

3-phase Sensorless BLDC Motor Control Development Kit with Qorivva MPC5606B MCU

by: Josef Kramoliš
Rožnov pod Radhoštěm, Czech Republic

1 Introduction

This application note describes a three-phase brushless DC (BLDC) motor control using a sensorless algorithm. This cost-effective solution is based on the [MPC5606B device](#), a member of the automotive body MPC5607B family of microcontrollers.

System concept, BLDC motor control theory and software implementation chapters were adopted from the application note titled *Three-phase Sensorless BLDC Motor Control Kit with the Qorivva MPC5604P* (document number AN4268) [2].

The system can control a three-phase BLDC motor without a positional feedback sensor. Application features include:

- Three-phase BLDC sensorless speed control
- Back-EMF (BEMF) sensing
- Application control user interface using the FreeMASTER debugging tool [8]
- Maximal motor current limitation

Contents

1	Introduction	1
2	System concept	2
3	BLDC Sensorless Control	3
3.1	Brushless DC motor	3
3.2	Principles of six-step BLDC motor control	4
3.3	BLDC motor control states	7
4	MPC560xB Controller Board Configuration	9
4.1	Enhanced modular input/output subsystem	10
4.2	Periodic interrupt timer (PIT)	12
4.3	Cross triggering unit (CTU)	13
4.4	Commutation function	14
5	Software Implementation	15
5.1	Introduction	15
5.2	Application flow	16
5.3	Speed evaluation and control	17
5.4	Zero-cross detection	19
5.5	Current limitation controller	19
5.6	State machine	19
5.7	Library functions	23
5.8	Setting the software parameters for a specific	23
6	FreeMASTER User Interface	24
6.1	Application start	24
7	Performance	26
8	Conclusion	26
9	References	27
10	Acronyms	28

2 System concept

The system can control a three-phase BLDC motor. The application meets the following performance specifications:

- Running on the 3-phase Sensorless BLDC Motor Control Development Kit with Qorivva MPC5606B MCU [1] (see Figure 1)
- Targeted at the MPC5606B microcontroller (refer to [3] for more information)
- Control technique incorporating:
 - Sensorless control of a three-phase BLDC motor
 - Zero-crossing technique
 - Closed-loop speed control
 - Closed-loop current control
 - Start-up with alignment
 - BEMF voltage sensing
 - 50 μ s sampling period with the FreeMASTER recorder
- FreeMASTER software control interface (motor start/stop, speed set-up)
- FreeMASTER software monitor
- DC bus overvoltage, undervoltage, and overcurrent protection

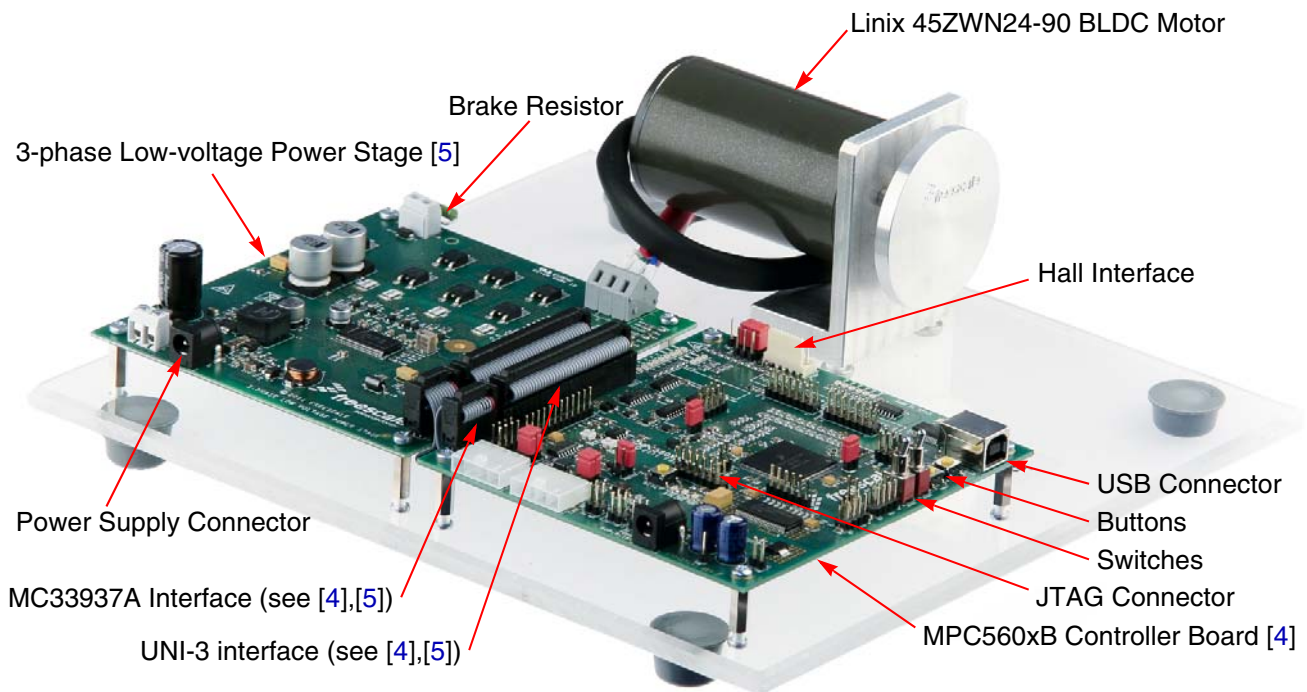


Figure 1. 3-phase sensorless motor control development kit with Qorivva MPC5606B MCU

3 BLDC Sensorless Control

3.1 Brushless DC motor

The BLDC motor is a rotating electric machine with a classic three-phase stator. As in an induction motor, the rotor has surface-mounted permanent magnets. There are no brushes on the rotor and the commutation is performed electronically at a certain rotor position. The displacement of the magnets on the rotor creates a trapezoidal BEMF shape, which means that a DC voltage with a rectangular shape (see Figure 2) can be used to create a rotational field with low torque ripples. The motor can have more than one pole-pair per phase. The number of pole-pairs per phase defines the ratio between the electrical revolution and the mechanical revolution. The rectangular shape of the applied voltage ensures control simplicity.

However, the rotor position must be known to align the applied voltage with the BEMF. Best efficiency is achieved when the BEMF and commutation events are aligned, in which case the motor behaves as a DC motor.

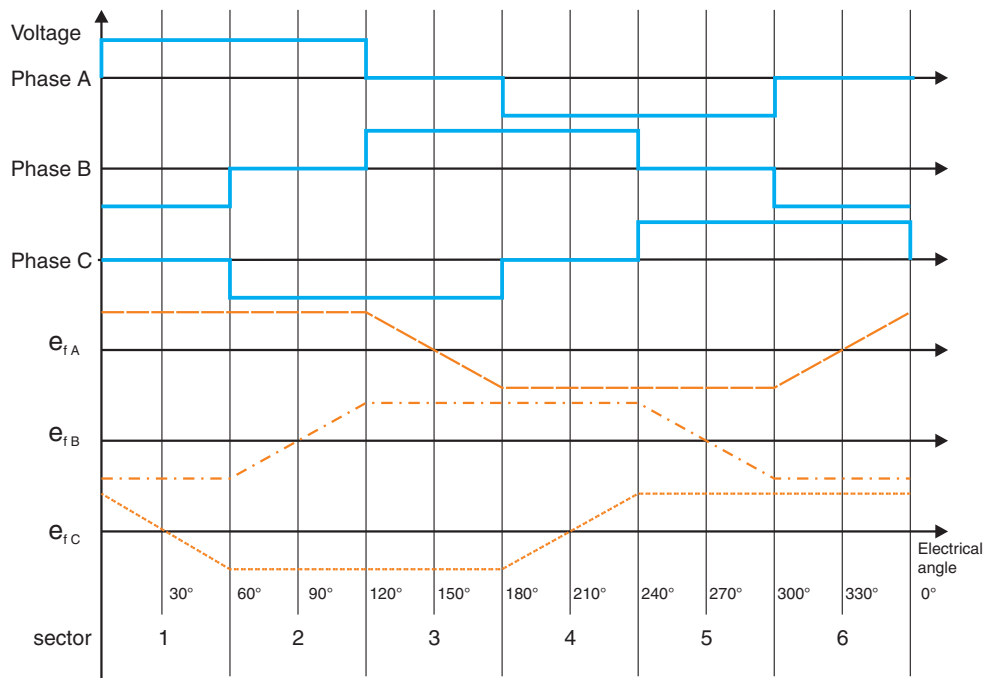


Figure 2. Three-phase voltage system for the BLDC motor

The main task for sensorless control of the BLDC motor is the estimation of rotor position. For rotor position estimation, this application uses a technique based on BEMF sensing. The key features of this technique are:

- Speed range from 5-10% up to 100% of nominal speed
- BEMF voltage must be high enough
- Based on the BEMF zero-crossing method

- Other advanced BEMF estimation techniques include:
 - System observers
 - Measurement of a non-conductive phase with multi-sampling

The following section discusses the concept of zero-crossing, as well as the methods and conditions for its correct evaluation.

3.2 Principles of six-step BLDC motor control

The three-phase BLDC motor operates in a two-phase model. Two phases that produce the highest torque are energized while the third phase is off. Which two phases are energized depends on the rotor position. The energized phases change every 60° (electrical degrees) as shown in Figure 2. The figure also displays ideal BEMF waveforms. The third phase can be used to observe the BEMF voltage to recognize the correct commutation event time point, as described later. Current commutation is executed by a six-step inverter as shown in the simplified form in Figure 3. Figure 2 and Table 1 demonstrates the switching sequence and current direction.

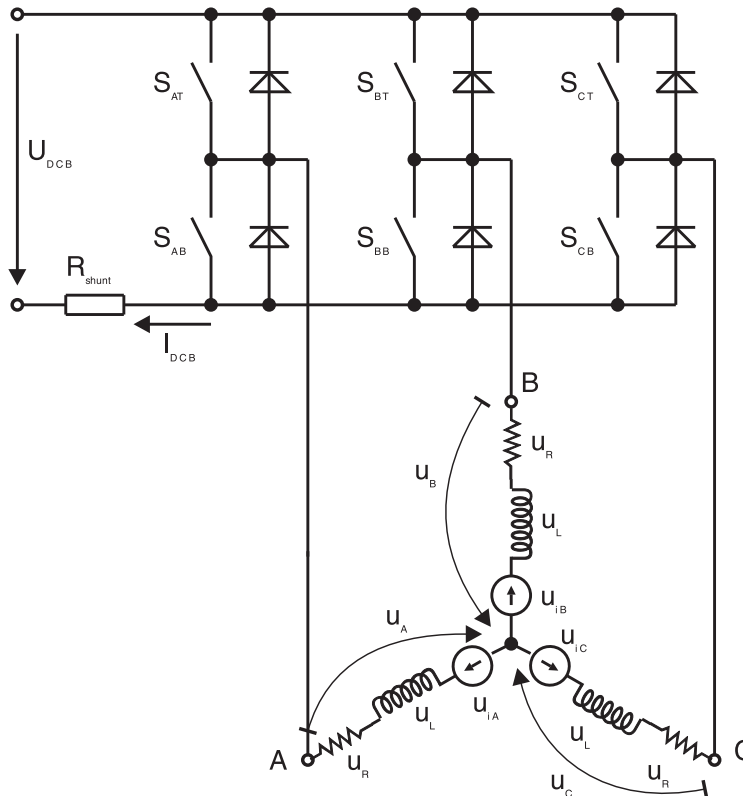


Figure 3. Power stage and motor topology

Table 1. Six-step switching sequence

Rotor Position	Sector Number	Switch Closed		Phase Current		
				A	B	C
0°-60°	1	S _{AT}	S _{BB}	+	-	off
60°-120°	2	S _{AT}	S _{CB}	+	off	-
120°-180°	3	S _{BT}	S _{CB}	off	+	-
180°-240°	4	S _{BT}	S _{AB}	-	+	off
240°-300°	5	S _{CT}	S _{AB}	-	off	+
300°-360°	6	S _{CT}	S _{BB}	off	-	+

To explain and simulate the idea of the BEMF sensing technique, this document provides a simplified mathematical model based on the basic circuit topology (see [Figure 3](#)). The voltage for a three-phase BLDC motor is supplied by a typical three-phase power stage designed using IGBT or MOSFET switches. The power stage switches are controlled by the MCU's on-chip timer module, which creates the desired control patterns. The goal of the model is to find out the dependency between the motor characteristics and switching angle. The switching angle is the angular difference between a real switching event and the ideal one. The motor control model consists of a three-phase power stage and a BLDC motor. The power for the system is provided by a DC bus voltage source U_{DCB} . Six semiconductor switches ($S_{A/B/C/T/B}$) deliver the rectangular voltage waveforms to the motor (see [Figure 2](#)). The semiconductor switches and diodes are simulated as ideal devices. The natural voltage level of the whole model is referenced to half of the DC bus voltage, which simplifies the mathematical expressions.

3.2.1 BEMF zero-crossing detection

[Figure 2](#) displays the motor phase winding voltage waveforms for ideal commutation. As seen, the commutation event is in the middle of two BEMF zero-crossings. The BEMF zero-crossing signal can then be used as a rotor position feedback to estimate the proper commutation time point.

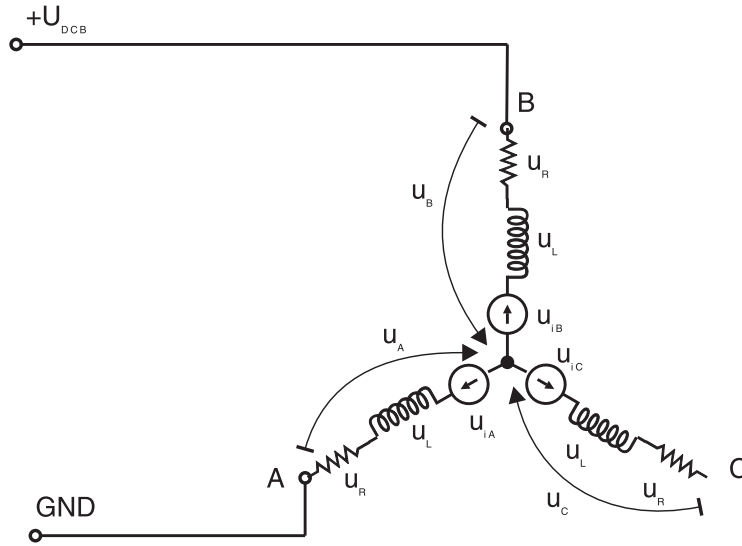


Figure 4. Zero-crossing detection and commutation diagram

The e_{1X} signals in Figure 2 are the BEMF voltages. These BEMF voltages are labeled as U_{iX} in Figure 4. This technique is based on the fact that only two phases of a motor are energized and the third non-conducting phase can be used to sense the BEMF voltage.

The following conditions are met:

$$S_{Ab}, S_{Bt} \leftarrow PWM\text{switching}$$

$$u_N = u_{DCB} - u_R - L \frac{di}{dt} - u_{iB}$$

$$u_N = u_R + L \frac{di}{dt} - u_{iA}$$

$$u_N = \frac{u_{DCB}}{2} - \frac{u_{iB} + u_{iA}}{2}$$

Eqn. 1

where U_N is the neutral point voltage of the star winding.

The voltage u_C can be calculated:

$$u_{iA} + u_{iB} + u_{iC} = 0$$

$$u_N = \frac{u_{DCB}}{2} + \frac{u_{iC}}{2}$$

$$u_C = u_N + u_{iC}$$

$$u_C = \frac{3}{2}u_{iC} + \frac{u_{DCB}}{2}$$

Eqn. 2

The voltage u_{iC} is null at zero-crossing, so the resultant form is:

$$u_C = \frac{u_{DCB}}{2}$$

Eqn. 3

3.2.1.1 BEMF measurement

Voltage dividers for BEMF voltage sensing shown in Figure 5 are provided on the 3-phase BLDC/PMSM Low-voltage Power Stage (see [5] for divider configuration).

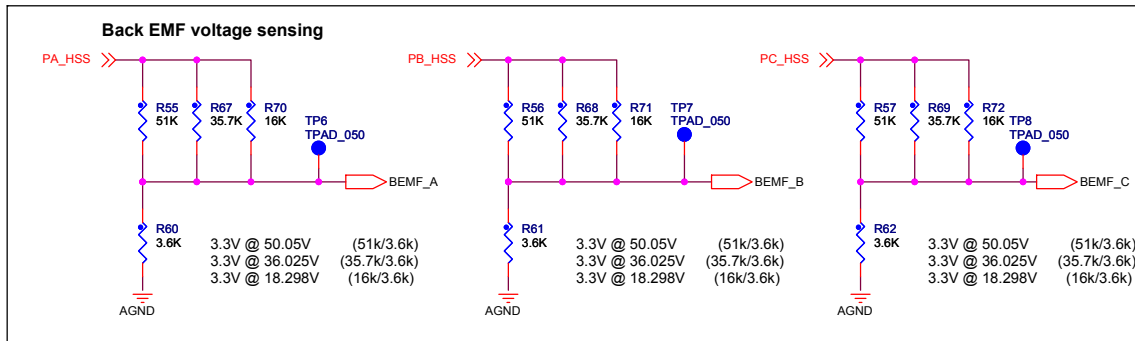


Figure 5. Back-EMF sensing circuit - dividers

Low-pass filters are provided for proper sampling of the BEMF voltage by the ADC module. For more information see the MPC560xB Controller Board User Guide [4].

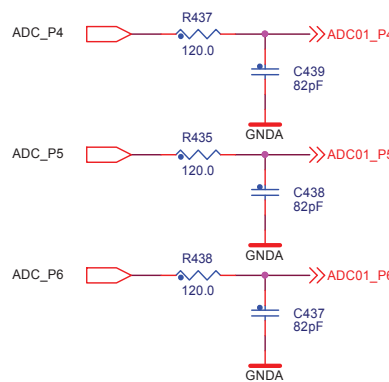


Figure 6. Back-EMF sensing circuit — low-pass filters

3.3 BLDC motor control states

To start and run the BLDC motor, the algorithm has to go through the following states:

- Alignment (initial position set-up)
- Start-up (forced commutations)
- Run (sensorless control with BEMF acquisition and zero-cross detection)

3.3.1 Alignment

As mentioned earlier, the main task for sensorless control of a BLDC motor is the rotor position estimation. Before starting the motor, the rotor position is not known. The main aim of the alignment task is to align the rotor to a known position. This known position is necessary to start rotation in the proper direction and to generate maximal torque during start-up. During alignment, all three phases are powered. Phase A is connected to the positive DC bus voltage, and phases B and C are connected to the negative DC bus

voltage. The alignment time depends on the mechanical constant of the motor, including load, and also on the applied motor current. In this state, the motor current (torque) is controlled by the PI controller every 50 μ s.

3.3.2 Start-up

In the start-up state, the motor commutation is controlled in an open-loop without any rotor position feedback. The commutation period is controlled with a linear open-loop starting ramp. The open-loop start is a short state at a very low speed where the BEMF is too small, so the zero-crossing event cannot be reliably detected.

3.3.3 Run

The running sensorless mode includes the BEMF acquisition with zero-crossing detection for the commutation control. The motor speed is controlled using zero-crossing period feedback to the speed PI regulator. The motor current is measured and filtered during a commutation event and used as feedback into the current controller. Its output limits the speed controller output to achieve the maximal motor current in the required range.

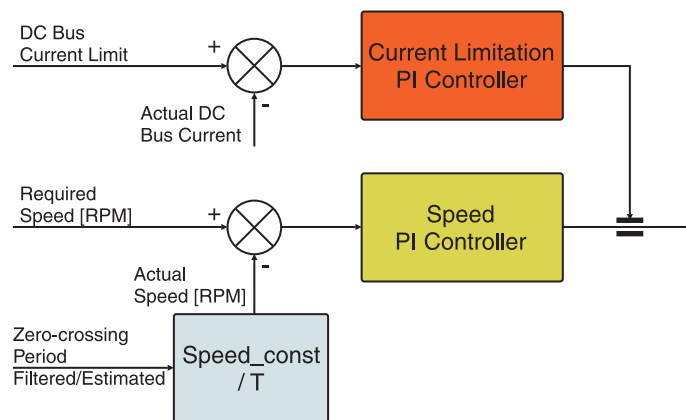


Figure 7. Speed control with torque limitation

4 MPC560xB Controller Board Configuration

The BLDC sensorless application framework meets the following technical specification:

- PWM output frequency is equal to 20 kHz, PWM signals driving the three-phase power stage MOSFET switches are provided by eMIOS_0 as follows:
 - Phase A high-side: eMIOS0_1
 - Phase A low-side: eMIOS0_2
 - Phase B high-side: eMIOS0_3
 - Phase B low-side: eMIOS0_4
 - Phase C high-side: eMIOS0_5
 - Phase C low-side: eMIOS0_6
- Current loop sampling period is equal to 50 μ s
- Speed loop sampling period is equal to 2.5 ms
- Three-phase BEMF voltage measurement uses three voltage dividers, one per inverter leg. Phase voltage is routed to ADC_1 (12-bit) as follows:
 - Phase A BEMF: ADC1_P[4]
 - Phase B BEMF: ADC1_P[5]
 - Phase C BEMF: ADC1_P[6]
- DC bus voltage routed to ADC_0 (10-bit) as follows:
 - DC bus voltage: ADC0_P[7]
- DC bus current (DC bus shunt resistor voltage) routed to ADC_0 (10-bit) as follows:
 - DC bus current: ADC0_P[8]

The BLDC sensorless motor control application uses the ADC, CTU, eMIOS, and PIT modules included in the MPC5606B. These modules are internally interconnected. The CTU provides ADC conversion triggered by the event generated by the eMIOS and PIT modules (see [Figure 8](#)). These modules are described below and a detailed description can be found in the reference manual titled *MPC5607B Microcontroller Reference Manual* (document number MPC5607BRM) [3].

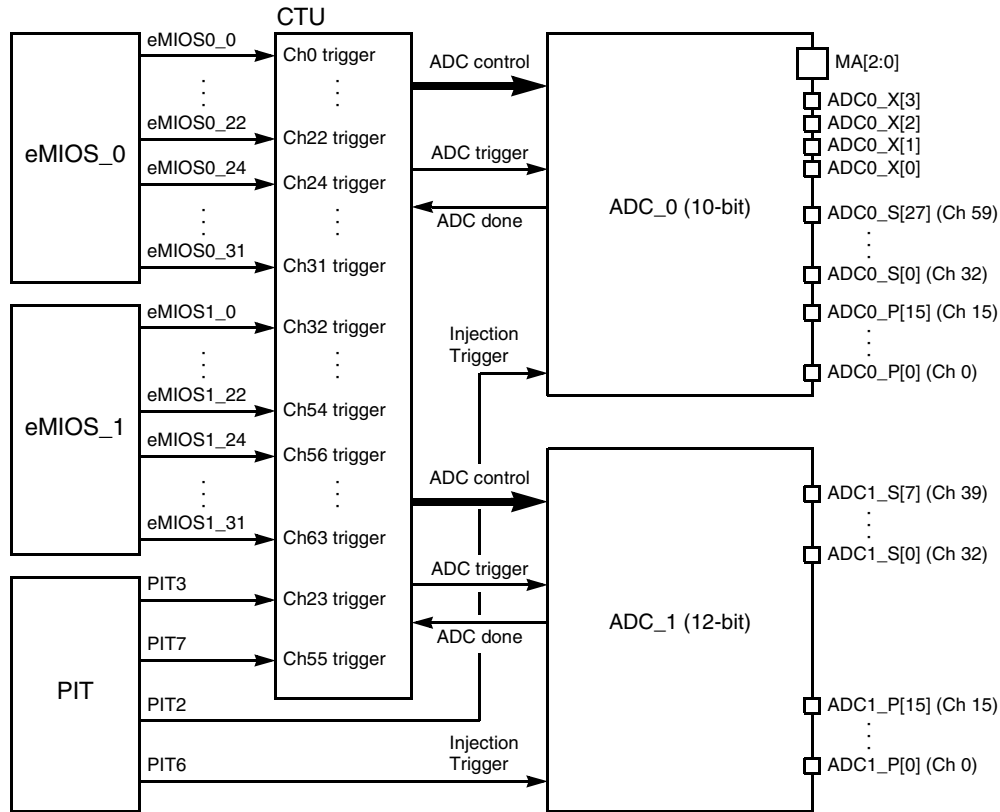


Figure 8. Timers/ADC module interconnection on the MPC5607B family of microcontrollers

4.1 Enhanced modular input/output subsystem (eMIOS)

The MPC5606B includes two eMIOS modules (eMIOS_0 & eMIOS_1). The input clock of both eMIOS modules is supplied from the peripherals clock set 3 (64 MHz system clock output divided by 1).

eMIOS_0 is used mainly for the generation of the PWM signals and ADC trigger signal. The application uses eMIOS_0 unified channels 0 to 7. Channel 0 serves as a time base (eMIOS_0 counter bus B) for channels 1 to 7. Channel 0 operates in the Modulus Counter mode (up counter with clear on match start). The match value written in the internal A1 register (accessible via the CADR[0] peripheral register) is equal to the required 50 μ s period of the PWM:

$$A1 = \text{PERIOD} = (f_{\text{eMIOS}_0} \times t_{\text{PWM}}) = (64 \text{ MHz} \times 50 \mu\text{s}) = 3200 = 0x0C80_{\text{hex}}$$

Channels 1 to 6 operate in the Output Pulse Width Modulation with Trigger mode (OPWMT). The odd-numbered channels are configured to generate high-side switch PWM signals, while the even-numbered channels are dedicated to low-side switch PWM signals. The duty cycle is given by setting the value of internal registers A1 and B2. The values of A1 and B2 (accessible via the CADR and CBDR peripheral registers) are set as follows (example for a 20% duty cycle):

$$A1 = (\text{PERIOD} \times (1 - \text{DUTY})) \div 2 = (3200 \times (1 - 0.2)) \div 2 = 1280 = 0x0500_{\text{hex}}$$

$$B2 = (\text{PERIOD} \times (1 + \text{DUTY})) \div 2 = (3200 \times (1 + 0.2)) \div 2 = 1920 = 0x0780_{\text{hex}}$$

A low-side switch PWM signal includes the dead-time, so the A1 and B2 values for the low-side switch PWM signals are set as follows (example for 500 ns dead-time):

$$DT = f_{eMIOS_0} \times t_{DT} = 64 \text{ MHz} \times 500 \text{ ns} = 32$$

$$A1_{LS} = A1 - DT = 1280 - 32 = 1248 = 0x04E0_{hex}$$

$$B2_{LS} = B2 + DT = 1920 + 32 = 1952 = 0x07A0_{hex}$$

eMIOS_0 channel 1 is also configured to generate a CTU trigger to start ADC_1 conversion of the selected BEMF voltage. The time of the trigger is defined by the value of the internal register A2 (accessible via the ALTCADR[1] peripheral register) and is set as follows (example for a 1.1 μs delay from the center of the PWM period):

$$DELAY = f_{eMIOS_0} \times t_{DELAY} = 64 \text{ MHz} \times 1.1 \mu\text{s} \cong 70$$

$$A2 = (PERIOD \div 2) + DELAY = (3200 \div 2) + 70 = 1670 = 0x0686_{hex}$$

Figure 9 displays the timing of two consecutive periods of complementary PWM signals with a 10% duty cycle in the first period with an update to a 50% duty cycle in the second period.

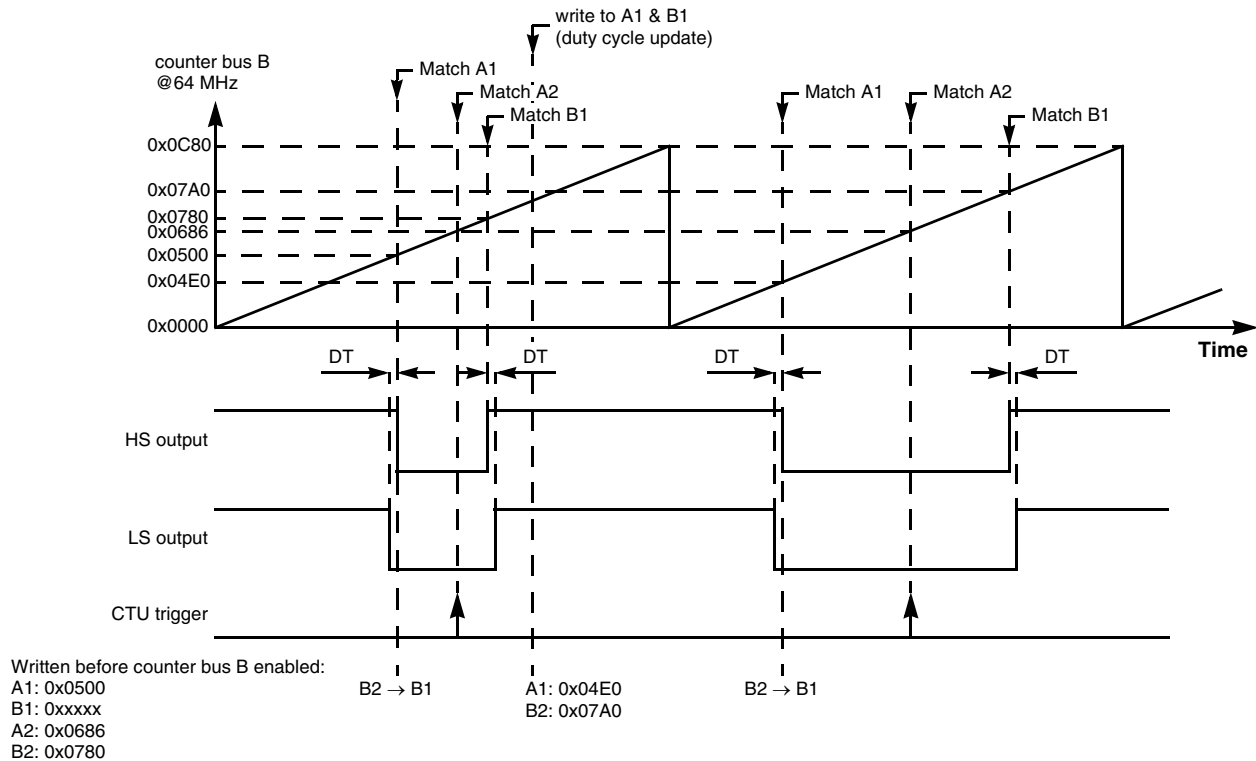


Figure 9. Complementary PWM generation timing

The duty cycle has to be updated in the current PWM period. A1 is not buffered and is updated immediately. B1 is buffered and its value is loaded from the buffer (B2) when an A1 match occurs.

The eMIOS_1 module channel 24 generates an interrupt request to change the PWM output with regards to the new motor commutation event. The time base for the counter is derived from peripheral clock set 3 (64 MHz system clock output divided by 1). The global eMIOS_1 prescaler divides the clock by 16 and the unified channel prescaler divides it further by 4. The time base for the commutation events is then:

$$f_{\text{COMM}} = ((64 \text{ MHz} \div 16) \div 4) = 1 \text{ MHz}$$

The channel 24 operates in the Single Action Output Compare mode. The value of the internal A2 register (accessible via the CADR[24] peripheral register) defines the time of the next commutation event and is immediately transferred to internal register A1. The FLAG bit is set when an A1 match occurs and generates an interrupt request to the e200z0 core. The interrupt service routine then performs the commutation function.

The channel 24 counter also serves as a free-running time base (counter bus E) for channel 25. The channel 25 operates in the Single Action Input Capture mode, in which the channel captures the counter bus E value into internal register A2 (accessible for reading via the CADR[25] peripheral register) when a rising or falling edge is detected on its dedicated input pin.

The eMIOS_1 channel 25 input signal is generated by eMIOS_0 channel 7 output (interconnected on the MPC560xB Controller Board) which is a square signal with consecutive rising and falling edges aligned with the eMIOS_0 channel 1 generated CTU trigger (operates in the Single Action output Compare mode with CADR[7] set to $0x0686_{\text{hex}}$, see [Figure 10](#)). At the time of the commutation, the CADR[25] holds the time of the last CTU trigger which is then used for the calculation of the next commutation event time.

4.2 Periodic interrupt timer (PIT)

The PIT module is used to generate a CTU trigger to start the ADC_0 injected chain conversion of DC bus voltage and DC bus current.

The PIT module input clock is supplied by the undivided 64 MHz system clock (PLL output). PIT channel 2 generates a CTU trigger pulse when its counter reaches 0 from the pre-loaded value stored in the LDVAL[2] peripheral register (down-counter). When the counter reaches 0, it re-loads the counter with the value from the LDVAL[2] peripheral register. The LDVAL is set in a way to be aligned with the eMIOS_0 channel 1 generated CTU trigger signal (see [Figure 10](#)).

The initial LDVAL is set as follows:

$$\text{LDVAL}_{\text{init}} = (\text{PERIOD} \div 2) + \text{DELAY} = (3200 \div 2) + 70 = 1670 = 0x0686_{\text{hex}}$$

Directly after the PIT 2 timer is enabled, the LDVAL is updated to a new value equal to the PWM period:

$$\text{LDVAL} = \text{PERIOD} = 3200 = 0x0C80_{\text{hex}}$$

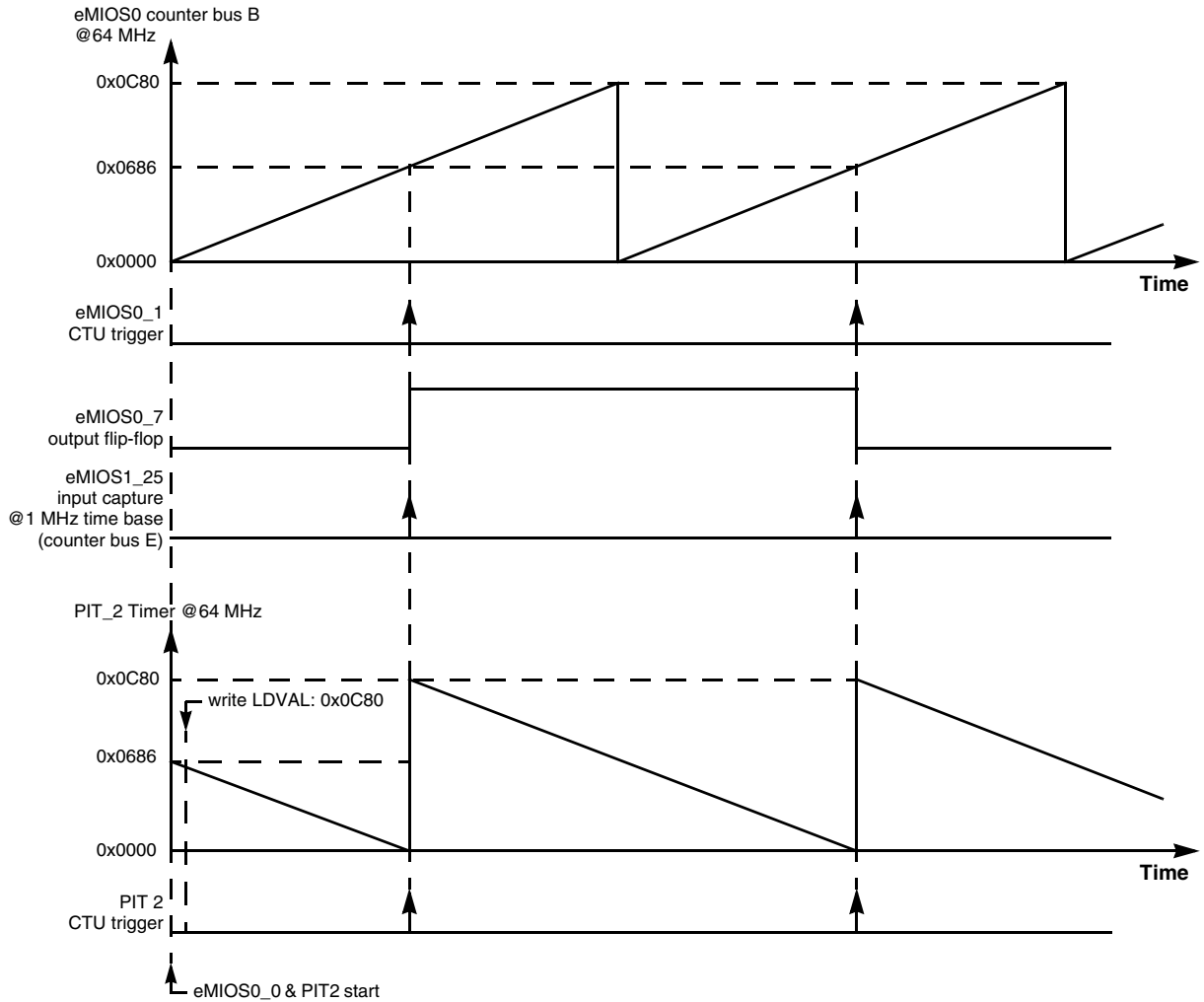


Figure 10. eMIOS and PIT CTU triggering alignment

4.3 Cross triggering unit (CTU)

The Cross Triggering Unit (CTU) on the MPC5606B is responsible for triggering ADC conversion based on an event generated by the eMIOS_0, eMIOS_1, and PIT modules.

The application uses only one CTU trigger source, which is the eMIOS_0 channel 1. A trigger event from the channel triggers selected BEMF voltage conversion by the ADC_1. The BEMF voltage ADC channel is selected by the CTU_EVTCFGR[1] peripheral register according to the current commutation sector.

4.4 Commutation function

The commutation function performs a re-configuration of the PWM signal output according to the commutation sector. This includes a reset of the counter bus B (eMIOS0_0), re-configuration of the eMIOS0_[1..6] according to the current commutation sector, and a PIT_2 reset. The commutation function is performed in the following steps:

1. Disable and reset counter bus B (set eMIOS_0 channel 0 to General Purpose Output mode)
2. Reconfigure eMIOS0_[1..6] mode of operation according to the current sector
 - Set complementary PWM pair channels to generate a PWM (force match B to get the output flip-flop to the PWM cycle initial state)
 - Set the appropriate PWM channels to the OFF state (Output Disable via the ODIS bit for high-side, General Purpose Output mode for low-side)
 - Set the appropriate PWM channel to the ON state (Output Disable via the ODIS bit for low-side)
3. Disable the PIT_2 timer
4. Set the PIT_2 LDVAL_{init} value
5. Re-start the eMIOS_0 counter bus B (set eMIOS0_0 back to Modulus Counter mode) and PIT_2 timer
 - This is performed by a special inline assembly function which starts the eMIOS0_0 and PIT_2 counters with a minimal delay between control registers byte writes
6. Set the PIT channel 2 LDVAL value
7. Reconfigure the CTU to trigger the conversion of a proper BEMF ADC channel relevant to the current commutation sector

5.2 Application flow

The application is interrupt driven running in real time. The main tasks of the motor control application are periodically running in one interrupt service routine, driven by the ADC_0 End of Injected Chain Conversion interrupt every 50 μ s. This includes both the fast current and slower speed control loops. The commutation of the motor stator flux is provided in the second interrupt service routine driven by the eMIOS_1 channel 24 output compare interrupt event. All tasks apart from the commutation function are executed in order, as described in the application state machine shown in [Figure 14](#), and the application flow charts [Figure 12](#) and [Figure 13](#).

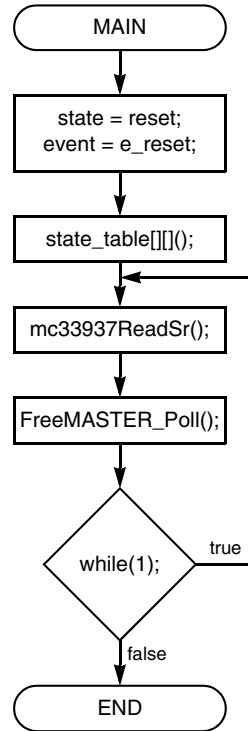


Figure 12. Main task flow chart

This type of application requires precise and deterministic sampling of analogue quantities and execution of all motor control functions. The state machine routines are called within a periodic interrupt routine. In reference to the state machine, the interrupts have to be configured and enabled at the end of the RESET state, where all peripheral initialization has to be executed. Consequently, the RESET state function is called before the main loop, as shown in [Figure 12](#). The background loop manages non-critical tasks, such as the FreeMASTER communication polling and the MC33937A FET pre-driver [6] status read-back.

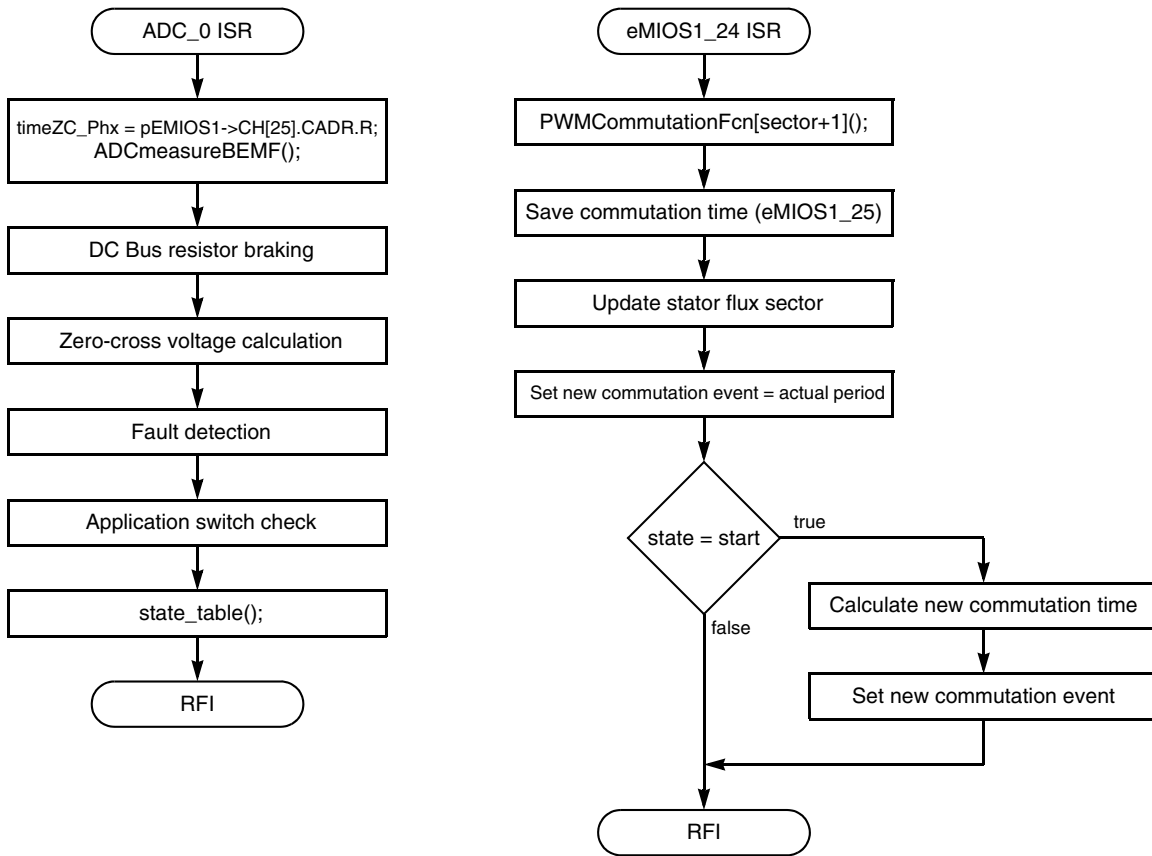


Figure 13. Application interrupt service routine flow charts

5.3 Speed evaluation and control

The application uses the eMIOS_1 channel 24 to achieve a precise commutation of the BLDC motor as described below.

When the zero-cross event is recognized, the eMIOS_1.CH[24].CADR peripheral register is filled with the new calculated value of the next commutation time. When the counter matches the CADR register value, the FLAG signal generates an interrupt request to the e200z0 core and the commutation function runs in the eMIOS1_24 interrupt service routine.

5.3.1 Speed evaluation

The speed is calculated in the Slow Control Loop which is part of the *BLDC_Fast_ISR* (ADC_0 ISR) routine. The zero-cross detection algorithm provides the actual commutation period duration for each commutation event. These variables refer to the eMIOS_1 channel 25 captured values. The eMIOS_1 channel 25 clock is set up to 1 MHz. To calculate the real time commutation period, you can write:

$$T_{REAL} = T \times T_{CLK} \tag{Eqn. 4}$$

$$T_{CLK} = \frac{1}{f_{CLK}} \quad \text{Eqn. 5}$$

$$T_{REAL} = \frac{T}{f_{CLK}} \quad \text{Eqn. 6}$$

where:

- T_{REAL} is the real commutation period
- T_{CLK} is the period of the eMIOS_1, channel 25, clock (eMIOS counter bus E)
- T is the value measured in the eMIOS_1 channel 25 ticks
- f_{CLK} is the eMIOS_1 channel 25 clock frequency

If you know the commutation period, you can calculate the period of one electrical revolution:

$$T_{elrev} = T_{REAL} \times N = \frac{T \times N}{f_{CLK}} \quad \text{Eqn. 7}$$

where:

- T_{elrev} is the real period of one electrical revolution
- N is number of commutations in one electrical period

To calculate the period of one mechanical revolution, the result of [Equation 7](#) must be multiplied by the number of pole-pairs:

$$T_{mechrev} = T_{elrev} \times p = \frac{T \times N \times p}{f_{CLK}} \quad \text{Eqn. 8}$$

and finally, you can calculate the mechanical speed in revolutions per minute:

$$\omega_{mech} = \frac{60}{T_{mechrev}} = \frac{60 \times f_{CLK}}{T \times N \times p} \quad \text{Eqn. 9}$$

If the clock rate is 1 MHz, the number of commutations per electrical revolution is 6, and the number of pole-pairs is 4, you can get the constant:

$$c = \frac{60 \times f_{CLK}}{N \times p} \quad \text{Eqn. 10}$$

Therefore, the speed is calculated as:

$$\omega_{mech} = \frac{c}{T} \quad \text{Eqn. 11}$$

where c is the mechanical speed constant, that is 2.5×10^6 .

To achieve a better resolution, the mechanical speed is multiplied by 1000.

5.3.2 Speed controller

The motor speed PI controller is called in the speed control loop, which is slower than the current control loop. The K_p and K_i constants are calculated from either the motor or the entire mechanical system parameters. The speed loop bandwidth was chosen as 20 kHz, and attenuation as 1.

5.4 Zero-cross detection

The zero-cross algorithm is executed in each *BLDC_Fast_ISR (ADC_0_ISR)* routine. The CTU and PIT2 trigger analog to digital conversion of stator flux related analogue quantities, such as the DC bus and related BEMF voltage, and the DC bus current. The time of the measurement is captured by the eMIOS_1 channel 25. [Figure 2](#) displays the BEMF voltage behavior for each commutation sector. The relevant zero-cross detection function is called with respect to the actual stator flux sector.

The zero-cross event occurs when the phase BEMF voltage crosses $U_{DCBUS} \div 2$, and this is basically in the middle of the actual commutation period. When this occurs, the next commutation event is calculated from actual zero-cross time and actual zero-cross period. The result is loaded into the eMIOS1.CH[24] CADR compare register to achieve a precise commutation of the stator flux. The algorithm also stores the motor current and the actual zero-cross period. These values are then used for speed and motor current calculation.

5.5 Current limitation controller

The motor current limitation controller is called in fast control loop. The parameters of the armature current PI controller are calculated assuming the armature current loop bandwidth and attenuation, and the motor physical constants.

5.6 State machine

The application state machine is implemented using a two-dimensional array of pointers to state functions, called `state_table[][]`, with the first parameter describing the current application event and the second parameter describing the actual application state. These two parameters select a particular pointer to a state machine function, which causes a function call whenever `state_table[][]()` is called.

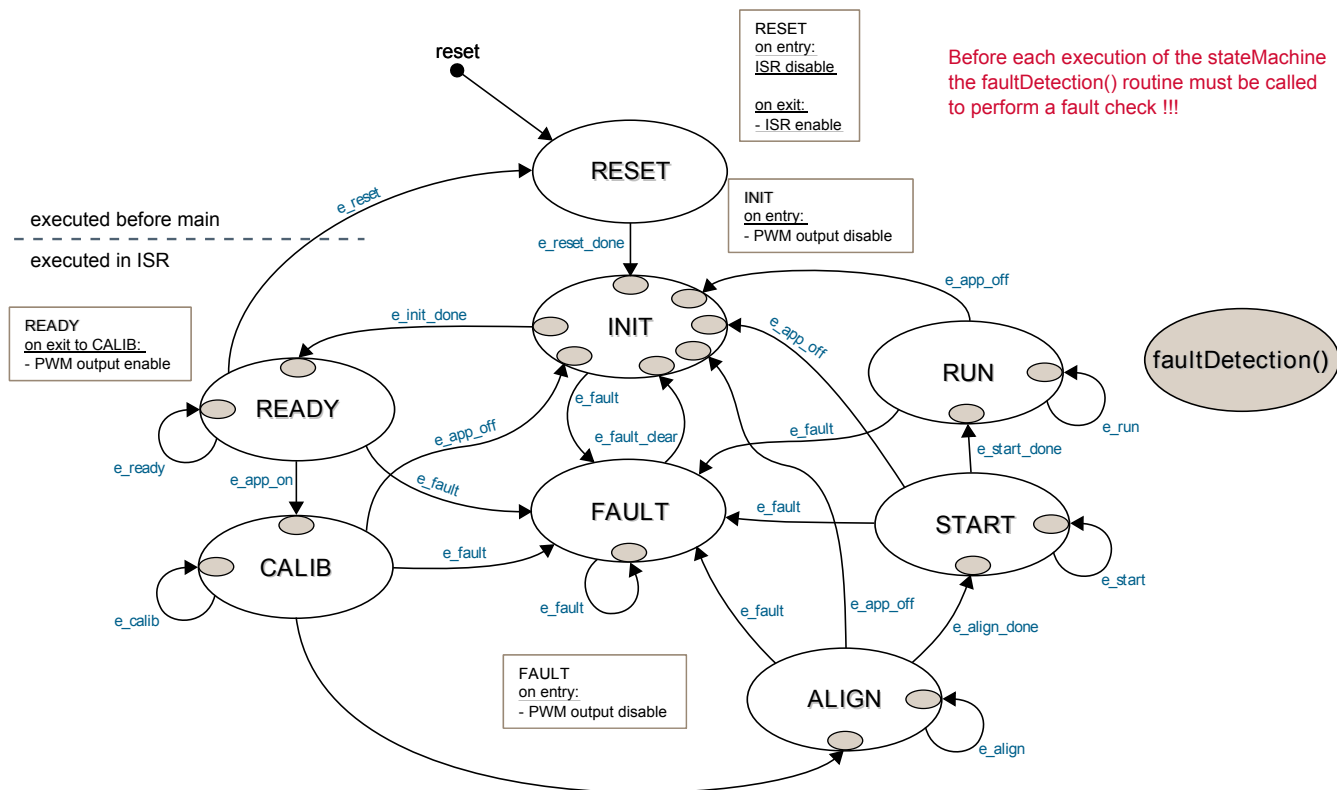


Figure 14. Application state machine

The application state machine consist of the following eight states selected using the variable *state* defined as *AppStates* enumeration type:

- RESET state = 0
- INIT state = 1
- FAULT state = 2
- READY state = 3
- CALIB state = 4
- ALIGN state = 5
- START state = 6
- RUN state = 7

To signalize or initiate a change of the state, fifteen application events are defined and selected using the variable *event* defined as *AppEvents* enumeration type:

- e_reset - event = 0
- e_reset_done - event = 1
- e_fault - event = 2
- e_fault_clear - event = 3
- e_init_done - event = 4
- e_ready - event = 5

- e_app_on - event = 6
- e_calib - event = 7
- e_calib_done - event = 8
- e_align - event = 9
- e_align_done - event = 10
- e_run - event = 11
- e_app_off - event = 12
- e_start_done - event = 14

5.6.1 RESET state

State RESET is the first state which is executed after the MCU exits the power-on reset state and enters the main() function. It is executed only once at the start of the main function, to perform system variables and all peripheral initialization. Before initializing all peripherals, all interrupts are disabled and enabled again at the RESET state end, with respect to the interrupt driven application as described before. This routine also includes initialization and set-up of the MC33905 system basis chip, which provides the power supply for the MPC560xB Controller Board, and the MC33937A FET pre-driver [6]. Both routines use the SPI peripheral and must be called after the DSPI and SIUL initialization routines.

5.6.2 INIT state

State INIT is similar to the RESET one-pass routine, which allows the user to set up application variables. At the end of the INIT state, the transition event is set to pass to the READY state if there is no fault pending. INIT is executed directly after the RESET state, or after the RUN state when the application is stopped.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the READY state is performed automatically at the end of the INIT routine.

5.6.3 FAULT state

The application goes to this state immediately when a fault is detected. The system allows all states to pass into the FAULT state by setting *event=e_fault*.

NOTE

The MPC5606B hardware does not implement any automatic mechanism to force eMIOS PWM outputs to a default safe state based on an external input fault signal. Additional external hardware logic is required to prevent damage to the 3-phase power stage in the case of a fault.

This is implemented on the MPC560xB Controller Board [4].

5.6.4 READY state

This READY state is used as the application initial state. The application checks only fault inputs and the application switch status to start the application.

Transition to the RESET state is performed by setting *event=e_reset*, which is executed automatically when you set the *switchAppReset* variable to true using FreeMASTER.

Transition to the INIT state is performed by setting *event=e_app_on*, which is executed automatically by setting *switchAppOnOff=true* using FreeMASTER. Its value can also be changed by switching on the external switch on the MPC560xB Controller Board.

5.6.5 CALIB state

The CALIB state provides calibration of the analogue quantities used by the sensorless motor control algorithm. The analog offsets are calibrated for all six voltage vectors applied to the three-phase bridge. When the calibration is finished for all six sectors (voltage vectors), the *alignmentTimer*, *svmsector*, and *torqueRequired* variables are initialized and the PWM_Alignment() function is called to set the PWM output. The variable event is set automatically to *event=e_calib_done*, which enables the transition to the ALIGN state.

5.6.6 ALIGN state

The ALIGN state provides the motor rotor alignment process described in [Section 3.3.1, “Alignment”](#). You can set up the ALIGNMENT_TIME macro value and the proper motor current, depending on the minimal mechanical system behavioral time (mechanical system inertia, motor time constants, and so on) to ensure the correct motor rotor position. The alignment current is controlled via the PI regulator, updated every PWM cycle. The required alignment current can be adjusted by the *torqueRequired* variable. When the counter *alignmentTimer* reaches zero, *switchAlignDone* is set to true, variables used for the next state are initialized, and the variable *event* is set automatically to *event=e_align_done*. This enables transition to the START state.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting *event=e_app_off*, which is executed automatically by setting *switchAppOnOff=false* using FreeMASTER or by switching off the external switch on the MPC560xB Controller Board.

5.6.7 START state

The START state provides the start rotor rotation sequence described in [Section 3.3.2, “Start-up”](#). The motor current PI controller function *Ureq=GFLIB_ControllePIpAW(torque_err, &i_controllerParams1)* is called every PWM cycle. Its parameters (proportional gain, integral gain, lower, and upper limits) can be set in the *i_controllerParams1* structure variable. The PI controller function is part of the Auto Math and Motor Control Library Set for MPC560xP and its detailed description is shown in the Automotive Math and Motor Control Library Set for Qorivva MPC560xP User Guide, MPC560xPMCLFIXUG [7].

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting *event=e_app_off*, which is executed automatically by setting *switchAppOn_Off=false* using FreeMASTER or by switching off the external switch on the MPC560xB Controller Board.

5.6.8 RUN state

The RUN state provides the motor speed and current regulation sequence as has been described in [Section 3.3.3, “Run”](#). The zero-cross detection function *ZCdetect[svmSector]()* is called every PWM cycle to manage the correct motor commutation process. In the slow control loop performed every 1 ms, the speed and current control loops are performed.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting *event=e_app_off*, which is executed automatically by setting *switchAppOn_Off=false* using FreeMASTER or by switching off the external switch on the MPC560xB Controller Board.

5.7 Library functions

The application source code uses the Auto Math and Motor Control Library Set for the MPC560xP with 32-bit fixed-point arithmetic [7], which is fully applicable on the MPC5606B. It has the same core type as the MPC560xP. The library contains three independent library blocks: GFLIB, GDFLIB, and GMCLIB. The General Functions Library (GFLIB) includes basic mathematical functions (such as sine, cosine, lookup table, ramp, and so on). Advance filter functions are part of the General Digital Filters Library (GDFLib), and standard motor control algorithms are part of the General Motor Control Library (GMCLIB).

5.8 Setting the software parameters for a specific motor

The default software parameter settings have been calculated and tuned for a hardware set-up with the LINIX 45ZWN24-90 BLDC motor.

All application parameters dedicated to the motor or application rating (maximum voltage, speed, and so on.) are defined in the *BLDC_appconfig.h* file and commented to help you to modify the parameters according to your own specific requirements.

6 FreeMASTER User Interface

The FreeMASTER Run-Time Debugging Tool [8] is used to control the application and monitor variables during run time.

Communication with the host PC is performed via USB. Because the FreeMASTER driver on the MPC5606B supports RS232 communication only, there must be a VCP driver for the FTDI FT232RL USB to serial UART interface integrated chip installed on the host PC. The driver creates a virtual USB COM port and can be downloaded from www.ftdichip.com.

The application configures the LINFlex module of the MPC5606B for a communication speed of 19200 bps. Therefore, FreeMASTER also has to be set to this communication speed.

6.1 Application start

- Install the USB driver to create a virtual COM port for emulation of RS232 communication on USB (Virtual COM Port Drivers at <http://www.ftdichip.com/Drivers/VCP.htm>).
- Install FreeMASTER (FreeMASTER Application Installation for PC - FMASTERSW at www.freescale.com).
- Connect the USB cable to the MPC560xB Controller Board and the host PC.
- Connect the power supply to the power stage. The MPC560xB Controller Board is supplied from the power stage. The BLDC motor used is designed for a 24 V phase voltage.
- Start the FreeMASTER project MPC5606B_BLDC.pmp located in the application software root directory.
- Set the communication port and speed (19200 Bd) in the FreeMASTER menu Project/Option (or by pressing Ctrl + T).
 - Virtual communication port appears in the Port (COM & LPT) section of the Windows Device Manager as USB Serial Port (COM n), where n is the number of the COM port.
- Enable communication by pressing the STOP button in the FreeMASTER toolbar, or by pressing Ctrl + K.
 - Successful communication is signalized in the status bar.
- If no actual faults are present in the system, all fault indicators shall be dark red. If there is a fault present, identify the source of the fault source and remove it. Successful recovery is signalized by the switching off of the fault indicator with the name of the fault. A small red LED-like indicator next to the fault indicator indicates that the fault remains latched in the system. To remove it, click on the green Fault Clear button or press SW501 and SW502 together on the MPC560xB Controller Board.
- Click the ON/OFF button or switch over the button SW503 on the MPC560xB Controller Board to run the application. The BLDC motor will start running. You are able to change the mechanical speed by writing to the *wRotElRequired* variable, by double-clicking on the speed gauge, or by pressing SW502 (speed up) or SW501 (speed down) buttons on the MPC560xB Controller Board.

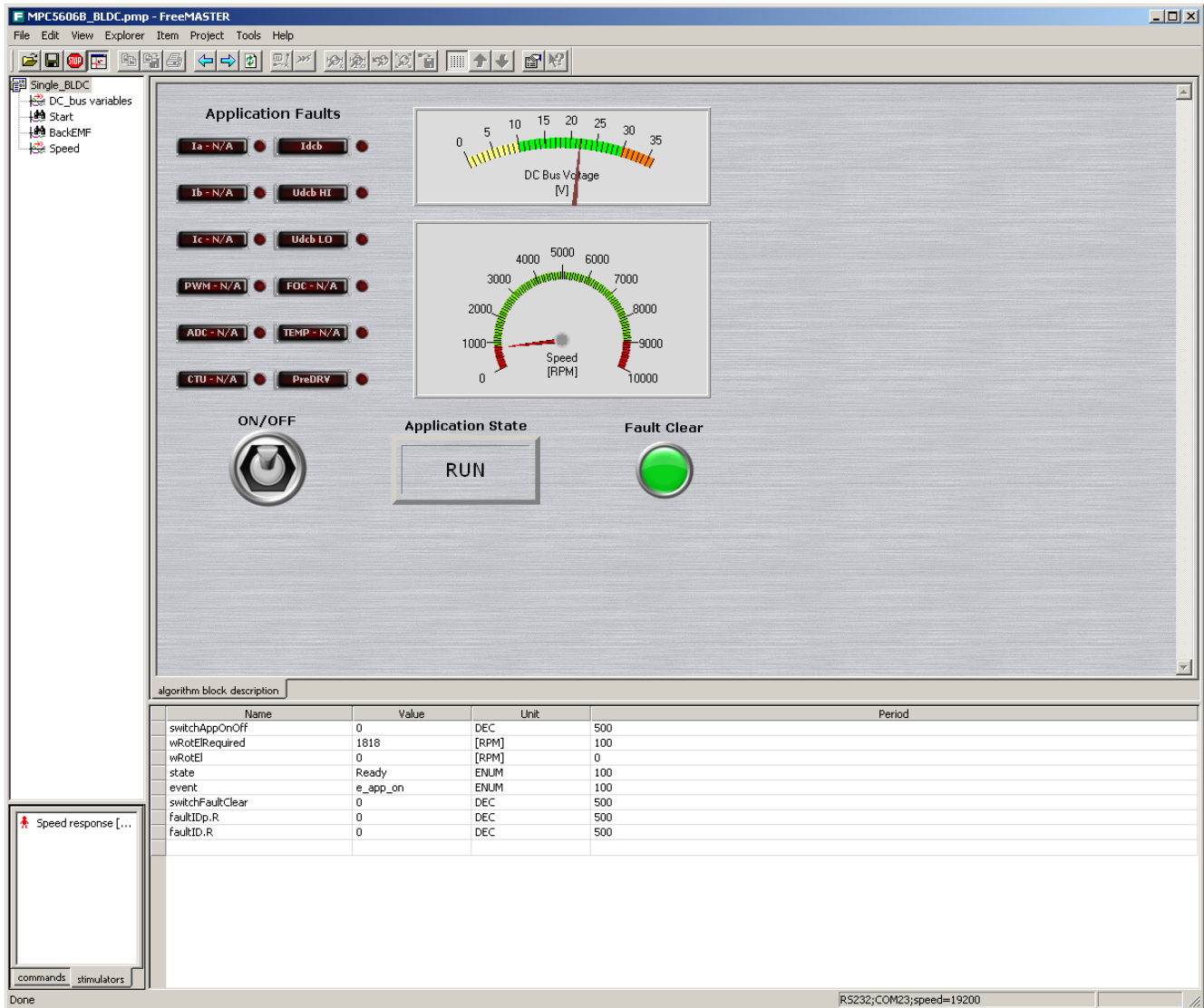


Figure 15. FreeMASTER control page screenshot

7 Performance

This chapter contains BLDC sensorless application performance analysis results done on the MPC5606B.

Table 2. Fast loop ISR and commutation function maximum time duration¹

Function	Duration
BLDC_Fast_ISR (every 50 μ s)	16.7 μ s
PWMCommutationFcn[>()	1.44 μ s

¹ Measured values doesn't include interrupt latency and interrupt prolog and epilog execution times.

Please note that in comparison with motor control dedicated devices like the MPC560xP, MPC564xL, and MPC567xK, it is necessary to force all six PWM eMIOS channels to a defined output state (based on the commutation sector) by the software on the MPC5606B. All six PWM outputs are updated sequentially with up to a five CPU cycle delay (dependent on the code optimization) between each eMIOS channel update. See [Section 4.4, "Commutation function," on page 14](#) for more details.

MPC560xP, MPC564xL, and MPC567xK include internal HW that can force PWM channels to predefine states simultaneously by an active edge of the internal signal (for example, timer output compare event) without SW intervention.

Application code was compiled using a Green Hills compiler with optimization options listed in [Table 3](#). The timing was measured on the e200z0 core at a 64 MHz system clock frequency using optimal flash read/write wait state control and address pipelining control settings.

Table 3. Compiler options

Compiler Option	Description
-Ospeed	Optimize for speed.
-Opeep	Peephole optimization.
-Opipeline	Pipeline optimization (pipeline instruction scheduling).
-isel	Automatic generation of isel integer conditional move instruction.
-sda	Small data area optimization with a threshold of 8 bytes.
-auto_sda	Automatic link-time allocation of data to small data area.

8 Conclusion

The described design demonstrates a way of using the MPC5606B microcontroller for BLDC sensorless motor control. The design is fully compatible with any member of the MPC5607B family of microcontrollers. The MPC5607B family of microcontrollers is thus a suitable choice for BLDC motor control in the automotive body area.

9 References

1. 3-phase BLDC Sensorless Motor Control Development Kit with Qorivva MPC5606B MCU at www.freescale.com/AutoMCDevKits
2. Application note — *3-phase Sensorless BLDC Motor Control Development Kit with the Qorivva MPC5604P MCU* (document number AN4268) at www.freescale.com/AutoMCDevKits
3. Reference Manual — *MPC5607B Microcontroller* (document number MPC5607BRM) at www.freescale.com
4. User Guide — *MPC560xB Controller Board* (document number MPC560XBMCBUG) at www.freescale.com/AutoMCDevKits
5. User Guide — *3-phase Low-voltage Power Stage* (document number 3PHLVPSUG) at www.freescale.com/AutoMCDevKits
6. Data Sheet — *MC33937 Three Phase Field Effect Transistor Pre-driver Data Sheet*, at www.freescale.com
7. Automotive Math and Motor Control Library Set for Qorivva MPC560xP with 32-bit fixed-point arithmetic, www.freescale.com/AutoMCLib
8. FreeMASTER Run-Time Debugging Tool, www.freescale.com/FREEMASTER

10 Acronyms

ADC	Analog to Digital Converter
BEMF	Back Electromotive Force
BLDC	Brushless DC Motor
CTU	Cross Triggering Unit
DC	Direct Current
DT	Dead Time
DSPI	Deserial Serial Peripheral Interface
eMIOS	Enhanced Modular Input/Output Subsystem
IGBT	Insulated Gate Bipolar Transistor
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MCU	Microcontroller Unit
FET	Field Effect Transistor
PC	Personal Computer
PI	Proportional-Integral
PIT	Periodic Interrupt Timer
PLL	Phase-Locked Loop
PWM	Pulse Width Modulation
RFI	Return From Interrupt
SIUL	System Integration Unit Lite
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VCP	Virtual COM Port

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:**Home Page:**

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale, the Freescale logo and Qorivva are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© Freescale Semiconductor, Inc. 2012. All rights reserved.