAINS230_F16** or **U_DCB_MIN_FAULT_MAINS115_F16. The** Positionsensing fault is checked with the Check Index Position process.

The Overvoltage and Overcurrent faults are set in the PWM A Fault interrupt.

6.3 State Diagram

The software can be split into the processes shown in **Section 6.2**.

The state diagrams of the following processes are described below:

- Application Control State Diagram
- PMSM Control State Diagram
- Fault Control State Diagram
- Analog Sensing State Diagram

6.3.1 DSP Initialization

- PWM: DSP initialization
- Application Control: DSP initialization
- PM Synchronous Motor Control: DSP initialization
- Analog Sensing: DSP initialization
- Brake Control: DSP initialization

- Fault Control: DSP initialization
- LED Indication: DSP initialization
- Button Control Initialization
- set manual application operating mode
- enable masked interrupts
- Application Control: Initialization Triggers, which sets all affected processes to Begin App. Initialization state

6.3.2 Application Control State Diagram

The process Application Control is displayed in **Figure 6-28**.

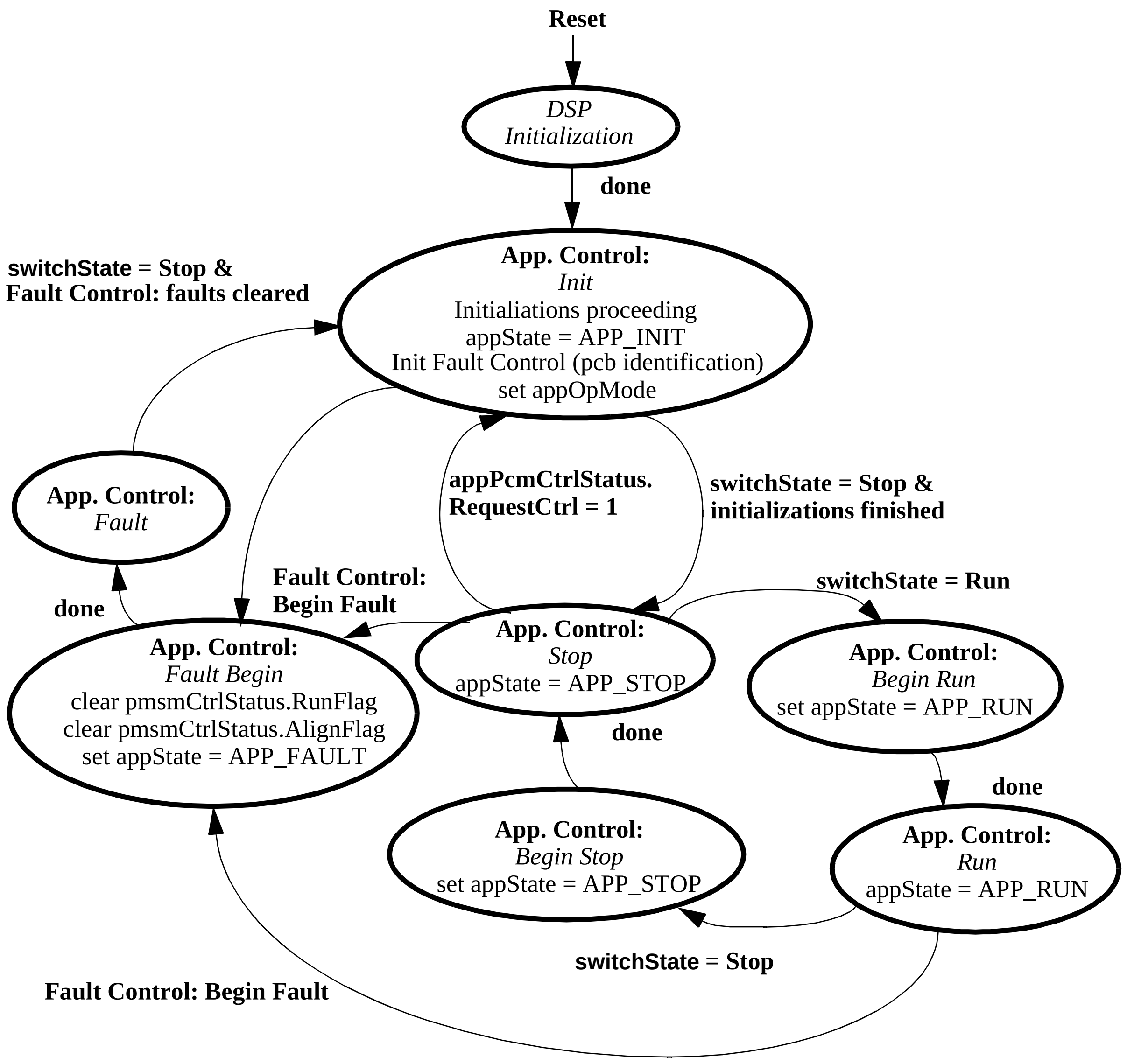


Figure 6-28. State Diagram - Application Control

After reset the DSP Initialization state is entered. The peripheral and variables initialization is provided in this state and the application operating mode **appOpMode** is set to *Manual Control*.

When the state is finished, Application Control Init state follows.As shown in **Figure 6-28** **appState = APP_INIT**, all subprocesses required initializations are proceeding, pcb identification is provided, the PWM is disabled, so there is no voltage applied on motor phases. Whenever **appPcmCtrlStatus.RequestCtrl** flag is set (from PC Master) the application operating mode **appOpMode** is toggled (the application operating mode can only be changed in this state). If the **switchState = Stop** the Application Control entries the *Stop* state.

The **switchState** is set according to manual switch on evm board or PC Master register **AppPcmCtrlStatus.StartStopCtrl** depending on application operating mode.

In the Stop state **appState = APP_STOP**, the PWM is disabled, so there is no voltage applied on motor phases. When **switchState = Run** the Begin Running state is processed. If there is a request for changing of application operating mode **appPcmCtrlStatus.RequestCtrl = 1**, the application Init is entered (the application operating mode request can only be accepted in the Init state or Stop state by transition to the Init state).

In the Begin Running state, all the processes provide settings to the Running state.

In the Running state the PWM is enabled, so voltage is applied on motor phases. The motor is running according to the state of all subprocesses. If **switchState = Stop**, the Stop state is entered.

If Fault Control subprocess Begin Fault state (any fault is detected), the Begin Fault state is entered. It sets **appState = APP_FAULT**, the PWM is disabled and the subprocess PMSM Control is set to Stop. The Fault state can only transit to the Init state, when **switchState = Stop**, and Fault Control subprocess has successfully cleared all faults.

6.3.3 PMSM Control State Diagram

A State Diagram of the Commutation Control process is illustrated in **Figure 6-29**.

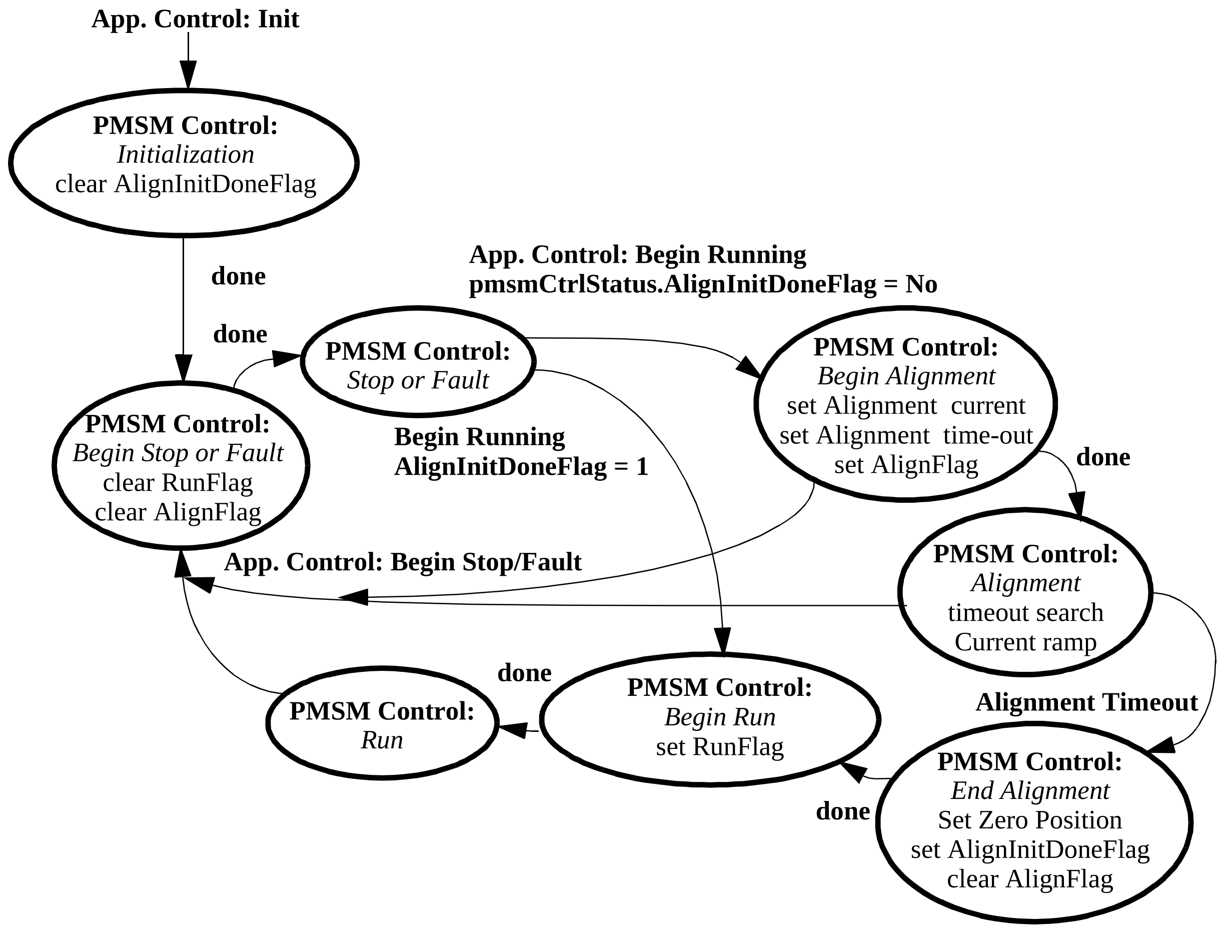

App. Control: Init
PMSM Control: Initialization clear AlignInitDoneFlag
done
done
PMSM Control: Stop or Fault
App. Control: Begin Running pmsmCtrlStatus.AlignInitDoneFlag = No
PMSM Control: Begin Alignment set Alignment current set Alignment time-out set AlignFlag
PMSM Control: Begin Stop or Fault clear RunFlag clear AlignFlag
Begin Running AlignInitDoneFlag = 1
done
PMSM Control: Alignment timeout search Current ramp
App. Control: Begin Stop/Fault
done
PMSM Control: Run
PMSM Control: Begin Run set RunFlag
Alignment Timeout
PMSM Control: End Alignment Set Zero Position set AlignInitDoneFlag clear AlignFlag
done

Figure 6-29. State Diagram - PMSM Control

When Application Control initializes, the PMSM Control subprocess initialization state is entered. The **AlignInitDoneFlag** is cleared which means that Alignment needs to proceed. The next PMSM Control state is *Begin Stop or Fault*. The **RunFlag** and **AlignFlag** are cleared and Stop or Fault state is entered. When Application Control: Begin Run the PMSM Control subprocess enters *Begin Alignment* or *Begin Run* state. It depends on if alignment initialization has already proceeded or not (flagged by **AlignInitDoneFlag**). The Alignment state is necessary for setting of zero position of position sensing. (see **Section 4.3.8**). In the state *Begin Alignment*, the Alignment current and duration are set (the Alignment is provided by setting of desired current for **d_axis** to **i_Sd_Alignment** and **q_axis** to 0). The Alignment state provides current control and time-out search. When Alignment Time-out occurs, the *End Alignment* is entered. In that state The Position Sensing Zero Position is set, so the position sensor is aligned with the real vector of the rotor flux. When the End Alignment state is done, the PMSM Control goes to a regular Run state, where the motor is running according to required speed. If Application Control state is set to *Begin Stop or Begin Fault*, the PMSM Control goes to the *Begin Stop or Fault* and then to the *Stop or Fault*.

6.3.4 Fault Control State Diagram

The State Diagram of the Fault Control subprocess is displayed in **Figure 6-30**. After the initialization the fault conditions are searched. If any fault occurs the **appFaultStatus** variable is set according to detected error and PWM is switched on (PWMEN bit = 0). Then the Fault state is entered (this state also causes Application Control: Fault state). If the faults are successfully cleared, this is signalled to the Application Control process. The Fault state is left when Application Control Init state is entered.

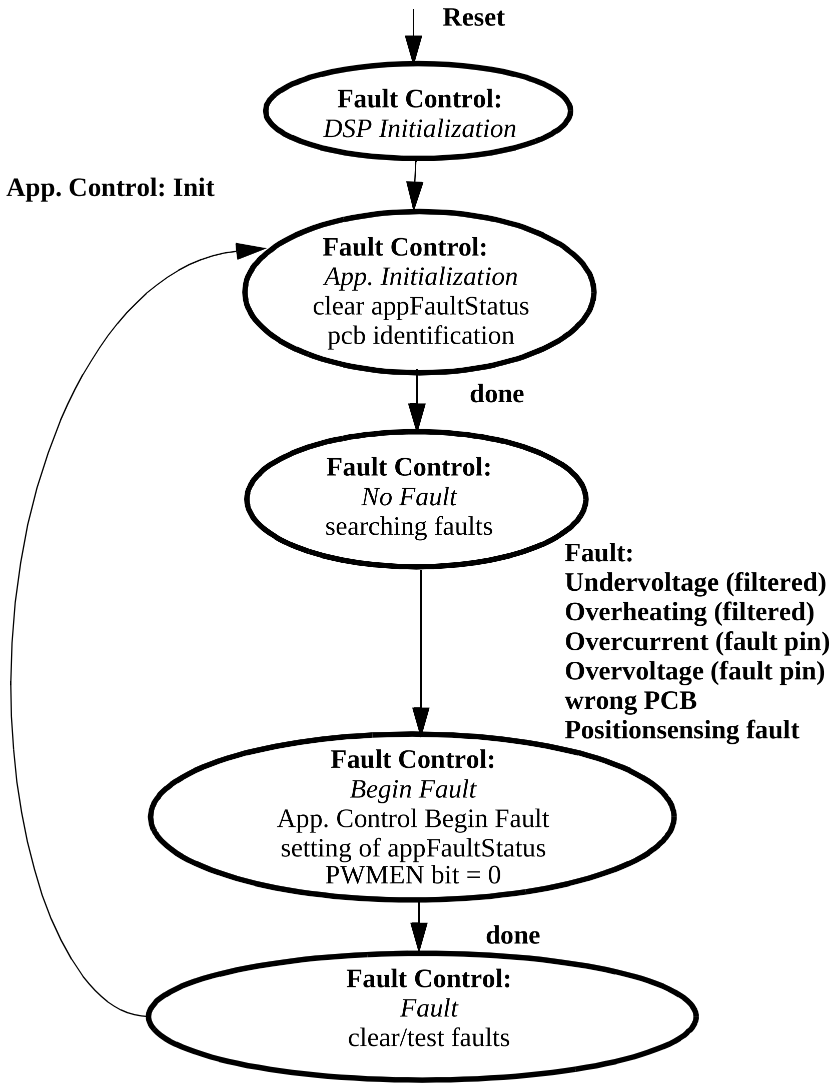


Figure 6-30. State Diagram Fault Control

6.3.5 Analog Sensing State Diagram

The State Diagram of the Analog Sensing subprocess is displayed in **Figure 6-31**. The DSP Initialization state initializes HW modules like ADC, synchronization with PWM, etc. In the *Begin Init* the Initialization is started, so the variables for initialization sum and **InitDoneFlag** are cleared. In the *Init Proceed* state the temperature, DC Bus voltage and phase current samples are sensed and summed. After required analog sensing, init samples are sensed, and the *Init Finished* state is entered. There the samples average is calculated from the sum (divided by number of analog sensing init samples). According to the phase currents average value the phase current offsets are initialized.

Then all variables sensing is initialized and the state *Init Done* is entered, so the variables from analog sensing are valid for other processes. In this state temperature and DC Bus voltage are filtered in first order filters.

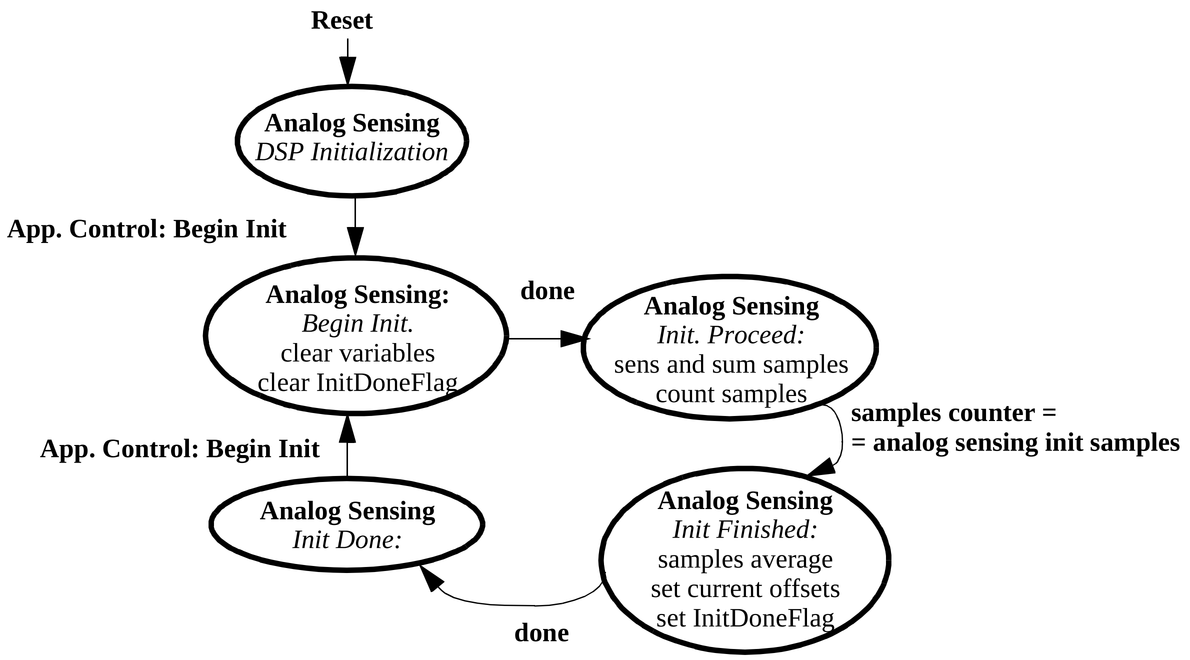


Figure 6-31. State Diagram - Analog Sensing

7. SDK Implementation

The Motorola Embedded SDK is a collection of APIs, libraries, services, rules and guidelines. This software infrastructure is designed to let DSP5680x software developers create high-level, efficient, portable code. This chapter describes how the PM Synchronous motor vector control application is written under SDK.

7.1 Drivers and Library Function

The PM Synchronous motor vector control application uses the following drivers:

- ADC driver
- Quadrature Timer driver
- Quadrature Decoder driver
- PWM driver
- LED driver
- SCI driver
- PC Master driver
- Switch driver (only for DSP56F805EVM &DSP56F807EVM)
- Brake driver

All drivers are included in *bsp.lib* library.

The PM motor control application uses the following library functions:

- cptrfmClarke (Clark transformation, *mcfunc.lib* library)
- cptrfmPark (Park transformation, *mcfunc.lib* library)
- cptrfmParkInv (Inverse Park transformation, *mcfunc.lib* library)
- mcElimDcBusRip (DC bus ripple elimination, *mcfunc.lib* library)
- mcPwmIct (3-ph sinewave modulation, *mcfunc.lib* library)
- rampGetValue (ramp generation, *mcfunc.lib* library)
- switchcontrol (switch control, *mcfunc.lib* library)
- boardId (board identification,*bsp.lib* library)

7.2 Appconfig.h File

The purpose of the *appconfig.h* file is to provide a mechanism for overwriting default configuration settings which are defined in the *config.h* file.

There are two *appconfig.h* files The first *appconfig.h* file is dedicated for External RAM (*..\ConfigExtRam* directory) and second one is dedicated for FLASH memory (*..\ConfigFlash* directory). In case of PM Synchronous motor vector control application both files are identical with the following exceptions:

- The *appconfig.h* for ExtRAM target contains PC Master Recorder buffer 25000 samples long, while *appconfig.h* for Flash target contains PC Master Recorder buffer only 100 samples long. This is due to the limited Flash memory size.
- The *appconfig.h* for DSP56F805EVM and DSP56F807EVM contains the definition of a switch driver, while the *appconfig.h* for DSP56F803EVM does not.

The *appconfig.h* file can be divided into two sections. The first section defines which components of SDK libraries are included to application, the second part of overwrites standard setting of components during their initialization.

7.3 Drivers Initialization

Each peripheral on the DSP chip or on the EVM board is accessible through a driver. The driver initialization of all used peripherals is described in this chapter. For detailed description of drivers see the document **Embedded SDK Targeting Motorola DSP5680x Platform**

To use the driver the following step must be done:

- include the driver support to the *appconfig.h*
- fill configuration structure in the application code for specific drivers (depends on driver type)
- initialize the configuration setting in *appconfig.h* for specific drivers (depends on driver type)
- call the *open* (create) function

Access to individual driver functions is provided by the *ioctl* function call.

7.4 Interrupts

The SDK serves the interrupt routines calls and automatically clears interrupt flags. The user defines the callback functions called during interrupts. The callback functions are assigned during the drivers initialization - `open()`. The callback function assignment is defined as one item of initialization structure which is used as a parameter of function `open()`. Some drivers define the callback function in *appconfig.h* file.

7.5 PC Master

PC Master was designed to provide the debugging, diagnostic and demonstration tool for development of algorithms and applications. It consists of components running on PCs and parts running on the target DSP, connected by an RS232 serial port. A small program is resident in the DSP that communicates with the PC-Master software to parse commands, return status information to the PC, and process control information from the PC. The PC-Master software executing on a PC uses Microsoft Internet Explorer as a user interface to the PC.

The PC Master application is part of the Motorola Embedded SDK and may be selectively installed during SDK installation.

To enable the PC Master operation on the DSP target board application, the following lines must be added to the appconfig.h file:

```
#define INCLUDE_SCI          /* SCI support */
#define INCLUDE_PCMASTER     /* PC Master support */
```

It automatically includes the SCI driver and installs all necessary services.

The baud rate of the SCI communication is 9600Bd. It is set automatically by the PC Master driver.

A detailed PC Master description is provided by the **PC Master User Manual**.

The actions controlled by the PC-Master are:

- Set PC Master Mode of the motor control system
- Set Manual Mode of the motor control system
- Start the motor
- Stop the motor
- Set the Required Speed of the motor

Variables read by the PC-Master software as a default and displayed to the user are:

- Required Speed of the motor
- Actual Speed of the motor
- Application status - Init/Stop/Run/Fault
- DC Bus voltage level
- Identified line voltage
- Fault Status - No_Fault/Overvoltage/Overcurrent/Undervoltage/Overheating
- Identified Power Stage

The profiles of required and actual speeds together with the desired I_d, I_q currents can be seen at a Speed Scope window.

The Speed Scope PC Master displays Recorders windows. Due to the limited on-chip memory, the Recorder can be used **ONLY** when the application is running from **External RAM**. The length of the recorded window may be set in “Recorder Properties” => bookmark “Main” => “Recorded Samples”. The dedicated memory space is defined in *appconfig.h* file of the ExtRAM target. The recorder samples are taken each 125 µsec in rate of PWM frequency.

The following speed recorder can be captured:

- Required Speed
- Actual Speed
- Desired I_d Current
- Desired I_q Current

8. Implementation Notes

Described here are any noteworthy pieces of information related to the final implementation of the preceding design. Noted here are the actual design files used, any deviations from the design requirements, special tricks used in the coding, optimizations performed etc.

8.1 Analogue Value Scaling

The PM synchronous motor vector control application uses a fractional representation for all real quantities except time.

The C-language standard does not have any fractional variable type defined. Therefore fractional operations are provided by Code Warrior intrinsics functions (e.g. mult_r()). As a substitution of the fractional type variables, the application uses types Frac16, resp. Frac32. These are in fact defined as integer 16-bit signed variables, resp. integer 32-bit signed variables. The difference between Frac16 and pure integer variables is that Frac16 and Frac32 declared variables should only be used with fractional operations (intrinsics functions).

A recalculation from real to a fractional form and Frac16, Frac32 value is given by following equations:

$$\text{Frac16 Value} = 32768 \cdot \frac{\text{Real Value}}{\text{Real quantity range}} \qquad \text{(EQ 8-1.)}$$

in case of Frac16 16-bit signed value and

$$\text{Frac32 Value} = 2147483648 \cdot \frac{\text{Real Value}}{\text{Real quantity range}} \qquad \text{(EQ 8-2.)}$$

in case of Frac32 32-bit signed value.

$$\text{Fractional Value} = \frac{\text{Real Value}}{\text{Real quantity range}} \qquad \text{(EQ 8-3.)}$$

in case of fractional form.

8.1.1 Voltage Scaling

The voltage scaling results from sensing circuit of used hardware (for details see **MEMC3BLDCPSUM/D - 3-phase Brushless DC High Voltage Power Stage** manual). The following table shows all voltage variables and constants.

Table 8-2. Voltage Variables and Constants with their Scaling

Name	Type	Real/ Fract	Value	Range	Note
VOLT_RANGE_MAX	C	R	407 V	-	max. measurable voltage (given by HW)
VOLT_RANGE_MIN	C	R	-407 V	-	min. measurable voltage (given by HW)
U_DC_BUS_NET230	C	R	310 V	<-407;407)	nominal DC Bus voltage at 230 V
U_DC_BUS_NET230_F16	C	F	FRAC1 6	<-1;1)	nominal DC Bus voltage at 230 V
U_DCB_MIN_FAULT_NET230	C	R	210 V	<-407;407)	under voltage fault limit at 230 V
U_DCB_MIN_FAULT_NET230_F32	C	F	FRAC3 2	<-1;1)	under voltage fault limit at 230 V
U_DC_BUS_NET115	C	R	115 V	<-407;407)	nominal DC Bus voltage at 115 V
U_DC_BUS_NET115_F16	C	F	FRAC1 6	<-1;1)	nominal DC Bus voltage at 115 V
U_DC_BUS_NET115	C	R	160 V	<-407;407)	nominal DC Bus voltage at 115 V
U_DC_BUS_NET115_F16	C	F	FRAC1 6	<-1;1)	nominal DC Bus voltage at 115 V
U_DCB_MIN_FAULT_NET115	C	R	105 V	<-407;407)	under voltage fault limit at 115 V
U_DCB_MIN_FAULT_NET115_F32	C	F	FRAC3 2	<-1;1)	under voltage fault limit at 115 V
U_DC_BUS_MAX_NET115	C	R	220 V	<-407;407)	max. DC Bus voltage at 115 V
U_DCB_THRESHOLD_NET115	C	R	220 V	<-407;407)	threshold DC Bus voltage for 115 V
U_DCB_THRESHOLD_NET115_F32	C	F	FRAC3 2	<-1;1)	threshold DC Bus voltage for 115 V
U_DCB_ON_BRAKE_NET230	C	R	385 V	<-407;407)	DC Bus voltage upper limit for brake at 230 V

Table 8-2. Voltage Variables and Constants with their Scaling

Name	Type	Real/ Fract	Value	Range	Note
U_DCB_ON_BRAKE_NET230_F16	C	F	FRAC16	<-1;1)	DC Bus voltage upper limit for brake at 230 V
U_DCB_OFF_BRAKE_NET230	C	R	370 V	<-407;407)	DC Bus voltage lower limit for brake at 230 V
U_DCB_OFF_BRAKE_NET230_F16	C	F	FRAC16	<-1;1)	DC Bus voltage lower limit for brake at 230 V
U_DCB_ON_BRAKE_NET115	C	R	200 V	<-407;407)	DC Bus voltage upper limit for brake at 115 V
U_DCB_ON_BRAKE_NET115_F16	C	F	FRAC16	<-1;1)	DC Bus voltage upper limit for brake at 115 V
U_DCB_OFF_BRAKE_NET115	C	R	190 V	<-407;407)	DC Bus voltage lower limit for brake at 115 V
U_DCB_OFF_BRAKE_NET115_F16	C	F	FRAC16	<-1;1)	DC Bus voltage lower limit for brake at 115 V
U_S_MAX_FW_LIMIT_NET230	C	R	130 V	<-407;407)	voltage limit for field weakening at 230 V
U_S_MAX_FW_LIMIT_NET230_F16	C	F	FRAC16	<-1;1)	voltage limit for field weakening at 230 V
U_S_MAX_FW_LIMIT_NET115	C	R	65 V	<-407;407)	voltage limit for field weakening at 115 V
U_S_MAX_FW_LIMIT_NET115_F16	C	F	FRAC16	<-1;1)	voltage limit for field weakening at 115 V
u_dc_bus	V	F	-	<-1;1)	DC Bus voltage
u_dc_bus_nominal_C	V	F	-	<-1;1)	nominal DC Bus voltage
u_dc_bus_filt	V	F	-	<-1;1)	filtered DC Bus voltage
u_SDQ	V	F	-	<-1;1)	stator voltage in DQ coordinates
u_SDQ_lin	V	F	-	<-1;1)	linear portion of stator voltage in DQ
u_SDQ_FeedForw	V	F	-	<-1;1)	feed forward portion of stator voltage in DQ
u_SAlphaBeta	V	F	-	<-1;1)	stator voltage in α,β
u_S_max_FWLimit	V	F	-	<-1;1)	max. voltage for field weakening
u_Reserve_FW	V	F	-	<-1;1)	vector reserve for field weakening

Table 8-2. Voltage Variables and Constants with their Scaling

Name	Type	Real/ Fract	Value	Range	Note
u_OverMax	V	F	-	<-1;1)	voltage reserve over maximal
u_dc_bus_min_fault_C	V	F	-	<-1;1)	under voltage
u_dc_bus_on_brake	V	F	-	<-1;1)	DC Bus voltage brake on
u_dc_bus_off_brake	V	F	-	<-1;1)	DC Bus voltage brake off

Notes: Type: C - constant, V - variable; **Real/Fract:** R - real quantity, F - fractional quantity; **Value:** real value, FRAC16 - equation **(EQ 8-1.)**, FRAC32 - equation **(EQ 8-2.)**

8.1.2 Current Scaling

The current scaling also results from sensing circuit of used hardware (for details see **MEMC3BLDCPSUM/D - 3-phase Brushless DC High Voltage Power Stage** manual). The following table shows all current variables and constants.

Table 8-3. Current Variables and Constants with Their Scaling

Name	Type	Real/ Fract	Value	Range	Note
CURR_MAX	C	R	2.93 A	-	max. measurable current (given by HW)
CURR_MIN	C	R	-2.93 A	-	min. measurable current (given by HW)
CURR_RANGE_MAX	C	R	5.86 A	-	max. current range limit
CURR_RANGE_MIN	C	R	-5.86 A	-	min. current range limit
I_S_MAX_EFFECTIVE	C	R	0.55 A	<-2.93;2.93>	max. stator current effective value
I_SDQ_MAX	C	R	0.78 A	<-2.93;2.93>	max. stator current amplitude
I_SDQ_MAX_F16	C	F	FRAC16	<-1;1)	max. stator current amplitude
I_SD_ALIGNMENT	C	R	0.55 A	<-2.93;2.93>	alignment current
I_SD_ALIGNMENT_F16	C	F	FRAC16	<-1;1)	alignment current
i_Sabc_comp	V	F	-	<-1;1)	3-phase stator currents compensated
i_SAlphaBeta_comp	V	F	-	<-1;1)	stator current in 2-phase system α,β
i_SDQ	V	F	-	<-1;1)	actual stator current in DQ coordinates
i_SDQ_desired	V	F	-	<-1;1)	desired stator current in DQ coordinates

Table 8-3. Current Variables and Constants with Their Scaling (Continued)

Name	Type	Real/ Fract	Value	Range	Note
i_Sd_Alignment	V	F	-	<-1;1)	required alignment current

Notes: Type: C - constant, V - variable; **Real/Fract:** R - real quantity, F - fractional quantity; **Value:** real value, FRAC16 - equation **(EQ 8-1.)**, FRAC32 - equation **(EQ 8-2.)**

8.1.2.1. Temperature Scaling

As shown in **Section 8.1.2.1.**, the temperature variable doesn’t have a linear dependency.

8.2 Other quantities

8.2.1 Position Scaling

Position scaling is described in **Section 4.3.3.1.**

Table 8-4. Position Variables and Constants with Their Scaling

Name	Type	Real/ Fract	Value	Range	Note
THETA_ALIGNMENT_EL	C	R	0°	<-180°;180°>	alignment angle
THETA_ALIGNMENT_EL_F16	C	F	FRAC 16	<-1;1)	alignment angle
PULSES_PER_REVOLUTION	C	R	1024	-	encoder pulses per revolution
MOTOR_POLE_PAIRS	C	R	3	-	motor pole pairs
PULSES_THETA_COEF_SCALE_ W16	C	F	see code	<-32768;32767>	position scale constant
PULSES_THETA_COEF_UF16	C	F	see code	<0;2)	position scale constant
theta_actual_el	V	F	-	<-1;1)	electrical rotor position
theta_align_el_C	V	F	-	<-1;1)	alignment angle

Type: C - constant, V - variable; **Real/Fract:** R - real quantity, F - fractional quantity; **Value:** real value, FRAC16 - equation **(EQ 8-1.)**, FRAC32 - equation **(EQ 8-2.)**

8.2.2 Speed Scaling

Position scaling is described in section **Section 4.3.3.2.**.

Table 8-5. Speed Variables and Constants with Their Scaling

Name	Type	Real/ Fract	Value	Range	Note
OMEGA_RANGE_MAX	C	R	6000 rpm	-	max. angular speed range limit (chosen)
OMEGA_RANGE_MIN	C	R	- 6000 rpm	-	min. angular speed range limit (chosen)
OMEGA_MAX_MAINS230	C	R	3000 rpm	<-6000;6000>	max. allowed angular speed at 230 V
OMEGA_MAX_MAINS230_F16	C	F	FRAC16	<-1;1)	max. allowed angular speed at 230 V
OMEGA_INCREMENT_MAINS230	C	R	100 rpm	<-6000;6000>	angular speed increment for button at 230 V
OMEGA_INCREMENT_MAINS230_F16	C	F	FRAC16	<-1;1)	angular speed increment for button at 230 V
OMEGA_MIN	C	R	50 rpm	<-6000;6000>	min. allowed speed
OMEGA_MIN_F16	C	F	FRAC16	<-1;1)	min. allowed angular speed
OMEGA_MAX_MAINS115	C	R	1100 rpm	<-6000;6000>	max. allowed angular speed at 115 V
OMEGA_MAX_MAINS115_F16	C	F	FRAC16	<-1;1)	max. allowed angular speed at 115 V
OMEGA_INCREMENT_MAINS115	C	R	50 rpm	<-6000;6000>	angular speed increment for button at 115 V
OMEGA_INCREMENT_MAINS115_F16	C	F	FRAC16	<-1;1)	angular speed increment for button at 115 V
OMEGA_RAMP	C	R	12000 rpm/s	-	angular speed ramp
OMEGA_RAMP_F16	C	F	see code	<-1;1)	angular speed ramp
omega_reqMAX_mech	V	F	-	<-1;1)	max. required speed (absolute value)
omega_reqMIN_mech	V	F	-	<-1;1)	min. required speed (absolute value)
omega_reqPCM_mech	V	F	-	<-1;1)	required speed from PCMaster
omega_increment_mech	V	F	-	<-1;1)	speed increment
omega_required_mech	V	F	-	<-1;1)	required speed set by buttons

Table 8-5. Speed Variables and Constants with Their Scaling (Continued)

Name	Type	Real/ Fract	Value	Range	Note
omega_desired_mech	V	F	-	<-1;1)	desired speed
omega_actual_mech	V	F	-	<-1;1)	actual speed

Notes: Type: C - constant, V - variable; **Real/Fract:** R - real quantity, F - fractional quantity; **Value:** real value, FRAC16 - equation **(EQ 8-1.)**, FRAC32 - equation **(EQ 8-2.)**

8.3 PI Controller Tuning

The application consists of four PI controllers. Two controllers are used for I_d, I_q currents, one for speed control and the last for field weakening. The controller's constants are given by simulation in Mathlab and were experimentally specified. The detailed description of controllers tuning is over this application note.

8.4 Subprocesses Relation and State Transitions

As shown in **Section 6.2** and **Section 6.3**, the SW is split into subprocesses according to their functionality. The code of the application is designed in order to be able to extract individual processes (e.g. Analog Sensing) and use them for customer applications. The C language function names dedicated to each process usually start with the name of the process (e.g **AnalogSensing**InitProceed()). They are also located in one place in the SW, so they can be easily used for other applications.

As can be seen in **Section 6.3**, the processes (subprocesses) state transients have some mutual relations (e.g., Application Control: Begin Initialization is a condition for transient of Analog Sensing process: Init Done to Begin Init state). In the code, the interface between processes is provided via "trigger" functions. This functions name convention is: <ProcessName><State>Trig().

The functionality will be explained on following example. The "trigger" function Process1StateTrig() is called from process1. The transient functions of process2, process3 etc., which need to be triggered by Process1State, are put inside of the Process1StateTrig().

9. DSP Usage

Table 9-6 shows how much memory is needed to run the 3-phase PM Synchronous Vector Control drive in using quadrature encoder. A part of the DSP memory is still available for other tasks.

Table 9-6. RAM and FLASH Memory Usage for SDK2.4 and CW4.1

Memory (in 16 bit Words)	Available DSP56F803 DSP56F805	Available DSP56F807	Used Application + Stack	Used Application without PC Master, SCI
Program FLASH	32K	60K	11520	7740
Data FLASH	4k	8K	617	617
Program RAM	512	2K	36	36
Data RAM	2K	4K	745 + 352 stack	452 + 352 stack

10. References

Design of Brushless Permanent-magnet Motors, J.R. Hendershot JR and T.J.E. Miller, Magna Physics Publishing and Clarendon Press, 1994

Brushless DC Motor Control using the MC68HC708MC4, AN1702/D, John Deatherage and Jeff Hunsinger, Motorola

DSP Evaluation Module Hardware User's Manual, DSP56F803EVMUM/D, Motorola

DSP Evaluation Module Hardware User's Manual, DSP56F805EVMUM/D, Motorola

DSP56F800 16-bit Digital Signal Processor, Family Manual, DSP56F800FM/D, Motorola

DSP56F80x 16-bit Digital Signal Processor, User's Manual, DSP56F801-7UM/D, Motorola

Evaluation Motor Board User's Manual, MEMCEVMBUM/D, Motorola

Web page: *motorola.com/semiconductors/motor*

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

Technical Information Center: 1–800–521–6274

HOME PAGE: http://www.motorola.com/semiconductors/