Application Note

AN2355/D
11/2002

Sensorless BLDC Motor Control on MC68HC908MR32 Software Description

By Libor Prokop and Leos Chalupa
Roznov System Application Laboratory
Roznov, Czech Republic

General Description

This application note describes 3-phase sensorless brushless dc (BLDC) motor control with back-EMF (electromotive force) zero-crossing sensing. The application serves as an example of a sensorless BLDC motor control system using Motorola's MC68HC908MR32 microcontroller unit (MCU). This system can be easily ported to other derivatives of the M68HC908MRx Family. The application note also illustrates the use of dedicated motor control on chip peripherals.

The application note gives a view on brushless dc motor control. It describes the application system concept with sensorless control techniques. The main part of this document gives a detailed description of the software for the BLDC sensorless back-EMF zero crossing application.

For application software use and parameter configuration to a customer motor, refer to the complementary application note entitled *Sensorless BLDC Motor Control on MC68HC908MR32 — Software Porting to Customer Motor* (Motorola document order number AN2356/D). That application note will also help to decide if the software and control method shown is suitable for a specific customer application.

BLDC Motor Control

BLDC Motor Targeted by This Application

The brushless dc motor (BLDC motor) is also referred to as an electronically commutated motor. There are no brushes on the rotor, and commutation is performed electronically at certain rotor positions. The stator magnetic circuit is usually made from magnetic steel sheets. Stator phase windings are inserted in the slots (distributed winding) as shown in **Figure 1**, or it can be wound as one coil on the magnetic pole. Magnetization of the permanent magnets and their displacement on the rotor are chosen in such a way that the back-EMF

(the voltage induced into the stator winding due to rotor movement) shape is trapezoidal. This allows a rectangular shaped 3-phase voltage system (see **Figure 2**) to be used to create a rotational field with low torque ripples.

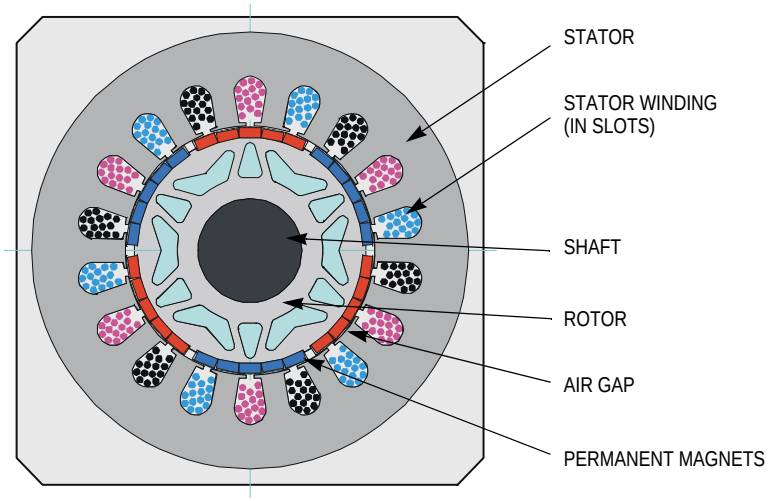

Figure 1. BLDC Motor Cross Section

The motor can have more than just one pole-pair per phase. This defines the ratio between the electrical revolution and the mechanical revolution. The BLDC motor shown has three pole-pairs per phase, which represent three electrical revolutions per one mechanical revolution.

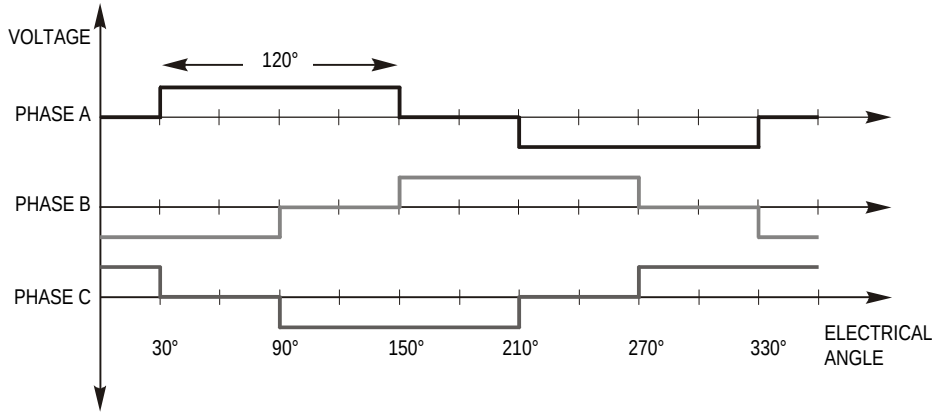

Figure 2. 3-Phase Voltage System

The easy to create rectangular shape of applied voltage ensures the simplicity of control and drive. But, the rotor position must be known at certain angles in order to align the applied voltage with the back-EMF (voltage induced due to movement of the PM). The alignment between back-EMF and commutation events is very important. In this condition, the motor behaves as a dc motor and

runs at the best working point. Thus, simplicity of control and good performance make this motor a natural choice for low-cost and high-efficiency applications.

Figure 3 shows a number of waveforms:

- Magnetic flux linkage
- Phase back-EMF voltage
- Phase-to-phase back-EMF voltage

Magnetic flux linkage can be measured. However, in this case it was calculated by integrating the phase back-EMF voltage (which was measured on the non-fed motor terminals of the BLDC motor). As can be seen, the shape of the back-EMF is approximately trapezoidal and the amplitude is a function of the actual speed. During speed reversal, the amplitude changes its sign and the phase sequence changes.

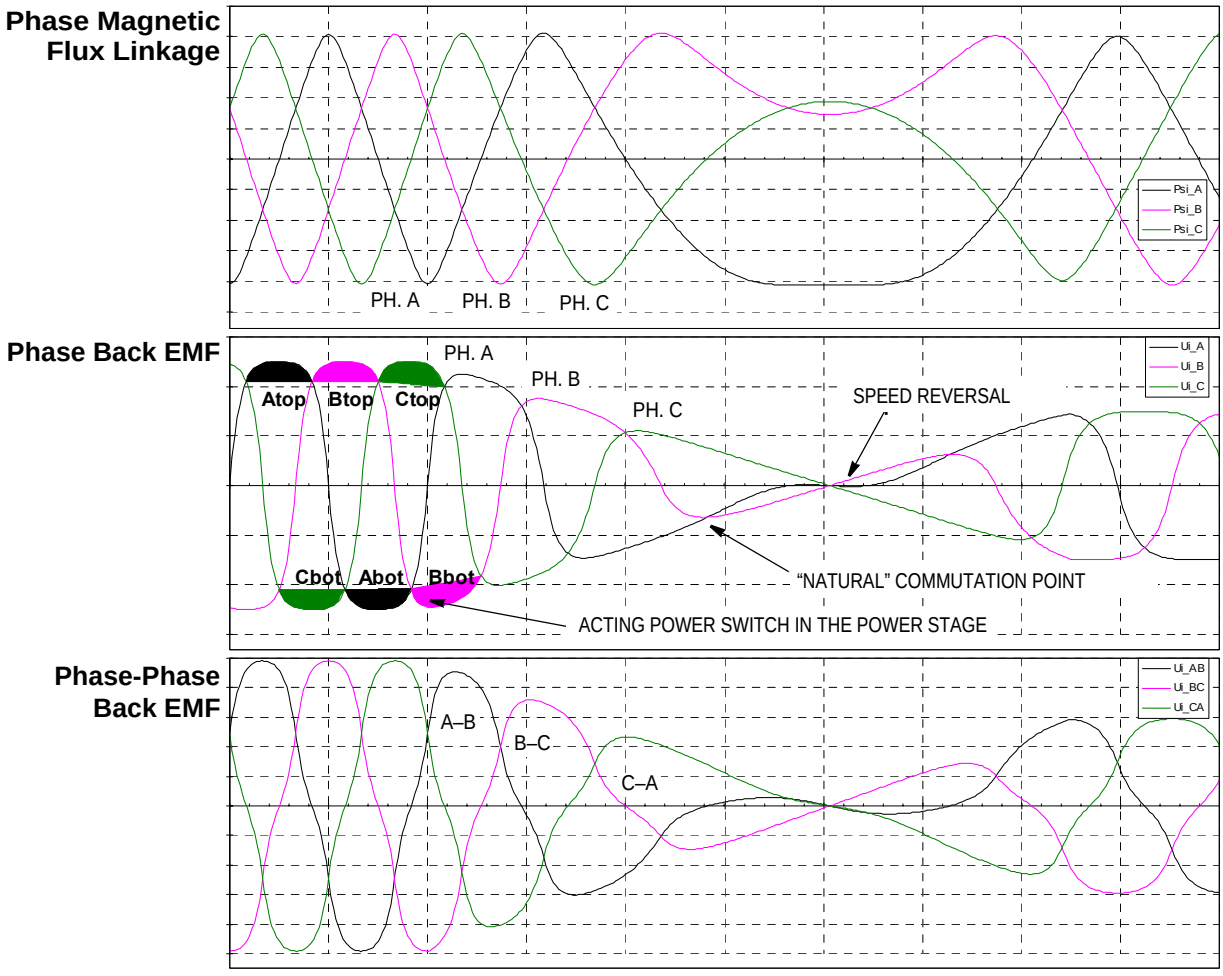

Figure 3. BLDC Motor Back EMF and Magnetic Flux

The filled areas in the tops of the phase back-EMF voltage waveforms indicate the intervals where the particular phase power stage commutations occur. The power switches are cyclically commutated through the six steps; therefore, this technique is sometimes called six step commutation control. The crossing points of the phase back-EMF voltages represent the natural commutation points. In normal operation the commutation is performed here. Some control techniques advance the commutation by a defined angle in order to control the drive above the pulse-width modulator (PWM) voltage control.

3-Phase BLDC Power Stage

The voltage for 3-phase BLDC motor is provided by a 3-phase power stage controlled by digital signals. Its topology is the one as for the AC induction motor (refer to **Figure 5**). The power stage is usually controlled by a dedicated microcontroller with on-chip PWM module.

Why Sensorless Control?

As explained in the previous section, rotor position must be known in order to drive a brushless dc motor. If any sensors are used to detect rotor position, sensed information must be transferred to a control unit (see **Figure 4**). Therefore, additional connections to the motor are necessary. This may not be acceptable for some applications. There are at least two reasons why you might want to eliminate the position sensors:

- Inability to make additional connections between position sensors and the control unit
- Cost of the position sensors and wiring

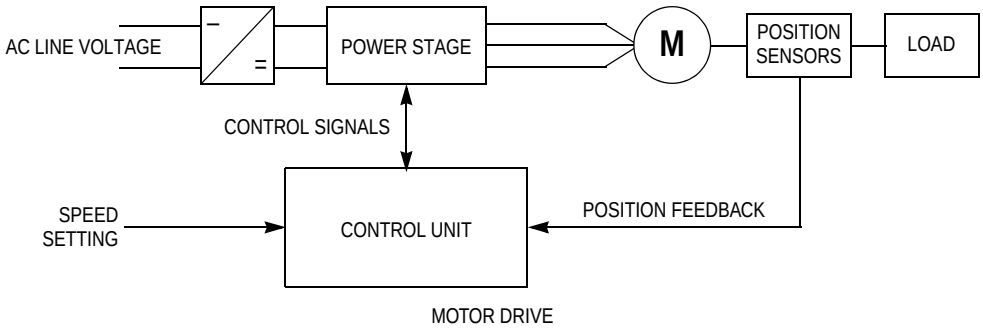

Figure 4. Classical System

Power Stage — Motor System Model

In order to explain and simulate the idea of back-EMF sensing techniques a simplified mathematical model based on the basic circuit topology has been created. See **Figure 5**.

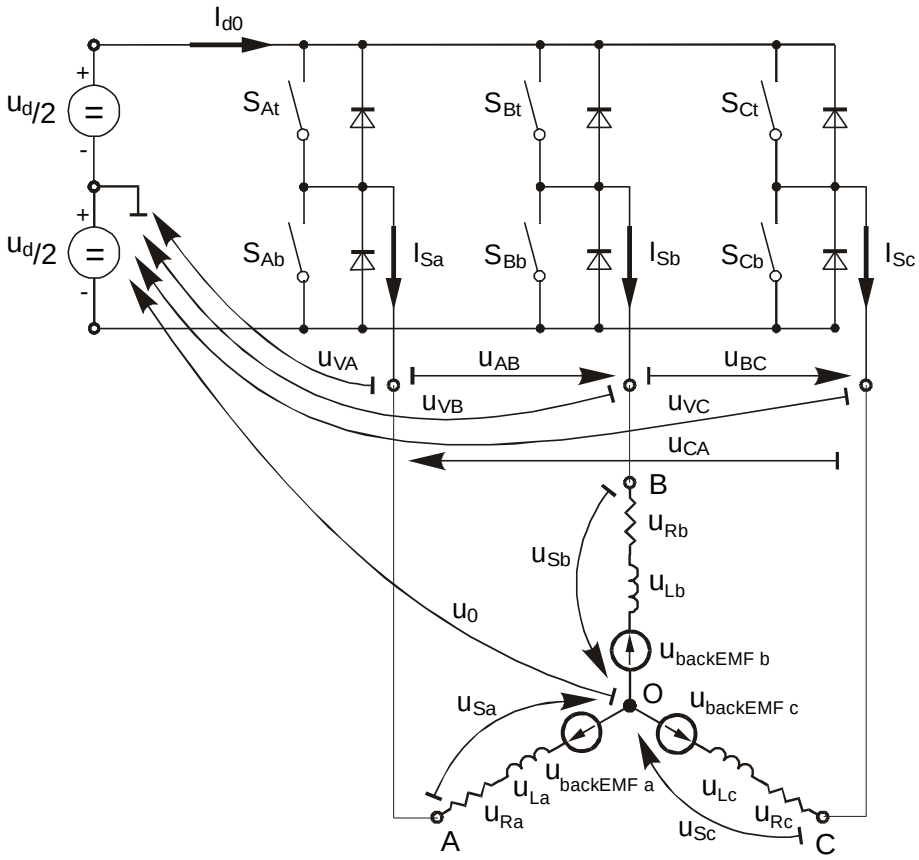

Figure 5. Power Stage — Motor Topology

The second goal of the model is to find how the motor characteristics depend on the switching angle. The **switching angle** is the angular difference between a real switching event and an ideal one (at the point where the **phase-to-phase** back-EMF crosses zero).

The motor-drive model consists of a normal 3-phase power stage plus a brushless dc motor. Power for the system is provided by a voltage source (U_d). Six semiconductor switches ($S_{A/B/C\ t/b}$), controlled elsewhere, allow the rectangular voltage waveforms (see **Figure 2**) to be applied. The semiconductor switches and diodes are simulated as ideal devices. The natural voltage level of the whole model is put at one half of the dc-bus voltage. This simplifies the mathematical expressions.

Stator Winding Equations

The BLDC motor is usually very symmetrical. All phase resistances, phase and mutual inductances, flux-linkages can be thought of as equal to, or as a function of the position θ with a 120° displacement.

The electrical BLDC motor model then consists of a set of the following stator voltage equations **Equation 1**.

$$\begin{bmatrix} u_{Sa} \\ u_{Sb} \\ u_{Sc} \end{bmatrix} = R_S \begin{bmatrix} i_{Sa} \\ i_{Sb} \\ i_{Sc} \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \Psi_{Sa} \\ \Psi_{Sb} \\ \Psi_{Sc} \end{bmatrix} \qquad \text{Equation 1}$$

The task of this section is to explain the background of the back-EMF sensing and to demonstrate how the zero crossing events can be detected. Parasitic effects that negatively influence the back-EMF detection are discussed and their nature analyzed.

Indirect Back EMF Sensing

Let us assume a usual situation, where the BLDC motor is driven in six-step commutation mode using PWM technique, where both top and bottom switches in the diagonal are controlled using the same signal (so called “hard switching PWM” technique). The motor phases A and B are powered, and phase C is free, having no current. So the phase C can be used to sense the back-EMF voltage. This is described by the following conditions:

$$\begin{gathered} S_{Ab}, S_{Bt} \leftarrow \mathrm{PWM} \\ u_{VA} = \mp\frac{1}{2}u_d, u_{VB} = \pm\frac{1}{2}u_d \\ i_{Sa} = -i_{Sb} = i, di_{Sa} = -di_{Sb} = di \\ i_{Sc} = 0, di_{Sc} = 0 \\ u_{backEMF\ a} + u_{backEMF\ b} + u_{backEMF\ c} = 0 \end{gathered} \qquad \text{Equation 2}$$

The branch voltage u_{Vc} can be calculated using the above conditions,

$$u_{VC} = u_{Sc} - \frac{1}{3}\left[\sum_{x=a}^{c} u_{backEMF\ x} + (L_{ac} - L_{bc})\frac{di}{dt} - u_{VC}\right] \qquad \text{Equation 3}$$

After evaluation the expression of the branch voltage u_{Vc} is as follows:

$$u_{VC} = \frac{3}{2}u_{backEMF\ c} - \frac{1}{2}(L_{ac} - L_{bc})\frac{di}{dt} \qquad \text{Equation 4}$$

The same expressions can also be found for phase A and B:

$$u_{VA} = \frac{3}{2}u_{backEMF\ a} - \frac{1}{2}(L_{ba} - L_{ca})\frac{di}{dt} \quad \text{Equation 5}$$

$$u_{VB} = \frac{3}{2}u_{backEMF\ b} - \frac{1}{2}(L_{cb} - L_{ab})\frac{di}{dt} \quad \text{Equation 6}$$

The first member in the equation **Equation 6** demonstrates the possibility to indirectly sense the back-EMF between the free (not powered) phase terminal and the zero point, defined at half of the dc-bus voltage (see **Figure 5**). Simple comparison of these two levels can provide the required zero crossing detection.

As shown in **Figure 5**, the branch voltage of phase B can be sensed between the power stage output B and the zero voltage level. Thus, back-EMF voltage is obtained and the zero crossing can be recognized.

When $L_{cb} = L_{ab}$, this general expressions can also be found:

$$u_{Vx} = \frac{3}{2}u_{backEMFx} \text{where } x = A,B,C \quad \text{Equation 7}$$

There are two necessary conditions which must be met:

- Top and bottom switches (in diagonal) have to be driven with the same PWM signal
- No current goes through the non-fed phase that is used to sense the back-EMF

Figure 6 shows branch and motor phase winding voltages during a 0–360° electrical interval. Shaded rectangles designate the validity of the equation **Equation 7**. In other words, the back-EMF voltage can be sensed during designated intervals.

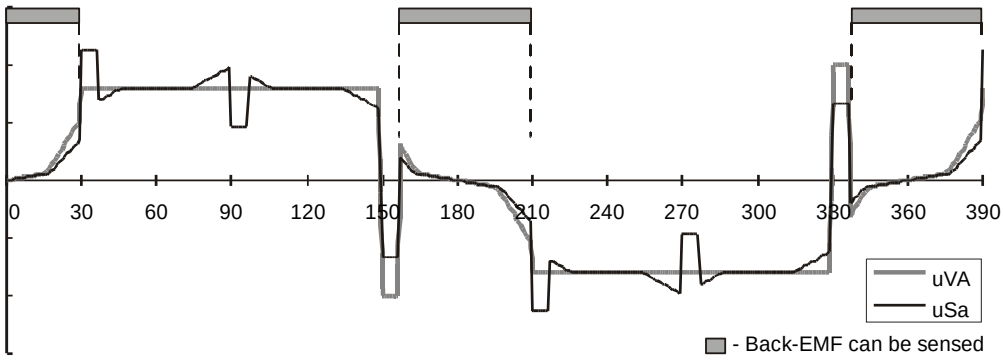

Figure 6. Phase Voltage Waveform

However simple this solution looks, in reality it is more difficult, because the sensed “branch” voltage also contains some ripples.

Effect of Mutual Inductance

As shown in previous equations **Equation 4** through **Equation 6**, the mutual inductances play an important role here. The difference of the mutual inductances between the coils which carry the phase current, and the coil used for back-EMF sensing, causes the PWM pulses to be superimposed onto the detected back-EMF voltage. In fact, it is produced by the high rate of change of phase current, transferred to the free phase through the coupling of the mutual inductance.

Figure 7 shows the real measured “branch” voltage. The red curves highlight the effect of the difference in the mutual inductances. This difference is not constant.

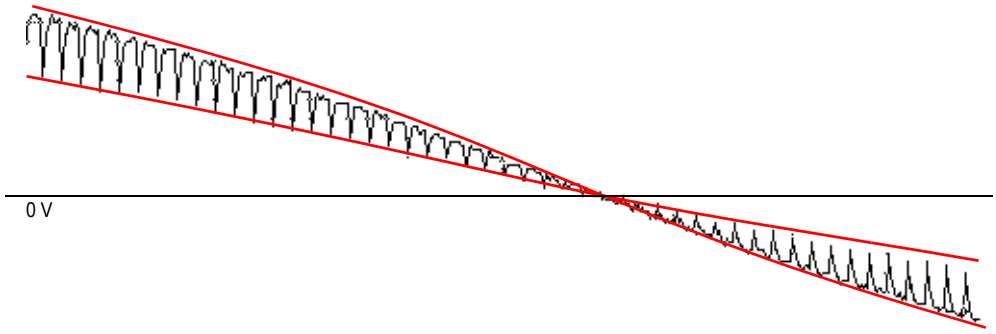

Figure 7. Mutual Inductance Effect

Due to the construction of the BLDC motor, both mutual inductances vary. They are equal at the position that corresponds to the back-EMF zero crossing detection.

The branch waveform detail is shown in **Figure 8**. Channel 1 in **Figure 8** shows the disturbed “branch” voltage. The superimposed ripples clearly match the width of the PWM pulses, and thus prove the conclusions from the theoretical analysis.

The effect of the mutual inductance corresponds well in observations carried out on the five different BLDC motors. These observations were made during the development of the sensorless technique.

NOTE: *The BLDC motor with stator windings distributed in the slots has technically higher mutual inductances than other types. Therefore, this effect is more significant. On the other hand the BLDC motor with windings wounded on separate poles, shows minor presence of the effect of mutual inductance.*

However noticeable this effect, it does not degrade the back-EMF zero crossing detection because it is cancelled at the zero crossing point. Simple additional filtering helps to reduce ripples further.

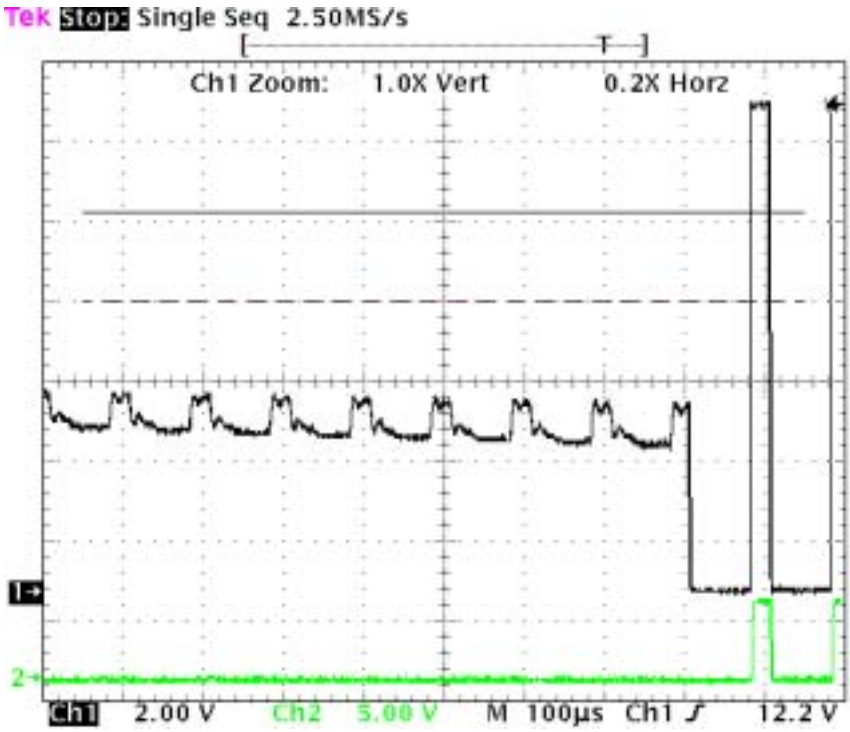

Figure 8. Detail of Mutual Inductance Effect

Effect of Mutual Phase Capacitance

The negative effect of mutual inductance is not the only one to disturb the back-EMF sensing. So far, the mutual capacitance of the motor phase windings was neglected in the motor model, since it affects neither the phase currents nor the generated torque. Usually the mutual capacitance is very small. Its influence is only significant during PWM switching, when the system experiences very high *du/dt*.

The effect of the mutual capacitance can be studied using the model shown in **Figure 9**.

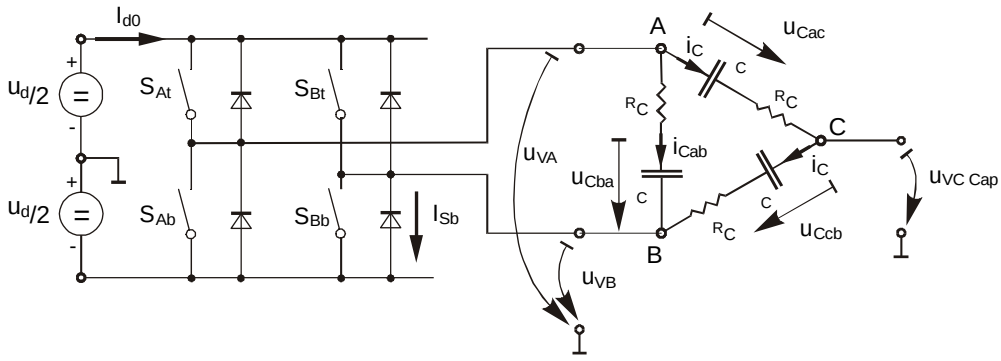

Figure 9. Mutual Capacitance Model

Let us focus on the situation when the motor phase A is switched from negative dc-bus rail to positive, and the phase B is switched from positive to negative. This is described by these conditions **Equation 8**:

$$S_{Ab}, S_{Bt} \leftarrow PWM$$
$$u_{VA} = -\frac{1}{2}u_d \rightarrow \frac{1}{2}u_d, u_{VB} = \frac{1}{2}u_d \rightarrow -\frac{1}{2}u_d \quad \text{Equation 8}$$
$$i_{Cac} = i_{Ccb} = i_C$$

The voltage that disturbs the back-EMF sensing, utilizing the free (not powered) motor phase C, can be calculated based the equation:

$$u_{VC\ Cap} = \frac{1}{2}(u_{Ccb} + u_{Cac} + 2R_C) - (u_{Ccb} + R_C) = \frac{1}{2}(u_{Cac} - u_{Ccb}) \quad \text{Equation 9}$$

The final expression for disturbing voltage can be found as follows:

$$u_{VC\ Cap} = \frac{1}{2}\left(\frac{1}{C_{ac}} - \frac{1}{C_{cb}}\right)\int i_C\,dt = \frac{1}{2}\left(\frac{C_{cb} - C_{ac}}{C_{cb} \cdot C_{ac}}\right)\int i_C\,dt \quad \text{Equation 10}$$

Equation 10 expresses the fact that only the unbalance of the mutual capacitance (not the capacitance itself) disturbs the back-EMF sensing. When both capacities are equal (they are balanced), the disturbances disappear. This is demonstrated in **Figure 10** and **Figure 11**.

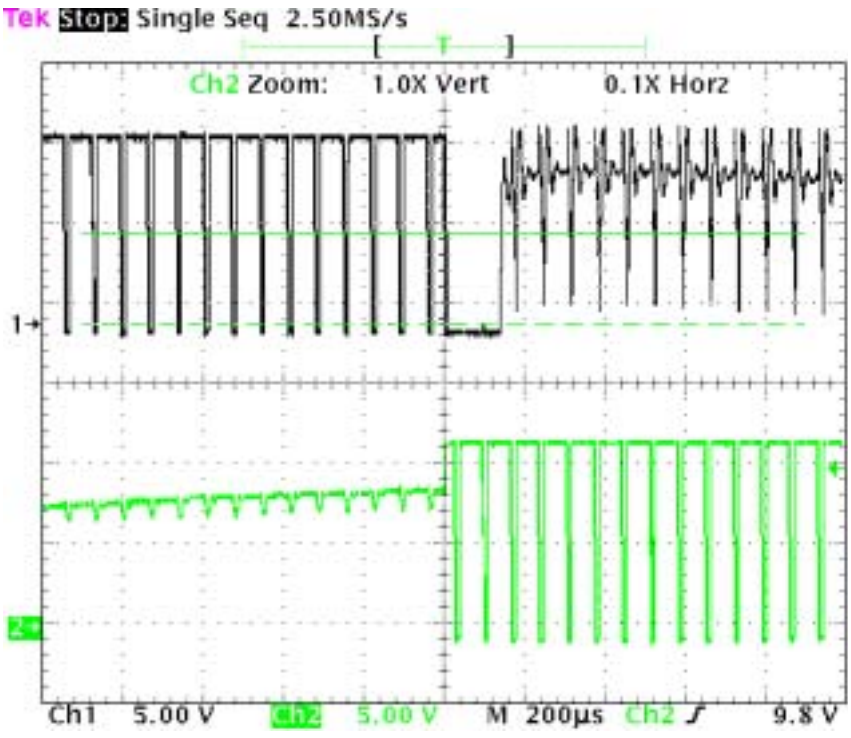

Figure 10. Distributed Back-EMF by Unbalanced Capacity Coupling

Channel 1 in **Figure 11** shows the disturbed “branch” voltage, while the other phase (channel 2) is not affected because it faces balanced mutual capacitance. The unbalance was purposely made by adding a small capacitor on the motor terminals, in order to better demonstrate the effect. After the unbalance was removed the “branch” voltage is clean, without any spikes.

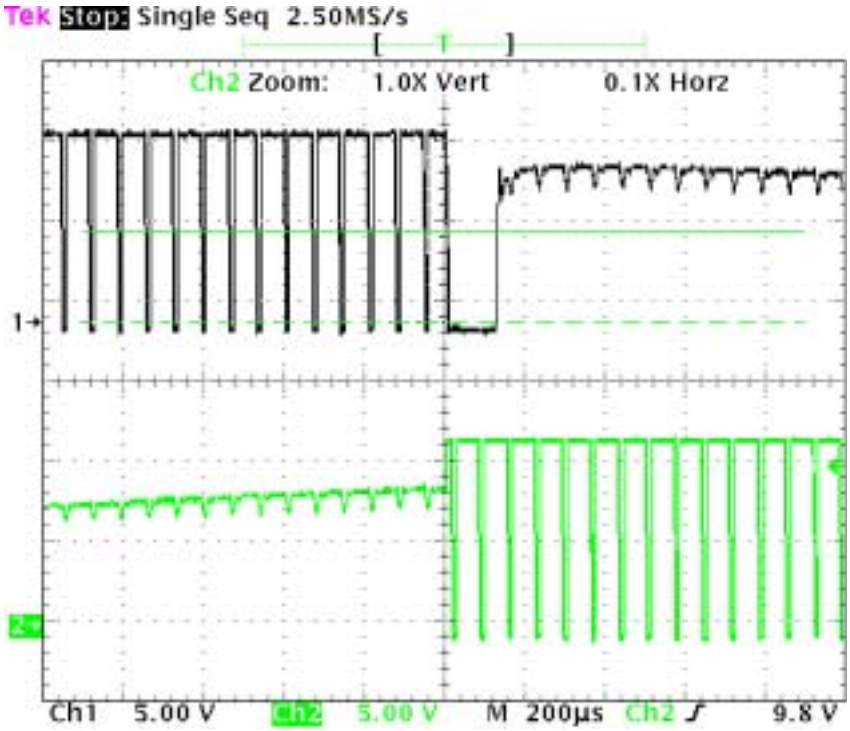

Figure 11. Balanced Capacity Coupling

NOTE: *The configuration of the phase windings end-turns has significant impact; therefore, it needs to be properly managed to preserve the balance in the mutual capacity. This is important, especially for prototype motors that are usually hand-wound.*

Failing to maintain balance in the mutual capacitance can easily disqualify such a motor from using sensorless techniques based on the back-EMF sensing. Usually, the BLDC motors with windings wound on separate poles show minor presence of the mutual capacitance. Thus, the disturbance is also insignificant.

Back-EMF Sensing Circuit

An example of the possible implementation of the back-EMF sensing circuit is shown in **Figure 12**.

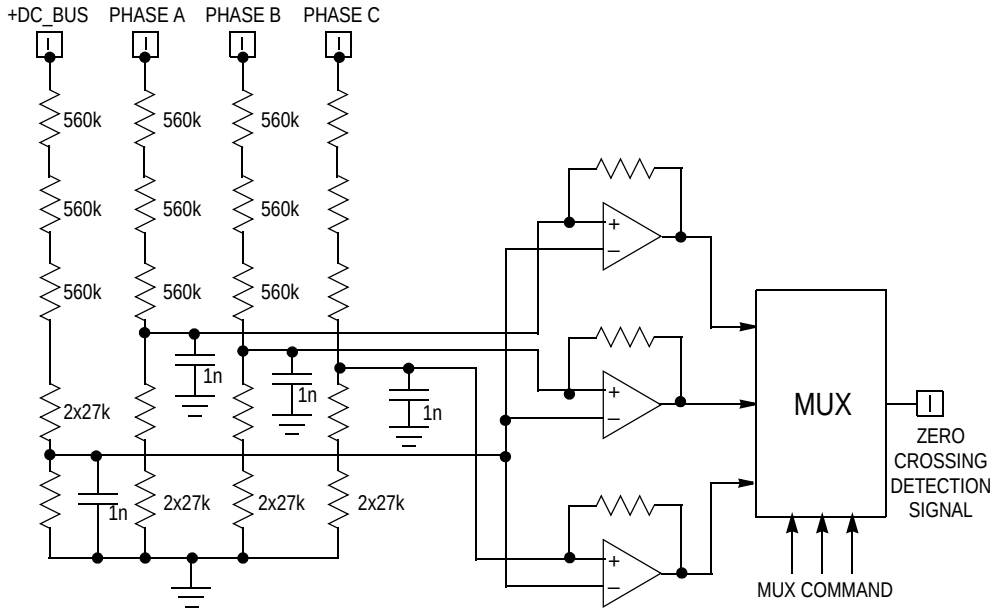

Figure 12. Back-EMF Sensing Circuit Diagram

As explained in the theoretical part of this application note, the phase zero crossing event can be detected at the moment when the branch voltage (of a free phase) crosses the half dc-bus voltage level. The resistor network is used to step down sensed voltages down to a 0–15 V voltage level. The comparators sense the zero voltage difference in the input signal. The multiple resistors reduce the voltage across each resistor component to an acceptable level. A simple RC filter prevents the comparators from being disturbed by high voltage spikes produced by IGBT switching. The multiplexer (MUX) selects the phase comparator output, which corresponds to the current commutation stage. This zero crossing detection signal is transferred to the timer input pin.

The comparator control and zero crossing signals plus the voltage waveforms are shown in **Figure 13**.

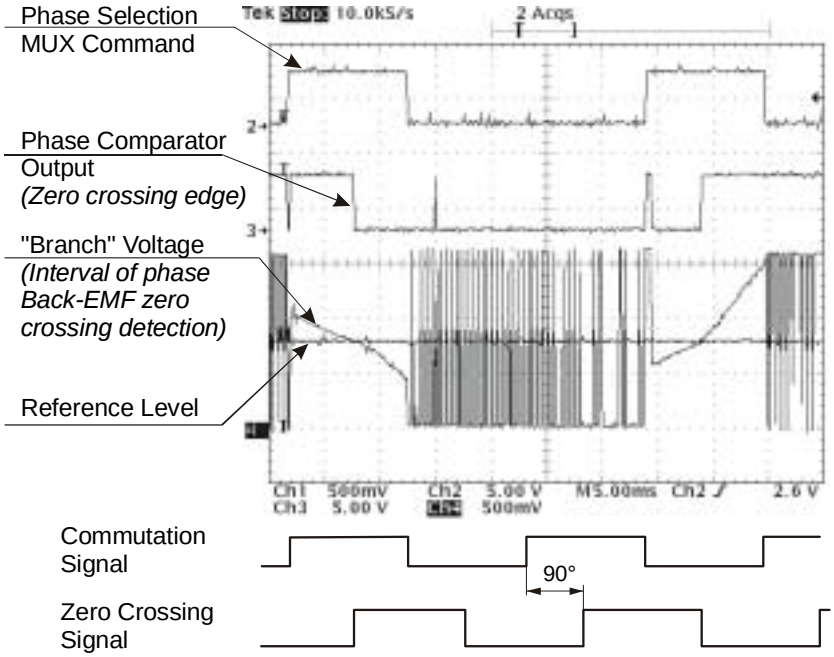

Figure 13. The Zero Crossing Detection

Application Specification

The concept of the application is that of a speed-closed loop drive using back-EMF zero crossing technique for position detection.

The system for BLDC motor control consists of hardware and software. The application uses universal Modular Motion Control Development Hardware boards, povided by Motorola for customer development support.

For hardware board descriptions refer to **References 3.**, **4.**, **5.**, **6.**, and **7.**.

There are three board and motor hardware sets for the application

- High-voltage hardware set — for variable line voltage 115–230 Vac and medium power (phase current < 2.93A)
- Low-voltage evaluation motor hardware set — for automotive voltage (12 V) and very low power (phase current < 4A)
- Low-voltage hardware set — for automotive voltage (12 V or possibly 42 V) and medium power (phase current < 50A)

The application software is practically the same for all three hardware platforms, needing only to be modified to include one of three constant customizing files containing hardware and motor parameter settings.

The application specifications are listed in **Table 1**.

Table 1. Application Specifications

Control Algorithm	3-phase trapezoidal BLDC motor control star or delta! connected
	Sensorless with back-EMF zero crossing commutation control
	Speed closed loop control
	Motoring mode
Target Processor	MC68HC908MR32
Software Language	C-language with some arithmetical functions in assembler
Application Control	Manual interface (start/stop switch, speed potentiometer control, LED indication)
	PC master software (remote) interface (via RS232 using PC computer)
Targeted Hardware	Software is prepared to run on three optional board and motor hardware sets: 1. High voltage hardware set at variable line voltage 115–230 Vac (software customizing file const_cust_hv.h) 2. Low voltage evaluation motor hardware set (software customizing file const_cust_evmm.h) 3. Low voltage hardware set (software customizing file const_cust_lv.h)
Software Configuration and Parameter Settings	Configuration to one of the three required hardware is provided by an include of the dedicated software customizing file (the software pack contains the files const_cust_hv.h, const_cust_lv.h, const_cust_evm.h, with predefined parameter settings for running on one of the optional board and motor hardware sets. The required hardware needs to be selected in code_fun.c file by one of these files #include)
	When software is configured for a different customer motor, the software configuration for any motor is provided in the dedicated customizing file according to the hardware board used

System Concept

Refer to **Figure 14** for the application block diagram.

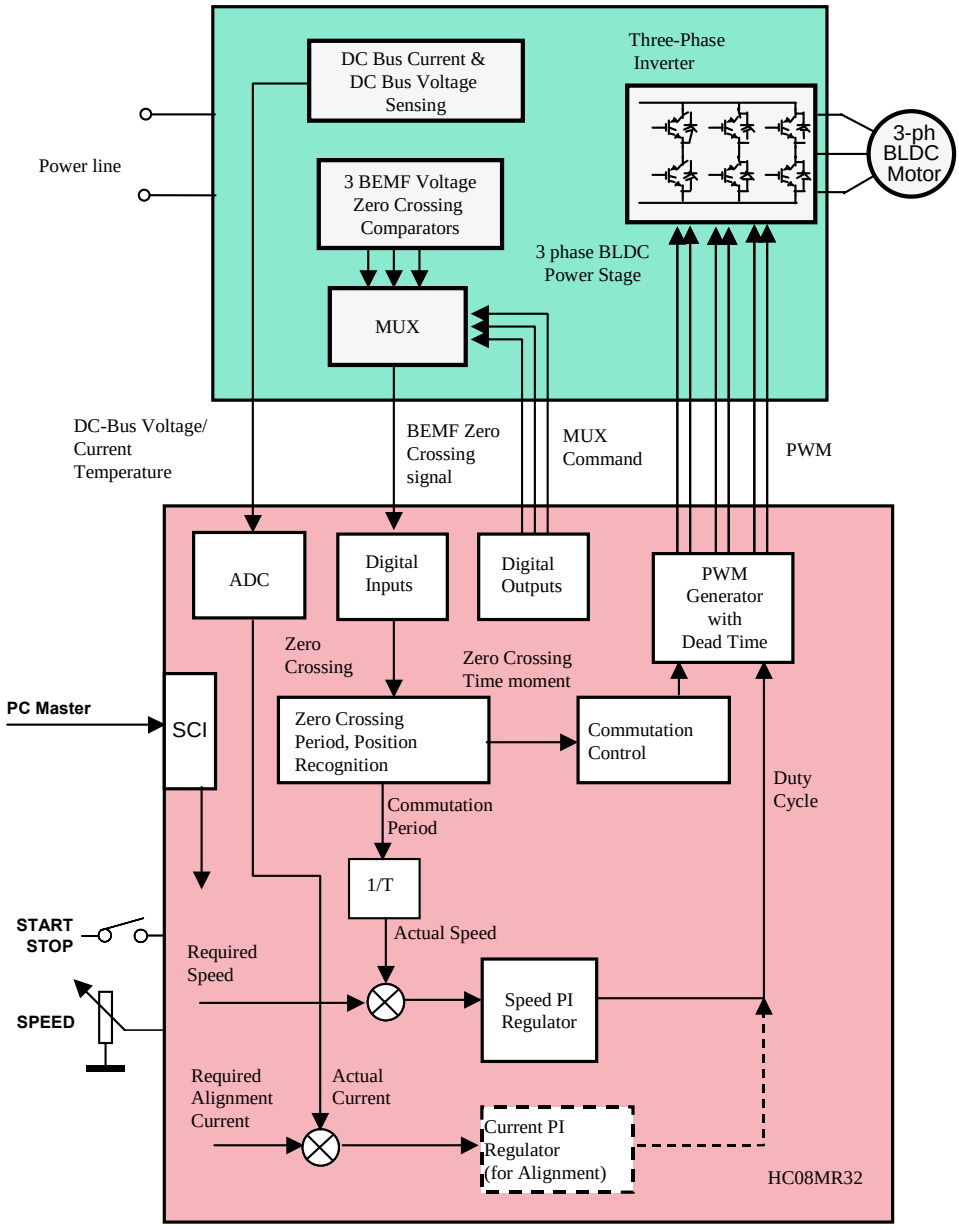

Figure 14. System Concept

The sensorless rotor position technique detects the zero crossing points of back-EMF induced in the motor windings. The phase back-EMF zero crossing points are sensed while one of the three phase windings is not powered. The information obtained is processed in order to commutate the energized phase pair and control the phase voltage, using pulse width modulation.

The back-EMF zero crossing detection enables position recognition. The resistor network is used to step down the sensed voltages to a 0–3.3 V voltage level. In order to filter high voltage spikes produced by the switching of the IGBTs (MOSFETs), zero crossing detection is synchronized with the middle of a central aligned PWM signal by the software. The software selects by MUX command the phase comparator output that corresponds to the current commutation step. The MUX circuit selects this signal, which is transferred to the MCU Input.

The voltage drop resistor is used to measure the dc-bus current which is chopped by the PWM. The signal obtained is rectified and amplified (0–3.3 V with 1.65 V offset). The internal MCU analog-to-digital (A/D) converter and zero crossing detection are synchronized with the PWM signal. This synchronization avoids spikes when the IGBTs (or MOSFETs) are switched, and simplifies the electric circuit.

During the rotor alignment state, the dc-bus current is controlled by the current PI regulator. In the other states (motor running), the phase voltage (PWM duty cycle) is controlled by the speed PI regulator.

The A/D converter is also used to sense the dc-bus voltage and drive temperature. The dc-bus voltage is stepped down to a 3.3 V signal level by a resistor network.

The six IGBTs (copack with built-in fly back diode), or MOSFETs, and gate drivers create a compact power stage. The drivers provide the level shifting that is required to drive high side switch. PWM technique is used to control motor phase voltage.

Control Technique

Sensorless Commutation Control

This section concentrates on sensorless BLDC motor commutation with back-EMF zero crossing technique.

In order to start and run the BLDC motor, the control algorithm has to go through the following states:

- **Alignment**
- **Starting (Back-EMF Acquisition)**
- **Running**

Figure 15 shows the transitions between the states. First the rotor is aligned to a known position; then the rotation is started without the position feedback. When the rotor moves, back-EMF is acquired so the position is known, and can be used to calculate the speed and processing of the commutation in the running state.

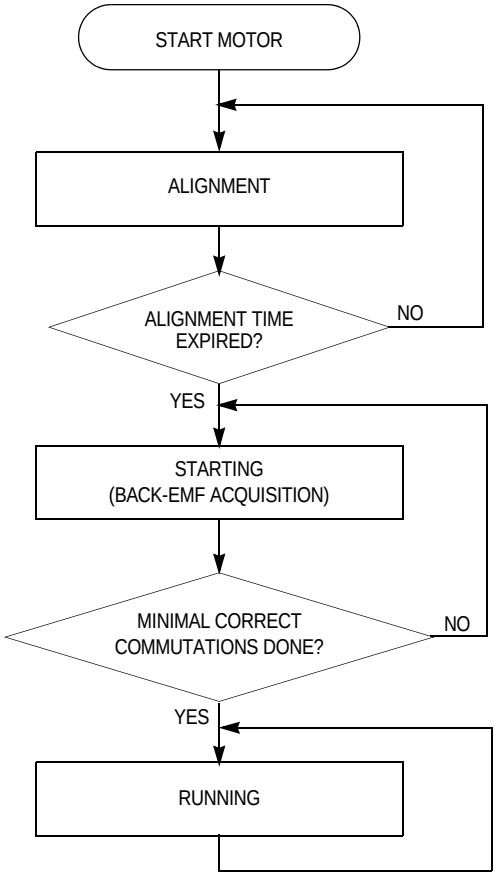

Figure 15. Commutation Control Stages

Alignment

Before the motor starts, there is a short time (depending on the motor's electrical time constant) when the rotor position is stabilized by applying PWM signals to only two motor phases (no commutation). The current controller keeps current within predefined limits. This state is necessary in order to create a high start-up torque. When the preset time-out expires then this state is finished.

The current controller subroutine, with PI regulator, is called to control dc-bus current. It sets the correct PWM ratio for the required current. The current PI controller works with constant execution (sampling) period. This period should be a multiple of the PWM period, in order to synchronize the current measurement with PWM:

Current controller period = n/PWM frequency

The BLDC motor rotor position with flux vectors during alignment is shown in **Figure 16**.

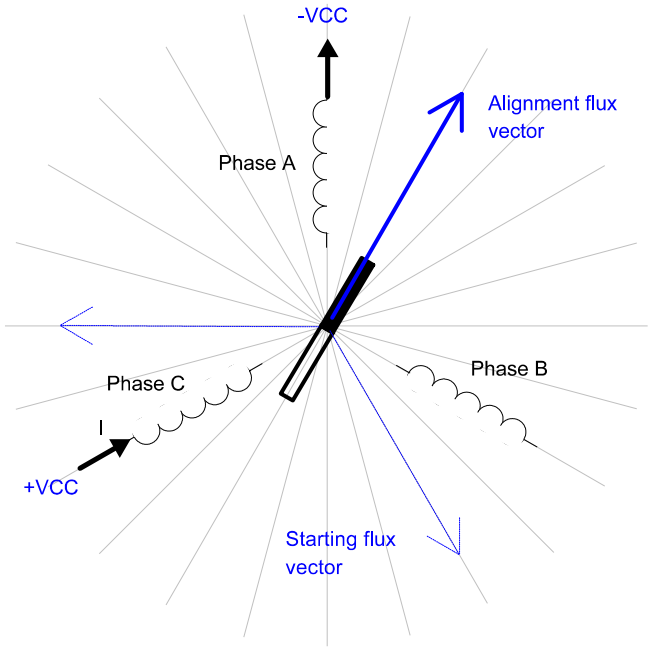

Figure 16. Alignment

Running

The commutation process is a series of states which assure:

- The back-EMF zero crossing is successfully captured
- The new commutation time is calculated
- The commutation is performed

The following processes need to be provided:

- BLDC motor commutation service
- Back-EMF zero crossing moment capture service
- Calculation of commutation time
- Interactions between these commutation processes

From diagrams an overview of how the commutation works can be understood. After commuting the motor phases, there is a time interval (Per_Toff[n]) when the shape of back-EMF must stabilized (after the commutation the fly-back diodes are conducting the decaying phase current; therefore, sensing of the back-EMF is not possible). Then the new commutation time (T2[n]) is preset. The new commutation will be performed at this time if the back-EMF zero crossing is not captured. If the back-EMF zero crossing is captured before the preset commutation time expires, then the exact calculation of the commutation time (T2*[n]) is made, based on the captured zero crossing time (T_ZCros[n]). The new commutation is performed at this new time.

If for any reason the back-EMF feedback is lost within one commutation period, corrective actions are taken in order to return to the regular states.

The flowchart explaining the principle of BLDC commutation control with back-EMF zero crossing sensing is shown in **Figure 17**.

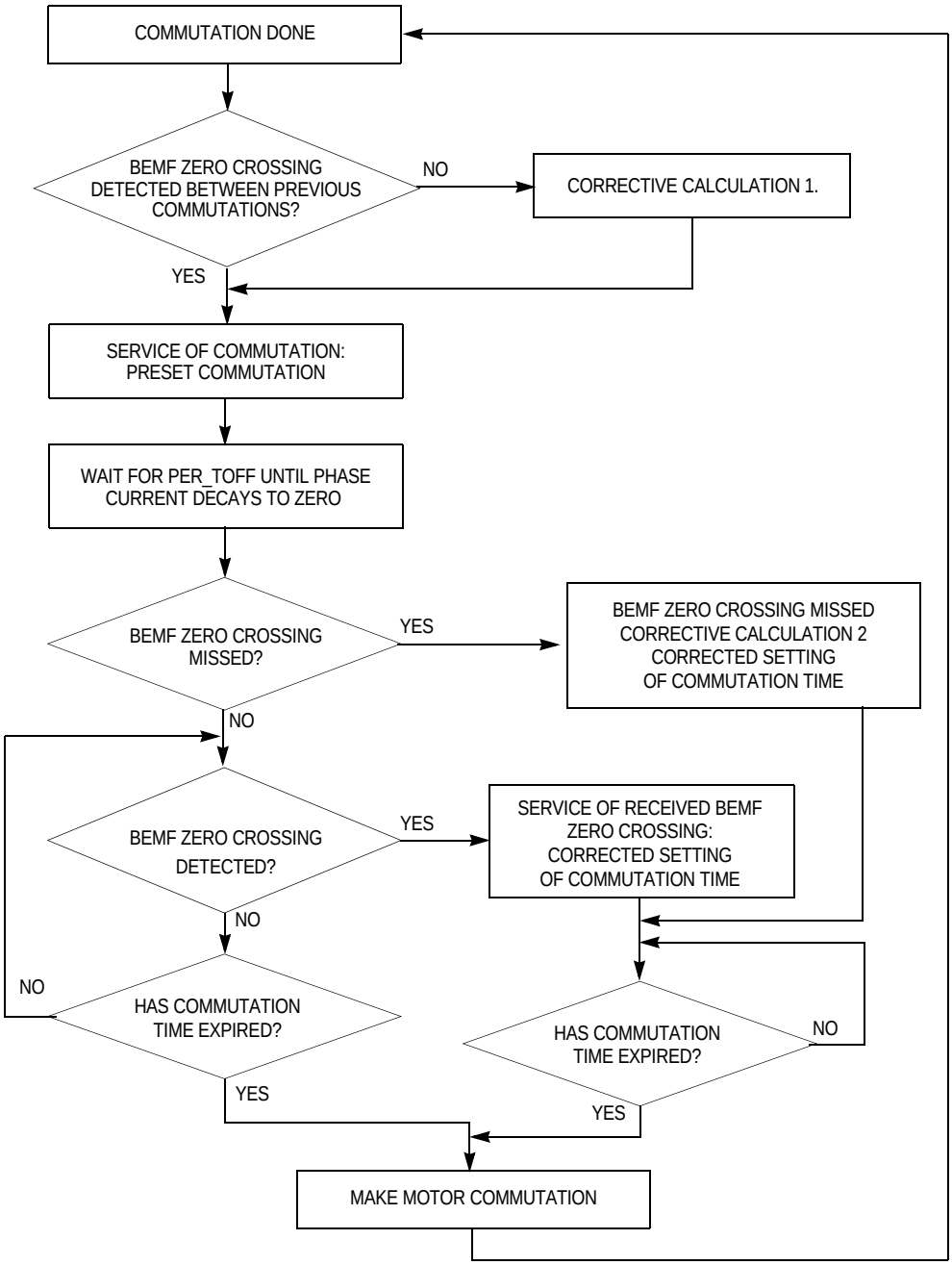

Figure 17. BLDC Commutation with Back-EMF Zero Crossing Sensing Flowchart

Running — Commutation Time Calculation

Commutation time calculation is shown in **Figure 18**.

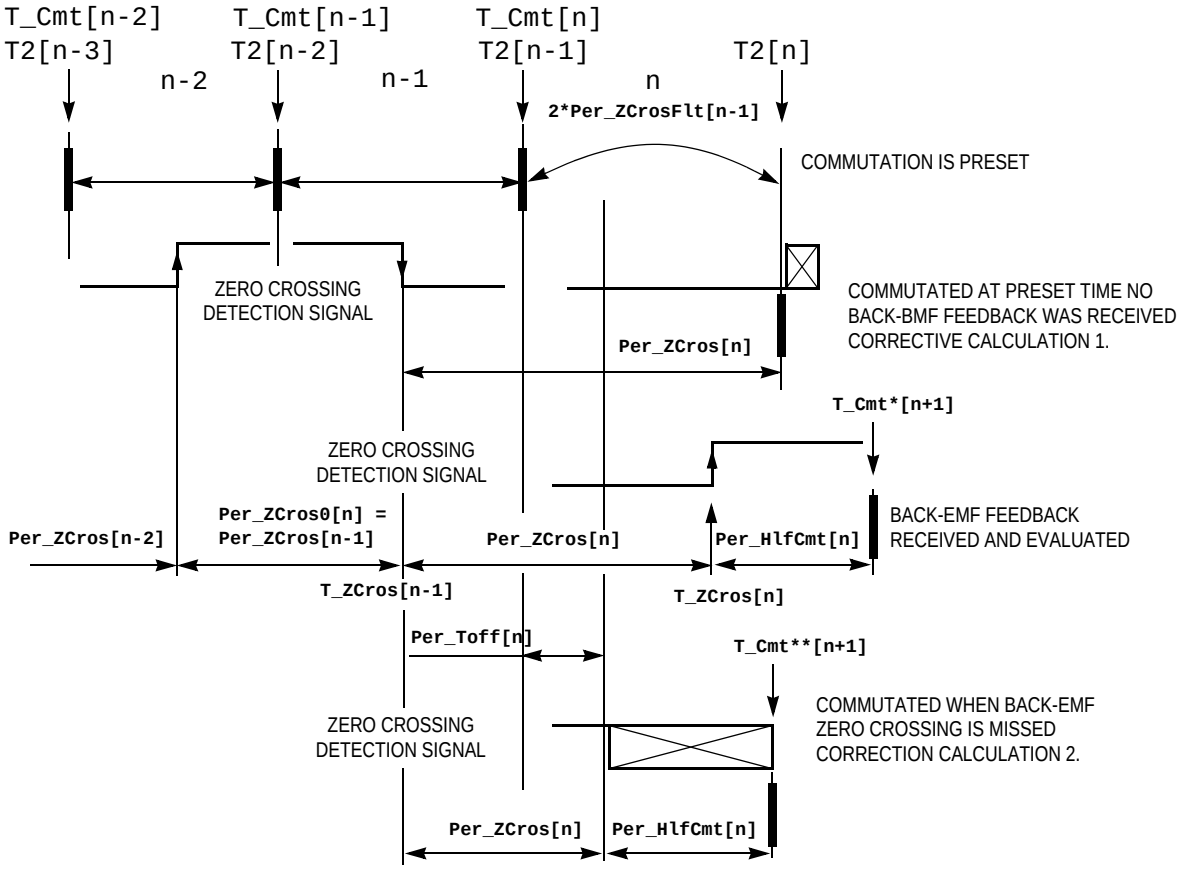

Figure 18. BLDC Commutation Time with Zero Crossing Sensing

The following calculations are made to calculate the commutation time (T2[n]) during the **Running state**:

- **Service of commutation** — The commutation time (T2[n]) is predicted:

```
T2[n] = T_Cmt[n] + 2*Per_ZCrosFlt[n-1]
If 2*Per_ZCrosFlt>Per_Cmt_Max
      then result is limited at Per_Cmt_Max
```

- **Service of received back-EMF zero crossing** — The commutation time (T2*[n]) is evaluated from the captured back-EMF zero crossing time (T_ZCros[n]):

```
Per_ZCros[n] = T_ZCros[n] - T_ZCros[n-1] = T_ZCros[n] - T_ZCros0
Per_ZCrosFlt[n] = (1/2*Per_ZCros[n]+1/2*Per_ZCros0)
HlfCmt[n] = 1/2*Per_ZCrosFlt[n]- Advance_angle =
          = 1/2*Per_ZCrosFlt[n]- C_CMT_ADVANCE*Per_ZCrosFlt[n]=
```

```
                Coef_HlfCmt*Per_ZCrosFlt[n]
                The best commutation was get with Advance_angle:
                60Deg*1/8 = 7.5Deg
                which means Coef_HlfCmt = 0.375 at Running state
                with default s/w setting
Per_Toff[n+1] = Per_ZCrosFlt*Coef_Toff and Per_Dis minimum
                Coef_Toff = 0.375 at Running state, Per_Dis = 150
                with default s/w setting
Per_ZCros0 <-- Per_ZCros[n]
T_ZCros0 <-- T_ZCros[n]
T2*[n] = T_ZCros[n] + HlfCmt[n]
```

- If no back-EMF zero crossing was captured during preset commutation period (T2P[n] then **Corrective Calculation 1.** is made:

```
T_ZCros[n] <-- CmtT[n+1]
Per_ZCros[n] = T_ZCros[n] - T_ZCros[n-1] = T_ZCros[n] - T_ZCros0
Per_ZCrosFlt[n] = (1/2*Per_ZCros[n]+1/2*Per_ZCros0)
HlfCmt[n] = 1/2*Per_ZCrosFlt[n]-Advance_angle =
Coef_HlfCmt*Per_ZCrosFlt[n]
                The best commutation was get with Advance_angle:
60Deg*1/8 = 7.5Deg
                which means Coef_HlfCmt = 0.375 at Running state!
Per_Toff[n+1] = Per_ZCrosFlt*Coef_Toff and Per_Dis minimum
Per_ZCros0 <-- Per_ZCros[n]
T_ZCros0 <-- T_ZCros[n]
```

- If back-EMF zero crossing is missed then **Corrective Calculation 2.** is made:

```
T_ZCros[n] <-- CmtT[n]+Toff[n]
Per_ZCros[n] = T_ZCros[n] - T_ZCros[n-1] = T_ZCros[n] - T_ZCros0
Per_ZCrosFlt[n] = (1/2*T_ZCros[n]+1/2*T_ZCros0)
HlfCmt[n] = 1/2*Per_ZCrosFlt[n]-Advance_angle =
Coef_HlfCmt*Per_ZCrosFlt[n]
                The best commutation was get with Advance_angle:
                60Deg*1/8 = 7.5Deg
                which means Coef_HlfCmt = 0.375 at Running state!
Per_ZCros0 <-- Per_ZCros[n]
T_ZCros0 <-- T_ZCros[n]
```

- Where:

```
T_Cmt = time of the last commutation
T2 = Time of the Timer 2 event (for Timer Setting)
T_ZCros = Time of the last zero crossing
T_ZCros0 = Time of the previous zero crossing
Per_Toff = Period of the zero crossing off
Per_ZCros = Period between zero crossings (estimates required
                commutation period)
Per_ZCros0 = Pervious period between zero crossings
Per_ZCrosFlt = Estimated period of commutation filtered
Per_HlfCmt = Period from zero crossing to commutation (half commutation)
```

The required commutation timing is provided by setting commutation constants **Coef_HlfCmt, COEF_TOFF**.

Starting (Back-EMF Acquisition)

The back-EMF sensing technique enables a sensorless detection of the rotor position; however, the drive must be first started without this feedback. This is due to the fact that the amplitude of the induced voltage is proportional to the motor speed. Hence, the back-EMF cannot be sensed at a very low speed and a special start-up algorithm must be performed.

In order to start the BLDC motor, the adequate torque must be generated. The motor torque is proportional to the multiplication of the stator magnetic flux, the rotor magnetic flux, and the sine of the angle between these magnetic fluxes.

It implies (for BLDC motors) the following:

1. The level of phase current must be high enough.
2. The angle between the stator and rotor magnetic fields must be 90°±30°.

The first condition is satisfied during the alignment state by maintaining dc-bus current at a level sufficient to start the motor. In the starting (back-EMF acquisition) state, the same value of PWM duty cycle is used as the one which has stabilized the dc-bus current during the align state.

The second condition is more difficult to fulfill without any position feedback information. After the alignment state, the stator and the rotor magnetic fields are aligned (0° angle). Therefore, two fast commutations (faster than the rotor can follow) must be applied to create an angular difference in the magnetic fields (see **Figure 19**).

The commutation time is defined by the start commutation period (**Per_CmtStart**). This allows starting the motor such that minimal speed (defined by state when back-EMF can be sensed) and is achieved during several commutations, while producing the required torque. Until the back-EMF feedback is locked, the commutation process (explained in **Running**) assures that commutations are done in advance, so that successive back-EMF zero crossing events are not missed.

After several successive back-EMF zero crossings:

- Exact commutation time can be calculated
- Commutation process is adjusted
- Control flow continues to the Running state

The BLDC motor is then running with regular feedback and the speed controller can be used to control the motor speed by changing the PWM duty cycle value.

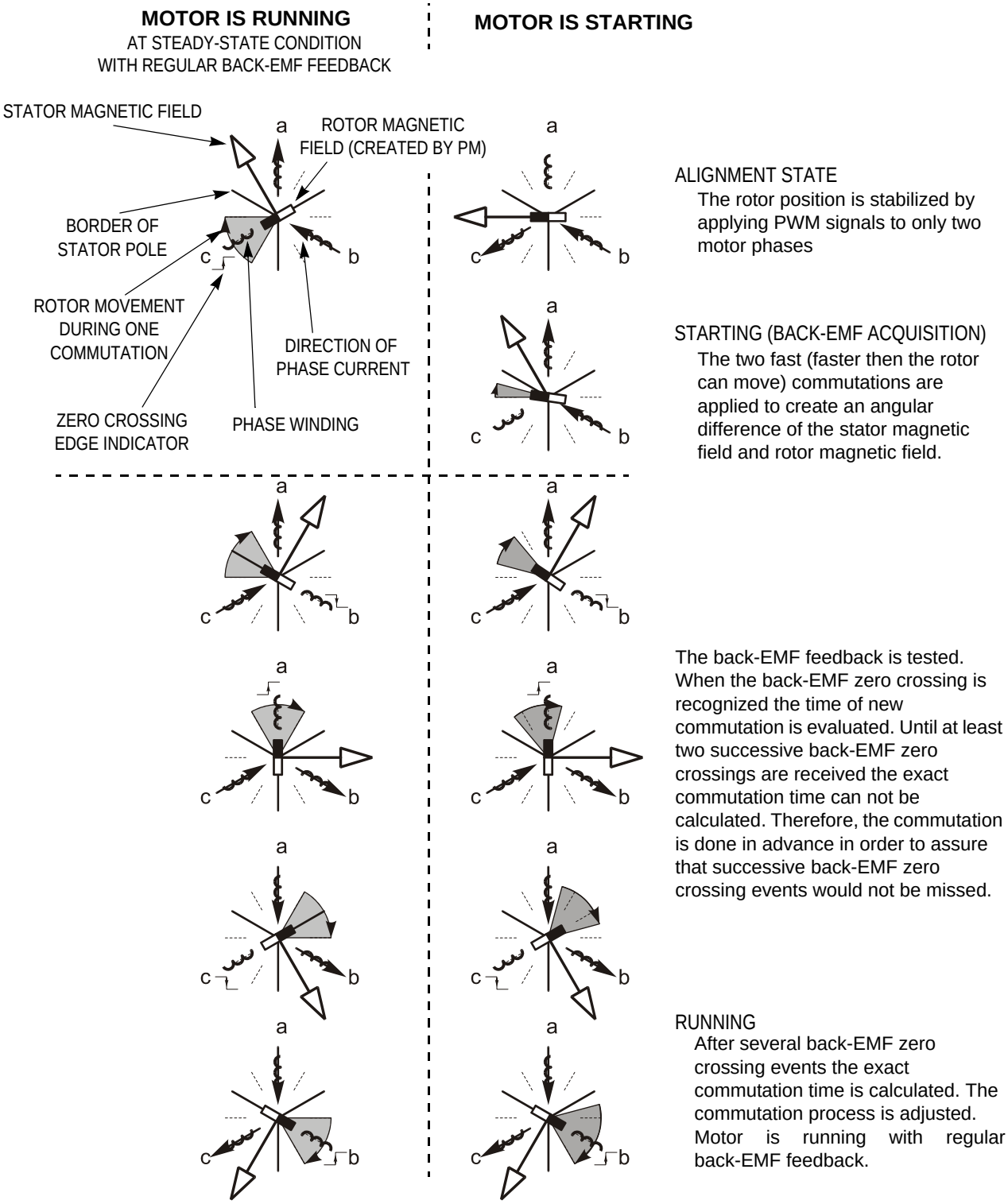

Figure 19. Vectors of Magnetic Fields

Figure 20 demonstrates the back-EMF during the start up. The amplitude of the back-EMF varies according to the rotor speed. During the starting (back-EMF acquisition) state the commutation is done in advance. In the running state the commutation is done at the right moments.

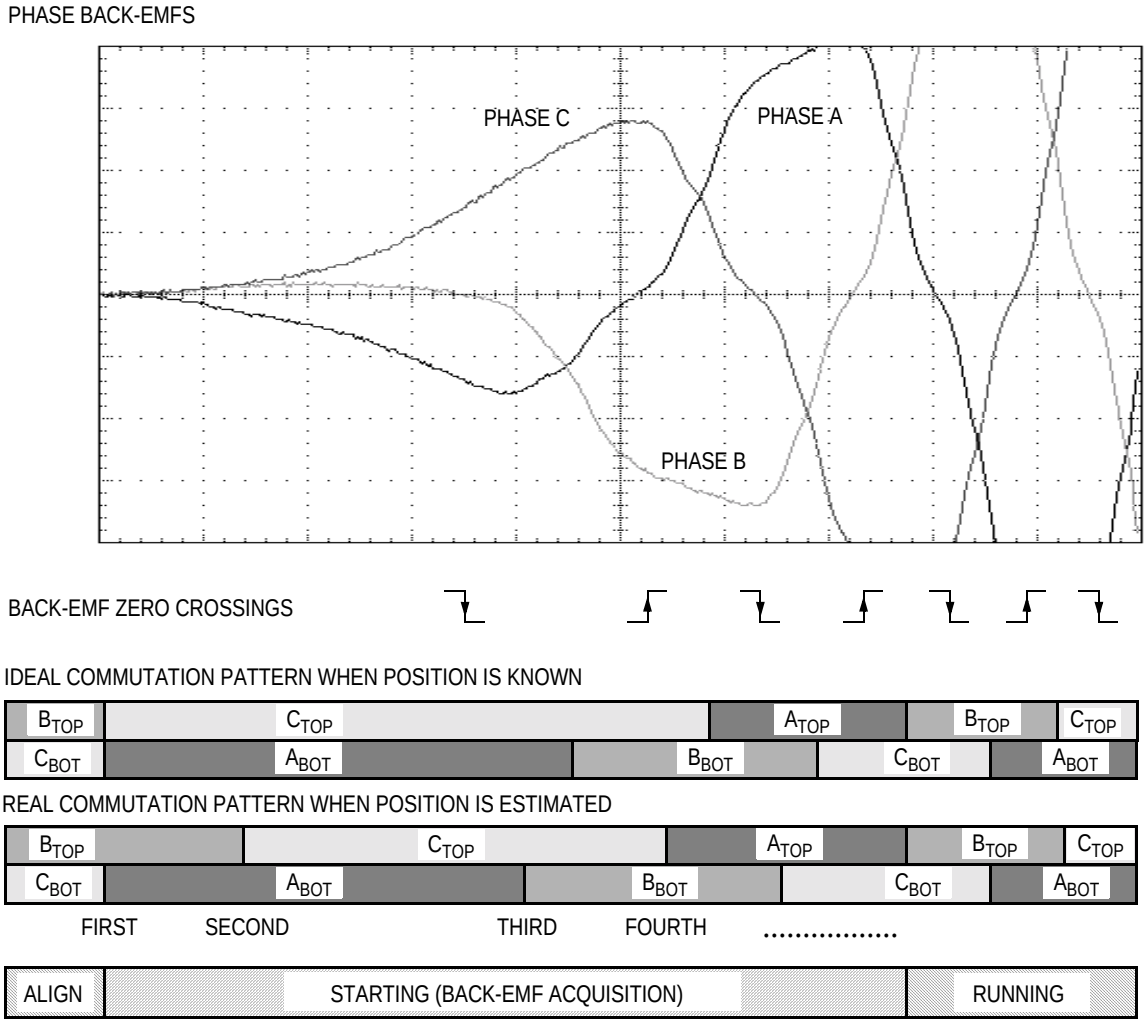

Figure 20. Back-EMF at Start Up

Figure 21 illustrates the sequence of the commutations during the starting (back-EMF acquisition) state. The commutation times T2[1] and T2[2] are calculated without any influence of back-EMF feedback. The commutation time calculations are explained in the following section.

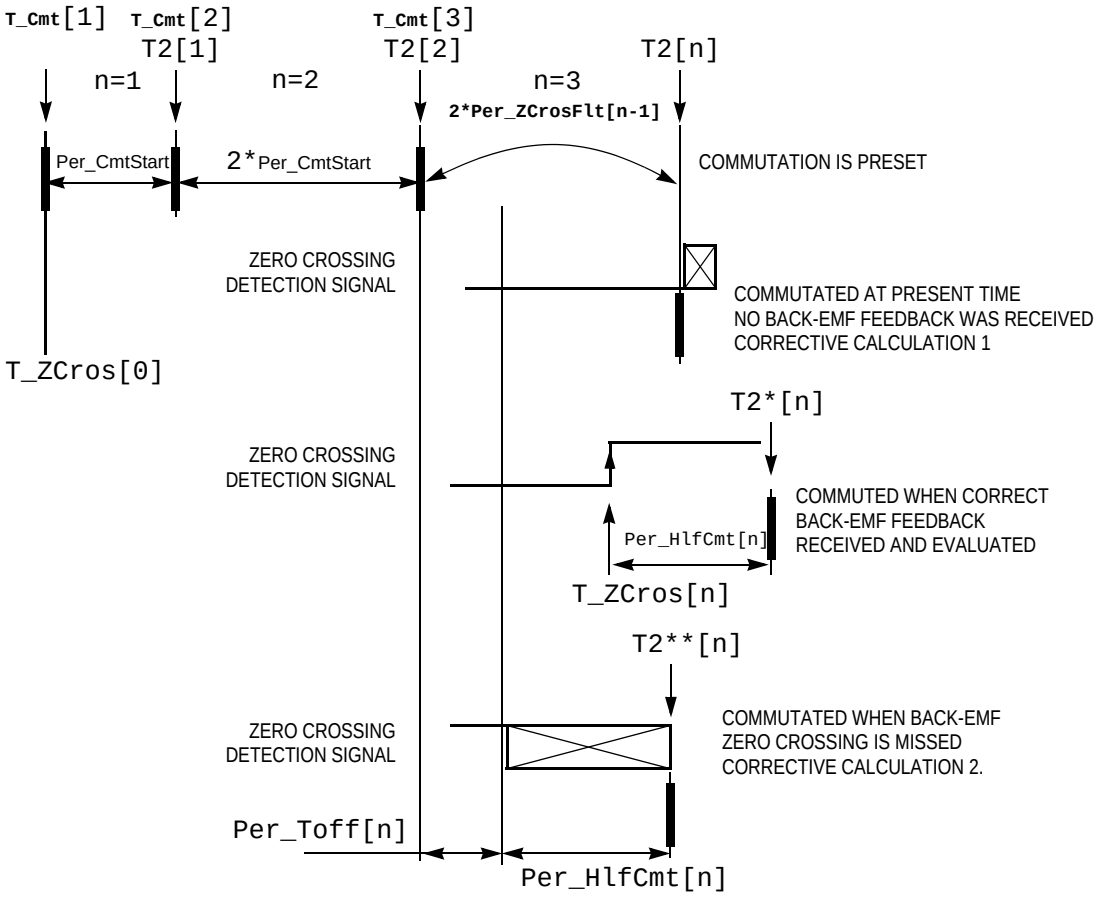

Figure 21. Calculation of the Commutation Times During the Starting (Back-EMF Acquisition) State

Starting — Commutation Time Calculation

Even the sub-states of the commutation process in the starting (back-EMF acquisition) state remain the same as in the running state. The required commutation timing depends on application state (starting state, running state). So the commutation time calculation is the same as that described in **Running — Commutation Time Calculation**, but the following computation coefficients are different:

```
coefficient Coef_HlfCmt = 0.125 with advanced angle Advance_angle:
60Deg*3/8 = 22.5Deg
            at Starting state!
Coef_Toff = 0.5 at Running state, Per_Dis = 150 with default s/w
setting
```

Speed Control

The speed close loop control is provided by a well known PI regulator. The required speed is calculated from speed input variable, as explained in **Process Desired Speed Setting**. The actual speed is calculated from the average of two back-EMF zero crossing periods (time intervals), received from the sensorless commutation control block. The speed regulator output is a PWM duty cycle.

The speed controller works with the constant execution (sampling) period **PER_T3_RUN_US**. A detailed explanation is provided in **Processs Speed Control**.

Application Control

The application can be controlled in two basic modes:

- Manual mode
- PC master software mode

In manual mode, it is controlled by an on-board start/stop switch and speed potentiometer. In PC master mode, it is controlled from a computer using PC master software. In both modes, the individual variables can be observed using the PC master software.

PC Master Software

PC master software was designed to provide the debugging, diagnostic, and demonstration tools for developing algorithms and applications. It consists of components running on PCs and parts running on the target MCU, connected by an RS232 serial port. A small program is resident in the MCU that communicates with the PC master software to parse commands, return status information, and process control information from the PC. The PC master software uses Microsoft[1] Internet Explorer as a user interface on the PC.

Communication with PC Master Software Specifications

SCI communication protocol with a default of 9.6 Kbaud, is used for communication as described in *User's Manual for PC Master Software*, Motorola 2000, found on the World Wide Web at:
http://e-www.motorola.com

PC master software controls and senses the status of the application with:

- PC master software — BLDC demonstration suitcase communication commands
- PC master software — BLDC demonstration suitcase communication bytes

After reset, the BLDC control MCU software is in manual mode. In order to control the system from PC master software, it is necessary to set PC master software mode, and then to provide the MCU software control from PC master software via application interface variables.

PC Master Software, BLDC Control MCU Software API, Communication Commands

Commands defined for the BLDC control MCU software are listed in **Table 2**. The commands are very simple. If the software executes the command, it responds with OK byte 00. If it is unable to execute the command, it responds with failed code 55. The commands "Set PC master software mode", "Set manual mode" can only be executed when the START/STOP switch on the demonstration suitcase is set to STOP and the motor is stopped. Otherwise, a failed response is sent.

Table 2. PC Master Software Communication Commands

Command	Command Code	Data Bytes	Demo Suitcase Action	Respons Byte	Response Description
Set PC master software mode	01	None	Setting of PC master software mode	00 55	OK Failed
Set manual mode	02	None	Setting of manual mode	00 55	OK Failed

1. Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries.

PC Master Software, BLDC Control MCU Software API, Communication Variables

The application interface, data variables used for the exchange between the BLDC control MCU software and PC master software, are shown in **Table 3**. These variables are used for status sensing and control. PC master software accesses these bytes directly from their physical memory addresses.

Table 3. PC Master Software API Variables

Name	Type	I/O	Representing Range	Description
Sys3	Sys3_Def	I/O	8flags	System variable #3
Motor_Ctrl	Motor_Ctrl_Def	I	8flags	Motor control variable
Motor_Status	Motor_Status_Def	O	8flags	Motor status variable
Failure	Failure_Def	O	8flags	Failure variable
Sp_Input	U8	I	< 0; 255>	Speed input variable used for required speed calculation
Speed_Range_Max_RPM	U16	O	< 0; 65535> [rpm]	Speed range maximum
Speed_Max_RPM	U16	O	< 0; 65535> [rpm]	Maximal speed limit
Speed_Min_RPM	U16	O	< 0; 65535> [rpm]	Minimal speed limit
Commut_Rev	U8	O	< 0; 255>	Commutations per motor revolution
Curr	S8	O	<-Curr_Range_Max_cA; Curr_Range_Max_cA)	dc-bus current
Curr_Range_Max_cA	S16	O	<-32768;32767> [A*10^-2]	Current range maximum [A*10^-2]

Type: *S8- signed 8 bit, U8- unsigned 8 bit,S16- signed 16bit, U16- unsigned 16bit*

The system registers **Sys3, Motor_Ctrl, Motor_Status, Failure** flags are described by definitions of **Sys3_Def, Motor_Ctrl_Def, Motor_Status_Def, Failure_Def:**

```
typedef union
{
  struct
    {
     unsigned int HV   : 1;  /* BIT0 High Voltage board Flag */
     unsigned int LV   : 1;  /* BIT1 Low Voltage board */
     unsigned int EVMm : 1;  /* BIT2 EVMm board */
     unsigned int BIT3 : 1;  /* BIT3 RESERVED */
     unsigned int PCMode : 1;/* BIT4 PCMaster/manual mode Flag */
     unsigned int BIT5 : 1;  /* BIT5 RESERVED */
     unsigned int BIT6 : 1;  /* BIT6 RESERVED */
     unsigned int Alignment : 1; /* BIT7 Alignment state Proceeding */
    } B;
/* |Alignment|***|***|PCMode|***|EVMm|LV||HV| */
     char R;
} Sys3_Def;
```

```
/* System register #3 Definition */

typedef union
{
  struct
    {
     unsigned int StartCtrl : 1; /* Switch Start set to START Flag */
     unsigned int BIT1      : 1; /* BIT1 RESERVED */
     unsigned int BIT2      : 1; /* BIT2 RESERVED */
     unsigned int BIT3      : 1; /* BIT5 RESERVED */
     unsigned int BIT4      : 1; /* BIT4 RESERVED */
     unsigned int BIT5      : 1; /* BIT6 RESERVED */
     unsigned int BIT6      : 1; /* BIT6 RESERVED */
     unsigned int ClearFail : 1; /* BIT7 Clear failure Status  */
    } B;
/* |ClearFail|***|***|***|***|***|***|StartCtrl| */
     char R;
} Motor_Ctrl_Def;
/* PC master software Motor Control Flags Definition */

typedef union
{
  struct
    {
     unsigned int Switch_Start : 1; /* BIT0 Switch START/STOP
                                       set to START Flag */
    unsigned int Running : 1;/* BIT1 Motor is running (Alignment, Start
                                                    or Running state) */
     unsigned int BIT2 : 1;  /* BIT2 RESERVED */
     unsigned int V120 : 1;  /* BIT3 120 V DC-Bus detected
                                (only for HV DC-Bus) */
     unsigned int BIT4 : 1;  /* BIT4 RESERVED */
     unsigned int BIT5 : 1;  /* BIT5 RESERVED */
     unsigned int BIT6 : 1;  /* BIT6 RESERVED */
     unsigned int BIT7 : 1;  /* BIT7 RESERVED */
    } B;
/* |***|***|***|***|V120|***|Running|Switch_Start| */
     char R;
} Motor_Status_Def;
/* PC master software Motor Status Flags register Definition */

typedef union
{
  struct
    {
     unsigned int OverCurrent : 1;     /* BIT0 Over-Current Failure */
     unsigned int OverHeating : 1;     /* BIT1 Over-Heating */
     unsigned int VoltageFailure : 1;  /* BIT2 Over-Voltage */
     unsigned int BIT3        : 1;     /* BIT5 RESERVED */
     unsigned int BIT4        : 1;     /* BIT4 RESERVED */
     unsigned int BIT5        : 1;     /* BIT6 RESERVED */
     unsigned int BoardIdFail : 1;     /* BIT6 pcb Identification
                                          Failure */
     unsigned int ErrCmt  : 1;         /* BIT7 error Commutation */
    } B;
 /* |ErrCmt|***|***|***|***|VoltageFailure|OverHeating|OverCurrent| */
     char R;
} Failure_Def;    /* Failure Flags register Definition */
```

Table 3 declares if the variable is used as an output or input from the BLDC control MCU software side. The variable is described and the unit defined.

When PC master software mode is set, the system start and stop is controlled by **StartCtrl** flag in **Motor_Ctrl** variable. When the application enters the fault state, the variable **Failure** displays the fault reason. Setting the **ClearFail** flag in **Motor_Ctrl** will exit the fault state.

The **Sp_Input** variable is used for speed control. In PC master software mode, it can be modified from PC master software (otherwise, it is set according to speed potentiometer value).

Desired speed [rpm] = **Sp_Input/255*(Speed_Max_RPM-Speed_Min_RPM) + Speed_Min_RPM**

So, the required motor commutation period is determined by the **Speed_Max_RPM** and **Speed_Min_RPM** variables. These are chosen according to which optional board and motor set by the BLDC control MCU software.

The variable **Speed_Range_Max_RPM** determines scaling of the speed variables.

The actual speed of the motor can be calculated from **Per_Speed_MAX_Range** and zero crossing period **Per_ZCrosFlt_T2**:

Actual speed [rpm] = **Speed_Range_Max_RPM*Per_Speed_MAX_Range/Per_ZCrosFlt_T2**

The variable **Commut_Rev** can be used for calculation of the BLDC motor commutation period:

Commutation Period [s] = 60 / Actual Speed [rpm] / **Commut_Rev**

The variable **Curr_Range_Max_cA** determines scaling of the current variables. So, the actual dc-bus current is:

dc-bus current [A] = **Curr** / 256*__Curr_Range_Max_cA__ / 100

Hardware Description

The hardware consists of the controller board accommodating MC68HC908MR32, power stage board, and motor. It is based on modular system boards as shown in:

- Application note entitled *Sensorless BLDC Motor Control on MC68HC908MR32 — Software Porting to Customer Motor* (Motorola document order number AN2356)
- Dedicated board manuals (they can be found on the Motorola web site at http:/motorola.com/semiconductors).

Software Description

This section describes the design of the software blocks of the drive. The software will be described in terms of:

- **Data Flow**
- **Main Software Flowchart**
- **State Diagram**

For more information on the control technique used see **Control Technique**.

Data Flow

The control algorithm obtains values from the user interface and sensors, processes them and generates 3-phase PWM signals for motor control, as can be seen on the data flow analysis shown in **Figure 22** and **Figure 23**.

Software Variables and Defined Constants

Important system variables are listed in **Table 4**.

Table 4. Software Variables

Name	Type	Representing Range	Description
Sys1	Sys1_Def	8flags	System variable #1
Speed_Min_U8	U8	< 0; Speed_Range_Max_RPM)	Minimal speed [system units]
Sp_Input	U8	< 0; 255>	Speed input variable used for required speed calculation
Coef_Speed_Inp	U8		Coeficient Sp_Inp to Speed_Desired calculation
Speed_Desired	U8	< 0; Speed_Range_Max_RPM)	Desired speed
PIParamsScl_U8_Speed	Structure		Speed PI regulator parameters
Per_Speed_MAX_Range	U16	[UNIT_PERIOD_T2_US]	Minimal commutation period of the speed range (at Speed_Range_Max_RPM)
Per_ZCrosFlt	U16	[UNIT_PERIOD_T2_US]	Zero crossing period — filtered
T2	U16(union)	[UNIT_PERIOD_T2_US]	Timer 2 variable
T_ZCros	U16	[UNIT_PERIOD_T2_US]	Zero crossing time [n]
T_ZCros0	U16(union)	[UNIT_PERIOD_T2_US]	Zero crossing time [n-1]
T_Cmt	U16	[UNIT_PERIOD_T2_US]	Commutation time
Curr	S8	<-Curr_Range_Max_cA; Curr_Range_Max_cA)	dc-bus current
Curr_Align	S8	<-Curr_Range_Max_cA; Curr_Range_Max_cA)	Required current during alignment state
PIParamsScl_S8_Currr	Structure		Current PI regulator parameters

Table 4. Software Variables (Continued)

Name	Type	Representing Range	Description
Volt	U8	<-VOLT_RANGE_MAX; VOLT_RANGE_MAX)	dc-bus voltage
V_TASC2	U8		Back-EMF zero crossing expecting edge
V_MUX	U8		Preset value of back-EMF zero crossing phase multiplexer

Type: S8- signed 8 bit, U8- unsigned 8 bit, S16- signed 16bit, U16- unsigned 16bit, (union)- 16 bits access or 2*8bit access

The system registers **Sys1** flags are described by definitions of **Sys3_Def:**

```
typedef union
{
  struct
    {
     unsigned int PC_F    : 1;  /* BIT0 Phase Commutation Flag */
     unsigned int Off_F   : 1;  /* BIT1 Offset timeout flag
                                   - Offset timeout can be measured */
     unsigned int ICR_F   : 1;  /* BIT2 Input Capture
                                      was succesfuly Received - Flag */
    unsigned int Rmp_F   : 1;  /* BIT3 Speed Ramp Flag - motor ramping */
     unsigned int Stop_F  : 1;  /* BIT4 Motor is going or is stopped */
     unsigned int Strt_F  : 1;  /* BIT5 Start Phase Flag */
     unsigned int Run_F   : 1;  /* BIT6 Motor Running with
                                       back-EMF feedback Flag */
     unsigned int FOK_F   : 1;  /* BIT7 Feedback within the righ time
Flag */
    } B;
               /* |FOK_F|Run_F|Strt_F|Stop_F|Rmp_F|ICR_F|Off_F|PC_F| */
  char R;
} Sys1_Def;       /* System register #1 Definition */
```

Main data flow is displayed in **Figure 22**. The processes are described in the following subsections.

Process Measurement

The process provides measurement of analog values using ADC. The measured inputs are: dc-bus current, dc-bus voltage, and speed input. The measurement is provided by the measurement handler. The state diagram is explained in **State Diagram**.

Start/Stop Switch Reading and Start/Stop Decision

The process reads the start stop switch and provides start condition and clear failure decisions, as explained in **Stand-By** and **Fault State**.

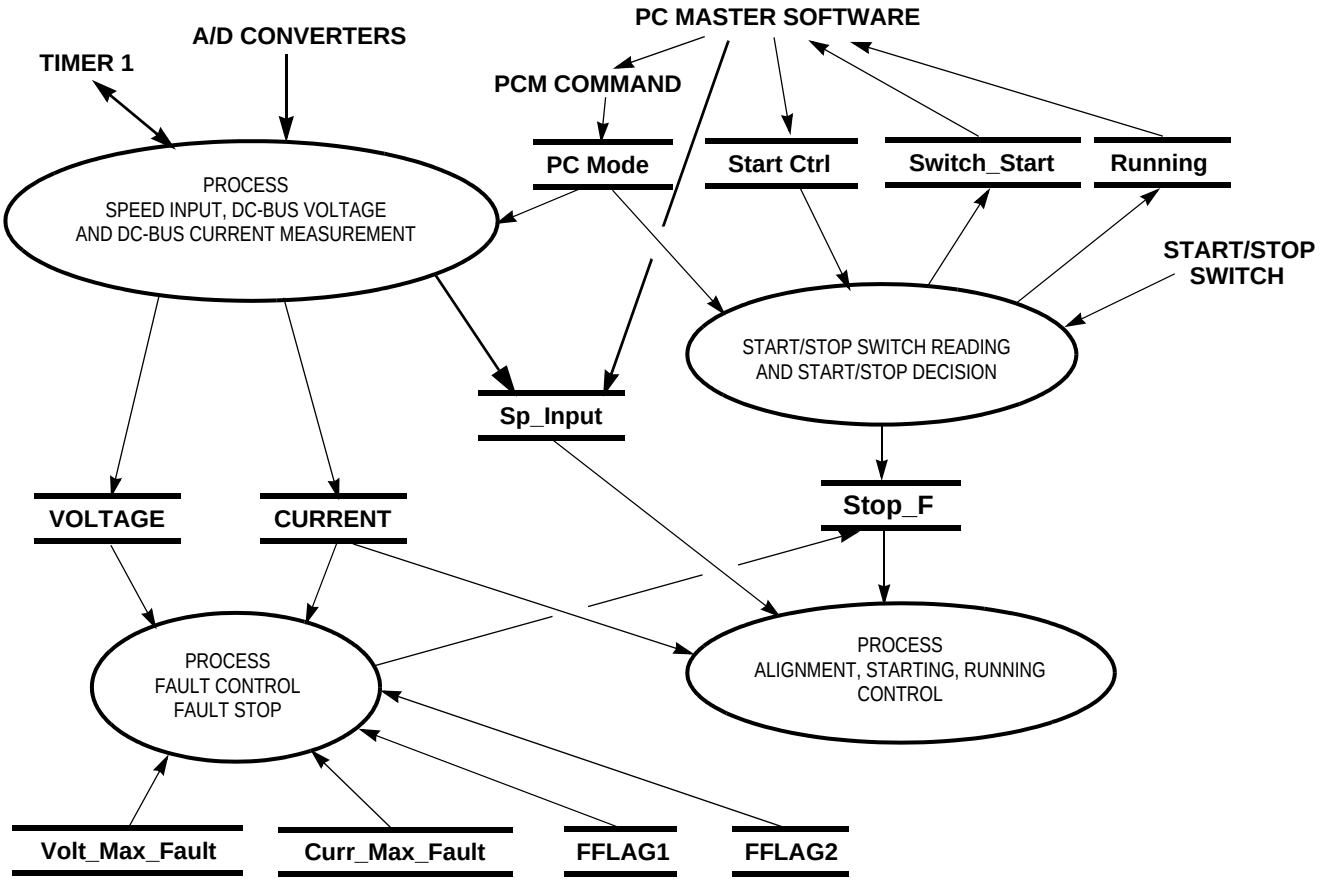

Figure 22. Main Data Flow — Part1

Process Fault Control Fault Stop

The process provides fault control and fault stop as described in **Fault State**, **Stand-By**, **Align State**, **Back-EMF Acquisition State**, and **Running State**.

The processes alignment, starting, and running control are displayed in **Figure 23**. The processes are described in the following subsections.

Process Back, EMF Zero Crossing Sensing

Back-EMF zero crossing process provides:

- Back-EMF zero crossing sampling in synchronization with PWM,
- Evaluates the zero crossing
- Records its time in T_ZCros

Further explanation is provided in **Data Flow** and **Figure 27**.

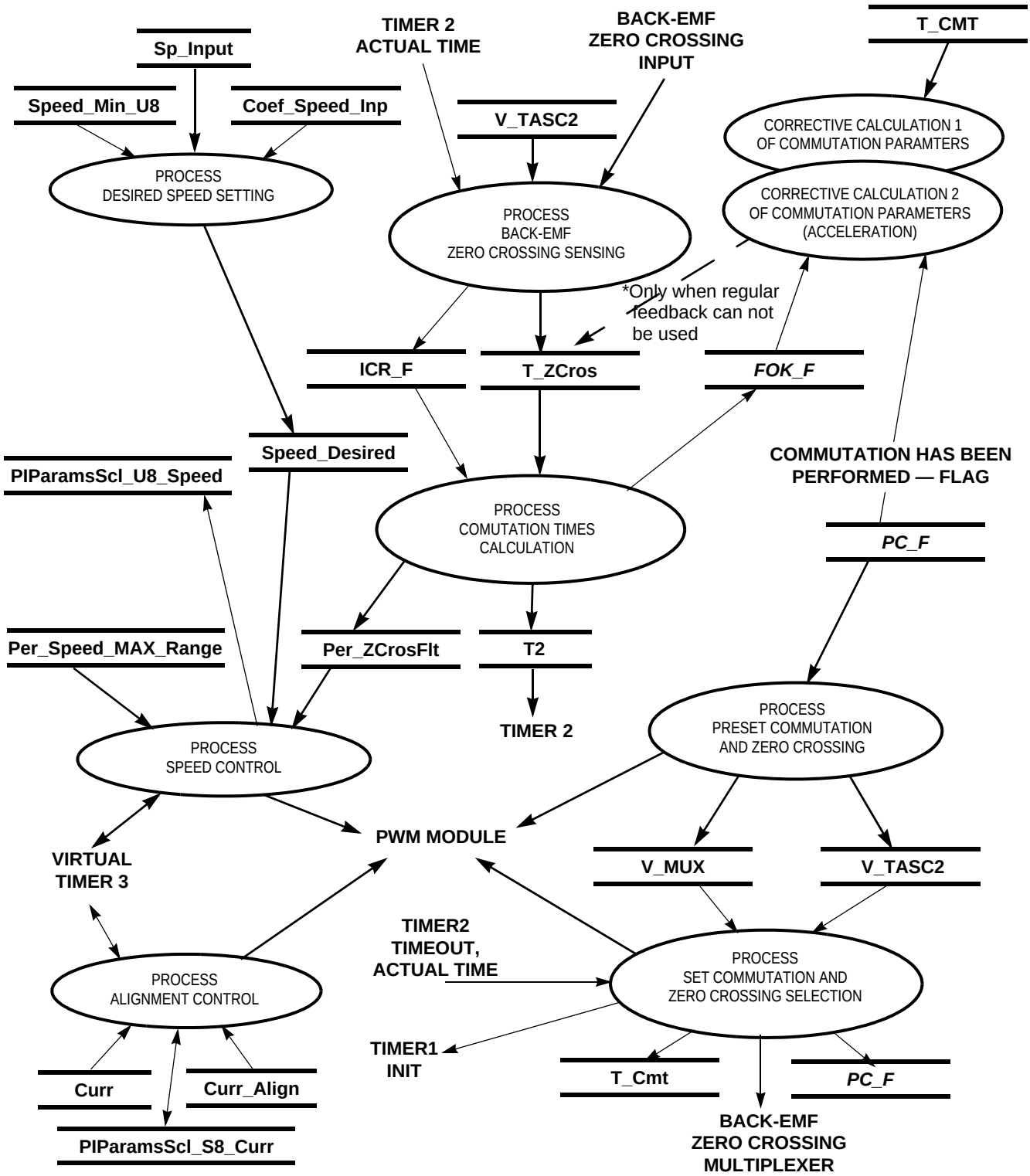

Figure 23. Main Data Flow — Part 2: Alignment, Starting, Running Control

Process Commutation Time Calculation, Corrective Calculation 1, Corrective Calculation 2

These processes provide calculations of commutation time intervals (periods) (**Per_ZCros**, **Per_ZCrosFlt**), from captured time (**T_Cmt**, **T_ZCros**, **T_ZCros0**), and sets Timer 2 with variable **T2**. These calculations are described in **Starting — Commutation Time Calculation** and **Running — Commutation Time Calculation**.

Process Desired Speed Setting

The desired speed, held in register **Speed_Desired**, is calculated from the following formula:

Speed_Desired = **Sp_Input***Coef_Speed_Inp/255 + **Speed_Min_U8**

Processs Speed Control

The general principle of the speed PI control loop is illustrated in **Figure 24**.

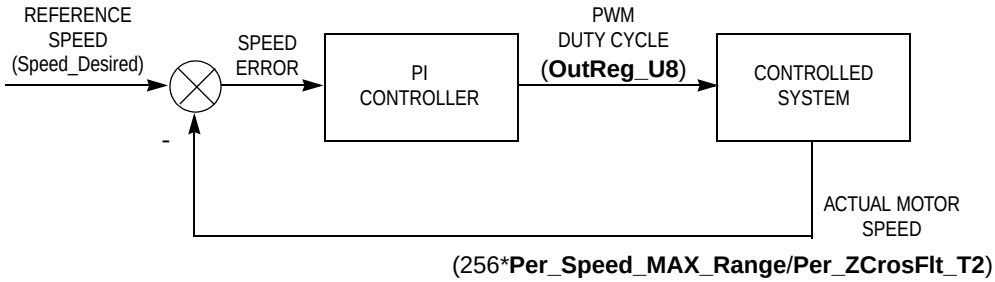

Figure 24. Closed Loop Control System

The speed closed loop control is characterized by the feedback of the actual motor speed.

The actual motor speed is calculated from zero crossing period:

Actual motor speed = 256***Per_Speed_MAX_Range/Per_ZCrosFlt_T2**

This information is compared with the reference set point and the error signal is generated. The magnitude and polarity of the error signal corresponds to the difference between the actual and desired speeds. Based on the speed error, the PI controller generates the corrected motor voltage in order to compensate for the error. The speed regulator parameters (gain...), internal, and input/output variables are located in the structure **PIParamsScl_U8_Speed**.

The speed controller works with a constant execution (sampling) period. The period is timed by timer 3, with the constant **PER_T3_RUN_US**.

PWM duty cycle is set for all six PWM channels according to regulator output, **OutReg_U8**. The maximum duty cycle is at **OutReg_U8** = 255. The implementation is described in **Implementation Notes — BLDC Speed Control and Calculation**.

Process Alignment Control

The process alignment control controls the current, Curr, using the PI regulator during alignment state (see **State Diagram**). The dc-bus current is regulated to required value Curr_Align. The current regulator parameters (gain...), internal, and input/output variables are located in the structure **PIParamsScl_S8_Curr**.

The current controller works with a constant execution (sampling) period. The period is timed by timer1, with the constant **PER_CS_T1_US**.

Processes Commutation and Zero Crossing Preset and Set

The processes commutation and zero crossing preset and set provides the BLDC commutation and zero crossing selection. Here the BLDC commutation means generation of the six step commutation which creates the voltage system shown in **Figure 2**. The required BLDC motor voltage system and commutation is provided using the MC68HC08MR32 PWM block.

The zero crossing selection means the selection of the required zero crossing phase as described in **Indirect Back EMF Sensing** and **Back-EMF Sensing Circuit**.The zero crossing selection is provided by the multiplexer setting.

As shown in **Figure 23**, the commutation and back-EMF zero crossing selection process is split into two actions:

- Preset commutation and zero crossing selection
  The preset means setting the buffered registers and RAM variables for commutation
- Set commutation and zero crossing selection
  The setting means loading the registers with buffered variables

The implementation is described in **Implementation Notes** - **BLDC Commutation and Zero Crossing Selection**.

Main Software Flowchart

The main software flowchart incorporates the main routine entered from the reset and interrupt states. The main routine includes initializing the MCU and the main loop. The flowcharts are shown in **Figure 25**, **Figure 26**, and **Figure 27**.

MCU Initialization is entered only after system reset. It provides initialization of system registers, ports, and CPU clock. The **MCU Initialization** is provided in **MCUInit()** function.

After **MCU Initialization** the **Application Initialization** is executed as **AppInit()** function, which performs the following actions. First the zero current offset of the dc-bus current measurement path is calibrated. This offset on the ADC input should be 1.65 V at zero current. This is implemented in the hardware design, in order to be able to measure negative and positive current values. The status registers are initialized and PWM generator is started. Also, timer 1 is started at the right moment to be synchronized with the PWM

generator. This way the current measurement is executed at the defined moment of the PWM signal.

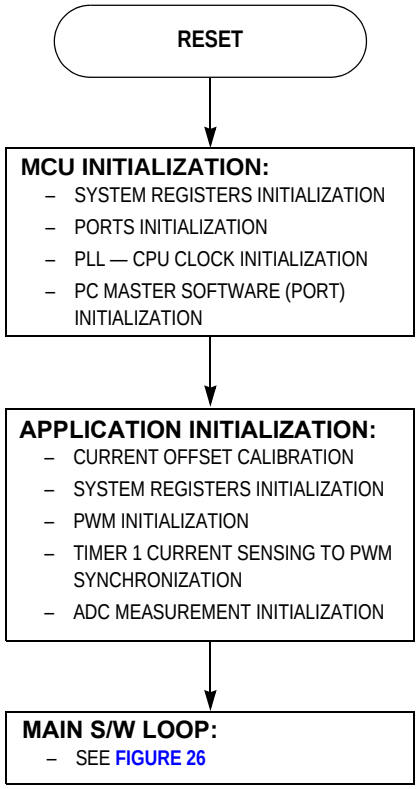

Figure 25. Main Software Flowchart

In the **Stand-By** state function, the start/stop switch is checked using **StSWReadStart ()** function. The **DecideStaSto ()** function is called to decide if the application should start. The start condition differs if manual or PC master software mode is set. When in manual mode (**PCMode** = 0), the start condition is the switch in the start position. When PC master software mode (**PCMode** = 1), the start condition is a start request from PC master software (**StartCtrl** = 1). In both modes, **Stop_F** is cleared when the software evaluates the start condition. When **Stop_F** is cleared, the software checks the over-voltage condition and the application starts.

The system **Alignment** and **Starting (Back-EMF Acquisition)** states are provided by **Alignment()** and **Start ()** functions in the **code_start.c** file, both are called from **main()**. The functionality during the start and running state is described in **Sensorless Commutation Control**. During the starting (back-EMF acquisition) state, the commutation time preset calculations are prepared in the **StrtCmtPreset()** function, and commutation time set calculations are provided by the **StrtCmtSet()** function.

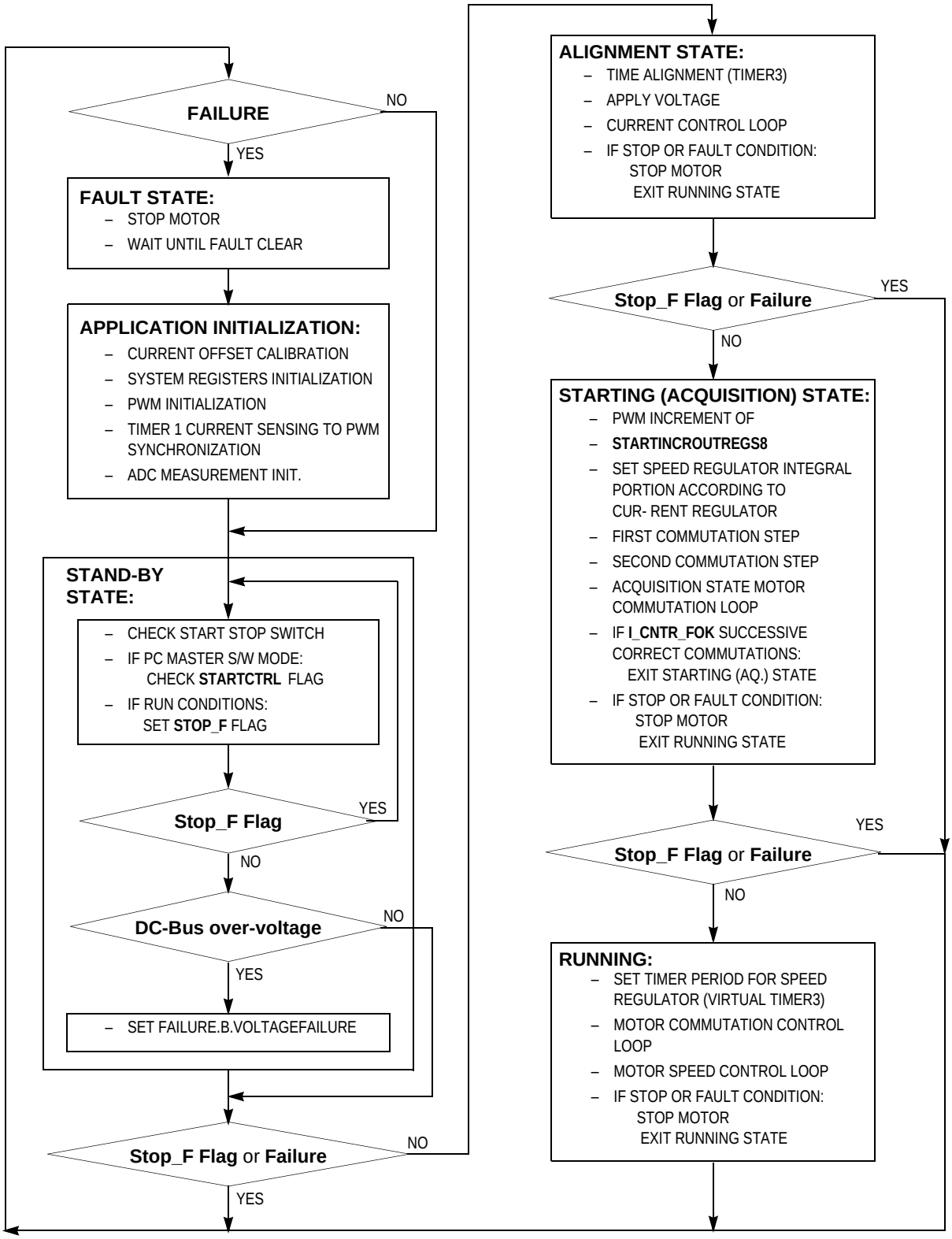

Figure 26. Main Software Flowchart — Main Software Loop

When the start is successfully completed, the **Running ()** function is called from **main()**. During the **Running** state, the commutation time preset calculations are provided by the **CmtPreset()** function, and commutation time set calculations are provided by the **CmtSet()** function.

During the **Running**, **Start** or **Alignment** states, the **DecideStop** () function is called to check drive stop conditions and can set the **Stop_F** flag. When the stop flag is set the **MotorStop ()** function is called to stop the motor, and running, start, or alignment state is left. The software enters stand-by state.

Also, the commutation error (**Cntr_Err**>= **MAX_ZC_ERR**) and over-current (**OverCurrent** flag = 1) fault are checked in **ERRHndl ()** and **OVCurrent ()** functions during the **Running**, **Start** or **Alignment** states. If any error is detected, the function **MotorStop ()** function is called. Then the software enters **Fault** state through the **Fault()** function. This is only left when the failures are cleared (variable **Failure** = 0). This decision is provided **DecideCleSto ()** function, called from **ErrorStop ()**. In manual control, the failures are cleared by setting the Start/Stop switch to Stop. In case of PC master software control, the failures are cleared by the **ClearFail** flag from the software. When the failures are cleared, the software enters **Application Initialization**.

Besides the main loop, there are three interrupts: timer 1, timer 2, and PWM reload interrupts. They are described in **Figure 27**.

Interrupt Timer 1 is periodically called with period **PER_CS_T1**. The interrupt function provides dc-bus current measurement and virtual timer 3 service, where timer 1 is providing the timer 3 time base. When over-current is discovered, the **OVC_F** flag is set. Finally, the ADC is set according to the **Nxt_Chnl** variable to prepare speed potentiometer or temperature sensing. This interrupt is provided by the **TIMACh1_Int()** function.

Interrupt Timer 2 sets commutation actions. If commutation is enabled (**CmtE_F** flag is set), the following actions are done:

- PWM commutation step
- Synchronization of timer 1, for current measurement with PWM
- Phase commutated flag **PC_F** is set
- Actual time (from timer counter register) = commutation time is recorded in **T_Cmt**.

This interrupt is provided by the **TIMACh3_Int()** function.

Interrupt PWM Reload provides back-EMF zero crossing sensing. The zero crossing input is sampled 2 or 3 times. The back-EMF state value is compared with expecting (rising/falling) edge. If the value corresponds with expecting edge, the zero crossing get flag **ICR_F** is set, and the actual time (from timer counter register) = zero crossing time is recorded in **T_ZCros**. This interrupt is provided by **PWMMC_Int()** function in **code_isr.c**.

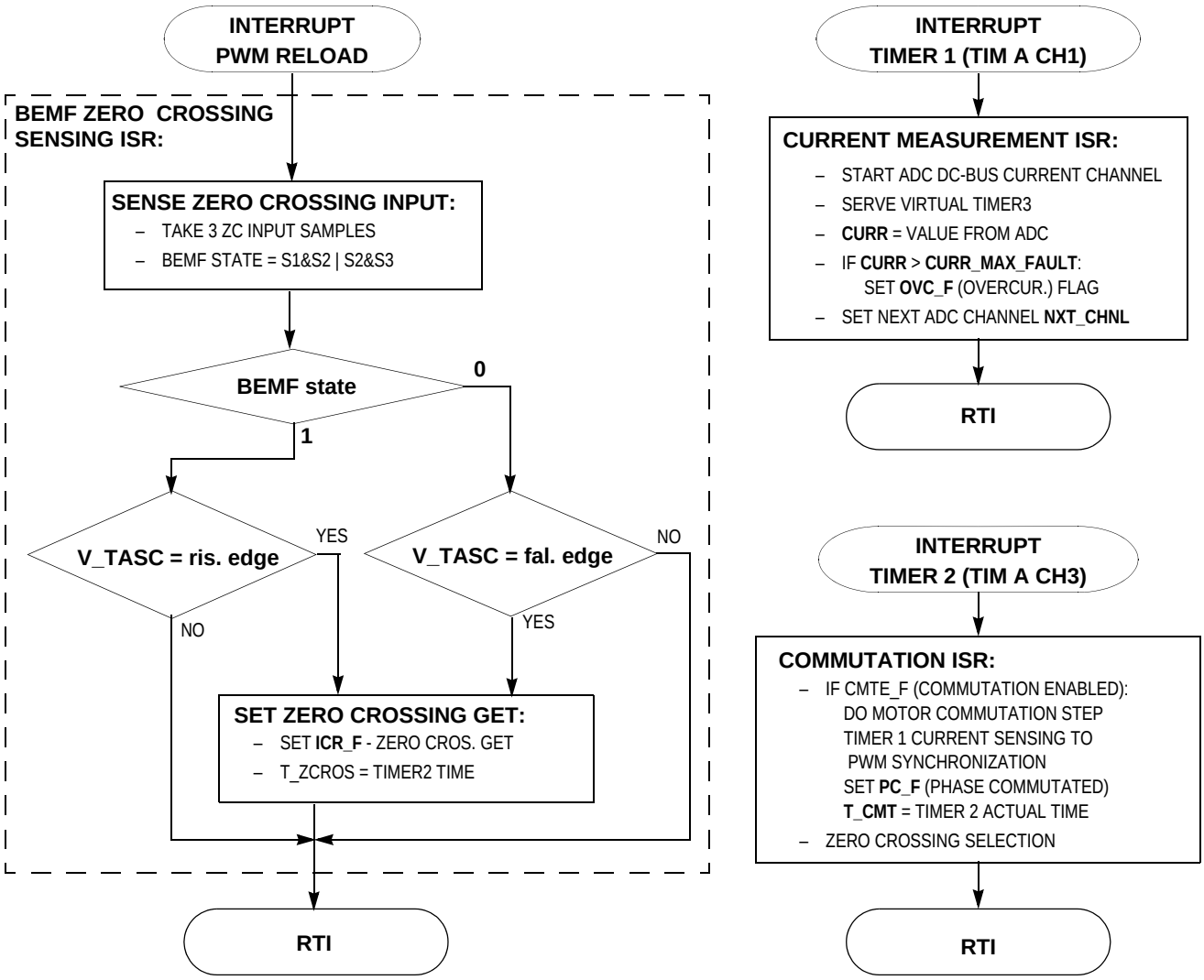

Figure 27. Software Flowchart — Interrupts

State Diagram

The motor control application can be in one of the eight states shown in Figure 28. Each of these states is described in the subsections following the figure.

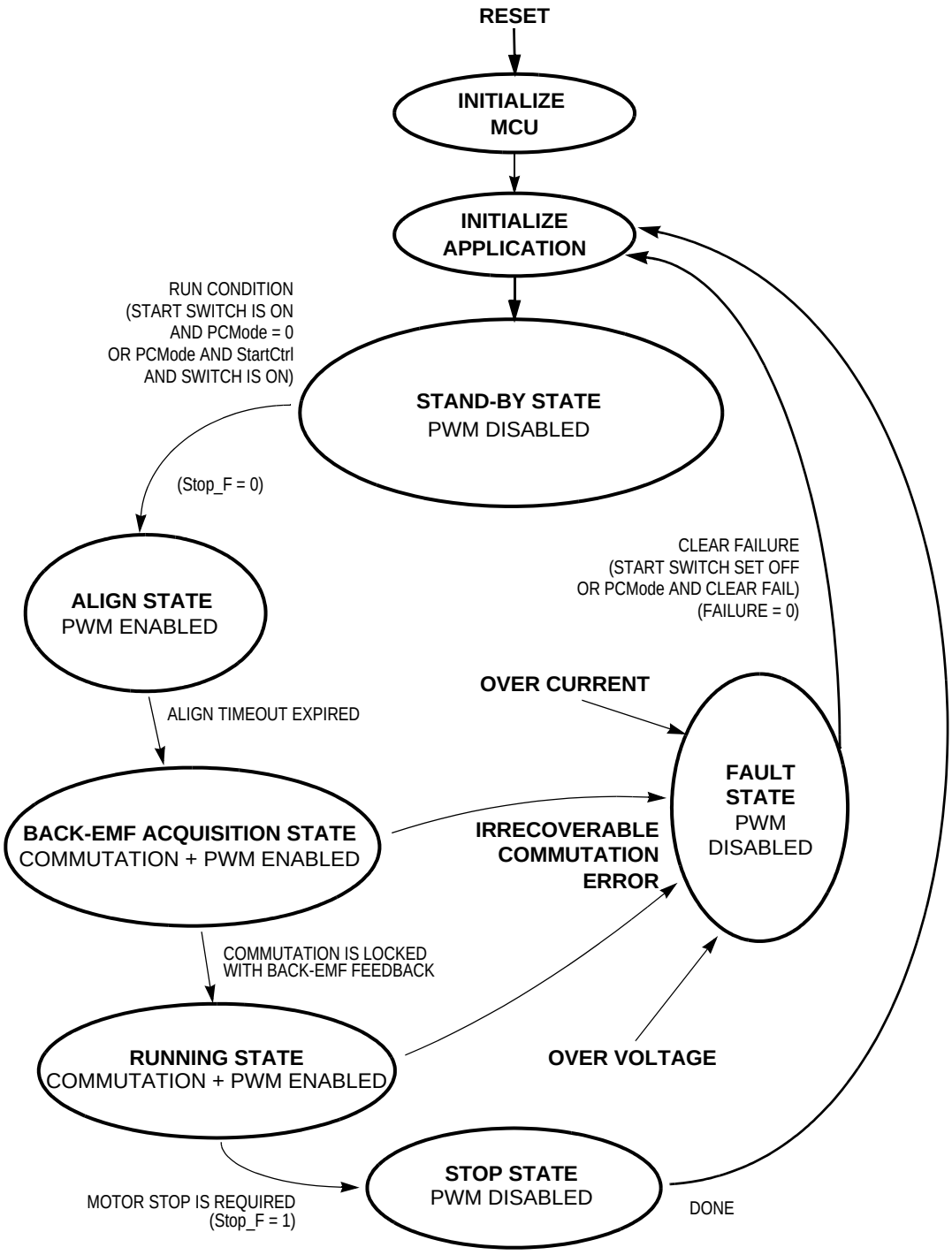

Figure 28. Application State Transitions

Initialize MCU

This state is entered after the MCU is reset, and performs the following actions:

- MCU ports are configured for the application
- Some application (system) variables are initialized
- MCU clock PLL is locked
- Hardware boards used are identified, and parameters initialized
- PC master communication software is initialized with SCI port
- ADC is initialized

and the state is exited.

Initialize Application

This state is used as an application reset, called following a return from fault or stop states.

In this state the following actions are done:

- Current measurement path calibrated and checked for error
- Some application (system) variables initialized
- Some MCU peripherals are configured (timer, OC function, PWM)
- PWM outputs for motor control are turned OFF
- Timer 1 (Tim A Ch1) is synchronized with the PWM cycle
- Speed input, dc-bus voltage and temperature measurement is initialized
- Ready LED is turned ON

and the state is exited.

Stand-By

State diagram for this software is shown in **Figure 29**.

Current measurement and current calibration when PWM is OFF

Before testing of the start switch, dc-bus current is measured when PWM is OFF. This way the dc-bus current measurement path is calibrated. The calibrated value **Offset0_Curr** is used for the final current calculation.

This offset on the ADC input should ideally be 1.65 V at 0 dc-bus current.

If the sensed value exceeds the limit (Offset_Max_Curr) when PWM is OFF, this indicates some hardware error, and the control flow enters into the fault state:

Start condition test

The start condition is tested. If in manual mode (PCMode = 0), the start condition is a movement in the switch from stop to start. In PC master software mode (PCMode = 1), the start condition in switch start position and StartCtrl = 1. If start condition valid then Strt_F is set, Stop_F is cleared, and the next state entered.

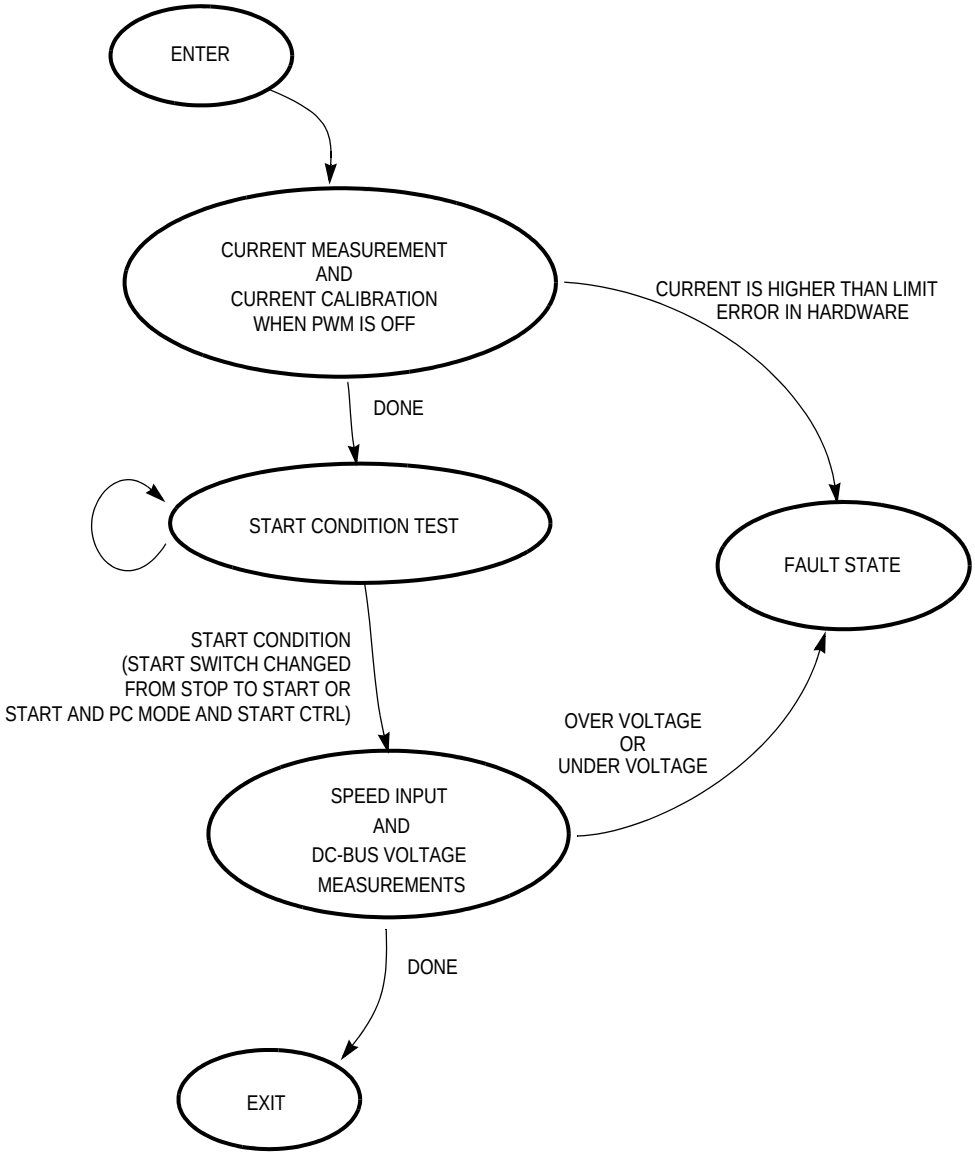

Figure 29. Stand-by State

Speed input and dc-bus voltage measurements

The dc-bus voltage is measured after the start switch is turned ON. This prevents the measurement being disturbed by the power turn ON. Where that dc-bus voltage is not within the limits, the control flow enters into the fault state.

Align State

In the align state the rotor position is stabilized by applying PWM signals to only two motor phases (no commutation). When preset time-out expires, then this state is finished. See **Figure 30**.

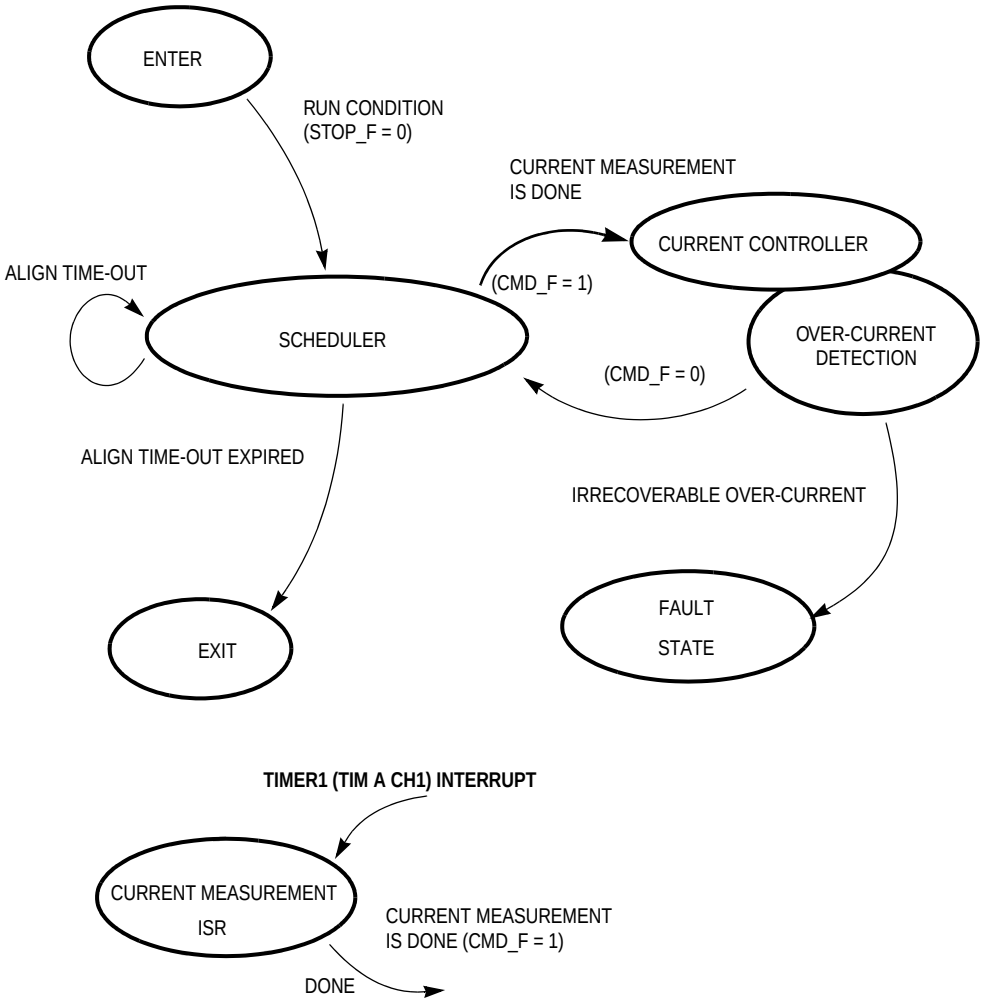

Figure 30. Align State

Scheduler

The scheduler handles the state transitions in the align state. The dc-bus current measurement is done in OC interrupt service routine, in order to keep synchronization with PWM cycle. After measurement is made, the scheduler allows calculation by the current controller and the over-current detection. The CMD_F (current measurement done) flag indicates that the new value of dc-bus current is ready to be processed by the current controller.

The time-out (software timer 3) of this state is defined in the software by the constants: **PER_T_ALIGN** and **PER_BASE_T3_ALIGN**.

Current Controller

The current controller subroutine is called every **PER_CS_T1_US** µs (128 µs with default software setting) after a new value of the dc-bus current has been obtained (**CMD_F**=1). It sets all six PVALx register pairs to get the right PWM ratio for the required current.

Timer 1 Interrupt

Once the synchronization of OC function with the PWM cycle has been achieved, it must be maintained because the current measurement is initiated here.

Over-Current Detection

The dc-bus current is periodically sensed and the over-current condition is evaluated. After a defined number successive over-current events **I_CNTR_OVC**, the control flow enters into the fault state.

Back-EMF Acquisition State

The back-EMF acquisition state provides the functionality described in **Starting (Back-EMF Acquisition)** and **Starting — Commutation Time Calculation**. **Figure 31** shows the state transitions for the state.

First Commutation

After the align state time out expires, voltage is applied to another phase pair. The first commutation is made and the PWM duty cycle is constant. This value has been defined by the current controller during the Aalign state.

The calculation of the commutation time is explained in **Starting — Commutation Time Calculation**.

Second Commutation

The commutation time (T2) is calculated from the previous commutation time and the start commutation period (**Per_CmtStart**). This time is set to timer 2. Then, the new PWM multiplexer logic data is readied for when the commutation interrupt performs the next commutation.

The calculation of the commutation time is explained in **Starting — Commutation Time Calculation**.

Measurements Handler

As explained in **Scheduler** and **Over-Current Detection**, the dc-bus current is scanned and over-current condition is evaluated. Where there is an over-current, the control enters the fault state (see **Fault State**). The voltage and the speed commands are scanned too. When this state is finished, the values of dc-bus current, dc-bus voltage, and speed command are updated.

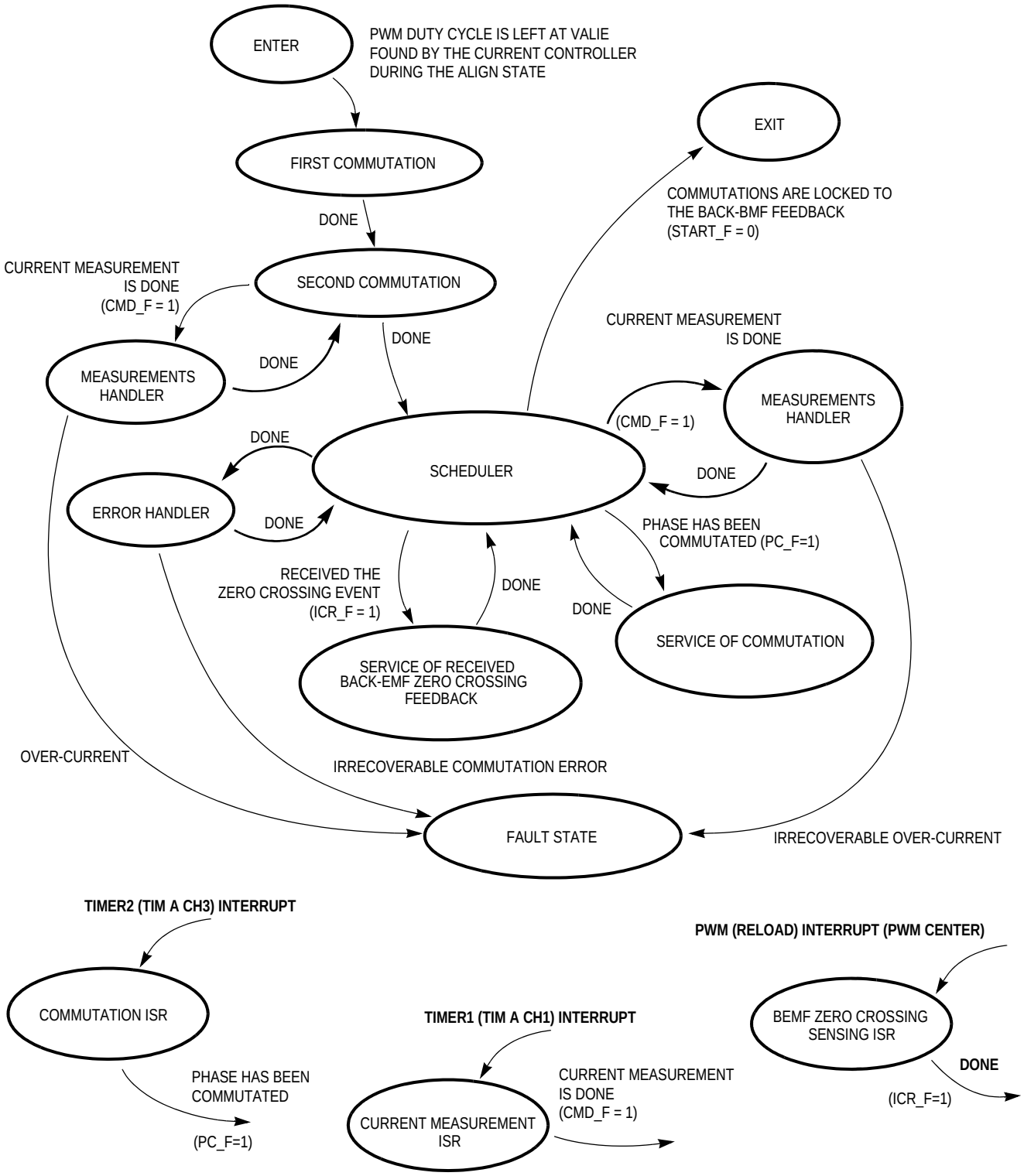

Figure 31. Back-EMF Acquisition

Service of Commutation

As already explained, the motor phase commutation is performed in the OC interrupt service routine. The phase commutated flag (**PC_F**=1) indicates this action to the scheduler, which allows the performed commutation to be serviced. Detailed explanation of this state is in **Processes Commutation and Zero Crossing Preset and Set**.

Service of Received Back-EMF Zero Crossing

The back-EMF zero crossing is detected by PWM middle function block. Then the appropriate flag (captured received the zero crossing event - ICR_F) is set by PWM centre interrupt service routine. This indicates to the scheduler that the zero crossing event must be served.

The following actions are taken:

1. Commutation parameters are recalculated more precisely based on the received feedback
2. Commutation time is preset to the output compare register of timer 2

For a better understanding of how the commutation process works, see **Sensorless Commutation Control**, (see **Starting — Commutation Time Calculation**).

BEMF Zero Crossing Sensing Interrupt Service Routine

This ISR is used to evaluate the back-EMF zero crossing. Back-EMF is evaluated here in order to synchronize zero crossing capture with the middle of central-aligned PWM. This technique rejects the noise caused by PWM from the back-EMF signal. When this ISR is initiated, then three samples of the zero crossing input (**BEMF_IN**) are taken and the state is evaluated. Based on the expected edge (**V_TASC2,** ELS2A_ELS2B) and the evaluated state of the **BEMF_IN** pin, the zero crossing event is verified. If it is accepted, then the captured time is stored in variable (**T_ZCros**) and the PWM ISR is finished. The appropriate flag (captured received the zero crossing event — **ICR_F**) is set.

Current Measurement Interrupt Service Routine

The output compare function is used to synchronize initiating the dc-bus current sampling with the PWM cycle, and also for the commutation timing.

Error Handler

If the BLDC motor is controlled properly, commutation events must be locked to the back-EMF zero crossing feedback. When that feedback is lost, commutation time is derived from previous commutation events. If feedback does not recover during a defined number of commutations (constant — C_MaxErr), then the situation is evaluated as irrecoverable commutation error, and the fault state is entered.

Measurement Handler

The measurement handler assures that the measurement process is done in the right order. The dc-bus voltage, speed command, and temperature are scanned sequentially. After the state has run three times, all the values for dc-bus current, dc-bus voltage, speed input, and temperature are updated. dc-bus current is scanned with a constant time period in the current measurement ISR, but the over-current condition is evaluated in the main software loop. After a defined number (I_OVC_Cnt) of successive over-current events, the control flow enters into the fault state.

Running State

The BLDC motor is run with regular feedback. The speed controller is used to control the motor speed by changing the value of PWM duty cycle. **Figure 32** shows the state transitions.

Measurements Handler

Explained in **Back-EMF Acquisition State**.

Service of Commutation

Explained in **Back-EMF Acquisition State**.

Service of Received Back-EMF Zero Crossing

Explained in **Back-EMF Acquisition State**. The difference is that the commutation parameters are recalculated with different constants (see **Running — Commutation Time Calculation**)

BEMF Zero Crossing Interrupt Service Routine

Explained in **Back-EMF Acquisition State**.

Current Measurement Interrupt Service Routine

Explained in **Back-EMF Acquisition State**.

Error Handler

Explained in **Back-EMF Acquisition State**.

Over Current

Explained in **Back-EMF Acquisition State**.

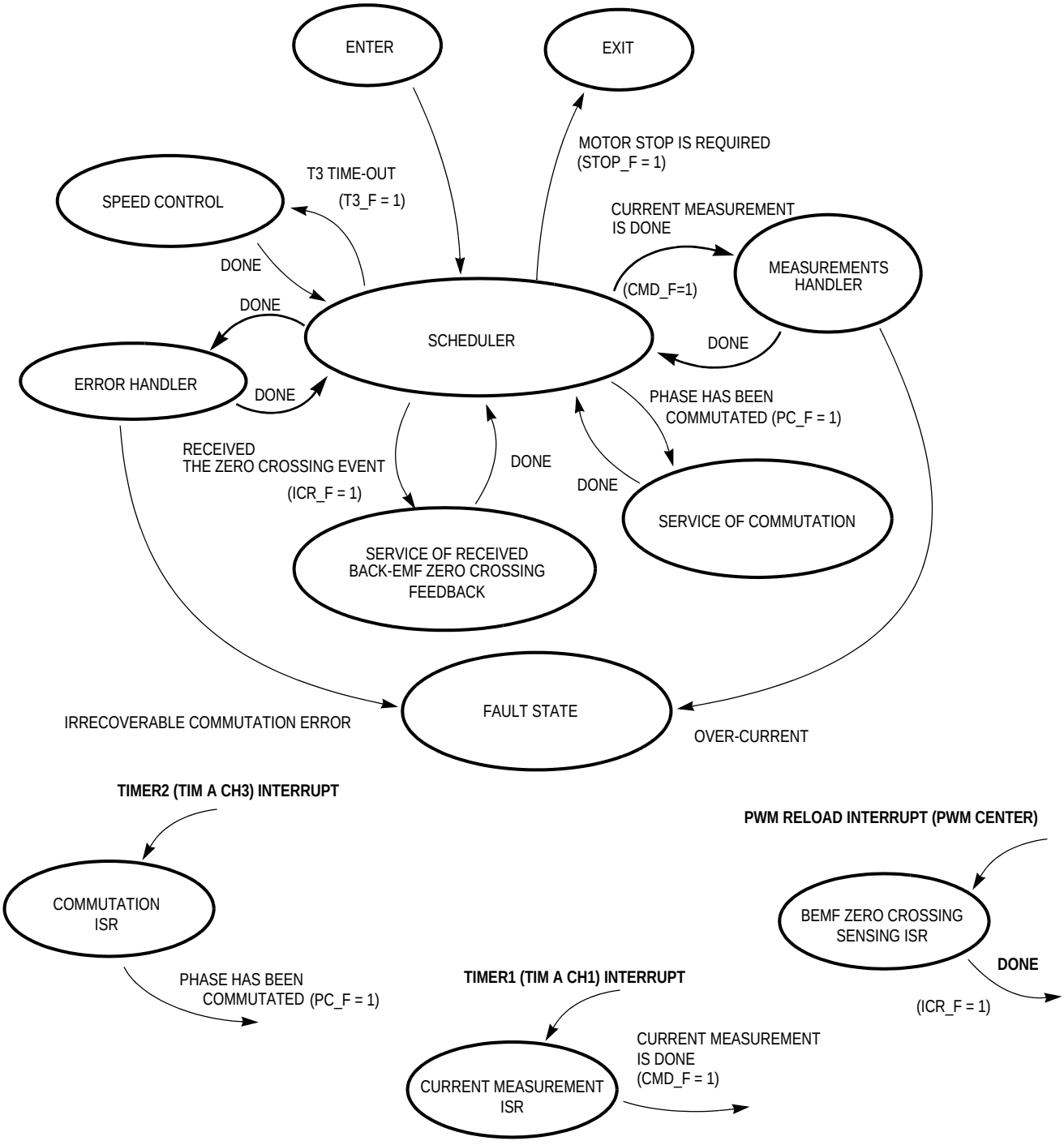

Figure 32. Running State

Stop State

When motor stop is required, the PWM signals are disabled and the power switches are switched off. The state diagram for this state is shown in **Figure 33**.

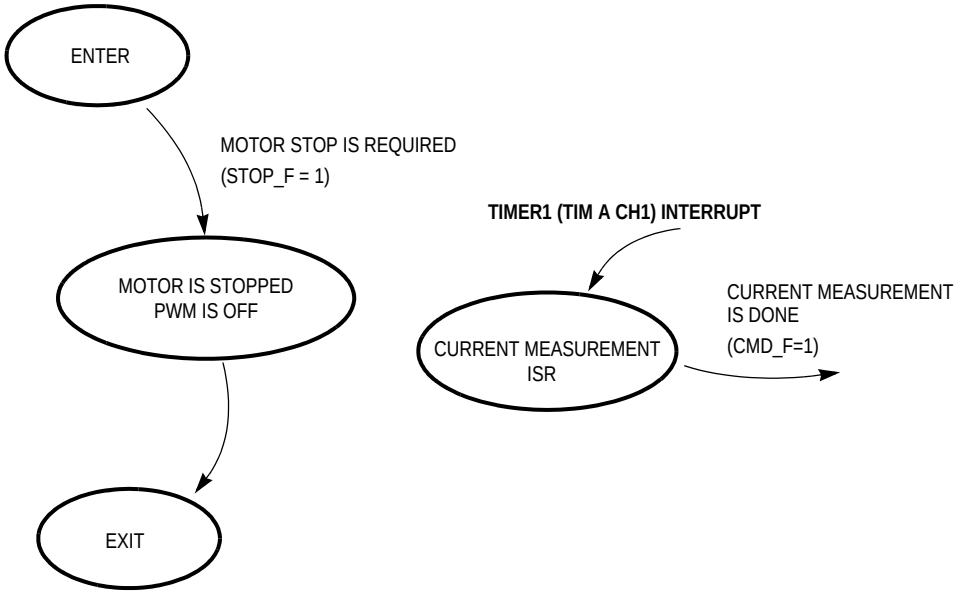

Figure 33. STOP State

Fault State

If over-voltage, over-current, or commutation fault occurs, the motor control PWM's are disabled and control enters the fault state, where it remains until RESET or clear failure (start switch set OFF or PCMode&ClearFail). The fault state is indicated by the Red LED diode blinking.

Clear Failure Test

The failure (Failure = 0) is tested. In manual mode (PCMode = 0), the switch in the stop position clears the failure. In PC master software mode (PCMode = 1), the failure is cleared by ClearFail flag or the switch in the stop position. If start condition is valid Strt_F is set, Stop_F is cleared, and the next state entered.

See **Figure 34**.

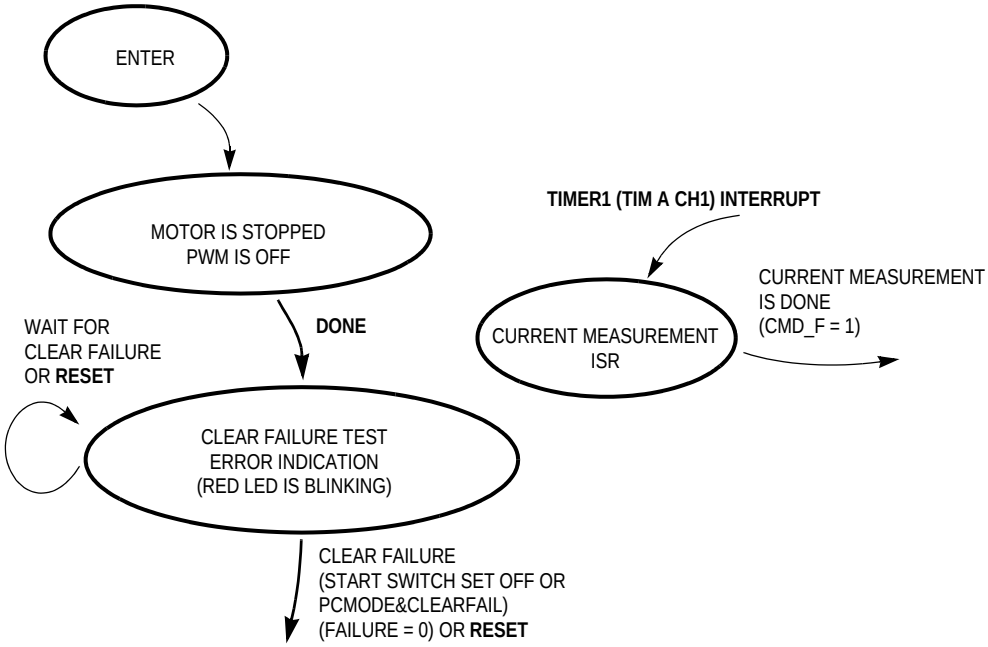

Figure 34. Fault State

Implementation Notes

Software Files

The software files and structure are described in an application note entited *Sensorless BLDC Motor Control on MC68HC908MR32 — Software Porting to Customer Motor* (Motorola document order number AN2356/D).

BLDC Commutation and Zero Crossing Selection

The required BLDC motor voltage system commutation is provided using the MC68HC08MR32 PWM block.

The zero crossing selection is provided by setting port F pins PTF1–PTF3 connected to the multiplexer.

As shown in **Data Flow**, the commutation and back-EMF zero-crossing selection process is split into two actions:

1. Preset BLDC commutation and BEMF zero-crossing selection
2. Set BLDC commutation and BEMF zero-crossing selection

Preset BLDC Commutation and BEMF Zero Crossing Selection

In each phase of the 6-step commutation two PWM channels (bottom and top switch) are active and the other four PWM channels are logical 0. The commutation preset is accomplished by setting the buffered registers PVAL1H, PVAL1L through PVAL6H, PVAL6L. To preset the active PWM channel, the MSB bits of the dedicated PVALxH registers are set to 0.To preset the PWM channel to the logical 0, the MSB bits of the dedicated PVALxH registers are set to 1. This is due to the signature bit functionality as described in *68HC908MR32, 68HC9908MR16 Advance Information* (Motorola document order number MC68HC908MR32/D) .This commutation preset is provided by the function **Commut ()**. In **Commut()**, the function field **Set_PWM(P_Step_Cmt] ()** calls one of **Shft_PWM0..5()** functions according to **P_Step_Cmt** value, so the PVALxH registers are set as required. This will not effect the PWM signals until **LDOK** bit is set. Also, the back-EMF zero crossing selection preset is provided in this function, by setting **V_MUX** variable according to **P_Step_Cmt** value.

Set BLDC Commutation and BEMF Zero Crossing Selection

The commutation set is provided in timer 2 interrupt function **TIMACh3_Int()** in the **code_isr.c** file. The **PWMEN** is toggled and **LDOK** bit set, in order to immediately restart the PWM generator and reload the PWM with the buffered PVAL1H, PVAL1L through PVAL6H, PVAL6L registers. Then the back-EMF zero crossing selection set is provided by setting PORTF according to **V_MUX** variable.

BLDC Speed Control and Calculation

Desired speed calculation and PWM duty cycle setting is quite simple, but there are some C language syntax tricks. Also, the scaling aspect needs to be taken into consideration.

Desired Speed Calculation

The 8 bit value **Speed_Desired** is calculated using 8-bit multiplication of two 8-bit variables, **Sp_Input** and **Coef_Speed_Inp**. The syntax is:

(unsigned char)((**Sp_Input***Coef_Speed_Inp**)>>8)

This syntax is used to generate optimal code using the MUL instruction.

PWM Duty Cycle

PWM duty cycle is set for all six PWM channels according to regulators output **OutReg_U8**. The maximal duty cycle is at **OutReg_U8** = 255. The registers **PVAL1H, PVAL1L** through PVAL6H, PVAL6L are set proportionally to the PWM modulus register **PMOD** = **MCPWM_MODULUS** constant (100% duty cycle is when **PVALx** = **PMOD**).

The **PWM_Val_Max** variable is:

PWM_Val_Max = **DUTY_PWM_MAX***MCPWM_MODULUS**

This variable is used for scaling of the regulator output **OutReg_U8**. The registers PVAL1H, PVAL1L through PVAL6H, PVAL6L are loaded with **PWM_Val** calculated from **OutReg_U8**:

PWM_Val = **PWM_Val_Max***OutReg_U8/256**

This calculation is provided with macro **umul_16x8_macro**.

Timers

Timer 1 and timer 2 are implemented using MC68HC08MR32 timers. Timer 3 is a software virtual timer using time base of timer 1.

Timer 1

Timer 1 is provided by timer A channel 1 set in output compare mode. In this mode the registers TACH1H and TACH1L are used for setting the output compare value, T1.

- TACNTH and TACNTL form a 16-bit timer A counter with an infinite counting 16-bit roll over.
- The timer A channel 1 interrupt is called whenever TACH1H = TACNTH and TACH1L = TACNTL. With each interrupt, the registers TACH1H and TACH1L are loaded with the new value, **T1 = T1+PER_CS_T1**, where **T1+PER_CS_T1** is a 16-bit addition with no saturation. So, the constant interrupt period **PER_CS_T1** of the timer T1 interrupts is generated.
- The timer unit **UNIT_PERIOD_T1_US** of timer A is determined by the MCU bus frequency (8 MHz with a 4-MHz oscillator and default software setting) and timer prescaler set in TASC. So, the default software value is 2 μs. The timer 1 interrupt function is provided by the **TIMACh1_Int ()** function.

Timer 2

Timer 2 is provided by timer A channel 3 set in output compare mode. In this mode, the registers TACH3H and TACH3L are used for setting the output compare value, T2.

- TACNTH and TACNTL form a 16-bit timer A counter, with infinite counting 16-bit roll over.
- The actual time is sensed from TACNTH and TACNTL base (e.g., time of the commutation, **T_Cmt**).
- The timer A channel 3 (timer 2) interrupt is called whenever TACH3H = TACNTH and TACH3L = TACNTL. So, the registers TACH3H and TACH3L are loaded with the T2 variable value.
- The value T2 (e.g., **T2 = T_Cmt + Per_HlfCmt**) is calculated using a 16-bit addition with no saturation. So, the time duration up to 65,536 **UNIT_PERIOD_T2_US** from actual time (TACNTH, TACNTL) can be timed at any TACNTH, TACNTL timer A counter value.
- The timer unit **UNIT_PERIOD_T2_US** of timer A is determined by the MCU bus frequency (8 MHz with 4-MHz oscillator and default software setting) and timer prescaler set in TASC. So, the default software value is 2 μs. The timer 2 interrupt function is provided by function **TIMACh3_Int()**.

References

1. *Sensorless BLDC Motor Control on MC68HC908MR32 - Software Porting to Customer Motor* (document order number AN2356/D), Motorola 2002
2. Motion Control Development Tools found on the World Wide Web at: http://e-www.motorola.com
3. *Motorola Embedded Motion Control MC68HC908MR32 Control Board User's Manual*, (document order number MEMCMR32CBUM/D), Motorola 2000
4. *Motorola Embedded Motion Control 3-Phase AC BLDC High-Voltage Power Stage User's Manual* (document order number MEMC3PBLDCPSUM/D), Motorola 2000
5. *Motorola Embedded Motion Control Optoisolation Board* (document order number MEMCOBUM/D), Motorola 2000
6. *Motorola Embedded Motion Control Evaluation Motor Board User's Manual* (document order number MEMCEVMBUM/D), Motorola 2000
7. *Motorola Embedded Motion Control 3-Phase BLDC Low-Voltage Power Stage User's Manual* (document order number MEMC3PBLDCLVUM/D), Motorola 2000

8. User's Manual for PC Master Software, Motorola 2000, found on the World Wide Web at:
   http://e-www.motorola.com
9. *68HC908MR32, 68HC908MR16 Advance Information* (document order number MC68HC908MR32/D), Motorola
10. *Low Cost High Efficiency Sensorless Drive for Brushless DC Motor using MC68HC(7)05MC4* (document order number AN1627), Motorola

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com