AN0064
Application Note

AT32 Motor Control Library User Guide

Introduction

This application note describes how to use AT32 motor control library in detail covering from the setup of the hardware and software environments, motor control library architecture, header file settings, a rich choices of functions to the application examples.

Applicable products:

Part number	AT32F4xx, AT32L0xx

Contents

List of tables

List of figures

1 Overview

This application note mainly covers the following content:
Target motor type: Three-phase permanent magnet synchronous motor (brushless DC motor BLDC)

Control methods:

- FOC vector control
- 120° square wave commutation

Three-phase PWM method:

- SVPWM
- 120° conduction PWM control

Phase current sensing method:

- 3-shunt current sensing
- 2-shunt current sensing
- 1-shunt current sensing and reconstruction method

Rotor position detection mode:

- Hall-effect position sensor
- Photoelectric incremental encoder

Initial rotor position estimation modes:

- Three-phase voltage vector mode
- Two-phase voltage vector mode

Rotor position estimation in FOC driving mode:

- BEMF estimation with Luenberger observer

Rotor position estimation in 120° square wave control mode:

- BEMF zero crossing point detection by comparator
- BEMF zero crossing point detection by ADC

Sensored FOC sine-wave control mode:

- Voltage vector control
- Torque control (current vector control)
- Speed control
- Field weakening control
- Positioning control
- Regenerative braking control

Sensorless FOC sine-wave control mode:

- Open loop startup
- Torque control (current vector control)
- Speed control
- Field weakening control
- Regenerative braking control

Sensored 120° square-wave commutation mode:

- 120° square wave voltage control
- Torque control (120° square wave current control)
- Speed control

Sensorless 120° square-wave commutation mode:

- 120° square wave voltage control
- Open loop startup
- Torque control (120° square-wave current control)
- Speed control

2 Environment requirements

2.1 Hardware environment

Hardware resources required include a PMSM (BLDC) motor, AT-Link or third-party debugger and a motor control board. If the AT-MOTOR-EVB evaluation board is to be used, please refer to the *AT32_LV_Motor_Control_EVB_User _Manual* for details on hardware configurations.

- PMSM (BLDC) motor
- Debugger
- Motor control board

2.2 Software environment

1. Set up an AT32 development environment according to the getting-started guidelines of the corresponding AT32 MCU series and their respective functions.
2. After a new project is created, add the header files and functions of the motor control library to the new project, and configure motor control mode, motor parameters, control board parameters, controller parameters and MCU peripheral parameters in the header files.
3. Modify ID files according to the Flash memory size of each AT32 MCU, see Table 1 below for details. For example, for AT32F413RCT7 with a 256KB Flash memory, its IROM1 start address is 0x8000000 with a size of 0x3F800, whereas its IROM2 starts at the address 0x803F800 with a size of 0x800. See Figure 1 for more information.
4. Write user program according to users' needs, and call the motor library functions in the program to quickly develop the motor control project.

Table 1. Correspondence between Flash size and ROM

Flash size	**1024K**	**512K**	**256K**
IROM1(adress)	0x8000000	0x8000000	0x8000000
IROM1(size)	0xFF800	0x7F800	0x3F800
IROM2(adress)	0x80FF800	0x807F800	0x803F800
IROM2(size)	0x800	0x800	0x800

Flash size	**128K**	**64K**	**32K**	**16K**
IROM1(adress)	0x8000000	0x8000000	0x8000000	0x8000000
IROM1(size)	0x1FC00	0x0FC00	0x07C00	0x03C00
IROM2(adress)	0x801FC00	0x800FC00	0x8007C00	0x8003C00
IROM2(size)	0x400	0x400	0x400	0x400

Note [1]: For keil v5.33, the AT32 BSP source code does not support V6.15 compiler. Thus please use keil compiler version 5 for compiling purposes.

Figure 1. ROM settings in AT32F413RCT7 (AT32 IDE)

```
AT32F413xC_FLASH.ld
_estack = 0x20008000;    /* end of RAM */

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
FLASH (rx)      : ORIGIN = 0x08000000, LENGTH = 0x3F800
MC_DATA(r)      : ORIGIN = 0x0803F800, LENGTH = 0x800
RAM (xrw)       : ORIGIN = 0x20000000, LENGTH = 32K
SPIM (rx)       : ORIGIN = 0x08400000, LENGTH = 16384K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH


  .mc_data :
  {
    . = ALIGN(4);
    *mc_flash_data_table.o(.rodata .rodata*)
    . = ALIGN(4);
  } >MC_DATA

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(4);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)

```

3 Motor control library files

Figure 2 shows the organization of the motor control library program. The lower-level hardware peripherals are controlled through BSP functions. The motor control functions and UI functions are built on BSP and low-level functions. The user program are created based on the motor control functions and UI functions, making it easier for users to call motor control functions to control MCU hardware peripherals and execute motor control program. The UI control functions are linked to the external PC UI software to transmit motor control state or change motor control parameters and commands in a real-time manner.

Figure 2. Motor control program architecture

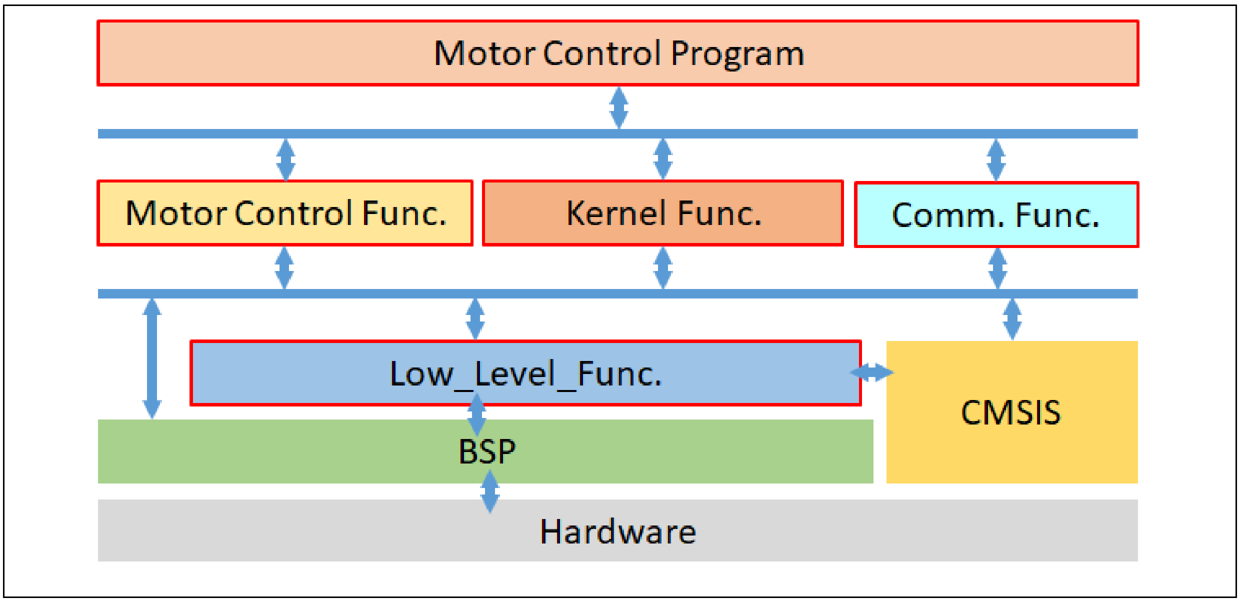


Figure 3 shows an overall structure of motor control library files and the relationship among them.

The "*motor_control_drive_param.h*" header files allow users to configure drive mode, and such parameters as motor, control board and controller.

The "*mc_hwio.h*" header files are used to set MCU peripheral parameters through the pins connecting MCU peripherals to the control board.

All of the parameter settings are then incorporated into the "*mc_foc_globals.h (mc_bldc_globals.h)*". The initial values of the variables are configured through the "*mc_foc_globals.c (mc_bldc_globals.c)*".

The "*mc_hwio.c*" is used to initialize MCU peripherals.

Figure 3. Structure of motor control library files

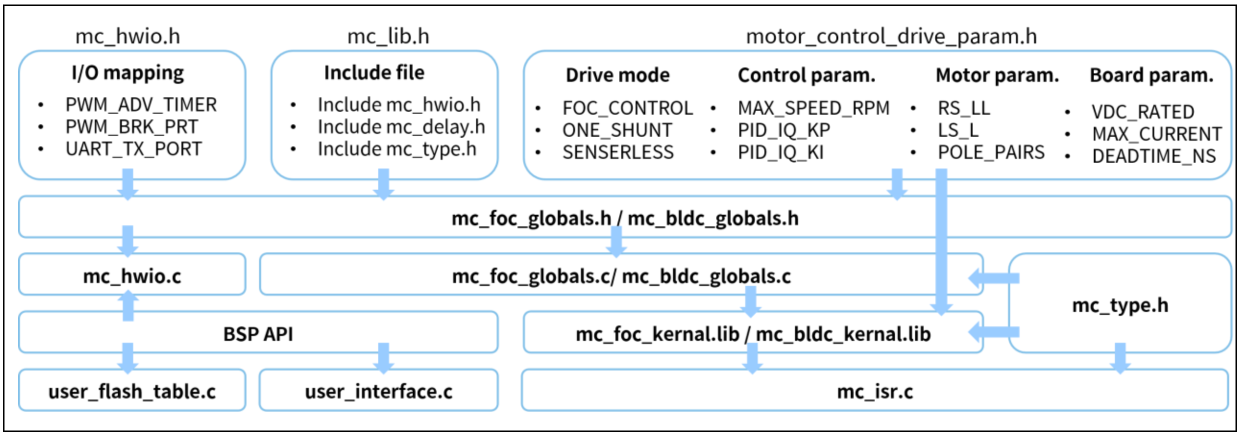


Table 2 presents a summary of motor control library files.

Table 2. Summary of motor control library files

File name	Description
FOC/BLDC general files	
mc_lib.h	Header file management
motor_control_drive_param.h	Set motor drive mode (current sampling mode, sensored mode,ect.), control parameters, motor parameters and board parameters
mc_hwio.c	Hardware peripheral configuration
mc_hwio.h	Hardware IO macro definitions
mc_isr.c	Motor control interrupt functions
mc_type.h	Global variables type and definition, enumeration definition
mc_delay.c	Delay functions
mc_delay.h	Declaration of delay functions
mc_comm_uart.c	Communication peripheral configuration
mc_comm_uart.h	Declaration and configuration of UART functions
mc_pid_control.c	PID controller functions
mc_pid_control.h	Declaration of PID controller functions
mc_curr_fdbk.c	Current sensing functions
mc_curr_fdbk.h	Declaration of current sensing functions
mc_math.c	Filter functions
mc_math.h	Declaration of filter functions
mc_hall.c	Hall sensor functions
mc_hall.h	Declaration of Hall sensor functions
FOC files	
mc_foc_kernal.lib	Motor control library core functions
mc_foc_kernal.h	Declaration of motor control library core functions
motor_control_foc.c	Motor control functions
motor_control_foc.h	Declaration of motor control functions
mc_foc.c	Vector control functions
mc_foc.h	Declaration of vector control functions
mc_encoder.c	Encoder functions
mc_encoder.h	Declaration of encoder functions
mc_field_weakening.c	Field weakening functions
mc_field_weakening.h	Declaration of field weakening functions
mc_foc_sensorless.c	Sensorless functions
mc_foc_sensorless.h	Declaration of sensorless functions
mc_foc_globals.c	Global variables definition and default value, global function declaration

mc_foc_globals.h	Global variables, global function declaration, macro definitions
user_interface_foc.c	Communication interface functions
user_interface_foc.h	Declaration of communication interface functions
mc_flash_data_table_foc.c	Write Flash data table
mc_flash_data_table_foc.h	Write Flash data table configuration
BLDC files	
mc_bldc_kernal.lib	Motor control library functions
motor_control_bldc.c	Motor control functions
motor_control_bldc.h	Declaration of motor control functions
mc_bldc.c	6-step square wave functions
mc_bldc.h	Declaration of 6-step square wave functions
mc_bldc_sensorless.c	Sensorless functions
mc_bldc_sensorless.h	Declaration of sensorless functions
mc_bldc_globals.c	Global variables definition and default value, global function definition
mc_bldc_globals.h	Global variables, global function declaration, macro definitions
user_interface_bldc.c	Communication interface functions
user_interface_bldc.h	Declaration of communication interface functions
mc_flash_data_table_bldc.c	Write Flash data table
mc_flash_data_table_bldc.h	Write Flash data table configuration

1) **motor_control_drive_param.h**

- This file is divided into four parts, which are used to configure drive mode, control parameters, motor parameters and board parameters.
- Table 3 presents the definition of drive modes. Users can configure the desired drive mode according to their hardware and motor configuration. For current sampling method, it is possible to select 3-shunt, 2-shunt or 1-shunt current sensing mode. For the positioning sensor, there are photoelectric incremental encoder, Hall sensor or sensorless available for selection. In addition, the field weakening control is optional according to users' needs.

Table 3. Drive mode definitions

Definition	Description
FOC_CONTROL	FOC vector control mode
SIX_STEP_CONTROL	Six-step square wave control mode
THREE_SHUNT	Three-shunt current sampling
TWO_SHUNT	Two-shunt current sampling
ONE_SHUNT	Single-shunt current sampling
COMPLEMENT	Enable complementality between low-side MOS transistor switch and upper-side PWM (see Figure 6)
GATE_DRIVER_LOW_SIDE_INVERT	Enable low-side MOS inverted output (see Figure 6)
EMF_COMPENSATE	EMF voltage compensate
INCREM_ENCODER	Photoelectric incremental encoder
ABZ	AB signal calibration with zero position (photoelectric incremental encoder)
AB	AB signal calibration without zero position (photoelectric incremental encoder)
M_METHOD	M method for speed measurement (photoelectric incremental encoder)
MT_METHOD	M/T method for speed measurement (photoelectric incremental encoder)
HALL_SENSORS	Hall sensor (general-purpose)
LOW_SPEED_CONTROL	Low-speed voltage control (6-step square wave control with hall sensor)
WITHOUT_CURRENT_CTRL	Voltage control without current loop
PHASE_ADVANCE	Phase advance (six-step square wave sensorless control)
FIELD_WEAKENING	Field weakening control
SENSORLESS	Sensorless control mode (general-purpose)
OPENLOOP_STARTUP	Open loop startup in sensorless control mode (general-purpose)
ALIGN_AND_GO_STARTUP	Align and go startup in sensorless control mode (general-purpose)
INIT_ANGLE_STARTUP	Initial angle detection startup in sensorless control mode (general purpose)
VOLT_SENSE	Voltage sensing in sensorless vector control mode
CONST_CURRENT_START	Constant current startup (6-step square wave sensorless control)
CONST_VOLTAGE_START	Constant voltage startup (6-step square wave sensorless control)
BLDC_SENSORLESS_ADC	BEMF detection with ADC in six-step square wave sensorless control mode
BLDC_SENSORLESS_COMP	BEMF detection with comparator in six-step square wave sensorless control mode

EMF_CONTINOUS_SAMPLE	Continous samping mode for BEMF zero-crossing detection (six-step square wave sensorless comparator mode)
CURRENT_LP_FILTER	Get d/q axis current low-pass filter signals (vector control)
INTERNAL_CLOCK_SOURCE	Use MCU internal oscillator as a clock source
E_BIKE_SCOOTER	E-bike/scooter mode
DC_CURRENT_LIMIT	DC current limit (for E_BIKE_SCOOTER use only)
RDS_AUTO_CALIBRATION	MOS's $R_{DS(on)}$ auto calibration (for E_BIKE_SCOOTER use only)

- Motor control parameters are presented in Table 4. It is possible to define corresponding parameters according to the hardware used, motor requirments and control characteristics. Debugging is also supported, such as, current d-axis, q-axis PI controller, speed PI controller.

Table 4. Control parameter definitions

Definition	**Description**
	BLDC/FOC general definitions
PWM_FREQ	PWM output frequency (unit: Hz)
MOTOR_CONTROL_MODE	Motor control mode (speed control, torque control, etc.; see *motor_control_mode* in *type.h* for details)
CTRL_SOURCE	Configure command control source (external source control/software control)
UI_UART_BAUDRATE	Baud rate of UI communication serial interface
UI_SAMPLE_CYCLE	Sampling frequency (unit: PWM counter timing base)
TUNE_TARGET_CURRENT	Tune target current value of PI parameters (unit: ampere)
TUNE_CURRENT_TOTAL_PERIOD	Tune total current pulse period of PI parameters (unit: ms)
TUNE_CURRENT_STEP_PERIOD	Tune current pulse step period of PI parameters (unit: ms)
SPEED_LOOP_FREQ	Speed loop frequency (unit: Hz)
MIN_SPEED_RPM	Minimum motor speed (unit: rpm)
MAX_SPEED_RPM	Maximum motor speed (unit: rpm)
MIN_CONTROL_SPEED	Minimum speed loop control speed (unit: rpm)
ACC_SPD_SLOPE	Slope of acceleration (unit: rpm/ms, when speed loop control frequency=1kHz)
DEC_SPD_SLOPE	Slope of deceleration (unit: rpm/ms, when speed loop control frequency=1kHz)
SP_MAX_VOLT	Maximum voltage of external command source (unit: voltage)
SP_THRESHOLD	Threshold voltage of external command source (unit: voltage)
SP_RUN_VALUE	Minimum voltage to start running in external source mode (unit: voltage)
SP_STOP_VALUE	Maximum voltage to stop running in external source mode (unit: voltage)
PID_SPD_KP_DIV_DIV	Speed control proportional gain divisor (Q16 mode)
PID_SPD_KI_DIV_DIV	Speed control integral gain divisor (Q16 mode)
PID_SPD_KP_DEFUALT	Speed control proportional gain (Q15 mode)
PID_SPD_KI_DEFUALT	Speed control integral gain (Q15 mode)
	FOC definitions

Definition	Description
STABLE_SPEED_RPM	Stable motor speed (unit: rpm) (for hall sensor use only)
SLICK_SPEED_RPM	Motor slick speed (unit: rpm) (for hall sensor use only)
POSITION_LOOP_FREQ	Position loop control frequency (unit: Hz)
MAX_POSITION_ANGLE	Maximum position angle (unit: Degree)
MIN_POSITION_ANGLE	Minimum position angle (unit: Degree)
CMD_TO_VAL_GAP	Define the counter gap value between rotor position and its target to use PID_POS_KI_DEFUALT_STABLE
PID_POS_KP_DEFUALT	Rotor position control proportional gain (Q15 mode)
PID_POS_KI_DEFUALT	Rotor position control integral gain (Q15 mode)
PID_POS_KI_DEFUALT_STABLE	Rotor position control integral gain (Q15 mode) (rotor position close to target position)
PID_POS_KD_DEFUALT	Rotor position control derivative gain (Q15 mode)
PID_POS_KP_GAIN_DIV	Rotor position control proportional gain divisor (Q16 mode)
PID_POS_KI_GAIN_DIV	Rotor position control integral gain divisor (Q16 mode)
PID_POS_KD_GAIN_DIV	Rotor position control derivative gain divisor (Q16 mode)
PID_ID_KP_DEFUALT	d-axis current control proportional gain (Q15 mode)
PID_ID_KI_DEFUALT	d-axis current control integral gain (Q15 mode)
PID_ID_KP_GAIN_DIV	d-axis current control proportional gain divisor (Q16 mode)
PID_ID_KI_GAIN_DIV	d-axis current control integral gain divisor (Q16 mode)
PID_IQ_KP_DEFUALT	q-axis current control proportional gain (Q15 mode)
PID_IQ_KI_DEFUALT	q-axis current control integral gain (Q15 mode)
PID_IQ_KP_GAIN_DIV	q-axis current control proportional gain divisor (Q16 mode)
PID_IQ_KI_GAIN_DIV	q-axis current control integral gain divisor (Q16 mode)
OLC_ANGLE_INC	Open loop angle incremental at each PWM output frequency
OLC_VOLT	Open loop voltage output (Q15 mode)
ENC_VOLT	Encoder aligned voltage (Q15 mode)
FW_MAX_ID_CURR	Maximum d-axis current in field weakening control (unit: ampere)
FW_VOLTAGE_REF	Maximum voltage before enabling field weakening (unit: 0.1%)
FW_KP_GAIN	Field weakening control proportional gain (Q15 mode)
FW_KI_GAIN	Field weakening control integral gain (Q15 mode)
FW_KP_GAIN_DIV	Field weakening control proportional gain divisor (Q16 mode)
FW_KI_GAIN_DIV	Field weakening control integral gain divisor (Q16 mode)
CURR_BANDWIDTH	d/q-axis current low-pass filter band width (unit: Hz)
OBS_SPD_BANDWIDTH	Speed low-pass filter band width of observer (unit: Hz)
OBS_GAIN1	Observer gain factor 1 (Q15 mode)
OBS_GAIN2	Observer gain factor 2 (Q15 mode)
PLL_KP_GAIN	PLL control proportional gain (Q15 mode)

Definition	Description
PLL_KI_GAIN	PLL control integral gain (Q15 mode)
PLL_KP_GAIN_DIV	PLL control proportional gain divisor (Q16 mode)
PLL_KI_GAIN_DIV	PLL control integral gain divisor (Q16 mode)
STARTUP_MAX_SPD	Open loop maximum speed before entering closed loop (unit: RPM)
STARTUP_CURRENT	The initial current command after entering closed loop (unit: ampere)
STARTUP_OL_VOLT	Open loop startup voltage (Q15 mode)
STARTUP_OL_SLOPE	Open loop startup acceleration (unit: rpm/s)
STARTUP_ALIGN_VOLT	Rotor alignment voltage before startup (Q15 mode)
STARTUP_ALIGN_TIME	Rotor alignment time before startup (unit: ms)
STARTUP_START_TIME	Startup time after rotor alignment (unit: ms)
DETECT_PULSE_WIDTH	Voltage pulse width for rotor initial angle detection (unit: us)
REVERSE_MAX_SPEED_RPM	Maximum reverse speed (unit: rpm) (for E_BIKE_SCOOTER use only)
REVERSE_CURRENT	Reverse current command (unit: ampere) (for E_BIKE_SCOOTER use only)
BRAKING_CURRENT	Brake current command (unit: ampere) (for E_BIKE_SCOOTER use only)
BLDC definitions	
I_SAMPLE_CHANGE_DUTY	Change PWM duty value of current sampling point (unit: PWM timer time base)
I_SAMPLE_MIN_DUTY	Minimum PWM DUTY value to sample current (unit: PWM timer time base)
SENSE_HALL_TIMES	Set phase-change times before open loop entesclosed loop (unit: times) (for BLDC sensorless only)
REBOOT_PERIOD_MS	Set a restart time when a startup failed (unit: ms) (for BLDC sensorless only)
SPEED_FILTER_TIMES	Speed moving average times (unit: times)
INIT_SPD_COUNT	Initial speed count value (for sensorless BLDC only)
START_CURRENT	Current value for constant current startup (unit: ampere) (for BLDC sensorless only)
START_VOLTAGE	Voltage value for constant voltage startup (unit: V)
START_PERIOD	Constant current/voltage startup duration (unit: usec)
OLC_INIT_SPD	Open loop initial speed (unit: rpm)
OLC_FINAL_SPD	Open loop final speed (unit: rpm)
OLC_TIMES	Set the times from initial speed to rise up to final speed (unit: times)
OLC_INIT_VOLT	Open loop initial voltage (unit: V)
OLC_VOLT_INC	Open loop incremental voltage (unit: V)
OLC_STARTUP_PERIOD	Set the period from open loop startup to closed loop (for BLDC sensorless use only)
LOCK_VOLT	Rotor alignment voltage before startup (Q15 mode) (for BLDC sensorless only)
LOCK_PERIOD	Rotor alignment time before startup (unit: ms) (for BLDC sensorless only)
PID_IS_KP_DIV_LOG	Bus current control proportional gain divisor (Q16 mode)
PID_IS_KI_DIV_LOG	Bus current control integral gain divisor (Q16 mode)

Definition	Description
PID_IS_KP_DEFUALT	Bus current control proportional gain (Q15 mode)
PID_IS_KI_DEFUALT	Bus current control integral gain (Q15 mode)
EMF_CHANGE_PERCENT_H	Duty value for BEMF zero-crossing detection to switch from low speed mode to high speed mode (unit: PWM timing base) (for sensorless BLDC only)
EMF_CHANGE_PERCENT_L	Duty value for BEMF zero-crossing detection to switch from high speed mode to low speed mode (unit: PWM timing base) (for sensorless BLDC only)
EMF_LOW_SPD_SAMPLE_POINT	BEMF zero-crossing sampling point DUTY value in low speed range (unit: PWM timer timing base) (for sensorless BLDC only)
EMF_HIGH_SPD_SAMPLE_DELAY	BEMF zero-crossing sampling point DUTY value in high speed range (unit: PWM timer timing base) (for sensorless BLDC only)
EMF_SAMPLE_INTERVAL	Sampling interval in continuous sampling mode (unit: PWM timer timing base) (for sensorless BLDC only)
EMF_LOW_SPD_CONT_SAMPLE_END	Low-speed sampling end point in continuous sampling mode (unit: PWM timer timing base) (for sensorless BLDC only)
EMF_HIGH_SPD_CONT_SAMPLE_DELAY	High-speed sampling end point in continuous sampling mode (unit: PWM timer timing base) (for sensorless BLDC only)
EMF_PHASE_ADV_SPD	Phase advance maximum speed (unit: rpm)
EMF_MIN_DELAY	Phase advance minimum delay (unit: usec)
EMF_AVOID_NOISE_INIT_PERIOD	First delay time for detecting BEMF zero-crossing point(unit: ms)
EMF_AVOID_NOISE_TIMES	Delay time for BEMF zero-crossing to avoid phase change noise (unit: PWM timer timing base)
EMF_LOW_SPD_OFFSET_RISING	BEMF zero-crossing point offset at rising edge in low speed range (unit: voltage digital conversion value) (for ADC detecting sensorless BLDC BEMF)
EMF_LOW_SPD_OFFSET_FALLING	BEMF zero-crossing point offset at falling edge in low speed range(unit: voltage digital conversion value) (for ADC detecting sensorless BLDC BEMF)
EMF_HIGH_SPD_OFFSET_RISING	BEMF zero-crossing point offset at rising edge in high speed range (unit: voltage digital conversion value) (for ADC detecting sensorless BLDC BEMF)
EMF_HIGH_SPD_OFFSET_FALLING	BEMF zero-crossing point offset at falling edge in high speed range(unit: voltage digital conversion value) (for ADC detecting sensorless BLDC BEMF)
MAX_VOLT_CMD	Maximum voltage command in low-speed voltage mode (Q15 mode) (for BLDC Hall sensor only)
MAX_V_CONTROL_SPD	Maximum speed shifting from low-speed voltage control to current control (unti: rpm) (for BLDC Hall sensor only)
MIN_I_CONTROL_SPD	Minimum speed shifting from current control to low-speed voltage control(unti: rpm) (for BLDC Hall sensor only)

- Table 5 presents the definitions relating to driver functions.

Table 5. Driver parameter definitions

Definition	Description
VDC_RATED	DC-BUS voltage
V_SENSE_GAIN	Voltage feedback ratio
ADC_REFERENCE_VOLT	ADC reference voltage (unit: voltage)
ADC_DIGITAL_SCALE_12BITS	ADC resolution
SYSTEM_CORE_CLOCK	System frequency (unit: Hz)
TMR_CLK	Timer clock frequency (unit: Hz)
CHANGE_PHASE_TMR_DIV	Phase change clock frequency division (for six-step square wave sensorless mode)
DEADTIME_CLK_SFT_BITS	Dead time frequency divison shift bits
DEADTIME_NS	Dead time (unit: ns)
MIN_INTERVAL_TIME	Mininum interval between two-phase signals when PWM shift (unit: ns)
MAX_CURRENT	Maximum peak phase current (unit: ampere)
MIN_CURRENT	Minimum peak phase current (unit: ampere)
DC_MAX_CURRENT	Maximum DC-BUS current (unit: ampere) (for DC_CURRENT_LIMIT use only)
CURRENT_SPAN_SHIFT	Number of shifts for current per-unit normalization
R_SHUNT	Shunt resistance (unit: Ω)
OP_GAIN	Current amplifier gain
CURR_OFFSET_VOLT	Zero current offset (unit: voltage)
RDC_SHUNT	DC shunt resistance (unit: Ω) (for DC_CURRENT_LIMIT only)
DC_OP_GAIN	DC current amplifier gain (for DC_CURRENT_LIMIT only)
IDC_OFFSET_VOLT	zero DC current offset (unit: voltage) (for DC_CURRENT_LIMIT only)
EMF_SENSE_GAIN	BEMF feedback ratio
OVER_CURRENT_VREF	Overcurrent threshold point (unit: voltage)
OVER_VOLT_THRESHOLD	Over-voltage threshold point (unit: voltage)
UNDER_VOLT_THRESHOLD	Under-voltage threshold point (unit: voltage)
V0_V	Parameter V0 of the approximate curve of NTC temperature and voltage relationship (note [2])
T0_C	Parameter T0 of the approximate curve of NTC temperature and voltage relationship (note [2])
dV_dT	Parameter dV/dT of the approximate curve of NTC temperature and voltage relationship (note [2])
OVER_TEMP_THRESHOLD	Over-temperature threshold point (unit: Celsius degrees)
MC_ERROR_MASK	Error detection mask

Note [2]: Approximate curve equation for voltage-temperature relation is V[V]=V0+dV/dT[V/Celsius](T-T0)[Celsius].*

- Table 6 presents the definitions relating to motor parameters.

Table 6. Motor parameter macro definitions

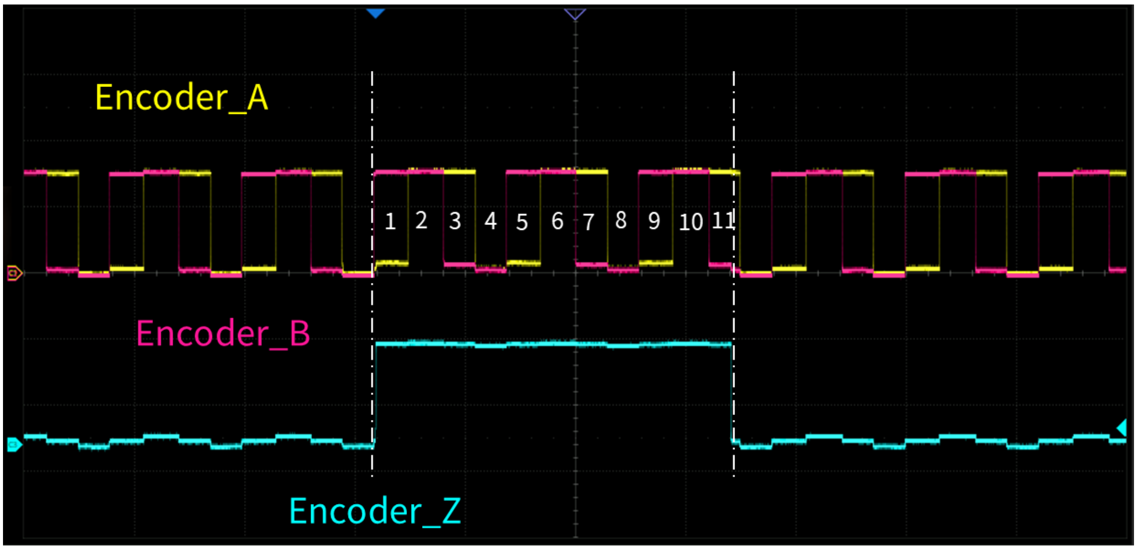
Definition	Description
BLDC/FOC general parameters	
RS_LL	Motor line-to-line winding resistance (unit: Ω)
LS_LL	Motor line-to-line winding inductance (unit: mH)
POLE_PAIRS	Number of pole-pairs
FOC parameters	
NOMINAL_CURRENT	Nominal current (unit: ampere)
ENCODER_PPR	Pulses per revolution of encoder (unit: pulse per revolution)
ENC_IDX_COUNT	Encoder index signal width (uint: count) (note [3])
ENC_STALL_TIME	Encoder stall time (unit: ms)
HALL_STATE_ONE_NEXT	The next state of HALL state 1 (forward) (see note 4)
HALL_STATE_TWO_NEXT	The next state of HALL state 2 (forward) (see note 4)
HALL_STATE_THREE_NEXT	The next state of HALL state 3 (forward) (see note 4)
HALL_STATE_FOUR_NEXT	The next state of HALL state 4 (forward) (see note 4)
HALL_STATE_FIVE_NEXT	The next state of HALL state 5 (forward) (see note 4)
HALL_STATE_SIX_NEXT	The next state of HALL state 6 (forward) (see note 4)
HALL_STATE_ONE_ANGLE	Electrical angle in HALL state 1 (see note 4)
HALL_STATE_TWO_ANGLE	Electrical angle in HALL state 2 (see note 4)
HALL_STATE_THREE_ANGLE	Electrical angle in HALL state 3 (see note 4)
HALL_STATE_FOUR_ANGLE	Electrical angle in HALL state 4 (see note 4)
HALL_STATE_FIVE_ANGLE	Electrical angle in HALL state 5 (see note 4)
HALL_STATE_SIX_ANGLE	Electrical angle in HALL state 6 (see note 4)
BLDC parameters	
ANGLE_INIT_DETECT_DUTY	PWM DUTY value detected at an initial angle (unit: PWM timing base)
KE	Motor KE value
OUTPUT_AH_BL_HALL_STATE	Output Hall state of A-shunt high-side PWM and B-shunt low-side ON (see note 5)
OUTPUT_AH_CL_HALL_STATE	Output Hall state of A-shunt high-side PWM and C-shunt low-side ON (see note 5)
OUTPUT_BH_CL_HALL_STATE	Output Hall state of B-shunt high-side PWM and C-shunt low-side ON (see note 5)
OUTPUT_BH_AL_HALL_STATE	Output Hall state of B-shunt high-side PWM and A-shunt low-side ON (see note 5)
OUTPUT_CH_AL_HALL_STATE	Output Hall state of C-shunt high-side PWM and A-shunt low-side ON (see note 5)
OUTPUT_CH_BL_HALL_STATE	Output Hall state of C-shunt high-side PWM and B-shunt low-side ON (see

Definition	Description
	note 5)
HALL_STATE_ONE_NEXT_CW	The next state of HALL state 1 (forward)
HALL_STATE_TWO_NEXT_CW	The next state of HALL state 2 (forward)
HALL_STATE_THREE_NEXT_CW	The next state of HALL state 3 (forward)
HALL_STATE_FOUR_NEXT_CW	The next state of HALL state 4 (forward)
HALL_STATE_FIVE_NEXT_CW	The next state of HALL state 5 (forward)
HALL_STATE_SIX_NEXT_CW	The next state of HALL state 6 (forward)
HALL_STATE_ONE_NEXT_CCW	The next state of HALL state 1 (reverse)
HALL_STATE_TWO_NEXT_CCW	The next state of HALL state 2 (reverse)
HALL_STATE_THREE_NEXT_CCW	The next state of HALL state 3 (reverse)
HALL_STATE_FOUR_NEXT_CCW	The next state of HALL state 4 (reverse)
HALL_STATE_FIVE_NEXT_CCW	The next state of HALL state 5 (reverse)
HALL_STATE_SIX_NEXT_CCW	The next state of HALL state 6 (reverse)

Note [3]: This is applicable to ABZ mode of photoelectric incremental encoder.

Figure 4 presents the relationship diagram of JK42BLS01-X056ED encoder with ABZ signals. If the width gap between Z signal's rising edge and its falling edge equals to 11 counts of AB signals, the ENC_IDX_COUNT is set to 11. (Usually photoelectric encoder has one or two counnts).

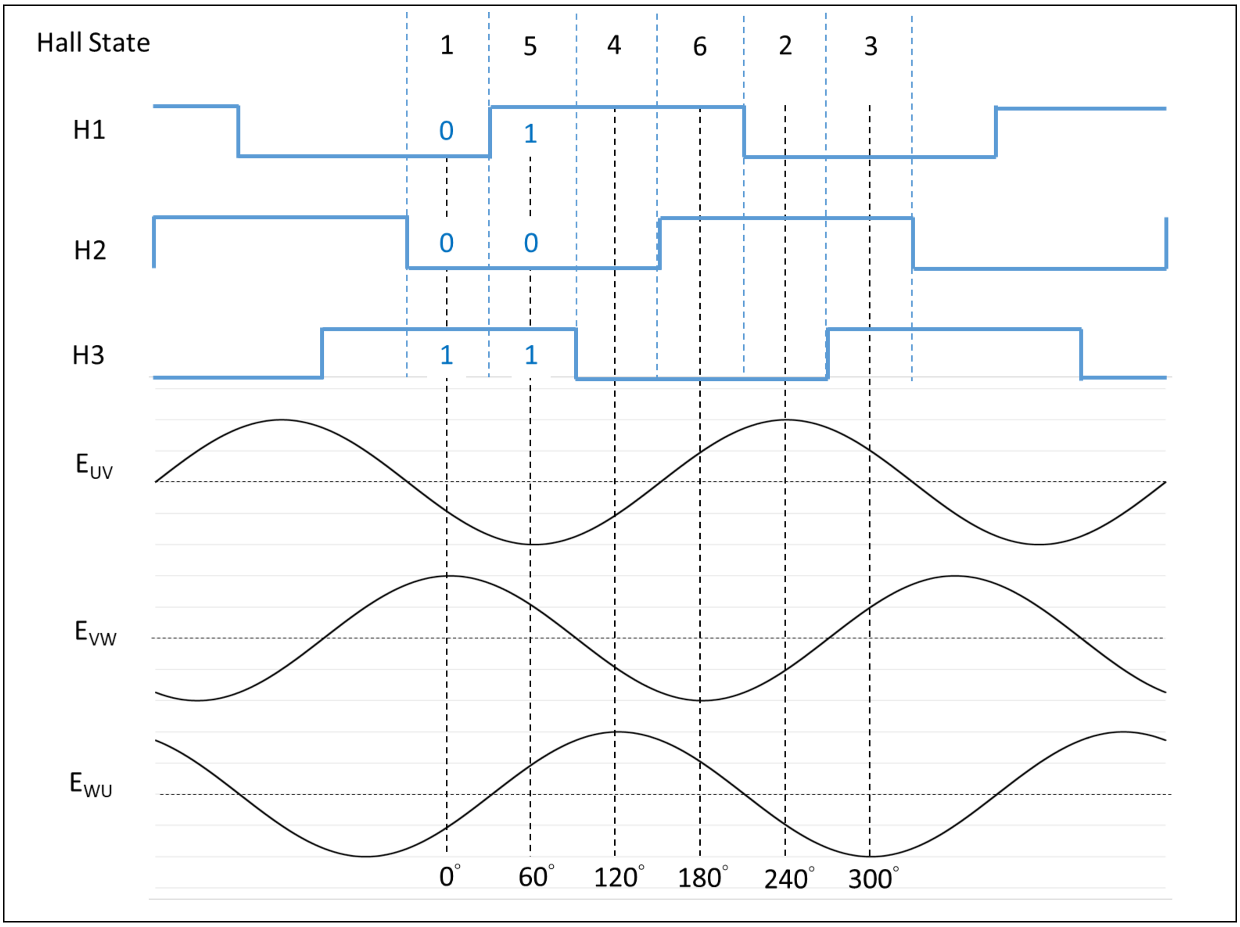
Figure 4. Encoder ABZ singal relationship



Note [4]: Hall state sequence and corresponding electrical angle can be configured according to BEMF and hall state.

Figure 5 presents the relationship diagram between BLDC (JK42BLS01-X056ED) BEMF, Hall state and electrical angle. The line-to-line BEMF for three-phase motor should be in consistent with this diagram. The zero degree of angle corresponds to the maximum BEMF between VW lines, while 60-degree angle corresponds to the minimum BEMF between UV lines, and so on. Different hall state values indicate different electrical angles. For example, hall state 5 indicates a 60-degree angle, which is written in the HALL_STATE_FIVE_ANGLE. Additionally, there is a need to define the hall state during motor runtime. For example, if the next state of hall state 1 is 5, this is written in the HALL_STATE_ONE_NEXT.

Figure 5. Relationship diagram between PMSM BEMF, Hall state and electrical angle



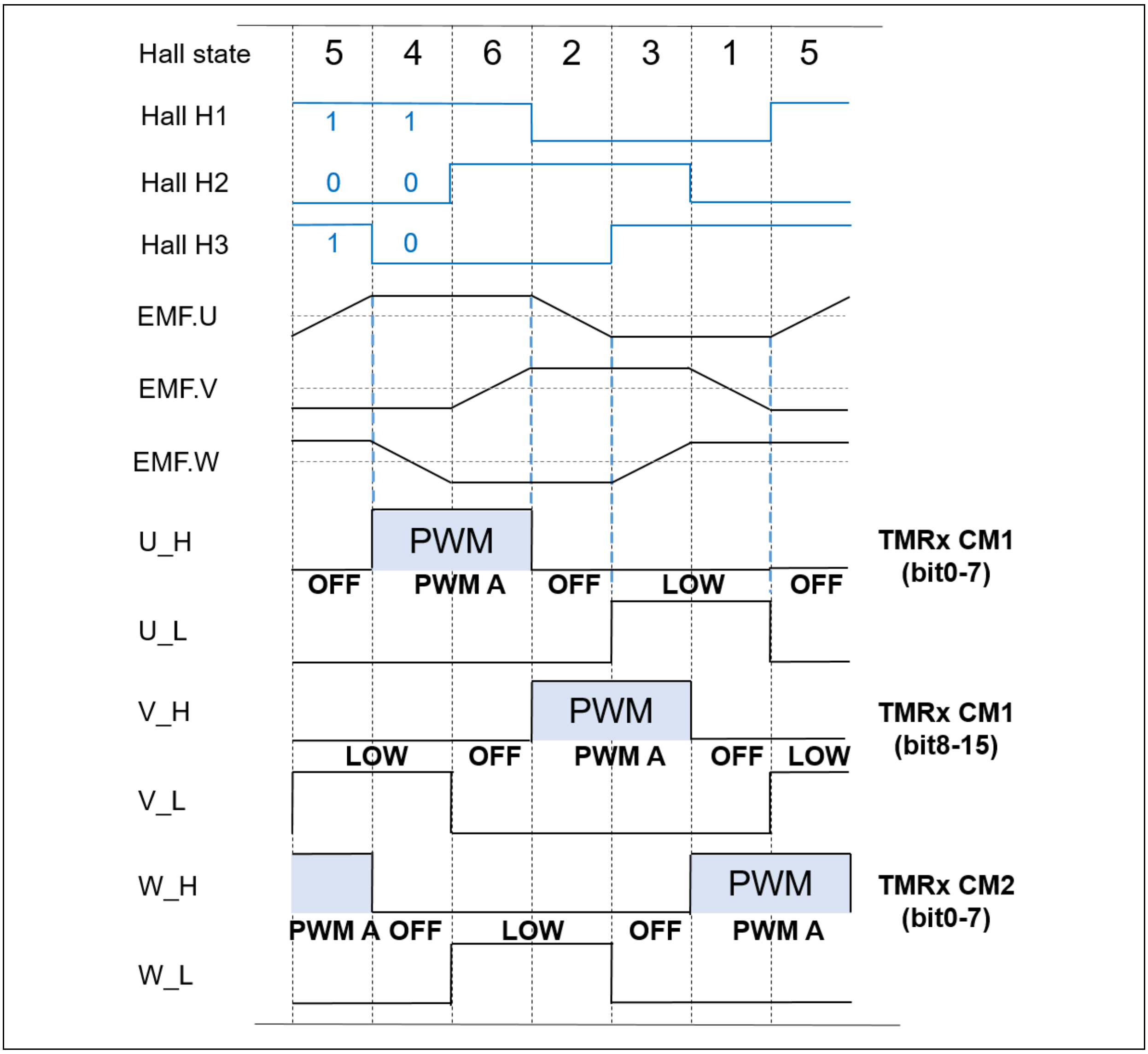
Note [5]: It is possible to set different MOS high-side and low-side ON/OFF state according to BEMF and hall state.

Figure 6 presents the relationship diagram between BLDC BEMF, hall state and MOS on/off state.

Taking Figure 6 as an example, a maximum A-phase BEMF combined with a minimum B-phase BEMF corresponds to hall state 4, so the OUTPUT_AH_BL_HALL_STATE is written with 4. Similarily, when the A-phase BEMF is at the highest level while the C-phase BEMF is at the lowest level, hall state is 6, so the OUTPUT_AH_BL_HALL_STATE is written with 6. In this way, a correct PWM waveform can be output by defining these 6 hall states.

When the driver uses a gate driver whose low-side input signals are without reverse feature, and a low-side PWM outputs (complementary mode is disabled) while a high-side PWM outputs, the corresponding PWM output is shown in Figure 6. If it uses the gate driver whose low-side input signals are with reverse type, and a low-side PWM outputs (complementary mode is enabled) while a high-side PWM outputs, the motor_control_drive_param.h can be used to configure the desired output mode according to users' needs. In our three examples of six-step square wave control mode, they use a mode in which the low-side is with reverse output and complementary feature is enabled.

Figure 6. Relationship diagram between BLDC BEMF, Hall state and MOS on/off



2) mc_hwio.h header file

- This file is used to configure macro definitions according to the user hardware IOs and peripherals. It also includes the declaration of the mc_hwio.c file functions.

3) mc_hwio.c file

- This file is used to configure peripherals such as TMR, ADC, DMA and GPIO, according to user hardware. In it also includes some basic control functions such as button, LED. See Table 7 for details.

Table 7. Peripheral configuration functions

Function	Description
nvic_config	Configure interrupt priority
tmr_pwm_init	Configure PWM output-related timer, crm clock, GPIO and DMA
gpio_hall_init	Configure Hall sensor GPIOs (for BLDC use only)
tmr_hall_init	Configure Hall sensor timer and crm clocks (for BLDC use only)
tmr_sensorless_change_phase_init	Configure phase change timer and crm clock in sensorless mode (for sensorless BLDC)
hall_timer_init	Configure Hall sensor timer, crm clock and GPIO (for FOC use ony)
encoder_time_init	Configure incremental photoelectric encoder timer, crm clock, GPIO and EXINT (for FOC only)
encoder_capture_timer_init	Photoelectric Encoder capture timer, crm clock, GPIO (for FOC/MT_METHOD only)
adc_ordinary_config	Configure ADC, DMA and GPIO of ADC ordinary channels
adc_preempt_config	Configure ADC, DMA and GPIO of ADC preempted channels
speed_timer_init	Configure speed control clock (tmr) and crm clock
uart_init	Configure UART-related crm clock, GPIO and UART
button_switch_init	Switch button GPIO and crm clock configuration (for BLDC hall sensor only)
button_exint_init	Button interrupt event EXINT configuration
led_config	LED GPIO and crm clock initialization configuration
led_on	LED ON
led_off	LED OFF
led_toggle	LED toggle (from on to off, or from off to on)
led_init	LED initialization configuration
led_blink	LED flashes
mode_switch_init	Switch button configuration
current_offset_tmr_setting	Current offset timer configuration
bldc_angle_init_config	TMR configuration of initial angle detection (for sensorless BLDC only)
tmr_sensorless_change_phase_init	TMR configuration for sensorless phase change (for sensorless BLDC only)
tmr_read_emf_init	TMR configuration for sensorless zero-crossing detection in continuous sampling mode (for sensorless BLDC only)
bldc_sensorless_detectEMF_config	TMR, ADC, and DMA configuration for sensorless zero-crossing detection (for sensorless BLDC only)
foc_angle_init_config	TMR and ADC configuration for initial angle detection (FOC sensorless only)

4) mc_isr.c file

- This file contains interrupt functions, as shown in Table 8.

Table 8. Motor control interrupt functions

Function	Description
ADVTMR_PWM_CYCLE_IRQ	Control loop interrupt (PWM update interrupt)
ADVTMR_PWM_BRK_IRQ	Brake input interrupt (PWM output disable)
ADC_SHUNT_SAMP_READY_IRQ	Current/BEMF sensing complete interrupt
ENCODER_CAPTURE_IRQ	Encoder capture interrupt
EXINT_ENCODER_IDX_IRQ	Encoder zero position interrupt
HALL_CAPTURE_IRQ	Hall signal input capture interrupt
SPEED_LOOP_TIMER_IRQ	Speed loop interrupt (for FOC use only)
SysTick_Handler	System interrupt (1ms), state machine program
BUTTON_EXINT_IRQHandler	Button interrupt
CHANGE_PHASE_IRQ	Sensorless BLDC phase change interrupt
READ_EMF_IRQ	Sensorless zero-crossing dection interrupt in continuous sampling mode (For BLDC use only)

5) mc_type.h file

- This file holds global enumeration/type definitions, as shown in Table 9.

Table 9. List of motor control library enumeration

Enumeration/Type	Description
BLDC/FOC general parameters	
firmware_id_type	Firmware ID type used to identify motor control mode
motor_control_mode	Motor control mode (open loop control, voltage control, Id tune mode, Iq tune mode, speed control, torque control, position control, encoder alignment mode, etc.)
ctrl_source_type	Control source (software control, external circuit control)
encoder_align_type	Encoder alignment variables
esc_state_type	Motor control process state machine (see Section 5.1 for details)
err_code_type	Motor error type (over-voltage, under-voltage, over-temperature, over-current, encoder error, Hall error, startup error)
shunt_nbr_type	Shunt current detection mode (1-shunt, 2-shunt or 3-shunt)
curr_offs_type	ADC current sampling offset value
current_type	Current variables
pid_ctrl_type	PID control loop variables
pid_ctrl_dc_type	PID control loop related structure variables
speed_type	Speed variables
value_type	Value type
hall_sensor_type	Hall sensor variables

ramp_cmd_type	Ramp command type
usart_data_index	UART queue variables
moving_average_type	Moving average variables
usart_config_type	UART peripheral configuration type
ui_wave_param_type	UI wave display parameters
FOC parameters	
qd_type	d,q type
abc_type	a,b,c type
alphabeta_type	α,β type
i_dc_type	Bus current type
trig_components_type	Trigonometric function type
voltage_type	Voltage type
adc_trigger_type	ADC trigger type
pwm_duty_type	PWM duty type
rotor_angle_type	Rotor angle type
encoder_type	Encoder type
open_loop_type	Open loop type
position_type	Position type
angle_type	Angle type
field_weakening_type	Field weakening type
lowpass_filter_type	Low-pass filter type
motor_volt_type	Voltage output type
motor_emf_type	BEMF detection type
state_observer_type	Sensorless observer type
sensorless_startup_type	Sensorless startup type
foc_angle_init_type	Initial angle detection type
rds_cali_type	MOS's $R_{DS(on)}$ calibration type
BLDC parameters	
angle_init_type	Initial angle detection type (for sensorless BLDC only)
start_state_type	State machine for initial angle detection (for sensorless BLDC only))
init_current_type	current type for Initial angle detection (for sensorless BLDC only)
angle_init_type	Initial angle detection type (for sensorless BLDC only)
i_bus_type	BUS current type
olc_type	Open loop type
emf_sample_type	BEMF sampling type (for sensorless BLDC only)
adc_sample_type	ADC sampling type (for sensorless BLDC only)

4 Motor control library functions

There are a six-step square wave motor control library (mc_bldc_kernal.lib) and a vector control motor control library (mc_foc_kernal.lib) available for uses to choose from according to their needs. All these two motor control libraries contain sensored and sensorless control functions. Yet it is worth noting that there is a need to perform software initialization settings by calling initlization functions before the use of motor control library. How to use motor control functions is detailed in the subsequent sections.

4.1 General-purpose motor control library functions

Initialization functions (they must be configured prior to motor control library use)

- **get_fw_id**

Table 10. get_fw_id

Name	Description
Function name	get_fw_id
Function prototype	firmware_id_type get_fw_id(void);
Function description	Get control mode ID (see note 6)
Input parameter	NA
Output parameter	NA
Return value	Control mode ID value
Required preconditions	NA
Called functions	NA

Example:

```
firmware_id = get_fw_id();
```

Note [6]: Control mode includes: sine-wave FOC control, six-step square wave control, sensorless, HALL sensor and encoder.

- **mc_param_init**

Table 11. mc_param_init

Name	Description
Function name	mc_param_init
Function prototype	flag_status mc_param_init(uint8_t fw_id);
Function description	Set initial parameters for motor control library
Input parameter	fw_id: control mode ID
Output parameter	NA
Return value	Set (successful) or Reset (failed)
Required preconditions	Get the fw_id by executing the get_fw_id
Called functions	NA

Example:

```
param_initial_rdy = mc_param_init(firmware_id);
```

PID controller functions

- **pid_controller**

Table 12. pid_controller

Name	Description
Function name	pid_controller
Function prototype	int16_t pid_controller(pid_ctrl_type *pid_handler, int32_t var_err);
Function description	PID controller (proportional-integral-derivative controller), comprising Proportional unit, Integral unit and Derivative unit
Input parameter 1	pid_handler: point to the structure pid_ctrl_type
Input parameter 2	var_err: error value
Output parameter	NA
Return value	PID controller outputs value
Required preconditions	NA
Called functions	NA

Example:

```
speed_ramp.cmd_final = pid_controller(&pid_pos, pos_err_temp);
```

Mathematical operation functions

- **atan2_fixed**

Table 13. atan2_fixed

Name	Description
Function name	atan2_fixed
Function prototype	int16_t atan2_fixed(int32_t y, int32_t x)
Function description	Fixed point arctan function
Input parameter 1	Y: y-axis signal
Input parameter 2	X: x-axis signal
Output parameter	NA
Return value	Signals after going through arctan function
Required preconditions	NA
Called functions	NA

Example:

```
local_degree = atan2_fixed(local_I.beta, local_I.alpha);
```

4.2 Motor control library functions in vector control mode

Current sampling functions

- current_read_foc_1shunt

Table 14. current_read_foc_1shunt

Name	Description
Function name	current_read_foc_1shunt
Function prototype	void current_read_foc_1shunt(current_type *curr_handler, voltage_type *volt_handler);
Function description	Read bus current value in vector control mode to re-build a 3-shunt current
Input parameter 1	curr_handler: point to the bus current offset value in the structure current_type
Input parameter 2	volt_handler: point to the voltage control area in the voltage_type
Output parameter	curr_handler: point to the 3-shunt current value in the structure current_type
Return value	NA
Required preconditions	NA
Called functions	adc_preempt_conversion_data_get

Example:

```
current_read_foc_1shunt(&current, &volt_cmd);
```

- rds_auto_calibration

Table 15. rds_auto_calibration

Name	Description
Function name	rds_auto_calibration
Function prototype	void rds_auto_calibration(rds_cali_type *rds_cali_handler, current_type *curr_handler, voltage_type *volt_handler);
Function description	MOS's $R_{DS(on)}$ auto calibration in E_BIKE_SCOOTER mode
Input parameter 1	rds_cali_handler: point to the d/q-axis current filter value in the structure rds_cali_type
Input parameter 2	curr_handler: point to the bus current filter value in the structure current_type
Input parameter 3	volt_handler: point to the d/a-axis voltage in the structure voltage_type
Output parameter	rds_cali_handler: point to the MOS's $R_{DS(on)}$ auto calibration in the structure rds_cali_type
Return value	NA
Required preconditions	NA
Called functions	lowpass_filtering

Example:

```
rds_auto_calibration(&Rds_Cali, &current, &volt_cmd);
```

Encoder functions

- enc_speed_get_MTmethod

Table 16. enc_speed_get_MTmethod

Name	Description
Function name	enc_speed_get_MTmethod
Function prototype	int32_t enc_speed_get_MTmethod(encoder_type *enc_handler, speed_type *spd_handler);
Function description	Get rotor speed from photoelectric encoder (M/T Method)
Input parameter 1	enc_handler: point to the parameters of the structure encoder_type
Input parameter 2	spd_handler: point to the paratmers of the structure speed_type
Output parameter	NA
Return value	Rotor speed
Required preconditions	NA
Called functions	NA

Example:

```
rotor_speed_encoder.val_temp = enc_speed_get_MTmethod(&encoder, &rotor_speed_encoder);
```

- position_cmd_ramp

Table 17. position_cmd_ramp

Name	Description
Function name	position_cmd_ramp
Function prototype	void position_cmd_ramp(position_type *pos_handler, ramp_cmd_type *cmd_ramp_handler, pid_ctrl_type *pid_handler);
Function description	Step position command is converted into S-curve command
Input parameter 1	pos_handler: point to the parameters of the structure position_type
Input parameter 2	cmd_ramp_handler: point to the parameters of the structure ramp_cmd_type
Input parameter 3	pid_handler: point to the parameters of the structure pid_ctrl_type
Output parameter	NA
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
position_cmd_ramp(&pos, &speed_ramp, &pid_pos);
```

Hall sensor functions

- hall_rotor_angle_get

Table 18. hall_rotor_angle_get

Name	Description
Function name	hall_rotor_angle_get
Function prototype	int16_t hall_rotor_angle_get(hall_sensor_type *hall_handler, rotor_angle_type *rotor_angle_handler);
Function description	Read estimated rotor angle from hall sensor
Input parameter 1	hall_handler: point to the rotor angle variable in the structure hall_sensor_type
Input parameter 2	rotor_angle_handler: point to the rotor angle in the structure rotor_angle_type
Output parameter	NA
Return value	Motor rotor angle
Required preconditions	NA
Called functions	NA

Example:

```
rotor_angle_hall.elec_angle_val = hall_rotor_angle_get(&hall, &rotor_angle_hall);
```

- hall_delta_theta_calculation

Table 19. hall_delta_theta_calculation

Name	Description
Function name	hall_delta_theta_calculation
Function prototype	int16_t hall_delta_theta_calculation(speed_type *rotor_speed_handler, rotor_angle_type *rotor_angle_handler, hall_sensor_type *hall_handler);
Function description	Get Hall senor rotor angle variation
Input parameter 1	hall_handler: point to HALL state in the structure hall_sensor_type
Input parameter 2	rotor_angle_handler: point to the rotor angle in the structure rotor_angle_type
Input parameter 3	rotor_speed_handler: pointer to the rotor speed in the structure speed_type
Output parameter	NA
Return value	Rotor angle variation
Required preconditions	NA
Called functions	NA

Example:

```
hall.theta_inc = hall_delta_theta_calculation(&rotor_speed_hall, &rotor_angle_hall, &hall);
```

PWM functions

- **svpwm_3shunt**

Table 20. svpwm_3shunt

Name	Description
Function name	svpwm_3shunt
Function prototype	pwm_duty_type svpwm_3shunt(voltage_type *volt_handler, pwm_duty_type *pwm_duty_handler);
Function description	Space vector pulse width modulation function (for 3-shunt use)
Input parameter 1	volt_handler: point to the parameters of the structure voltage_type
Input parameter 2	pwm_duty_handler: point to the parameters of the structure pwm_duty_type
Output parameter	NA
Return value	pwm_duty_type: 3-phase PWM duty value
Required preconditions	NA
Called functions	NA

Example:

```
pwm_duty = svpwm_3shunt(&volt_cmd, &pwm_duty);
```

- **svpwm_2shunt**

Table 21. svpwm_2shunt

Name	Description
Function name	svpwm_2shunt
Function prototype	pwm_duty_type svpwm_2shunt(voltage_type *volt_handler, pwm_duty_type *pwm_duty_handler);
Function description	Space vector pulse width modulation function (for 2-shunt use)
Input parameter 1	volt_handler: point to the parameters of the structure voltage_type
Input parameter 2	pwm_duty_handler: point to the parameters of the structure pwm_duty_type
Output parameter	NA
Return value	pwm_duty_type: 3-phase PWM duty value
Required preconditions	NA
Called functions	NA

Example:

```
pwm_duty = svpwm_2shunt(&volt_cmd, &pwm_duty);
```

- **svpwm_1shunt**

Table 22. svpwm_1shunt

Name	Description
Function name	svpwm_1shunt
Function prototype	pwm_duty_type svpwm_1shunt(voltage_type *volt_handler, pwm_duty_type *pwm_duty_handler);
Function description	Space vector pulse width modulation function (for 1-shunt use)
Input parameter 1	volt_handler: point to the parameters of the structure voltage_type
Input parameter 2	pwm_duty_handler: point to the parameters of the structure pwm_duty_type
Output parameter	NA
Return value	pwm_duty_type: 3-phase PWM duty value
Required preconditions	NA
Called functions	pwm_shift

Example:

```
pwm_duty = svpwm_1shunt(&volt_cmd, &pwm_duty);
```

- **pwm_shift**

Table 23. pwm_shift

Name	Description
Function name	pwm_shift
Function prototype	adc_trigger_type pwm_shift(int16_t *dTa, int16_t *dTb, int16_t *dTc, int16_t Ta, int16_t Tb, int16_t Tc, pwm_duty_type *pwm_duty_handler);
Function description	Pulse width modulation shift control (for 1-shunt use)
Input parameter 1	Ta: max. 3-phase duty cycle
Input parameter 2	Tb: second-largest 3-phase duty cycle
Input parameter 3	Tc: min. 3-phase duty cycle
Input parameter 4	pwm_duty_handler: pointer to the parameters of the structure pwm_duty_type
Output parameter 1	dTa: offset value for the maximum 3-phase duty cycle
Output parameter 2	dTb: offset value for the second-largest 3-phase duty cycle
Output parameter 3	dTc: offset value for the mininum 3-phase duty cycle
Return value	adc_trigger_type: current sampling time points
Required preconditions	NA
Called functions	NA

Example:

```
adc_trig = pwm_shift(&dTa, &dTb, &dTc, Ta, Tb, Tc, &local_pwm_duty);
```

- **foc_circle_limitation**

Table 24. foc_circle_limitation

Name	Description
Function name	foc_circle_limitation
Function prototype	void foc_circle_limitation(voltage_type *volt_handler);
Function description	Synthetic vector voltage maximum output limitation
Input parameter	volt_handler: pointer to the parameters of the structure voltage_type parameters
Output parameter	volt_handler: pointer to the Q-axis voltage of the strucutre voltage_type
Return value	NA
Required preconditions	NA
Called functions	arm_sqrt_q15

Example:

```
foc_circle_limitation(&volt_cmd);
```

- foc_vq_limitation

Table 25. foc_vq_limitation

Name	Description
Function name	foc_vq_limitation
Function prototype	void foc_vq_limitation(voltage_type *volt_handler, pid_ctrl_type *pid_handler);
Function description	Vector voltage VQ maximum output limitation
Input parameter	volt_handler: pointer to the parameters of the structure voltage_type
Output parameter	pid_handler: pointer to the parameters of the structure pid_ctrl_type
Return value	NA
Required preconditions	NA
Called functions	arm_sqrt_q15

Example:

```
foc_vq_limitation(&volt_cmd, &pid_iq);
```

Sensorless vector control functions

- startup_openloop

Table 26. startup_openloop

Name	Description
Function name	startup_openloop
Function prototype	flag_status startup_openloop(sensorless_startup_type *startup_handler, voltage_type *volt_handler);
Function description	Open loop startup function
Input parameter 1	volt_handler: pointer to the parameters of the structure voltage_type parameters
Input parameter 2	startup_handler: pointer to the parameters of the structure sensorless_startup_type parameters
Output parameter	NA
Return value	flag_status: return SET (successful) or RESET (failed)
Required preconditions	NA
Called functions	NA

Example:

```
startup.closeloop_rdy = startup_openloop(&startup, &volt_cmd);
```

- startup_alpha_axis

Table 27. startup_alpha_axis

Name	Description
Function name	startup_alpha_axis
Function prototype	flag_status startup_alpha_axis(sensorless_startup_type *startup_handler, voltage_type *volt_handler);
Function description	Align and go startup function
Input parameter 1	volt_handler: pointer to the parameters of the structure voltage_type parameters
Input parameter 2	startup_handler: pointer to the parameters of the structure sensorless_startup_type parameters
Output parameter	NA
Return value	flag_status: return SET (successful) or RESET (failed)
Required preconditions	NA
Called functions	NA

Example:

```
startup.closeloop_rdy = startup_alpha_axis(&startup, &volt_cmd);
```

- flag_status startup_angle_init

Table 28. flag_status startup_angle_init

Name	Description
Function name	flag_status startup_angle_init
Function prototype	flag_status startup_angle_init(sensorless_startup_type *startup_handler, voltage_type *volt_handler);
Function description	Align and go startup function with initial angle
Input parameter 1	volt_handler: pointer to the parameters of the structure voltage_type parameters
Input parameter 2	startup_handler: pointer to the parameters of the structure sensorless_startup_type parameters
Output parameter	NA
Return value	flag_status: return SET (successful) or RESET (failed)
Required preconditions	NA
Called functions	NA

Example:

```
startup.closeloop_rdy = startup_angle_init(&startup, &volt_cmd);
```

- flag_status startup_angle_init2

Table 29. flag_status startup_angle_init2

Name	Description
Function name	flag_status startup_angle_init2
Function prototype	flag_status startup_angle_init2(sensorless_startup_type *startup_handler, voltage_type *volt_handler);
Function description	Open loop startup function with initial angle
Input parameter 1	volt_handler: pointer to the parameters of the structure voltage_type parameters
Input parameter 2	startup_handler: pointer to the parameters of the structure sensorless_startup_type parameters
Output parameter	NA
Return value	flag_status: return SET (successful) or RESET (failed)
Required preconditions	NA
Called functions	NA

Example:

```
startup.closeloop_rdy = startup_angle_init2(&startup, &volt_cmd);
```

- foc_sensorless_angle_init

Table 30. foc_sensorless_angle_init

Name	Description
Function name	foc_sensorless_angle_init
Function prototype	void foc_sensorless_angle_init(foc_angle_init_type *angle_detect_handler, current_type *curr_handler);
Function description	Initial angle detection function
Input parameter 1	angle_detect_handler: pointer to the parameters of the structure foc_angle_init_type parameters
Input parameter 2	curr_handler: pointer to the parameters of the structure current_type parameters
Output parameter	angle_detect_handler: pointer to the motor initial angel of the structure foc_angle_init_type
Return value	NA
Required preconditions	NA
Called functions 1	tmr_output_enable
Called functions 2	tmr_channel_value_set
Called functions 3	tmr_counter_enable

Example:

```
foc_sensorless_angle_init(&angle_detector, &current);
```

- **current_angle_init_3shunt**

Table 31. current_angle_init_3shunt

Name	Description
Function name	current_angle_init_3shunt
Function prototype	void current_angle_init_3shunt(foc_angle_init_type *angle_detect_handler, current_type *curr_handler);
Function description	3-shunt current sampling function (for motor initial angle detection)
Input parameter 1	angle_detect_handler: pointer to the parameters of the structure foc_angle_init_type parameters
Input parameter 2	curr_handler: pointer to the parameters of the structure current_type parameters
Output parameter	angle_detect_handler: pointer to the current value of the structure foc_angle_init_type
Return value	NA
Required preconditions	NA
Called functions	adc_preempt_conversion_data_get

Example:

```
current_angle_init_3shunt(&angle_detector, &current);
```

- **current_angle_init_2_1shunt**

Table 32. current_angle_init_2_1shunt

Name	Description
Function name	current_angle_init_2_1shunt
Function prototype	void current_angle_init_2_1shunt(foc_angle_init_type *angle_detect_handler, current_type *curr_handler);
Function description	Bus current sampling function (for motor initial angle detection)
Input parameter 1	angle_detect_handler: pointer to the parameters of the structure foc_angle_init_type parameters
Input parameter 2	curr_handler: pointer to the parameters of the structure current_type parameters
Output parameter	angle_detect_handler pointer to the current value of the structure foc_angle_init_type
Return value	NA
Required preconditions	NA
Called functions	adc_preempt_conversion_data_get

Example:

```
current_angle_init_2_1shunt(&angle_detector, &current);
```

- **obs_pll_execute**

Table 33. obs_pll_execute

Name	Description
Function name	obs_pll_execute
Function prototype	int16_t obs_pll_execute(state_observer_type *state_obs_handler, int16_t hBemf_alfa_est, int16_t hBemf_beta_est);
Function description	Quadrature-component-based PLL function
Input parameter 1	state_obs_handler: pointer to the parameters of the structure state_observer_type parameters
Input parameter 2	hBemf_alfa_est: the estimated BEMF α-axis voltage
Input parameter 3	hBemf_beta_est: the estimated BEMF β-axis voltage
Output parameter	NA
Return value	Rotor speed
Required preconditions	NA
Called functions 1	arm_sin_q15
Called functions 2	arm_cos_q15
Called functions 3	pi_controller

Example:

```
hRotor_Speed = obs_pll_execute(state_obs_handler, state_obs_handler->hBemf_alpha_est,
state_obs_handler->hBemf_beta_est);
```

- **rotor_angle_sensorless**

Table 34. rotor_angle_sensorless

Name	Description
Function name	rotor_angle_sensorless
Function prototype	int16_t rotor_angle_sensorless(state_observer_type *state_obs_handler, motor_volt_type *motor_volt_handler);
Function description	Rotor angle observer
Input parameter 1	state_obs_handler: pointer to the parameters of the structure state_observer_type parameters
Input parameter 2	motor_volt_handler: pointer to the parameters of the structure motor_volt_type parameters
Output parameter	NA
Return value	Rotor angle
Required preconditions	Execturing motor_volt_calc or motor_volt_read returns driver output α-axis and β-axis voltage
Called functions	obs_pll_execute

Example:

```
state_observer.elec_angle = rotor_angle_sensorless(&state_observer, &motor_voltage);
```

4.3 Motor control library functions in 6-step square wave mode

6-step square wave functions

- **calc_adc_sample_point**

Table 35. calc_adc_sample_point

Name	Description
Function name	calc_adc_sample_point
Function prototype	void calc_adc_sample_point (adc_sample_type *adc_sample,int16_t pwm_duty);
Function description	ADC sample point calculation function (including current sample point and BEMF sample point, see note [7])
Input parameter 1	adc_sample: point to the parameters of the structure adc_sample_type
Input parameter 2	pwm_duty: current PWM duty cycle
Output parameter	adc_sample: point to the adc_sample_point of the structure adc_sample_type
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
adc_sample_point_set(&adc_sample, pwm_comp_value);
```

Note [7]: BEMF sample point is only applicable to sensorless mode. In sensored mode, only current sample point is calculated.

6-step square wave sensorless functions

- **bldc_sensorless_angle_init**

Table 36. bldc_sensorless_angle_init

Name	Description
Function name	bldc_sensorless_angle_init
Function prototype	void bldc_sensorless_angle_init(angle_init_type *angle_init);
Function description	Bus current sampling function (for motor initial angle detection)
Input parameter	angle_init: point to the parameters of the structure angle_init_type parameters
Output parameter	angle_init: point to the current of the structure angle_init_type
Return value	NA
Required preconditions	NA
Called functions 1	tmr_channel_value_set
Called functions 2	disable_mosfet
Called functions 3	tmr_output_enable
Called functions 4	tmr_counter_enable

Example:

```
bldc_sensorless_angle_init(PWM_ADVANCE_TIMER,&angle_init,angle_init_step);
```

- **angle_init_estimation**

Table 37. angle_init_estimation

Name	Description
Function name	angle_init_estimation
Function prototype	uint8_t angle_init_estimation(angle_init_type *angle_init);
Function description	Initial angle estimation
Input parameter 1	angle_init: pointer to the parameters of the structure angle_init_type
Input parameter 2	NA
Output parameter	angle_init: point to the parameters of the structure angle_init_type
Return value	Return six-step square wave control phase (from 1 to 6) corresponding to the detected initial angle
Required preconditions	bldc_sensorless_angle_init
Called functions	NA

Example:

```
hall.state = angle_init_estimation(&angle_init);
```

- **change_phase_period_ramp**

Table 38. change_phase_period_ramp

Name	Description
Function name	change_phase_period_ramp
Function prototype	void change_phase_period_ramp(adc_sample_type *adc_sample);
Function description	Delay phase change period ramp increase/decrease after zero-crossing detection
Input parameter 1	adc_sample: point to the parameters of the structure adc_sample_type
Input parameter 2	NA
Output parameter	adc_sample: point to the phase change command of the structure adc_sample_type
Return value	NA
Required preconditions	NA
Called functions	NA

Example:

```
change_phase_period_ramp(adc_sample);
```

5 Motor control library application example structure

5.1 State machine overview

The state machine flow chart is shown in Figure 7. It includes such state as Idle, Safety ready, Angle init, Starting, Running, Free run, I_tune, Enc_align and Error.

5.1.1 State descriptions

Idle*

This refers to the initial state of the state machine. In this mode, the motor remains in static state. The state machine returns to “Idle” state when an error condition is resolved or motor stops running.

Safety ready*

This state indicates that the motor can be started safely for all parameters have been set and the current offset obtained during the“Idle” state period.

Angle init

For a sensorless motor, the state of an initial angle is detected prior to startup. The motor runs in “Starting” state as soon as the initial angle is obtained. This state is unique to the sensorless mode. It can be skipped if there is no need of motor initial angle detection.

Starting

After the initial angle is detected in sensorless control mode, it is possible to configure motor drive mode, peripheral parameters, among others. In this state, it is also possible to set the condition for entering closed loop control mode to enhance system robustness.

Running*

This means motor is running in this mode. With the UI interface, users are able to adjust the corresponding parameters such as target speed and current, or send a command to halt the motor.

Free run

This mode is smiliar to “Stop” mode. In this mode, the driver stops output, decreasing motor speed to zero. This state is maintained until a complete stop of the motor. After a full stop, the motor then returns to “safety ready” mode

I_tune*

This mode is used to fine-tune current PID controller parameters. In this mode, it is possible to set appropriate target current, KP and KI of current loop, to generate a step current, via UI interface.

Enc_align

This represents the zero-poistion calibration state of an encoder. With the UI interface, users are able to enable zero-position calibration feature. Typically, after a driver reset, it is necessary to activate zero-position calibration. If no such action implemented, an auto zero-position calibration will be triggered before motor startup. This state is skipped for a motor without encoder.

Error*

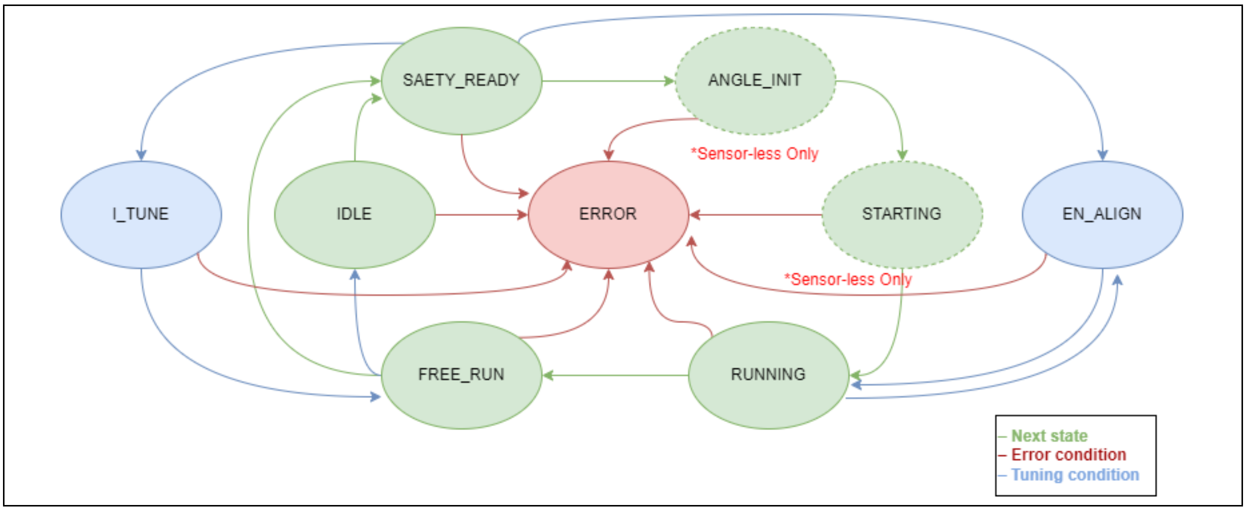
Jump to this state in case of any error.

Note [8]: The states marked with “” are executed continuously until other event (user operation or any fault) is generated.*

5.1.2 State machine procedure

The state machine flowchart is presented in Figure 7. “Green” circles represents a state machine switch process in normal conditions, while “Blue” circles presents a switch process under special circumstances. For example, users can switch to I_TUNE state to adjust the current PID controller parameters, and switch to ENC_ALIGN state to calibrate encoder before startup.The “Red” one indicates the system error. The state machine jumps to the ERROR state if any error (such as over-voltage, over-current, etc.) occurs during program running.

Figure 7. State machine process flow



6 Revision history

Table 39. Document revision history

Date	Version	Revision note
2022.11.18	2.0.0	Initial release.
2022.11.18	2.0.1	Updated some descriptions and terms
2022.11.24	2.0.2	Updated some descriptions and terms
2022.12.01	2.0.3	Updated code definitions and descriptions
2022.12.10	2.0.4	Updated motor library codes
2022.12.23	2.0.5	Revised document formats.
2022.12.29	2.0.6	Revised text errors
2023.03.02	2.0.7	Added keil V5.33 compiling limitation, software requirements, united function description formats, code descriptions.
2023.04.20	2.0.8	Added macro definition of positioning control, interrupt functions, control functions and relevant descriptions.
2023.10.05	2.1.0	Updated the contents relating to motor control program architecture, functions and files.