Freescale Semiconductor
Application Note

Document Number: AN4548
Rev. 1, 02/2016

Dual 3-Phase Sensorless BLDC Kit with MPC5643L MCU

by: Petr Konvicny

Contents

1 Introduction

This application note describes design of a dual 3-phase BLDC motor control drive using a sensorless algorithm. The design is targeted at automotive applications. This cost-effective solution is based on NXP MPC5643L device dedicated to automotive motor control and safety systems.

The system is designed to drive two independent 3-phase BLDC motors without a positional feedback sensor. Features of this design include:

- Sensorless 6-step commutation control of a BLDC motor that can be configured for single-motor or dual-motor application
- BEMF sensing and zero-crossing detection
- Application control user interface using the FreeMASTER debugging tool
- Maximal motor current limitation

2 System concept

The system is designed to drive two independent 3-phase BLDC motors using the NXP MTRCKTDBN5643L motor-control development kit. The application meets the following performance specifications:

- Targeted at the MPC5643L Controller (refer to the dedicated user manual for the MPC5643L, available at www.nxp.com)

- Running on the MPC5643L Control drive board (refer to the dedicated user manual for the MPC5643L Controller Board)
- Control technique incorporating:
  - Sensorless control of a 3-phase brushless DC motor
  - Zero-crossing technique
  - Closed-loop speed control
  - Closed-loop current control
  - Open-loop start-up sequence with alignment
  - BackEMF voltage sensing
  - 50 µs sampling period with the FreeMASTER recorder
- Floating-point implementation
- FreeMASTER software control interface (motor start/stop, speed set-up)
- FreeMASTER software monitor
- DC-Bus over-voltage and under-voltage, over-current, overload protection

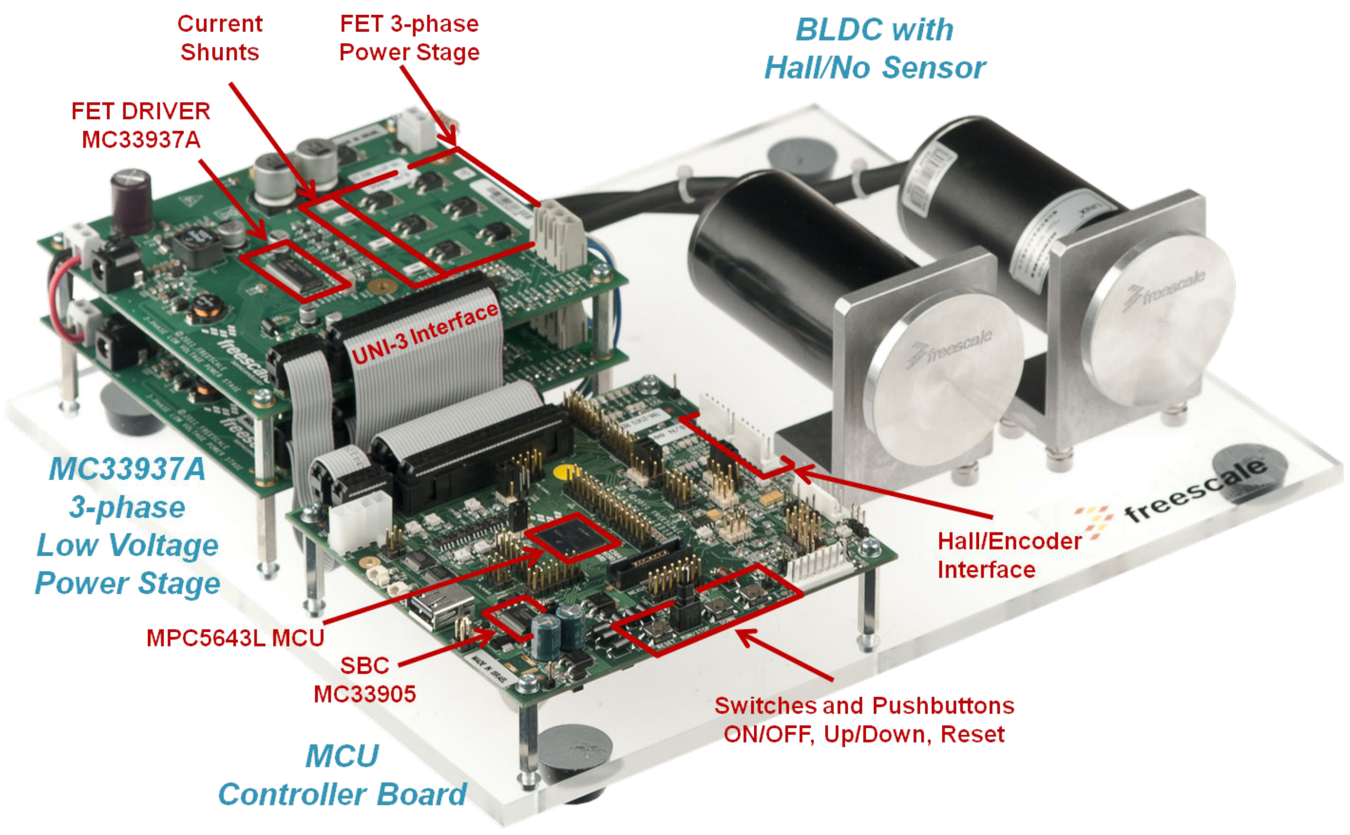


Figure 1. Dual 3-phase sensorless BLDC kit with MPC5643L MCU

3 BLDC sensorless control

3.1 Brushless DC motor

The BLDC motor is a rotating electric machine with a standard 3-phase stator similar to an induction motor. The phases mounted on the stator are connected to form a way or delta connection. The rotor has surface-mounted permanent magnets. The motor can have more than one pole pair per phase. The pole pair per phase defines the ratio between the electrical revolution and the mechanical revolution. The BLDC motor is equivalent to an inverted DC brushed motor, where the magnet rotates while the conductors remain stationary. In the DC brushed motor, the commutator and brushes reverse the current polarity in such way that the stator and rotor magnetic fields are perpendicular. However, in the brushless DC motor, a power transistor (which must be switched in synchronization with the rotor position) performs the polarity reversal. This process is also known as electronic commutation. The displacement of the magnets on the rotor creates a trapezoidal back-electromotive force (Back-EMF) shape when the rotor is spinning. Neglecting the higher-order harmonic terms, the Back-EMF in the motor phase (ea, eb, ec) is indicated in the following figure. Each Back-EMF has a constant amplitude for 120 electrical degrees, followed by a 60 electrical degree transition in each half-cycle.

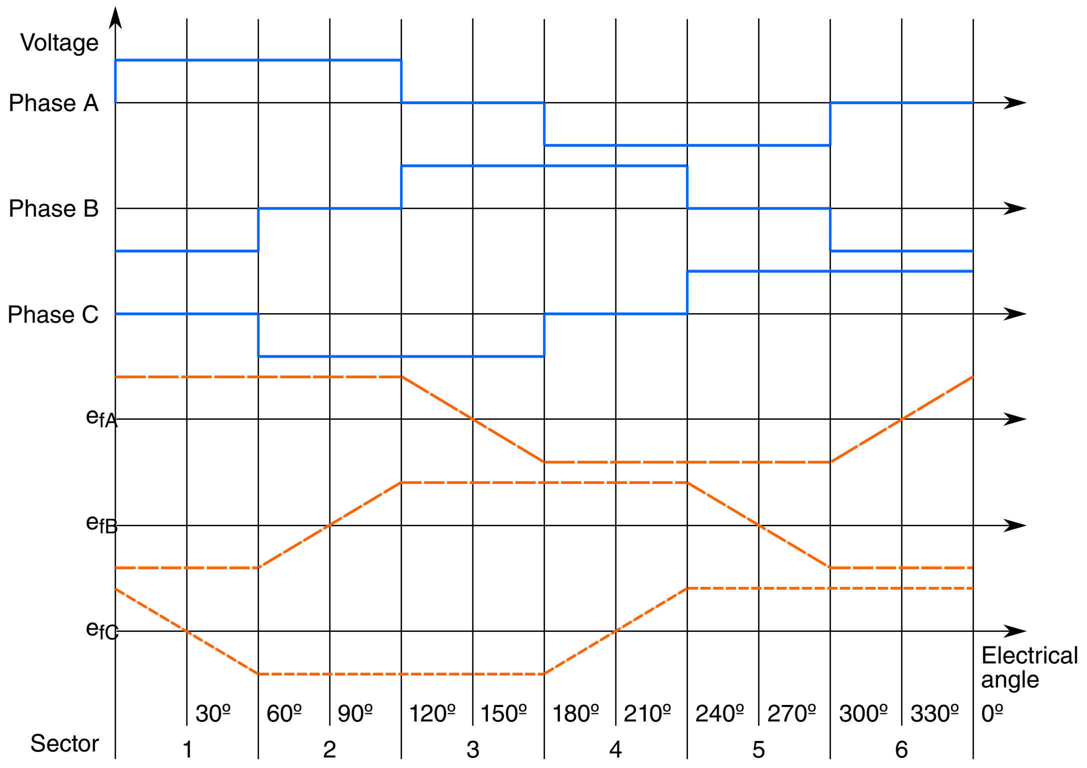


Figure 2. 3-phase voltage system for a BLDC motor

The applied DC voltage must be aligned and synchronized with the motor Back-EMF signal, as shown in the above figure. In other words, the rotor position must be either directly measured by a position sensor or estimated indirectly using a sensorless technique. This application note describes a technique based on Back-EMF sensing for rotor position estimation. To track the rotor position correctly some conditions have to be met:

- Speed range from 5–10% to 100% of nominal speed
- The Back-EMF signal must be high enough
- Possibility of the Back-EMF zero-crossing detection
- Other advanced Back-EMF estimation techniques:
  - System Observers
  - Measurement of a nonconductive phase with multisampling

Whatever a zero-crossing is, the reasons and conditions for its correct evaluation are shown in the following sections.

3.2 Principles of 6-step BLDC motor control

The 6-step BLDC control, also known as the commutation control, provides a mechanism to energize the phases according to the rotor position with the quasi-square current waveforms. Because only six discrete outputs per electrical cycle are required (as shown in Figure 2), six semiconductor power switches are sufficient to create quasi-square current waveforms for the phases. Six semiconductor power switches form a 3-phase power inverter, designed using the IGBT or MOSFET switches. The power for the system is provided by the DC-bus voltage UDCB. The semiconductor switches and diodes are modeled as ideal devices in this figure.

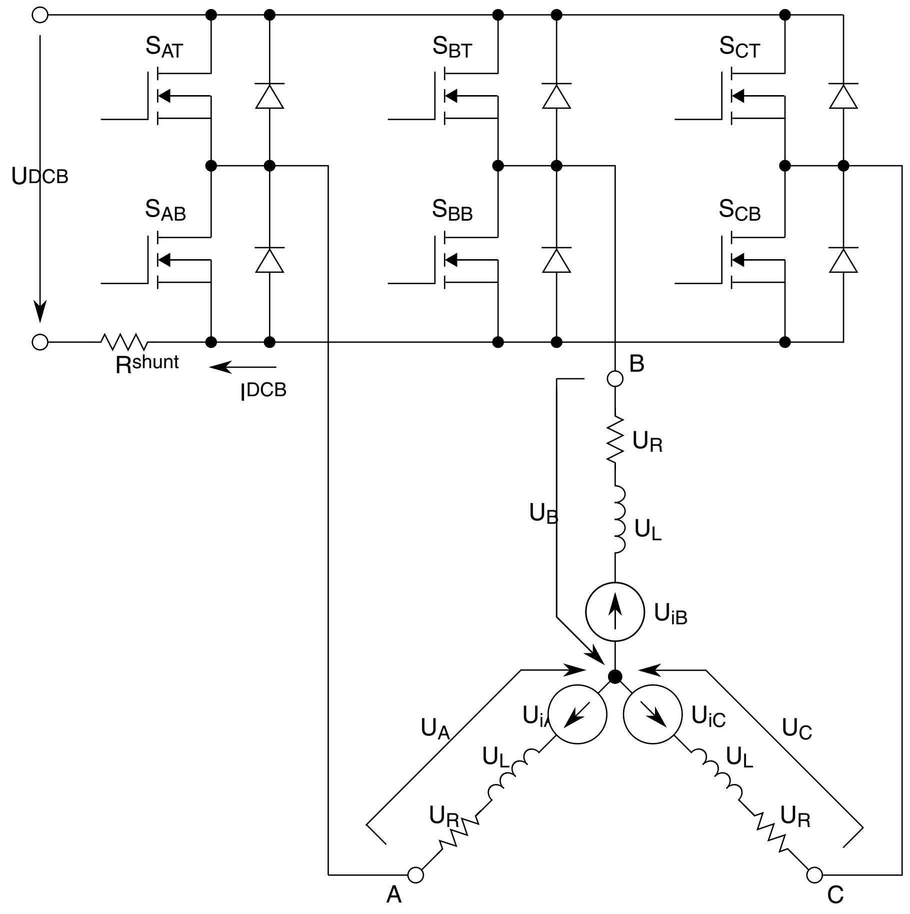


Figure 3. Power stage and motor topology

Six-step commutation is a very common method for driving a 3-phase way-connected BLDC motor. In the six-step commutation control, the BLDC motor is operated in a two-phase model. Two phases are energized while the third phase is disconnected as the space between the magnet poles passes over it and produces zero Back-EMF voltage. The selection of the two energized phases is carried out by a position sensor or a position observer. This table shows the output current waveforms for a 3-phase inverter and the switching devices that conduct during the six switching intervals per cycle.

Table 1. 6-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 Back-EMF sensing techniques, a simplified mathematical model founded on the basic circuit topology (see Figure 3) is provided. The voltage for a 3-phase BLDC motor is supplied by a typical 3-phase power stage designed using IGBT or MOSFET switches. The power stage switches are controlled by the MCU on-chip flexPWM 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 drive model consists of a 3-phase power stage and a brushless DC motor. The power for the system is provided by a DC-bus voltage source UDCB. Six semiconductor switches (SA/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 Back-EMF zero-crossing detection

Figure 2 shows the motor phase winding voltage waveforms for the right commutation. The right commutation event should be in the middle of two Back-EMF zero-crossings. So, the Back-EMF zero-crossing signal can simply be used as a rotor position feedback to estimate the right commutation time instance.

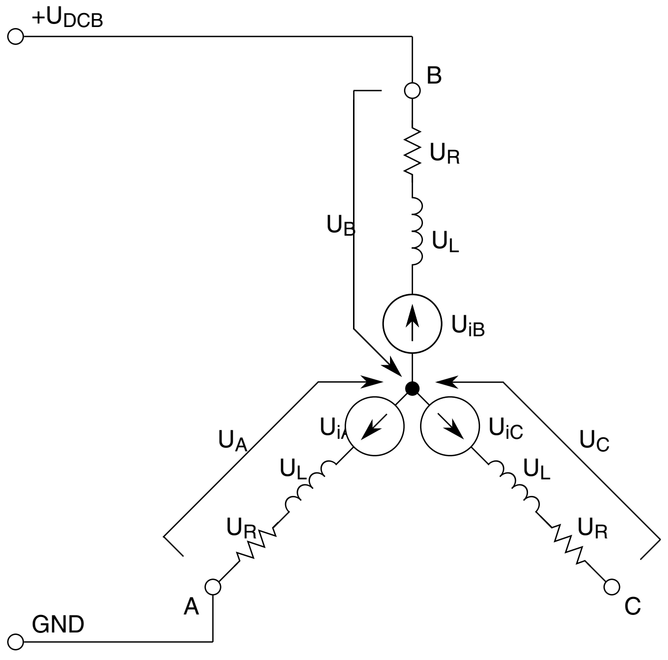


Figure 4. ZC detection and commutation diagram

The e_{1x} signals in Figure 2 are the Back-EMF voltages. These are the U_{iX} voltages in Figure 4.

This technique is established on the fact that only two phases of a motor are energized and the third non-fed phase can be used to sense the Back-EMF voltage.

The following conditions are met:

$$S_{Ab}, S_{Bt} \leftarrow PWMswitching$$

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

The voltage u_C can be calculated:

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

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

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

3.2.1.1 Back-EMF measurement

Figure 5 and Figure 6 show the Back-EMF sensing circuit that is realized on each power stage and controller board side. Each phase voltage is adjusted into the ADC converter range as shown in Figure 5. The user can set up the 3-phase voltage input ranges easily by the divider ratio.

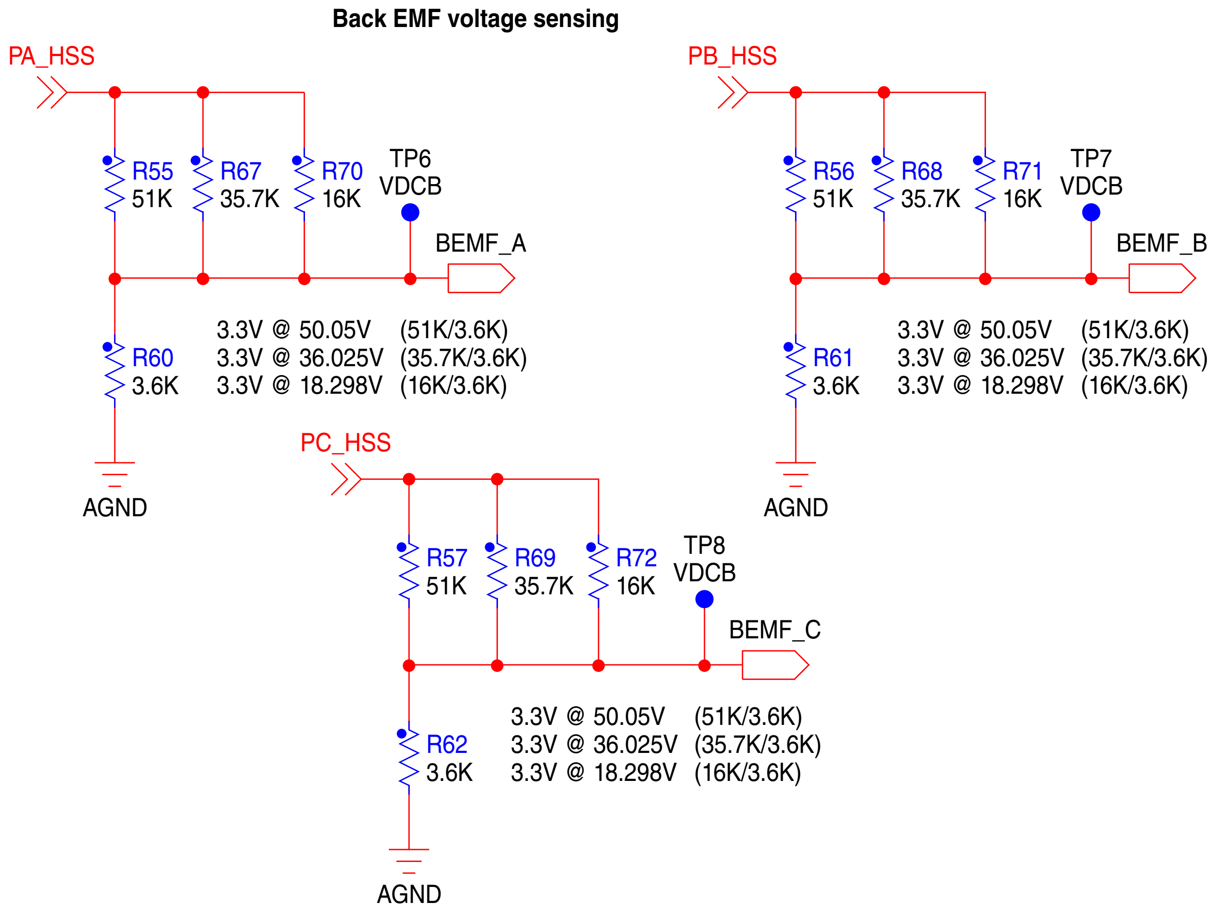


Figure 5. Back-EMF sensing circuit - dividers

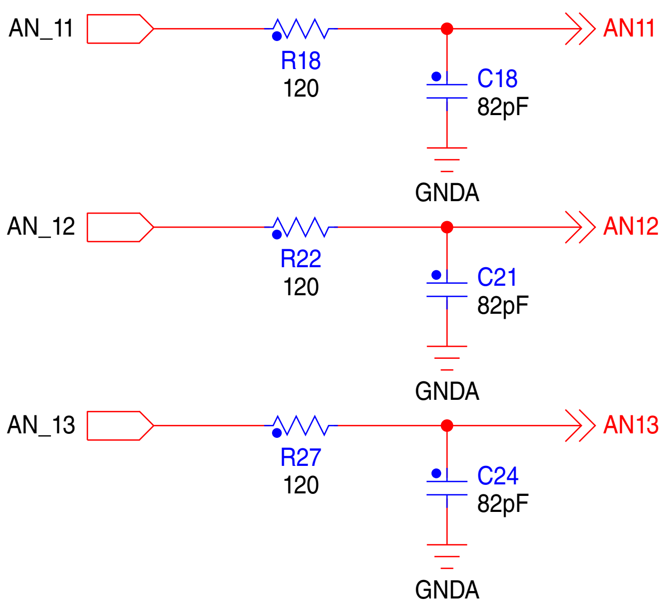


Figure 6. Back-EMF sensing circuit - low pass filters

3.3 States of BLDC drive

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

- Alignment (initial position setting)
- Start-up (forced commutation)
- Run (sensorless running with Back-EMF acquisition and zero-crossing detection)

3.3.1 Alignment

It has come up before that the main task of a BLDC motor for sensorless control is the position estimation. Before starting the motor, the rotor position is not known. The main aim of the alignment state is to align the rotor to a known position. This known position is necessary to start rotation in the proper direction and to generate a maximal torque during startup. 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 DCBus 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 on every PWM reload event.

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 should be a short state and at a very low speed where the Back-EMF is too small, so that the zero-crossing events cannot be reliably detected.

3.3.3 Run

Running Sensorless mode includes the Back-EMF 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 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.

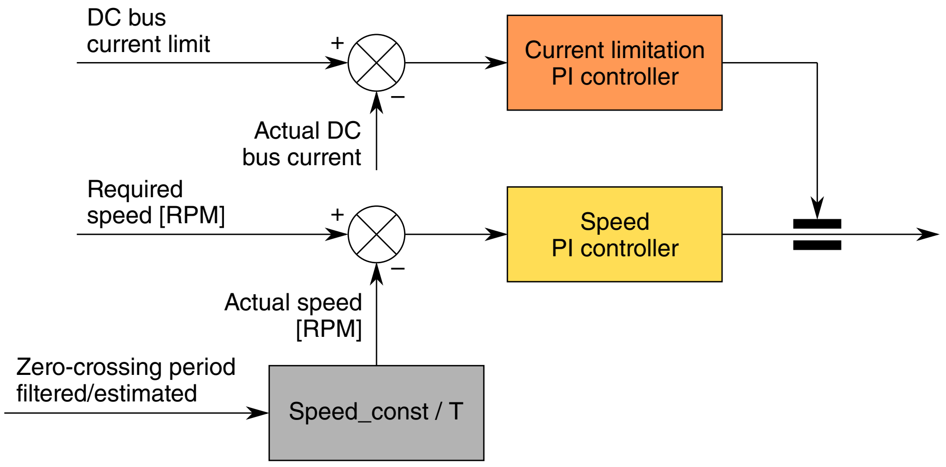


Figure 7. Speed control with torque limitation

4 MPC5643L - Controller Board configuration

The BLDC sensorless application framework is designed to meet the following technical specifications:

- MPC5643L Controller Board is used (refer to the dedicated user manual for the MPC5643L Controller Board)
- Two 3-phase low-voltage power stages with an MC33937 pre-driver is used
- PWM output frequency = 20 kHz
- current loop sampling period = 50 µs
- speed loop sampling period = 2.5 ms
- 3-phase Back-EMF voltage measurement using three dividers each per inverter leg. Phase voltage is routed to ADC0 and ADC1 as follows:
  - Motor 1 - phase A Back-EMF: ADC0/1 - CH11
  - Motor 1 - phase B Back-EMF: ADC0/1 - CH12
  - Motor 1 - phase C Back-EMF: ADC0 - CH2
  - Motor 2- phase A Back-EMF: ADC0/1 - CH13
  - Motor 2- phase B Back-EMF: ADC0/1 - CH14
  - Motor 2- phase C Back-EMF: ADC1 - CH2
- DC-Bus voltage measurement routed to ADC0 as follows:
  - Motor 1 - DC-Bus voltage: ADC0 - CH1
  - Motor 2 - DC-Bus voltage: ADC0 - CH3
- DC-Bus current measurement routed to ADC1 as follows:
  - Motor 1 - DC-Bus current: ADC1 - CH1
  - Motor 2 - DC-Bus current: ADC1 - CH3

The MPC5643L device includes special modules (flexPWM, CTU, ADC, and eTIMER) dedicated for motor control applications. These modules are directly interconnected and can be set-up in-line with any type of application or requirements. Figure 8 shows module interconnections. The modules are described below and a detailed description can be found in the MPC5643L reference manual.

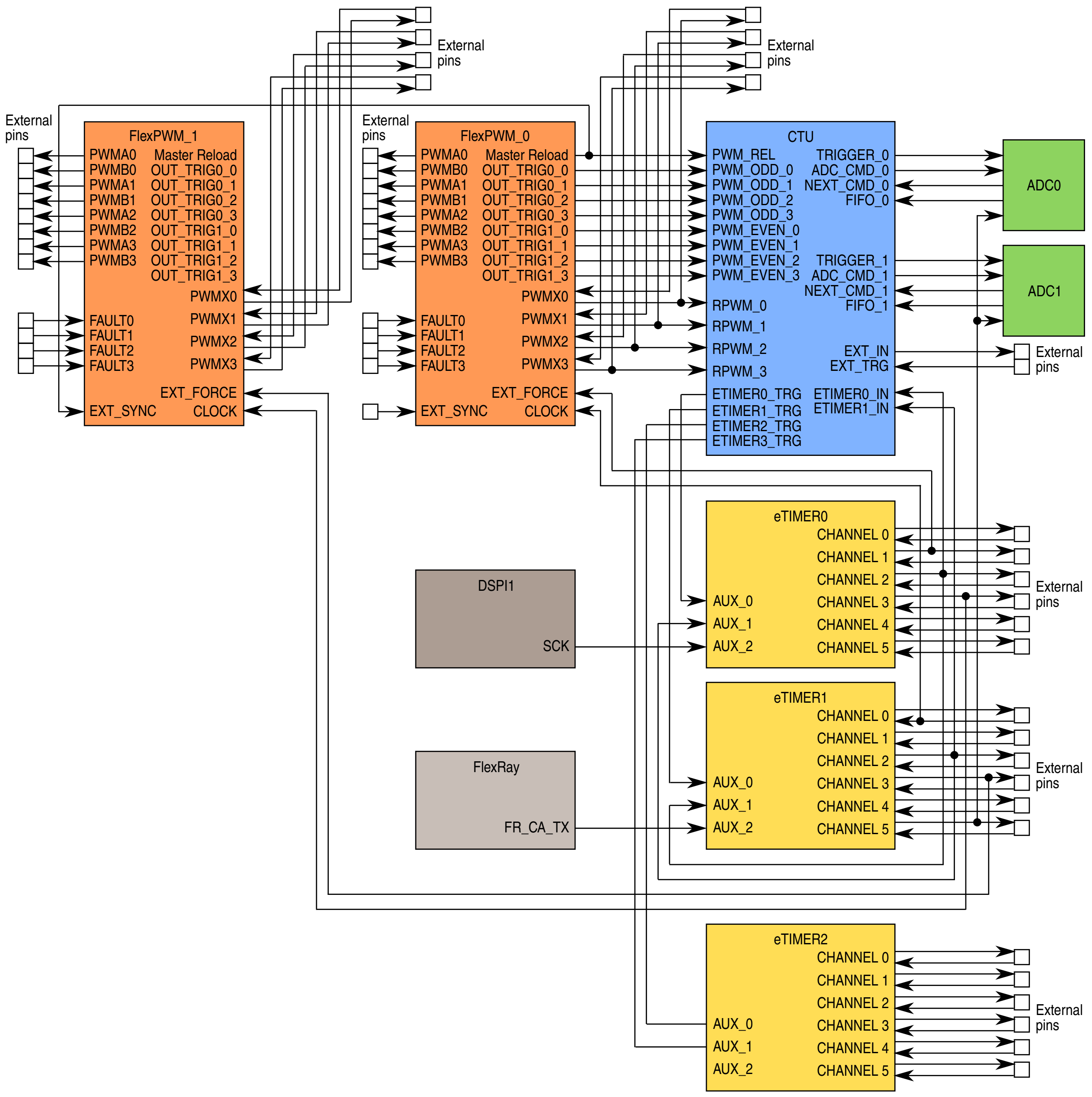


Figure 8. MPC5643L motor control peripheral modules connection

4.1 FlexPWM

The MPC5643L device includes two PLLs. PLL1 is used to generate the motor clock time domain of 120 MHz. The Clock Generation Module generates the reference clock MC_PLL_CLK for all the motor control modules (flexPWM, CTU, ADC0 and 1, eTimer0 and 1). The MPC5643L device contains two independent flexPWM modules. Each module can handle one 3-phase electrical motor and another pair of PWM outputs is also available.

The FlexPWM module zero can synchronize second module while using MRS signal as showed in Figure 8.

The FlexPWM0 sub-module #0 is configured to run as a master and to generate the Master Reload Signal (MRS) and counter synchronization signal (master sync) for other submodules in module 0 and also for FlexPWM1 and its sub-modules. The MRS signal is generated at every occurrence of sub-module #0, VAL1 compare, that is, a half cycle reload. All double buffered registers are updated on occurrence of an MRS, therefore, the update of a new PWM duty cycle is done after every PWM period.

The application uses centre-aligned PWMs. The VAL0 register defines the centre of the period and is set to zero and the INIT register to the negative value of VAL1. Suppose the PWM clock frequency is 120 MHz and the required PWM output 20 kHz, then the VAL1, VAL0, and INIT registers are set as follows:

- VAL1 = (120000000/20000)/2 = 3000 = $0x0BB8_{hex}$
- VAL0 = 0
- INIT = –VAL1 = –3000 = $0xF448_{hex}$

The duty cycle is given by setting the value of the registers VAL2 and VAL3. The VAL2 register value is the negative of VAL3.

- VAL3 = (DC * PERIOD)/2 = (0.1 * 6000)/2 = 300 = $0x012C_{hex}$
- VAL2 = –VAL3 = –300 = $0xFED4_{hex}$

The synchronization between module 0 and 1 is done by MRS from module 0. This signal is internally connected to “external sync” input module 1. The module 1 is set to generate the MRS in the middle of the PWM period. PWM signals from module 0 and 1 that are shifted by half of period can be seen in Figure 9. This leads to the uniform use of energy from power source and less EMI radiation.

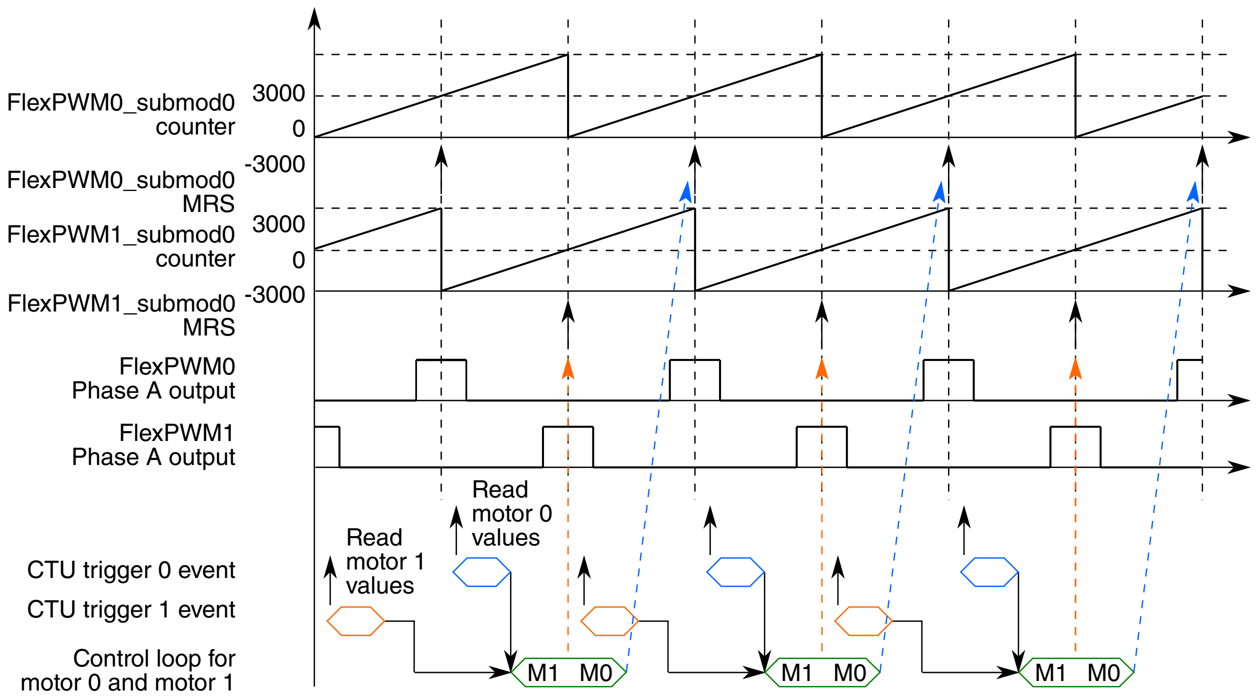


Figure 9. MPC5643L PWM timing generation diagram

4.2 Cross Triggering Unit (CTU)

The CTU module works in triggered mode. The MRS from the flexPWM0 submodule0 is selected from the input selection register (TGSISR) to reload the TGS counter register with the value of the TGS Counter Reload Register (TGSCRR). The TGS is able to generate up to eight events. Each trigger can be delayed from an MRS occurrence, the delay is set in the TGS Compare Registers.

The MRS signal is generated after every PWM period. The counter can count up to 6000_{DEC} and the INIT value is zero.

The application uses two trigger events for measuring the Back-EMF, DC-Bus voltage, and DC-Bus current for motor 0 and 1. The other triggers are free and can be used for triggering other application events.

- T0CR = 120_{DEC} - motor 0 analog quantities
- T2CR = 3120_{DEC} - motor 1analog quantities

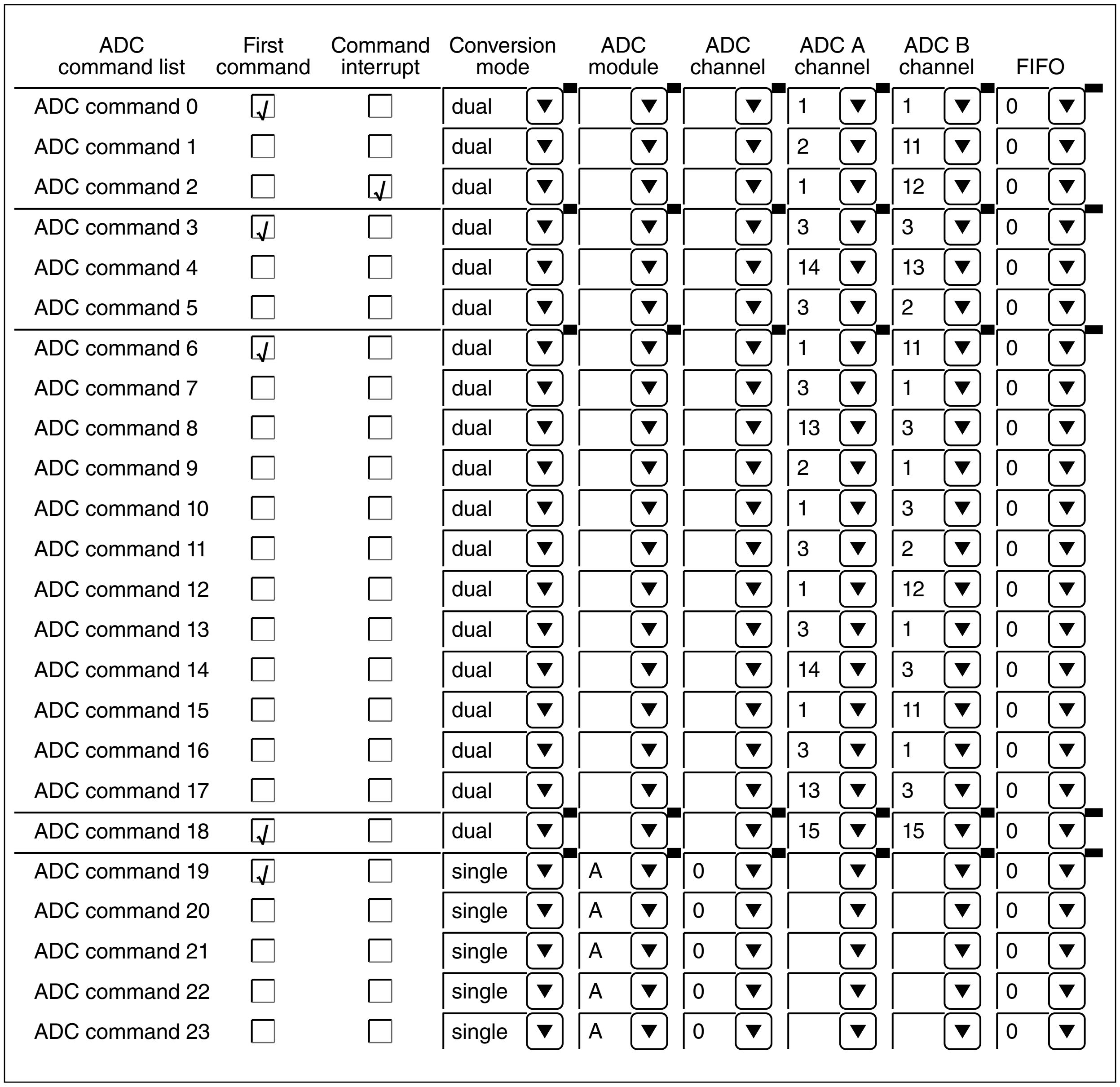
T0CR and T2CR values are set up with respect to the real delays in the system as shown in Figure 9. The minimal delay value is given by the dead-time value for a rising edge, the power transistor turn-on delay, and the rise time and settling time of the Back-EMF RC filter.

The CTU Scheduler subUnit (SU) generates the trigger event according to the occurred trigger event. The following trigger event is generated:

- ADC command output:

T0CR generates an ADC command event output for motor 0, with the command offset initially set to zero. This is used as the synchronization signal to the ADC (ADC commands #0 for Back-EMF voltages, DC-Bus voltage, and DC-Bus current measurement).

T2CR generates an ADC command event output for motor 1, with the command offset initially set to three. This is used as the synchronization signal to the ADC (ADC commands #3 for Back-EMF voltages, DC-Bus voltage, and DC-Bus current measurement).



ADC command list	First command	Command interrupt	Conversion mode	ADC module	ADC channel	ADC A channel	ADC B channel	FIFO
ADC command 0	✓		dual			1	1	0
ADC command 1			dual			2	11	0
ADC command 2		✓	dual			1	12	0
ADC command 3	✓		dual			3	3	0
ADC command 4			dual			14	13	0
ADC command 5			dual			3	2	0
ADC command 6	✓		dual			1	11	0
ADC command 7			dual			3	1	0
ADC command 8			dual			13	3	0
ADC command 9			dual			2	1	0
ADC command 10			dual			1	3	0
ADC command 11			dual			3	2	0
ADC command 12			dual			1	12	0
ADC command 13			dual			3	1	0
ADC command 14			dual			14	3	0
ADC command 15			dual			1	11	0
ADC command 16			dual			3	1	0
ADC command 17			dual			13	3	0
ADC command 18	✓		dual			15	15	0
ADC command 19	✓		single	A	0			0
ADC command 20			single	A	0			0
ADC command 21			single	A	0			0
ADC command 22			single	A	0			0
ADC command 23			single	A	0			0

Figure 10. CTU ADC command list configuration

4.3 eTimer

The eTimer module0.channel1 generates, through OFLAG output, a forced signal for the flexPWM module 0, which changes the PWM output with regard to the new motor commutation state, as shown in Figure 8. The time base for the counter is derived from MC_PLL_CLK. The prescaler register divides the MC_PLL_CLK by 128. The time base for commutation events is:

— f_{COMM} = (120000000/128) = 937500 Hz

The value of the COMP1 register defines the time of the next commutation event. The eTimer output (OFLAG) is set at the compare occurrence, and generates an external forced signal for the flexPWM module. The external forced signal updates the PWM outputs in-step with the preloaded state according to the newly applied sector pattern.

The same approach is implemented on the flexPWM module 1 (second BLDC motor). The force signal is derived from eTimer module1.channel3 OFLAG output.

This ensures both independently controlled motor commutation without any delay. Both events generate interrupt. The PWM module settings are preset for next commutation sectors in this interrupt service routine and are stored in the double buffered registers. The flexPWM module waits for the next force signal that rewrites new PWM setup into registers.

5 Software implementation

5.1 Introduction

This section describes the software design of the dual BLDC sensorless control algorithm. Figure 11 shows the application block diagram.

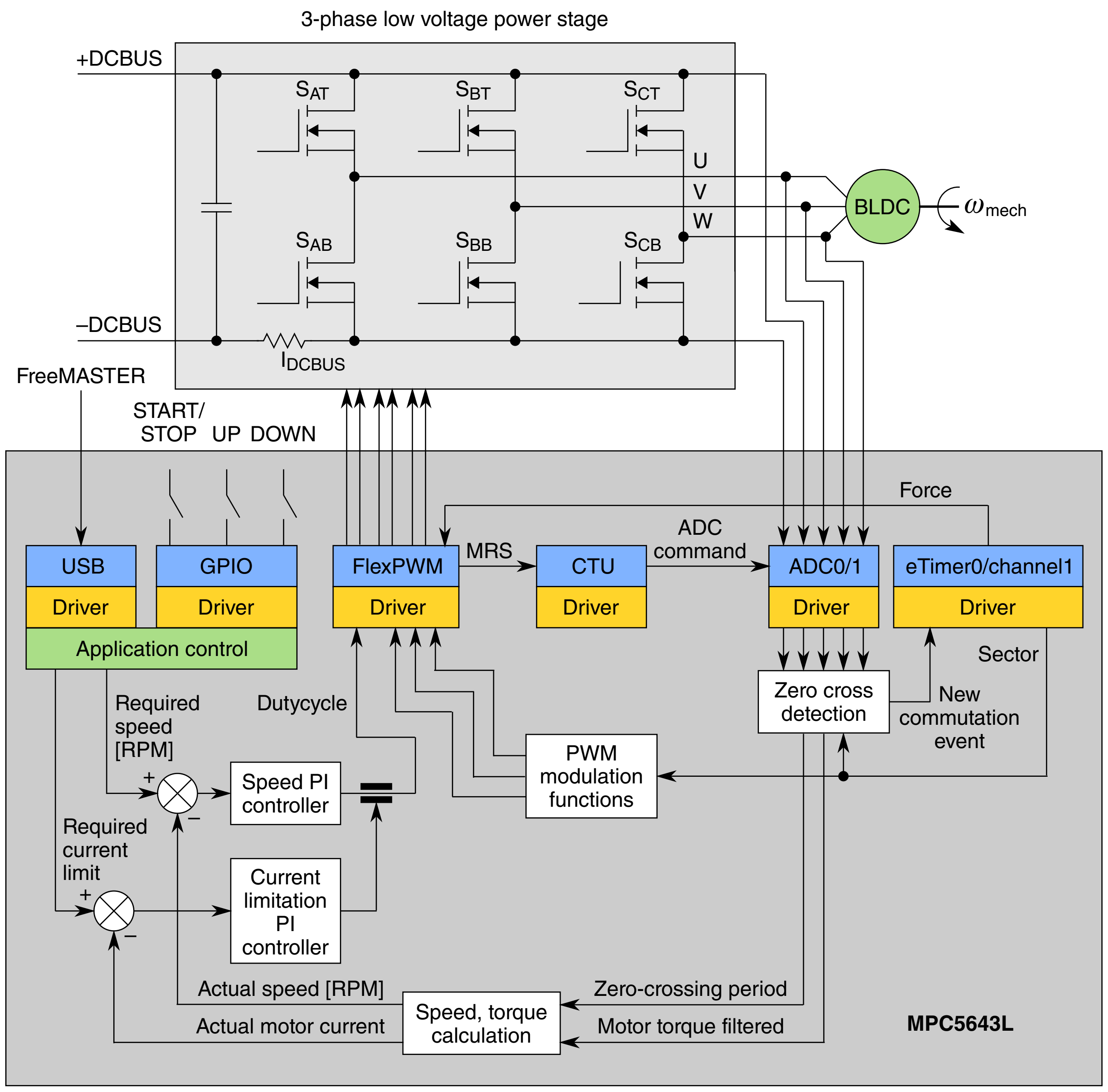


Figure 11. System block diagram

The application is optimized for using MPC5643L motor control peripherals to achieve small core impact. The motor control peripherals (flexPWM, CTU, eTimer, and ADC modules) are internally linked/set up together to work independently from the core and to achieve deterministic sampling of analog quantities and precise commutation of the stator field. The software part of the application consists of different blocks that will be described below. The entire application behavior is scalable by the FreeMASTER tool from a PC.

The MPC564xL device has embedded floating-point unit (FPU), supporting scalar and vector SIMD single-precision floating-point operations and also signal processing extension unit, supporting SIMD fixed-point operations. Both FPU and signal processing extension unit use the 64-bit general-purpose register file. The application has been implemented in fixed-point arithmetic and also in floating point arithmetic.

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 CTU-ADC command interrupt request every 50 µs, see Figure 9. 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 an eTimer0.Channel 1 interrupt event for motor 0 and eTimer1.Channel3 interrupt routine for motor 1. 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.

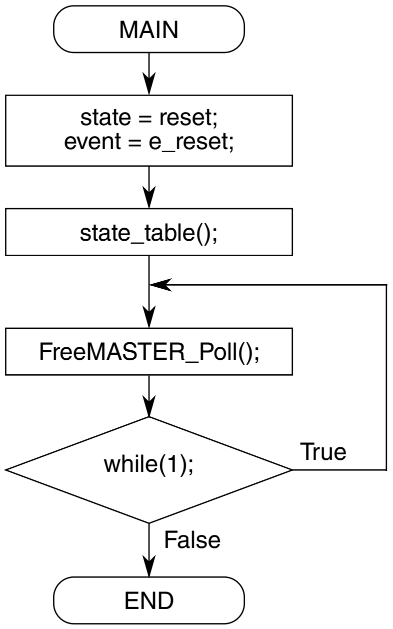


Figure 12. Main task flow

This type of application requires precise and deterministic sampling of analog quantities, and to execute all motor control functions the state machine routines are called within a periodic interrupt routine. In reference to the state machine, the interrupt has to be set-up and allowed at the end of the RESET state, where all peripheral settings also have to be done. Consequently, the RESET state is called before the main loop, as shown in Figure 12. The background loop handles non-critical tasks, such as the FreeMASTER communication polling and the MOSFET pre-driver fault service routine.

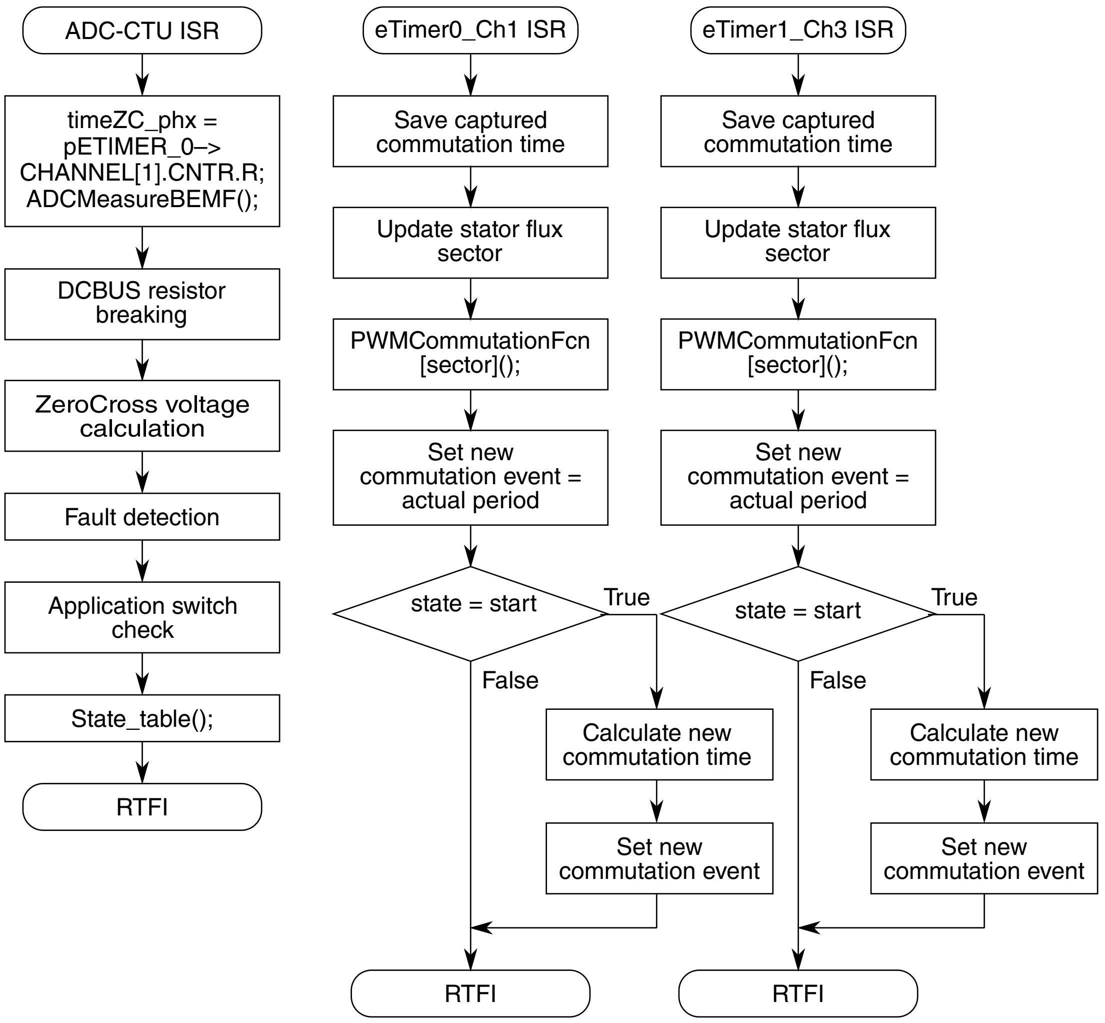

ADC-CTU ISR
timeZC_phx = pETIMER_0–> CHANNEL[1].CNTR.R; ADCMeasureBEMF();
DCBUS resistor breaking
ZeroCross voltage calculation
Fault detection
Application switch check
State_table();
RTFI
eTimer0_Ch1 ISR
Save captured commutation time
Update stator flux sector
PWMCommutationFcn [sector]();
Set new commutation event = actual period
state = start
True
False
Calculate new commutation time
Set new commutation event
RTFI
eTimer1_Ch3 ISR
Save captured commutation time
Update stator flux sector
PWMCommutationFcn [sector]();
Set new commutation event = actual period
state = start
True
False
Calculate new commutation time
Set new commutation event
RTFI

Figure 13. CTU-ADC IRS routine flow chart

5.3 Speed evaluation and control

The application uses eTIMER0.channel1 resp. eTIMER1.Channel3 to achieve a precise commutation of both BLDC motors as described below.

When the zero-cross event is recognized, the eTimer0.channel1.COMP1 resp. eTimer1.channel3.COMP1 register is filled by a new calculated value of the next commutation time. When a counter matches the COMP1 register value, the OFLAG signal forces corresponding flexPWM module with a new set-up without any delay or CPU load.

5.3.1 Speed evaluation

The speed is calculated in the Slow Control Loop, which is part of the *BLDC_Fast_ISR* routine. The zero-cross detection algorithm provides the actual commutation period duration for each commutation event. These variables are referred to the eTIMER0.channel1 resp. eTimer1.channel3. The clock for both eTIMER0.channel1 or eTimer1.channel3 is set up to 937500 Hz. So, to calculate the real-time commutation period:

$$T_{REAL} = T \times T_{CLK}$$

$$T_{CLK} = \frac{1}{f_{CLK}}$$

$$T_{REAL} = \frac{T}{f_{CLK}}$$

where:

- T_{REAL} is the real commutation period
- T_{CLK} is the period of the eTIMER0.channel1 or eTimer1.channel3
- T is the value measured in the eTIMER0.channel1 or eTimer1.channel3 increments
- f_{CLK} is the eTIMER0.channel1 or eTimer1.channel3 clock rate

If commutation period is known, the period of one electrical revolution can be calculated by:

$$T_{elrev} = T_{REAL} \times N = \frac{T \times N}{f_{CLK}}$$

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 above equation must be multiplied by the number of pole-pairs:

$$T_{mechrev} = T_{elrev} \times p = \frac{T \times N \times p}{f_{CLK}}$$

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

$$\omega_{mech} = \frac{60}{T_{mechrev}} = \frac{60 \times f_{CLK}}{T \times N \times p}$$

If the clock rate is 937500 Hz, the number of commutations per electrical revolution is 6, and the number of pole-pairs is 4; we can get the constant:

$$c = \frac{60 \times f_{CLK}}{N \times p}$$

Therefore, the speed is calculated as:

$$\omega_{mech} = \frac{c}{T}$$

where:

- c is the mechanical speed constant, that is, $14.0625\text{x}10^6$.

To achieve a better resolution, the mechanical speed is multiplied by 1000 in fixed-point implementation.

5.3.2 Speed controller

The motor speed PI controller is called in every 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 whole mechanical system parameters. The speed loop bandwidth was chosen as 20 Hz and attenuation as 1. Unfortunately, the parameters of the LINIX motor were unknown prior the test, therefore the constants of the PI controller have been set experimentally.

5.4 Zero-cross detection

The zero-cross algorithm is executed in each *BLDC_Fast_ISR* routine. The CTU module triggers stator flux sector related analog quantities, such as the actual DC-BUS and related phase BEMF voltage, the DC-BUS current, and time of measurement. Figure 2 shows the behavior of the BEMF voltage for each sector. The relevant zero-cross function is called with respect to the actual stator flux sector.

The zero-cross event occurs when the phase BEMF voltage crosses the $U_{DCBUS}/2$, and it is half 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 eTimer0.channel1 resp. eTimer1.channel3 compare register to achieve precise commutation of the stator flux. The algorithm also stores the motor current and the actual zero-cross period. These values are used for speed and motor current calculations.

5.5 Current limitation controller

The motor current limitation controller is called in every fast control loop, which is after every 50 µs. The parameters of the armature current PI controller are calculated assuming the armature current loop bandwidth and attenuation and motor physical constants.

5.6 State machine

The application state machine is implemented using a two-dimensional array of pointers to a function variable 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.

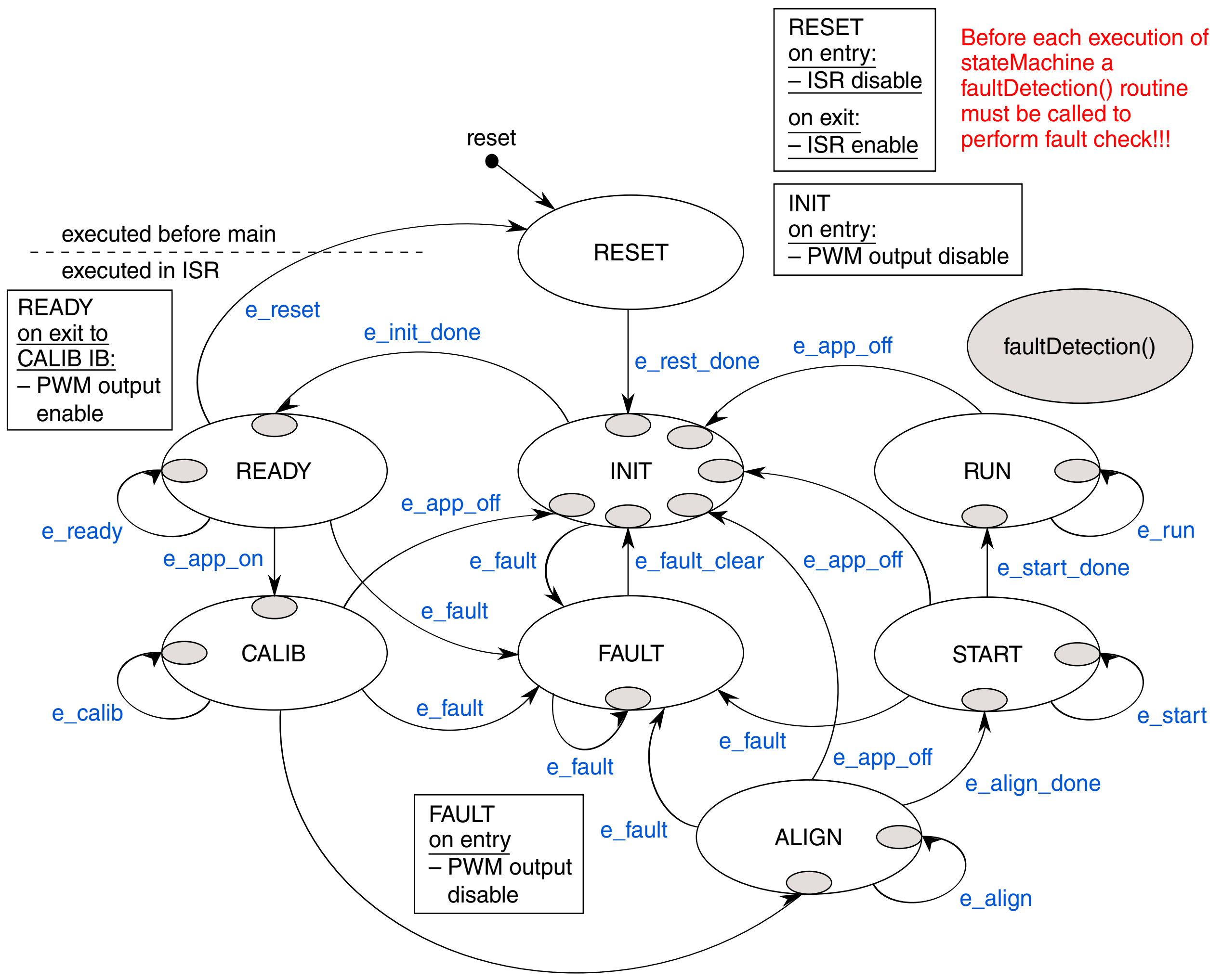


Figure 14. Application state machine

The application state machine consists of the following eight states, selected using the variable table state defined as *AppStates*:

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

To signalize/initiate a change of the state, 15 application events are defined and selected using the variable event defined as *AppEvents*:

- 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 - event = 13
- 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 into the main() function. It is executed only once at the start of the main function to provide system variables and all peripherals settings. Before configuring all peripherals, all interrupts are disabled and enabled at the RESET state end, with respect to interrupt driven application as described before. This routine also includes initialization and setting of the MC33905 System Basis Chip that provides power supply for the MPC5643L controller board and MC33937 MOSFET pre-driver. Both routines use SPI peripheral and must be called after the DSPI and SIU configuration routines.

5.6.2 INIT state

State INIT is similar to the RESET routine one pass routine, which allows users to set up application variables, and at the end the transition event is set to the READY state if there is no fault event. It is executed directly after the RESET state or after a RUN state when an 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.

5.6.4 READY state

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

Transition to the RESET state is performed by setting the variable event to event=e_reset, which is done automatically when the user sets switchAppReset true using FreeMASTER.

Transition to the INIT state is performed by setting the variable event to event=e_app_on, which is done automatically on the rising edge of switchAppOnOff=true using FreeMASTER, or a second way to change its value is to switch on the external switch on the MPC5643L controller board.

5.6.5 CALIB state

The CALIB state provides calibration of the analog quantities used by the sensorless motor control algorithm. The analog offsets are calibrated for all six voltage vectors applied to the 3-phase bridge. When the calibration is done 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, this enables transition to the Alignment state.

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

Transition to the INIT state is performed by setting the variable event to event=e_app_off, which is done automatically on the falling edge of switchAppOnOff=false using FreeMASTER, or the other way to change its value is to switch off the external switch on the MPC5643L controller board.

5.6.6 ALIGN state

The ALIGN state provides the motor rotor alignment process as it has been shown in Alignment. The user can set up the variable ALIGNMENT_TIME and the propriety motor current depending on the minimal mechanical system behavior (mechanical system inertia, motor time constants, and so on) time to assure 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 and variables used for the next state are initialized, and the variable event is automatically set to event=e_align_done. This enables transition to the START state.

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

Transition to INIT state is performed by setting event to event=e_app_off, which is done automatically on falling edge of switchAppOnOff=false using FreeMASTER or the other way to change its value is to switch off the external switch on the MPC5643L controller board.

5.6.7 START state

The START state provides the start rotor rotation sequence as has been shown in Start-up. The motor current PI controller function Ureq=GFLIB_ControllerPIpAW(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. The PI controller function is part of the MPC5643L Motor Control Library and its detailed description is shown in the MPC5643L_MCLib manual.

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

Transition to the INIT state is performed by setting the variable event to event=e_app_off, which is done automatically on the falling edge of switchAppOnOff=false using FreeMASTER, or the other way to change its value is to switch off the external switch on the MPC5643L controller board.

5.6.8 RUN state

The RUN state provides the motor speed and current regulation sequence as has been described in Run. The zero-cross detection function ZCdetection[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 the variable event to event=e_app_off, which is done automatically on the falling edge of switchAppOnOff=false using FreeMASTER, or the other way to change its value is to switch off the external switch on the MPC5643L controller board.

5.7 Library functions

The application source codes use the Auto Math and Motor Control Library Set for the MPC564xL family of microcontrollers with 32-bit fixed-point or with single precision floating-point arithmetic depending on the implementation. The library contains three independent library blocks GFLIB, GDFLIB, and GMCLIB. GFLIB includes basic mathematical functions, such as Sine, Cosine, LUT, ramp, and so on. Advance filter functions are part of the General Digital Filters Library and standard motor control algorithms are part of the General Motor Control Library.

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 motor.

All application parameters dedicated to the motor or application ratings (max. voltage, velocity, and so on) are defined in the *BLDC_appconfig.h* file and commented to help the user modify the parameters according to their own specific requirements.

All hardware specific parameters dedicated to the hardware boards and processor (pin assignment, clock setting, peripheral settings, and so on) are defined in the *MPC5643L_appconfig.h* file.

6 Application performance

The fixed-point implementation takes 18.4 µs average and 25.4 µs maximum. The floating-point implementation takes 5.6 µs average and 9.5 µs maximum for one motor application, which is maximum 20% of CPU load by fast loop calculation frequency 20 kHz. Dual motor application takes 10.8 µs and 17.8 µs maximum.

Application code was compiled using a Green Hills compiler with optimization options listed in Table 2. The timing was measured on the e200z4 core at a 80 MHz system clock frequency using optimal flash read/write wait state control and address pipelining control settings.

Table 2. Compiler options

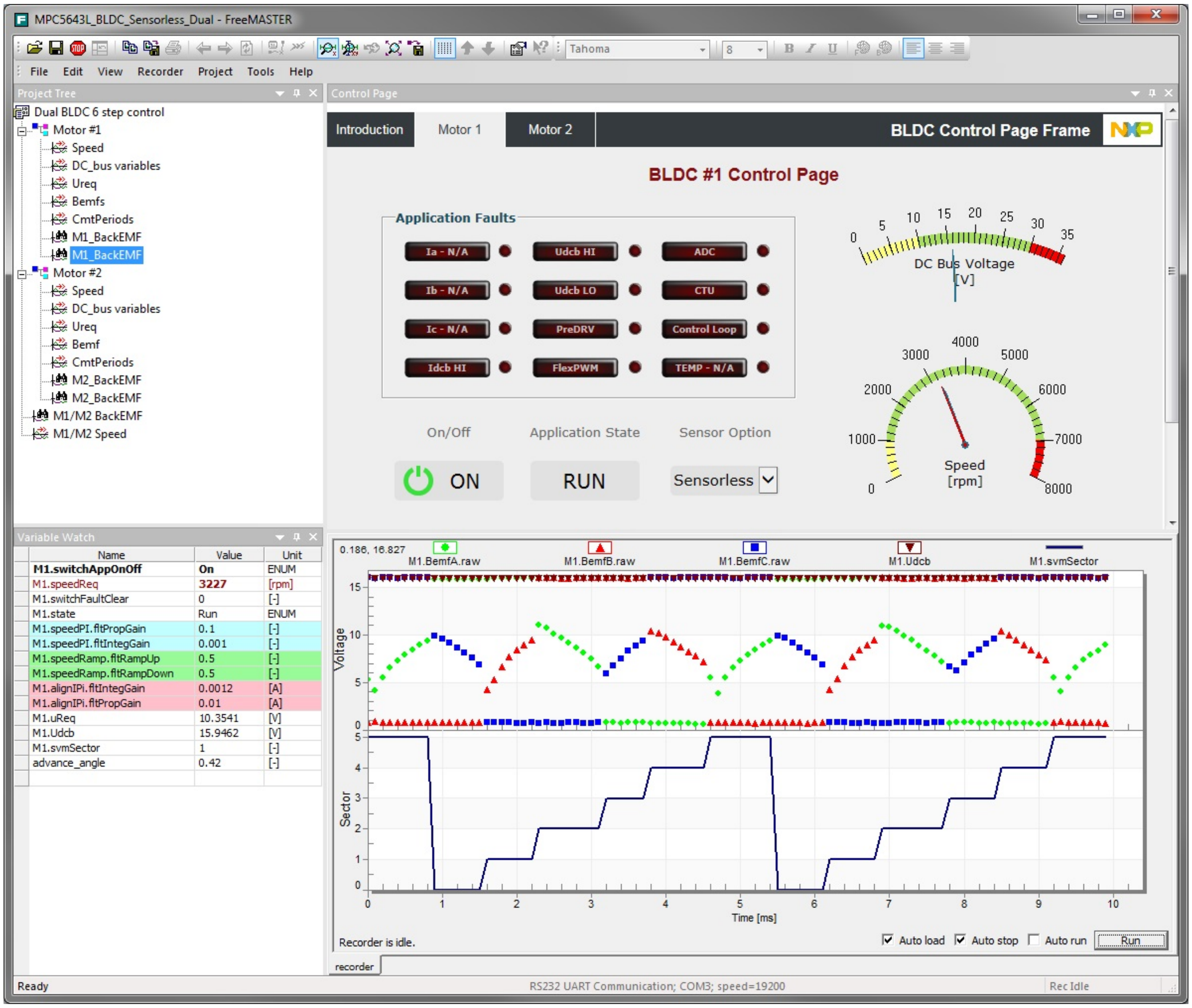
Compiler Option	Description
-Ospeed	Optimize for speed.
-Opeep	Peephole optimization.
-Opipeline	Pipeline optimization (pipeline instruction scheduling).
-isel	Automatic generation of isel integer conditional move instruction.
-sda	Small data area optimization with a threshold of 8 bytes.
-auto_sda	Automatic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

7 FreeMASTER user interface

The FreeMASTER debugging tool is used to control the application and monitor variables during run time.

Communication with the host PC is via USB. However, because FreeMASTER supports RS232 communication, there must be a driver for the physical USB interface CP2102 installed on the host PC that creates a virtual COM port from the USB. The driver can be installed from www.silabs.com.

The application configures the LINflex module of the MPC5643L for a communication speed of 19200 bps. Therefore, FreeMASTER also has to be set to this speed.



Name	Value	Unit
M1.switchAppOnOff	**On**	ENUM
M1.speedReq	**3227**	[rpm]
M1.switchFaultClear	0	[-]
M1.state	Run	ENUM
M1.speedPI.fltPropGain	0.1	[-]
M1.speedPI.fltIntegGain	0.001	[-]
M1.speedRamp.fltRampUp	0.5	[-]
M1.speedRamp.fltRampDown	0.5	[-]
M1.alignIPi.fltIntegGain	0.0012	[A]
M1.alignIPi.fltPropGain	0.01	[A]
M1.uReq	10.3541	[V]
M1.Udcb	15.9462	[V]
M1.svmSector	1	[-]
advance_angle	0.42	[-]

Figure 15. FreeMASTER control page for controlling the application

7.1 Application start

- Install the USB driver to create a virtual COM port for emulation of RS232 communication (“CP210x USB to UART Bridge VCP Drivers“ from www.silabs.com - downloaded from this link)
- Connect a USB cable to the MPC5643L controller board and to the host PC
- Connect the power supply to the power-stage. The controller board is supplied from the power stage. The BLDC motor used is designed for 24 V phase voltage.
- Start the FreeMASTER project located in \FreeMASTER_control\MPC5643L_BLDC_Sensorless_Dual.pmp for dual BLDC motor application or MPC5643L_BLDC_Sensorless_Single.pmp for single BLDC motor application.
- Enable communication by pressing the “STOP” button in the toolbar in FreeMASTER, or by pressing “CTRL+K”
- Successful communication is signalized in the status bar. If communication is not established, check the USB connection between the PC and the MPC5643L controller board and the communication port and speed. The communication port and speed can be set up in menu Project\Options (or by pressing “CNTR+T”). The communication speed has to be set to 19200 Bd.
- If no actual faults are present in the system, all LED-like indicators in the "Application Faults" field shall be dark red. If there is a fault present, identify the source of the fault and remove it. Successful removal is signalized by the switching off the respective LED-like indicator.
- If all of the LED-like indicators are off, clear the pending faults by pressing the "FAULT CLEAR" button.
- Click the On/Off button or switch over the ON/OFF switch on the MPC5643L controller board to run the application. The BLDC motor starts running. The mechanical speed can be changed by writing to the desiredSpeed variable.

8 Conclusion

The described design shows simplicity and efficiency in use of the MPC5643L microcontroller for independent two BLDC motor controls, and introduces it as an appropriate candidate for different low-cost two motor applications in the automotive area. Both types of implementation show the mentioned microcontroller is developed to achieve application safety requirements.

9 References

1. 3-phase BLDC Sensorless Motor Control Development Kit with MPC5643L, available at http://www.nxp.com/AutoMCDevKits
2. *MPC5643L Microcontroller Reference Manual* (document MPC5643LRM)
3. *MPC5643L Controller Board User Manual* (document MPC5643LMCBUM)
4. *3-Phase BLDC/PMSM Low Voltage Power Stage User Manual* (document 3PHLVPSUG)
5. *MC33937 Three Phase Field Effect Transistor Pre-driver Data Sheet* (document MC33937)
6. Automotive Math and Motor Control Library Set for MPC564xL, http://www.nxp.com/AutoMCLib
7. FreeMASTER Run-Time Debugging Tool, http://www.nxp.com/FREEMASTER

10 Revision history

This section documents the changes done to this document.

Table 3. Revision history

Revision	Date	Substantive changes
0	08/2012	Initial release.
1	02/2016	Updated Figure 1, added Figure 15, and various small updates.

Document Number AN4548
Revision 1, 02/2016