AN0064
Application Note

AT32 Motor Control Library User Guide

Introduction

This application note introduces how to use AT32 motor control library and details the hardware and software requirements, motor control library architecture, header file declaration and settings, individual functions and the program structure of practical application examples.

Applicable products:

Part number	AT32F4xx, AT32L0xx

Contents

List of tables

List of figures

1 Motor control library algorithm

The motor control library algorithm mainly contains:

Target motor type: Three-phase permanent magnet synchronous motor (PMSM) / 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 modes:

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

Rotor position detection methods:

- Hall-effect position sensor
- Photoelectric incremental encoder

Rotor Initial position estimation methods:

- Three-phase voltage vectors mode
- Two-phase voltage vectors mode

Estimation of rotor position in FOC driving mode:

- BEMF estimation with Luenberger observer

Estimation of rotor position in 120° square wave control mode:

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

Available methods in sensored FOC control mode:

- Voltage vector control
- Torque control (current vector control)

- Speed control
- Field weakening control
- Positioning control
- Regenerative braking control

Available methods in sensorless FOC control mode:

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

Available methods in sensored 120° square wave commutation mode:

- 120° square wave voltage control
- Torque control (120° square wave current control)
- Speed control
- Field weakening control
- Regenerative braking control

Available methods in sensorless 120° square wave commutation mode:

- 120° square wave voltage control
- Torque control (120° square wave current control)
- Rotation speed control
- Field weakening control
- Regenerative braking control

2 Environmental requirements

2.1 Hardware environment

A PMSM (BLDC) motor, AT-Link or third-party debugger and a motor control board; if the AT-MOTOR-EVB evaluation board is used, please refer to the *AT32_LV_Motor_Control_EVB_User_Manual* for relevant hardware configurations.

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

2.2 Software environment

1. It is recommended to build an AT32 development environment according to the getting started guide of the corresponding AT32 MCU series and to configure the MCU enhanced functions.
2. After a new project is created, add the relevant header files and motor control library to the new project, and set the motor control mode, motor parameters, control board parameters, controller parameters and MCU peripheral parameters to the corresponding header files.
3. Users can write their own control program according to their application requirements, and call the motor library functions in the program to quickly complete the motor control project by using the optimized motor control library functions.

3 Motor control library documents

Figure 1 shows the relationship between the motor control library application functions and MCU basic function (BSP), UI communication program, user defined control function and custom functions in a motor control project. The motor control library API, user defined API and UI program are based on AT32 BSP functions, and the user control program is based on the motor control library API, user defined API and UI program. Therefore, users can easily call the motor control library API functions to control MCU peripheral so as to implement motor control program. In addition, it is possible to transmit motor control status or modify control parameters and commands in a real-time manner through the linkage between UI control program and external PC UI program.

Figure 1. Motor control program architecture

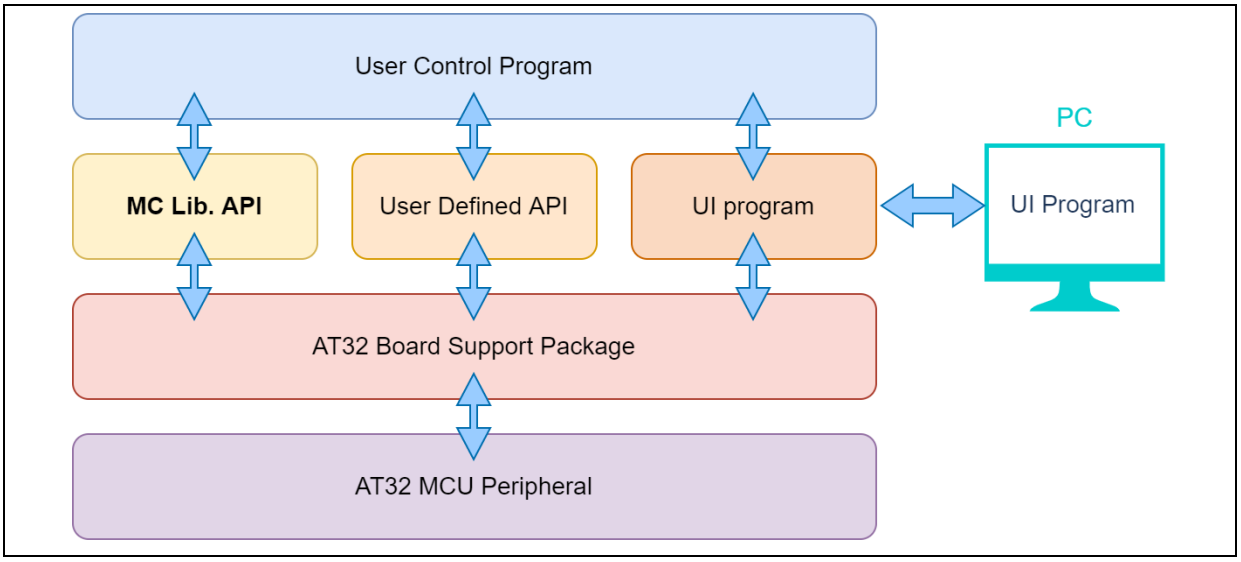


Figure 2 shows the structure of motor control library documents and their relationships. The four files, i.e., mc_lib.h, mc_motor_param.h, mc_drive_param.h and mc_ctrl_param.h, contain the motor control modes/methods, motor parameters, control board parameters and controller parameters to be input by users, and users can set MCU peripheral parameters in mc_hwio.h according to the connection relations between MCU peripherals and control board circuits. Relevant setting parameters are integrated in mc_foc_globals.h (mc_bldc_globals.h), and then their initial variables are set by the functions of mc_foc_globals.c (mc_bldc_globals.c) and used by motor control library functions. For the MCU peripherals initialization, it is executed in mc_hwoio.c.

Figure 2. Structure of motor control library documents

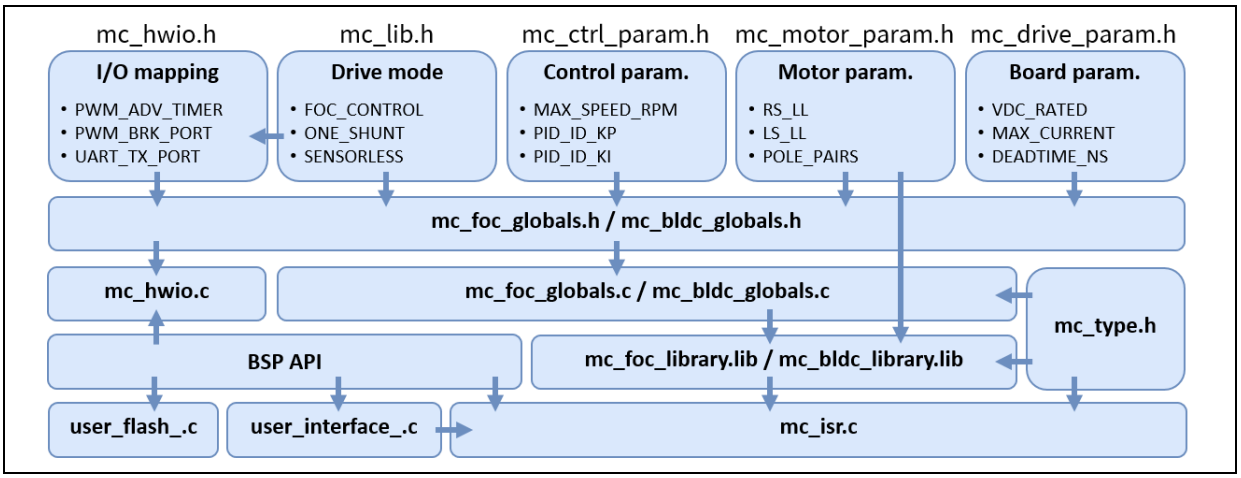


Table 1 contains description of all motor control library documents.

Table 1. Description of motor control library documents

File name	Description
FOC/BLDC common documents	
mc_lib.h	Unified header file management User defined motor drive modes (current sampling mode, sensor mode, etc.)
mc_ctrl_param.h	Control-related parameters
mc_drive_param.h	Drive-related parameters, such as maximum sensing voltage/current
mc_motor_param.h	Motor-related parameters, such as number of motor poles
mc_hwio.c	Hardware peripheral configuration
mc_hwio.h	Hardware peripheral mapping definitions
mc_isr.c	Relevant motor control interrupt functions
mc_flash_data_table.c	Writing parameters table into Flash
mc_flash_data_table.h	Relevant configuration for writing parameters table into Flash
mc_type.h	Global variables types and definitions, enumeration declaration
mc_delay.c	Delay-related functions
mc_delay.h	Declaration of delay-related functions
mc_comm_uart.c	Configuration of peripherals related to communication interface
mc_comm_uart.h	Declaration and configuration of communication UART related functions
FOC dedicated documents	
mc_foc_library.lib	FOC motor control functions library
mc_foc_globals.c	Global variables declaration and default values, global functions declaration
mc_foc_globals.h	Global variables, global functions declaration, macro definition
user_interface_foc.c	Functions related to communication interface
user_interface_foc.h	Declaration of functions related to communication interface
BLDC dedicated documents	
mc_bldc_library.lib	Six-step motor control functions library
mc_bldc_globals.c	Global variables declaration and default values, global functions declaration
mc_bldc_globals.h	Global variables, global functions declaration, macro definition
user_interface_bldc.c	Functions related to communication interface
user_interface_bldc.h	Declaration of functions related to communication interface

1) mc_lib.h header file

- Table 2 lists the definitions of drive modes that can be set according to the hardware and motor configuration. The current sampling mode is set according to hardware configuration (select one from 3-shunt, 2-shunt and 1-shunt current sampling modes), and the position sensor is configured according to the actual hardware (select the photoelectric incremental encoder or Hall sensor). In addition, the field weakening control is optional.

Table 2. Drive mode definitions

Definition item	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
INCREM_ENCODER	Photoelectric incremental encoder
WITH_INDEX	With zero position index (photoelectric incremental encoder)
WITHOUT_INDEX	Without zero position index (photoelectric incremental encoder)
HALL_SENSORS	Hall sensor
FIELD_WEAKENING	Field weakening control
SENSORLESS	Sensorless control
OPENLOOP_STARTUP	Open loop startup in sensorless FOC control mode
ALPHA_AXIS_STARTUP	alignment at α axis before startup in sensorless FOC control mode
INIT_ANGLE_STARTUP	Initial angle detection before startup in sensorless FOC control mode
BLDC_SENSORLESS_ADC	BEMF detection with ADC in sensorless six-step square wave control mode
BLDC_SENSORLESS_COMP	BEMF detection with comparator in sensorless six-step square wave control mode

2) mc_ctrl_param.h header file

- Motor control parameters are defined in this file. Table 3 lists the definitions of control parameters that can be defined according to the hardware, motor specifications and control-related characteristics. Some parameters can be set by the UI tuning, such as PI controllers of d-axis and q-axis currents, speed PI controller, etc.

Table 3. Control parameter definitions

Definition item	Description
BLDC/FOC common definitions	
MOTOR_CONTROL_MODE	Motor control mode (speed control, torque control, etc.; refer to motor_control_mode in type.h for details)
UI_UART_BAUDRATE	Baud rate of UI communication serial port

Definition item	Description
TUNE_TARGET_CURRENT	Target current value for current loop PI control parameters tuning (unit: ampere)
TUNE_CURRENT_TOTAL_PERIOD	Total period of current pulse for current loop PI control parameters tuning (unit: ms)
TUNE_CURRENT_STEP_PERIOD	Active period of current pulse for current loop PI control parameters tuning (unit: ms)
SPEED_LOOP_FREQ	Speed control loop frequency (unit: Hz)
MIN_SPEED_RPM	Minimum motor speed (unit: rpm)
MAX_SPEED_RPM	Maximum motor speed (unit: rpm)
MIN_CONTROL_SPEED	Minimum control speed (unit: rpm)
ACC_SPD_SLOPE	Slope of acceleration (unit: rpm/period of speed control loop)
DEC_SPD_SLOPE	Slope of deceleration (unit: rpm/period of speed control loop)
CTRL_SOURCE	Command source setting (external source/software control)
SP_MAX_VOLT	Maximum voltage of external command source (unit: voltage)
SP_THRESHOLD	Threshold voltage of external command source (unit: voltage)
SP_RUN_VALUE	Lowest voltage to startup in external source mode (unit: voltage)
SP_STOP_VALUE	Threshold voltage to stop in external source mode (unit: voltage)
PID_SPD_KP_DIV_LOG	Speed controller proportional gain divisor (Q16 mode)
PID_SPD_KI_DIV_LOG	Speed controller integral gain divisor (Q16 mode)
PID_SPD_KP_DEFUALT	Speed controller proportional gain (Q15 mode)
PID_SPD_KI_DEFUALT	Speed controller integral gain (Q15 mode)
OLC_ANGLE_INC	Angle increment per PWM switching period in open loop mode
OLC_VOLT	Open loop control output voltage (Q15 mode)
ENC_VOLT	Encoder alignment voltage (Q15 mode)
FOC dedicated definitions	
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_GAIN_DIV	d axis current control 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_GAIN_DIV	q axis current control gain divisor (Q16 mode)
FW_VOLTAGE_REF	Threshold 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_GAIN_DIV	Field weakening control gain divisor (Q16 mode)
CURR_BANDWIDTH	Current low-pass filter band width (unit: Hz)
OBS_SPD_BANDWIDTH	Speed low-pass filter band width of observer (unit: Hz)

Definition item	Description
OBS_GAIN1	Observer gain factor 1 (Q15 mode)
OBS_GAIN2	Observer gain factor 2 (Q15 mode)
PLL_KP_GAIN	Q-PLL control proportional gain (Q15 mode)
PLL_KI_GAIN	Q-PLL control integral gain (Q15 mode)
PLL_GAIN_DIV	Q-PLL control gain divisor (Q16 mode)
STARTUP_MAX_SPD	Open loop maximum speed before switching to closed loop (unit: rpm)
STARTUP_CURRENT	The initial current command after switching to sensorless closed loop (unit: ampere)
STARTUP_OL_VOLT	Startup voltage in open loop control (Q15 mode)
STARTUP_OL_SLOPE	Frequency acceleration in open loop control (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 (unit: ms)
DETECT_PULSE_WIDTH	Voltage pulse width for rotor initial angle detection (unit: us)
BLDC dedicated definition	
START_CURRENT	Current value for constant current startup (unit: ampere) (for BLDC sensorless only)
START_PERIOD	Running duration in constant current startup mode (unit: 1/PWM_FREQ sec) (for BLDC sensorless only)
I_SAMPLE_CHANGE_DUTY	Threshold of duty value for changing the current sampling point (unit: PWM timer time base)
I_SAMPLE_MIN_DUTY	The minimum PWM DUTY value for performing current sampling (unit: PWM timer time base)
SENSE_HALL_TIMES	The commutation times before switching to closed loop control from open loop control (unit: times) (for BLDC sensorless only)
EMF_CHK_TIMES	The number of consecutive judgments of the back EMF zero crossing point (unit: times) (for BLDC sensorless only)
REBOOT_PERIOD_MS	The restart time when the startup fails (unit: ms) (for BLDC sensorless only)
INIT_SPD_COUNT	Initial speed count value (for sensorless BLDC only)
PID_IS_KP_DIV_LOG	Current control proportional gain divisor (Q16 mode)
PID_IS_KI_DIV_LOG	Current control integral gain divisor (Q16 mode)
PID_IS_KP_DEFUALT	Current control proportional gain (Q15 mode)
PID_IS_KI_DEFUALT	Current control integral gain (Q15 mode)
EMF_CHANGE_PERCENT_H	Sensing mode switching point for the back-EMF zero-crossing detection timing from low speed mode changing to high speed mode (unit: PWM timer time base) (for sensorless BLDC only)

Definition item	Description
EMF_CHANGE_PERCENT_L	Sensing mode switching point for the back-EMF zero-crossing detection timing from low speed mode changing to high speed mode (unit: PWM timer time base) (for sensorless BLDC only)
EMF_SAMPLE_POINT_DUTY	Duty value of Back-EMF zero-crossing point detecting timing in high speed mode (unit: PWM timer time base) (only for BLDC sensorless)
SAMPLE_DELAY_DUTY	Duty value of Back-EMF zero-crossing point detecting timing in low speed mode (unit: PWM timer time base) (only for BLDC sensorless)
EMF_ADC_SAMPLE_DELAY	The interval between two consecutive samples in back EMF zero-crossing point detection (unit: PWM timer time base) (only for BLDC sensorless)

3) mc_drive_param.h header file

- This file mainly includes drive-related parameter definitions, such as dead time, shunt resistor and current amplifier gain. Table 4 lists definitions of these parameters.

Table 4. Driver parameter definitions

Definition item	Description
BLDC/FOC common parameters	
VDC_RATED	DC-BUS voltage
V_SENSE_GAIN	Divider ratio of voltage feedback
MAX_CURRENT	Maximum peak current of motor (unit: ampere)
MIN_CURRENT	Minimum peak current of motor (unit: ampere)
NOMINAL_CURRENT	Motor rated current (unit: ampere)
CURRENT_SPAN_SHIFT	Number of left shifts for current per-unit normalization
ADC_REFERENCE_VOLT	ADC reference voltage (unit: voltage)
ADC_DIGITAL_SCALE_12BITS	ADC resolution
SYSTEM_CORE_CLOCK	System frequency (unit: Hz)
TMR_CLK	Clock frequency (unit: Hz)
PWM_FREQ	PWM output frequency (unit: Hz)
DEADTIME_CLK_SFT_BITS	shift bit number of frequency divider for dead time generator
DEADTIME_NS	Dead time (unit: ns)
R_SHUNT	Shunt resistance (unit: Ω)
OP_GAIN	Current amplifier gain
CURR_OFFSET_VOLT	Zero current offset (unit: voltage)
EMF_SENSE_GAIN	Divider ratio of BEMF feedback
OVER_POWER_THRESHOLD	BUS overcurrent set point (unit: voltage)
OVER_CURRENT_VREF	Phase overcurrent set point (unit: voltage)
OVER_VOLT_THRESHOLD	Over-voltage set point (unit: voltage)
UNDER_VOLT_THRESHOLD	Under-voltage set point (unit: voltage)
V0_V	Parameter V0 of the temperature and voltage linear approximation function of NTC voltage divider circuit (note [1])
T0_C	Parameter T0 of the temperature and voltage linear approximation function of NTC voltage divider circuit (note [1])
dV_dT	Parameter dV/dT of the temperature and voltage linear approximation function of NTC voltage divider circuit (note [1])
OVER_TEMP_THRESHOLD	Over-temperature set point (unit: Celsius degrees)
MC_ERROR_MASK	Error detection mask
BLDC dedicated parameters	
EMF_LOW_SPD_OFFSET_RISING	Sensing offset of back EMF zero crossing point when sampling in low speed interval (rising edge)

EMF_LOW_SPD_OFFSET_FALLING	Sensing offset of back EMF zero crossing point when sampling in low speed interval (falling edge)
EMF_HIGH_SPD_OFFSET_RISING	Sensing offset of back EMF zero crossing point when sampling in high speed interval (rising edge)
EMF_HIGH_SPD_OFFSET_FALLING	Sensing offset of back EMF zero crossing point when sampling in high speed interval (falling edge)

Note [1]: Approximate curve equation of voltage-temperature relation is V[V]=V0+dV/dT[V/Celsius](T-T0)[Celsius].*

4) mc_motor_param.h header file

- This file mainly includes definitions of motor parameters, such as number of pole-pairs, encoder parameters and Hall sensor parameters. Refer to Table 5 for details.

Table 5. Motor parameter definitions

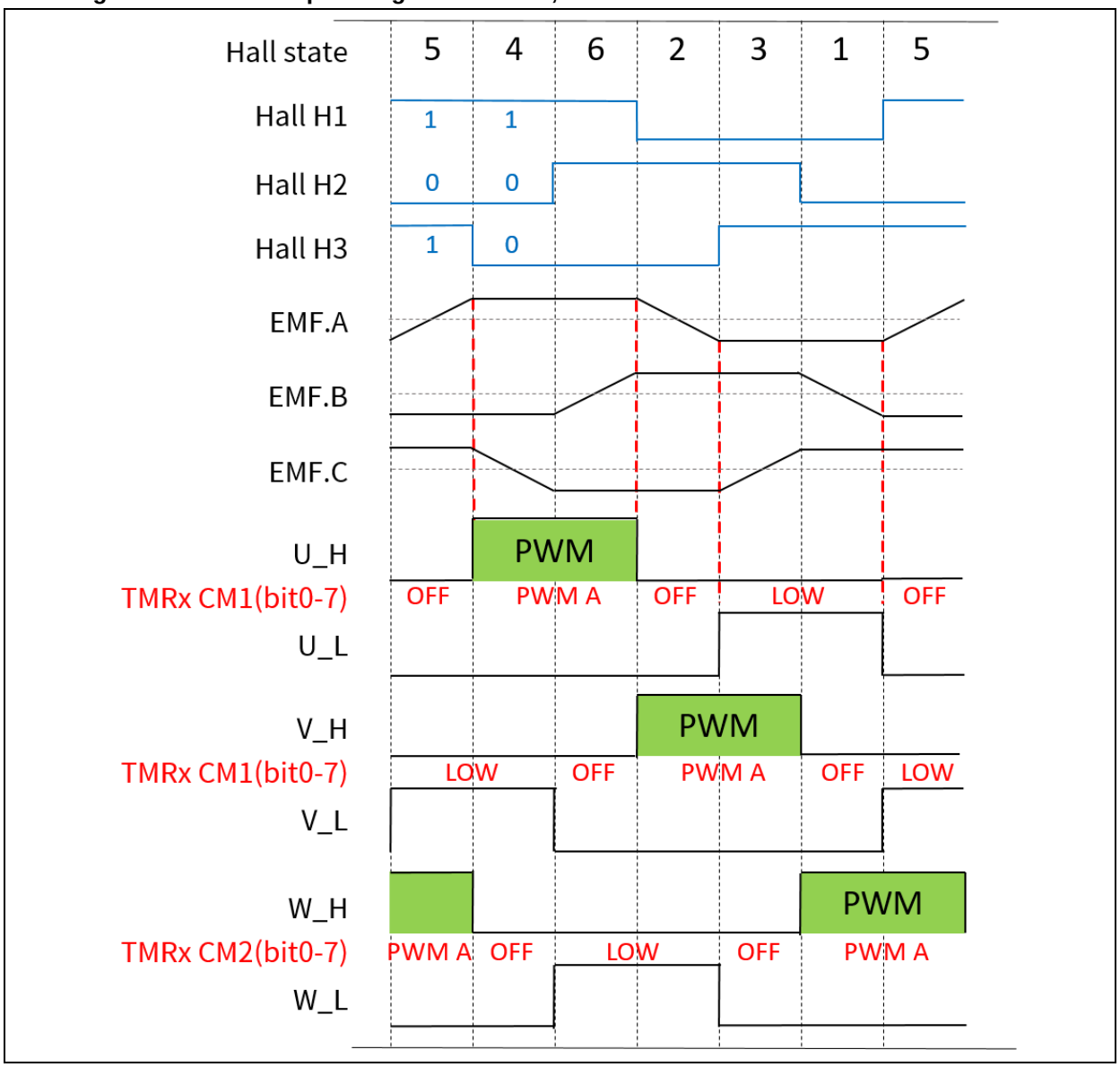
Definition item	Description
BLDC/FOC common parameters	
RS_LL	Motor line-to-line winding resistance (unit: Ω)
LS_LL	Motor line-to-line winding inductance (unit: mH)
POLE_PAIRS	Number of pole-pairs
ENCODER_PPR	Pulses per revolution of encode
ENC_STALL_TIME	Encoder stall time (unit: ms)
FOC dedicated parameters	
HALL_STATE_ONE_NEXT	The next state of HALL state 1 (forward)
HALL_STATE_TWO_NEXT	The next state of HALL state 2 (forward)
HALL_STATE_THREE_NEXT	The next state of HALL state 3 (forward)
HALL_STATE_FOUR_NEXT	The next state of HALL state 4 (forward)
HALL_STATE_FIVE_NEXT	The next state of HALL state 5 (forward)
HALL_STATE_SIX_NEXT	The next state of HALL state 6 (forward)
HALL_STATE_ONE_ANGLE	Electrical angle in HALL state 1
HALL_STATE_TWO_ANGLE	Electrical angle in HALL state 2
HALL_STATE_THREE_ANGLE	Electrical angle in HALL state 3
HALL_STATE_FOUR_ANGLE	Electrical angle in HALL state 4
HALL_STATE_FIVE_ANGLE	Electrical angle in HALL state 5
HALL_STATE_SIX_ANGLE	Electrical angle in HALL state 6
BLDC dedicated parameters	
HALL_STATE_1_PWM_MODE_CM1	PWM control configuration of TMRx_CM1 register in Hall state 1 (note [2])
HALL_STATE_2_PWM_MODE_CM1	PWM control configuration of TMRx_CM1 register in Hall state 2 (note [2])
HALL_STATE_3_PWM_MODE_CM1	PWM control configuration of TMRx_CM1 register in Hall state 3

Definition item	Description
	(note [2])
HALL_STATE_4_PWM_MODE_CM1	PWM control configuration of TMRx_CM1 register in Hall state 4 (note [2])
HALL_STATE_5_PWM_MODE_CM1	PWM control configuration of TMRx_CM1 register in Hall state 5 (note [2])
HALL_STATE_6_PWM_MODE_CM1	PWM control configuration of TMRx_CM1 register in Hall state 6 (note [2])
HALL_STATE_1_PWM_MODE_CM2	PWM control configuration of TMRx_CM2 register in Hall state 1 (note [2])
HALL_STATE_2_PWM_MODE_CM2	PWM control configuration of TMRx_CM2 register in Hall state 2 (note [2])
HALL_STATE_3_PWM_MODE_CM2	PWM control configuration of TMRx_CM2 register in Hall state 3 (note [2])
HALL_STATE_4_PWM_MODE_CM2	PWM control configuration of TMRx_CM2 register in Hall state 4 (note [2])
HALL_STATE_5_PWM_MODE_CM2	PWM control configuration of TMRx_CM2 register in Hall state 5 (note [2])
HALL_STATE_6_PWM_MODE_CM2	PWM control configuration of TMRx_CM2 register in Hall state 6 (note [2])
HALL_STATE_1_PWM_OUT_CCTRL	PWM control configuration of TMRx_CCTRL register in Hall state 1
HALL_STATE_2_PWM_OUT_CCTRL	PWM control configuration of TMRx_CCTRL register in Hall state 2
HALL_STATE_3_PWM_OUT_CCTRL	PWM control configuration of TMRx_CCTRL register in Hall state 3
HALL_STATE_4_PWM_OUT_CCTRL	PWM control configuration of TMRx_CCTRL register in Hall state 4
HALL_STATE_5_PWM_OUT_CCTRL	PWM control configuration of TMRx_CCTRL register in Hall state 5
HALL_STATE_6_PWM_OUT_CCTRL	PWM control configuration of TMRx_CCTRL register in Hall state 6

Note [2]: The PWM timer registers TMRx_CM1 and TMRx_CM2 configurations support to set upper/lower arm MOSFET switch states according to motor BEMF or Hall states.

Figure 3 shows the relationship among BLDC BEMF, Hall state and MOSFET conduction states. For example, in Hall state 5, the MOSFET states of phase A upper and lower arms are all OFF, and lower arm MOSFET of phase B is ON; therefore, define the HALL_STATE_5_PWM_MODE_CM1 as TMR_PWM_1OFF_2LOW, to set that phase A MOSFET state is OFF (1OFF) and phase B MOSFET state is LOW (2LOW) in Hall state 5. In addition, phase C MOSFET switch state is set at TMRxCM2; therefore, define HALL_STATE_5_PWM_MODE_CM2 as TMR_PWM_3PWMA, to set phase C upper arm MOSFET operates in PWM mode.

Figure 3. Relationship among BLDC BEMF, Hall state and MOSFET conduction states



In sensorless control, users can configure the TMRx_CM1 and TMRx_CM2 registers of PWM timer according to the relationship between back EMF voltages and switch states of three-phase MOSFET. The setting method is the same as Hall sensor control mode.

5) mc_hwio.h header file

- This file is mainly used for configuration definitions based on the mapping of user hardware IO interface and MCU peripherals, and it also includes function declarations in mc_hwio.c file.

6) mc_hwio.c file

- This file is mainly configured the MCU peripherals based on user hardware, such as TMR, ADC, DMA and GPIO, it also includes some basic control functions such as button, LED, PWM ON/OFF. Refer to Table 6 for details.

Table 6. Peripheral configuration related functions

Function	Description
nvic_config	Interrupt priority configuration
tmr_pwm_init	PWM timer, crm clock, GPIO and DMA configuration
gpio_hall_init	Hall sensor GPIO configuration
tmr_hall_init	Hall sensor timer and crm clock configuration for BLDC six-step control
hall_timer_init	Hall sensor timer, crm clock and GPIO configuration for FOC control
encoder_time_init	Incremental optical encoder timer, crm clock, GPIO and EXINT configuration
adc_ordinary_config	ADC ordinary channel ADC, DMA and GPIO configuration
adc_preempt_config	ADC preemptive channel ADC, DMA and GPIO configuration
gpio_output_init	Output GPIO configuration
uart_init	UART crm clock, GPIO and USART configuration
motor_board_init	MCU initialization function base on drive board
button_exint_init	Button interrupt event EXINT configuration
at32_button_state	Function for reading button GPIO state
err_led_init	Error LED GPIO initialization and crm clock configuration
err_led_on	Set error LED ON
err_led_off	Set error LED OFF
at32_led_toggle	Set error LED blinking
start_bldc	Six-step control related timers configuration before motor startup
current_offset_tmr_setting	timer configuration for current offset calibration
bldc_angle_init_config	timer configuration for initial angle detection of six-step control mode
pwm_switch_off	Disable PWM output configuration
pwm_switch_on	Enable PWM output configuration
foc_angle_init_config	timer and ADC configuration for initial angle detection of FOC control

7) mc_isr.c file

- This file includes the peripheral interrupt handler functions for motor control program. Refer to Table 6 for details.

Table 7. Motor control related interrupt functions

Function	Description
ADVTMR_PWM_CYCLE_IRQ	Control loop interrupt function (PWM update interrupt)
ADVTMR_PWM_BRK_IRQ	Brake input interrupt function (PWM output disable)
ADC_SHUNT_SAMP_READY_IRQ	Current/BEMF sensing complete interrupt function
EXINT_ENCODER_IDX_IRQ	Encoder zero position calibration interrupt function
HALL_CAPTURE_IRQ	Hall signals capture interrupt function
SysTick_Handler	System interrupt function (1ms), includes state machine program
BUTTON_EXINT_IRQHandler	Button interrupt function

8) mc_type.h file

- This file includes the type declarations of global structure variables and enumerations. Refer to Table 8 for details.

Table 8. Motor control library variable types

Enumeration	Description
BLDC/FOC common parameters	
firmware_id_type	Firmware ID type enumerations according to different motor control modes
motor_control_mode	Motor control mode enumerations (open loop control, voltage control, Id regulating mode, Iq regulating mode, speed control, torque control, position control, encoder alignment mode, etc.)
ctrl_source_type	Control source enumerations (software control, external circuit control)
encoder_align_type	Encoder alignment status enumerations
esc_state_type	Motor control process state machine (see Section 5.1 for details)
err_code_type	Motor error type enumerations (over-voltage, under-voltage, over-temperature, over-current, encoder error, Hall error, startup error)
curr_offs_type	Current loop offset structure variable type
current_type	Current-related structure variable type
voltage_type	Voltage-related structure variable type
adc_trigger_type	ADC trigger related structure variable type
pid_ctrl_type	PID control loop related structure variable type
pid_ctrl_dc_type	PID control loop related structure variable type
speed_type	Speed-related structure variable type
value_type	Structure type distinguished by program state, e.g. old, filtered

hall_sensor_type	Hall sensor related structure variable type
ramp_cmd_type	Ramp function-related structure variable type
queue_index	UART queue related structure variable type
moving_average_type	Moving average related structure variable type
FOC dedicated parameters	
qd_type	d,q type structure variable
abc_type	a,b,c axes structure variable type
alphabeta_type	α,β axes structure variable type
trig_components_type	Trigonometric function structure variable type
pwm_duty_type	PWM duty related structure variable type
rotor_angle_type	Rotor angle-related structure variable type
encoder_type	Encoder-related structure variable type
open_loop_type	Open loop related structure variable type
field_weakening_type	Field weakening related structure variable type
lowpass_filter_type	Low-pass filter related structure variable type
motor_volt_type	Voltage output related structure variable type
state_observer_type	Sensorless observer related structure variable type
sensorless_startup_type	Sensorless startup related structure variable type
foc_angle_init_type	Initial angle detection related structure variable type
BLDC dedicated parameters	
angle_init_type	Initial angle detection related structure variable type (for sensorless BLDC only)
start_state_type	State machine enumeration for initial angle detection in STARTING (for sensorless BLDC only))
init_current_type	Initial angle detection current structure variable type (for sensorless BLDC only)
angle_init_type	Initial angle detection related structure variable type (for sensorless BLDC only)
i_bus_type	BUS current related structure variable type
emf_sample_type	BEMF sampling related structure variable type
adc_sample_type	ADC sampling related structure variable type

4 Usage of motor control library function

Users can select six-step square wave motor control library (mc_bldc_library.lib) or vector control motor control library (mc_foc_library.lib) according to the motor control mode. These two motor control libraries include sensored/sensorless control functions. Users need to call the initialization function to perform initial setup of control program when using the motor control library. This section introduces function contents and how to use these functions.

4.1 General-purpose motor control library function

Initial setting related functions (*initialization functions required to use motor control library)

- uint8_t get_fw_id(void)

Get the program firmware ID, which indicates the control mode (note [3]).
Parameter: none.
Return value: return the firmware ID.

Note [3]: The control modes include FOC control, six-step square wave control, sensorless mode, HALL sensor, Encoder, etc.

- flag_status mc_param_init(uint8_t fw_id)

Initial parameter setting for motor control library functions
Parameter: Parameter1 “fw_id” is the firmware ID, which can be obtained by executing *get_fw_id()*.
Return value: return the initial setting complete flag, where “SET” indicates that the initial setting is completed, and “RESET” indicates initial setting failure.

Current sampling related functions

- flag_status current_offset_init(current_type* curr_handler, uint8_t shunt_nbr)

It is the current sampling initialization function, where the variable “shunt_nbr” is used for selecting the single-shunt, two-shunt, or three-shunt mode for current sampling function initialization and to get the initial value of three-phase current. The return value shows the initial setting complete flag, where “SET” indicates successful setting and “RESET” indicates setting failure.

- void current_read_foc_3shunt(current_type *curr_handler, pwm_duty_type *pwm_duty_handler)

This function reads the three-phase current value in vector control mode for a drive hardware with three-shunt current sensing circuit. It automatically selects the most appropriate sampled two of three phase currents under changing duty PWM, and the third phase current value is calculated by current balance method.

- void current_read_foc_2shunt(current_type *curr_handler)

This function reads the two-phase current value in vector control mode for a drive hardware with two-shunt current sensing circuit. The third phase current value is calculated by current balance method.

- void current_read_foc_1shunt(current_type *curr_handler, voltage_type *volt_handler)

This function reads the three-phase current value in vector control mode for a drive hardware with single-shunt current sensing circuit. It captures two phase current values on the single-shunt circuit at different sampling timings. The third phase current value is calculated by current balance method.

- void current_read_bldc(current_type *curr_handler)

This function reads BUS current value in six-step square wave control mode for a drive hardware with single-shunt current sensing circuit.

Hall position sensor related functions

- err_code_type read_hall_state(hall_sensor_type *hall_handler)

This function is used to read the Hall position sensor states, and detects for wrong signal or wire disconnection in motor control (it returns an error code if any error is detected).

- int16_t hall_rotor_angle_get(hall_sensor_type *hall_handler, rotor_angle_type *rotor_angle_handler)

  This function reads a continuous rotor angle value which is calculated based on the Hall states for FOC vector control. The return value is the motor rotor angle.

- int16_t hall_delta_theta_calculation(speed_type *rotor_speed_handler, rotor_angle_type *rotor_angle_handler, hall_sensor_type *hall_handler)

This function returns the calculated increment of rotor angle based on the states and frequencies of Hall signals. This value is one of the hall_rotor_angle_get function argument.

- err_code_type hall_at_zero_speed(hall_sensor_type *hall_handler)

This function is used for Hall sensor state initialization when the motor speed is zero. It also detects for wrong signal or wire disconnection of Hall position sensor, and returns an error code if any error is detected.

PID controller related functions

- void pid_set_kp(pid_ctrl_type *pid_handler, int16_t kp_gain)

This function sets the kp parameter of PID controller.

- void pid_set_ki(pid_ctrl_type *pid_handler, int16_t ki_gain)

This function sets the ki parameter of PID controller.

- void pid_set_kd(pid_ctrl_type *pid_handler, int16_t kd_gain)

This function sets the kd parameter of PID controller.

- void pid_set_intergral(pid_ctrl_type *pid_handler, int32_t integral_value)

This function sets the integral value of PID controller.

- int16_t pid_get_kp(pid_ctrl_type *pid_handler)

This function returns the kp parameter of PID controller.

- int16_t pid_get_ki(pid_ctrl_type *pid_handler)

This function returns the ki parameter of PID controller.

- int16_t pid_get_kd(pid_ctrl_type *pid_handler)

This function returns the kd parameter of PID controller.

- int16_t pi_controller(pid_ctrl_type *pid_handler, int32_t var_err)

This function acts as the PI controller (proportional-integral controller) with fixed dynamic output clamp, and it returns the output value of controller

- int16_t pi_controller_dyna_clamp(pid_ctrl_dc_type *pid_handler, int32_t var_err)

This function acts as the PI controller (proportional-integral controller) with a dynamic output clamp, and it returns the output value of controller.

- int16_t pid_controller(pid_ctrl_type *pid_handler, int32_t var_err)

This function acts as the PID controller (proportional-integral-derivative controller) with fixed dynamic output clamp, and it returns the output value of controller.

- void command_ramp(ramp_cmd_type *cmd_ramp_handler)

This function is used to convert a step command to a specified ramp command. The input parameters are step command and slope, and the output is the converted ramp command.

Filter related functions

- moving_average_type* moving_average(uint16_t order)

It is the initial function of a moving average filter, which is used together with the *moving_average_update* function.

Parameter: “order” is the number of input values set for average computation.

Return value: return a buffer address in *moving_average_type** type.

- int16_t moving_average_update(moving_average_type* filter, int16_t data)

This function returns the filtering result of data processed by moving average filter.

Parameter 1: “filter” is a buffer address in *moving_average_type** type, this address is obtained from the *moving_average()* function.

Parameter 2: “data” is the original data to be filtered.

Return value: the average filtering result.

- int16_t ma_filter(int16_t input_handler, int16_t average_handler, uint16_t PowOf2)

This function is a fast moving average filter that calculates moving average value by bit-shift operation. It is mainly used in the applications requiring fast multi-point moving average calculation.
Return value: the average filtering result.

- void lowpass_filter_init(lowpass_filter_type *lowpass_handler)

This function is the initial function of a low-pass filter and the initialization results will be stored in the structure variable which is pointed by handler. It is used together with the *lowpass_filtering* function.
Parameter: “lowpass_handler” is a handler address in *lowpass_filter_type** type. This handler is point to a structure variable with the sampling frequency and filtering bandwidth values.

Return value: none.

- int16_t lowpass_filtering(lowpass_filter_type *lowpass_handler, int16_t input_handler, int16_t output_handler)

This function is a low-pass filter. The “lowpass_handler” is an address pointer in lowpass_filter_type * type. The pointer variable is initialized by the *lowpass_filter_init* function. The return value is a filtering result.

4.2 Motor control library functions in vector control mode

Encoder-related functions

- void encoder_count_reset(encoder_type *enc_handler)

This function is used to clear photoelectric incremental encoder counter when its index signal is detected. Executing this function during encoder alignment procedure will set the idx_reset_flag to ensure a complete alignment process.

- err_code_type encoder_alignment_index (voltage_type *volt_handler, encoder_type *enc_handler)

This function is used for the alignment procedure of photoelectric incremental encoder with index pulse. In the alignment process, the motor runs until the index pulse of encoder is detected. An error code is returned if the alignment fails.

- void encoder_alignment(voltage_type *volt_handler, encoder_type *enc_handler)

This function is used for the alignment procedure of photoelectric incremental encoder without index pulse. The motor shaft is forcedly aligned to d-axis, and then the encoder counter is cleared.

- int16_t enc_rotor_angle_get(encoder_type *enc_handler)

This function returns the motor rotor angle converted from pulses counting value of encoder after alignment. This function is required in vector control.

- int32_t enc_rotor_speed_get(encoder_type *enc_handler, speed_type *spd_handler)

This function returns the motor speed based on the counting value and timing of encoder pulses after alignment, this function is required in vector control.

- err_code_type enc_error_check(encoder_type *enc_handler, ramp_cmd_type *cmd_ramp_handler, current_type *curr_handler)

This function is used to detect errors of photoelectric incremental encoder during motor running. It detects for wrong signals and wire disconnection of encoder, and returns an error code if any error is detected.

Pulse width modulation (PWM) related functions

- pwm_duty_type svpwm_3_2shunt(voltage_type *volt_handler, pwm_duty_type *pwm_duty_handler)

It is a space vector modulation function, which is an algorithm to control pulse width modulation (PWM). It calculates three-phase PWM duty in vector control for a hardware with three-shunt or two-shunt current sensing circuits, and it returns three-phase PWM duty.

- pwm_duty_type svpwm_1shunt(voltage_type *volt_handler, pwm_duty_type *pwm_duty_handler)

It is a space vector modulation function, which is an algorithm to control pulse width modulation

(PWM). It calculates three-phase PWM duties in vector control for a hardware with single-shunt current sensing circuit, and it returns three-phase PWM duty.

- 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)

This function controls the phase shift of pulse width modulation (PWM). It is applied to sample two phase currents on the single-shunt circuit in vector control, and it returns the ADC sampling point.

Field weakening related functions

- void fw_clear(field_weakening_type *fw_handler, voltage_type *volt_handler)

It is the initialization function of field weakening control, in which the parameters related to field weakening control are set to zero.

- qd_type fw_curr_ref(field_weakening_type *fw_handler, qd_type Iqdref)

This function is used to set the maximum field weakening current to avoid motor demagnetization. In vector control, the field weakening control with limitation of maximum field weakening current is enabled when the output voltage reaches the specified percentage. It returns the current commands of q and d axes.

Vector control related functions

- trig_components_type trig_functions (int16_t angle_handler)

This function calculates the sine/cosine function of trigonometry. The input variable “angle_handler” is in Q15 format and the input range “[0 32767]” is represented by [0 2π]. The *trig_components_type* is the returned variable type (in Q15 format), with the output range of [-32768 32767], which includes sin/cos outputs to represent the sine/cosine calculation results, respectively.

- alphabeta_type foc_clarke_trans(abc_type *abc_handler)

This function is used for clarke transformation in vector control to transform the three-phase vector to two-phase orthogonal vector. It is used for the three-phase current or voltage vector transformation, and returns the transformed two-phase orthogonal vector.

- void foc_park_trans(current_type *curr_handler, trig_components_type *theta_handler)

This function is used for park transformation in vector control to project the vector in two-phase orthogonal stationary frame to the orthogonal rotating reference frame. It is used in the current vector transformation.

- void foc_inver_park_trans(voltage_type *volt_handler, trig_components_type *theta_handler)

This function is used for inverse park transformation in vector control to project the vector in orthogonal rotating reference frame to the two-phase orthogonal stationary frame. It is used in the voltage vector transformation.

- void foc_circle_limitation(voltage_type *volt_handler)

This function limits the maximum output of the voltage vector composition in vector control, and the combined output voltage is confined to a circular rotating voltage vector.

- int16_t foc_open_loop_ctrl(voltage_type *volt_handler, open_loop_type *openloop_handler)

It is the open loop voltage control function in vector control, and the input parameters give the specified open loop voltage and angle increment to calculate the voltage vector angle and determine

the control voltage for users to perform open loop voltage control. It returns the open loop rotor angle.

Sensorless vector control related functions

- void observer_pll_clear(state_observer_type *state_obs_handler)

It is the initialization function of FOC sensorless observer and quadrature-component-based phase locked loop estimator, in which the sensorless control parameters are set to zero.

- flag_status startup_openloop(sensorless_startup_type *startup_handler, voltage_type *volt_handler)

It is the motor sensorless startup function, which is used to set the startup voltage, maximum speed and acceleration at startup, and select the open loop startup mode. It returns the motor startup complete flag, where “SET” indicates successful startup and “RESET” indicates startup failure.

- flag_status startup_alpha_axis(sensorless_startup_type *startup_handler, voltage_type *volt_handler)

It is the motor sensorless startup function, which is used to set the maximum speed at startup, alignment voltage, alignment time and startup time, and select to start motor after aligned to α axis. It returns the motor startup completed flag, where “SET” indicates successful startup and “RESET” indicates startup failure.

- flag_status startup_angle_init(sensorless_startup_type *startup_handler, voltage_type *volt_handler)

It is the angle detection function for motor sensorless startup, which is used to detect initial rotor angle and then set the maximum speed at startup, startup voltage and startup time. After the initial angle is obtained, this function instructs drive to output a fixed voltage vector to start motor to avoid inverse motor rotation. It returns the motor startup complete flag, where “SET” indicates successful startup and “RESET” indicates startup failure.

- flag_status startup_angle_init2(sensorless_startup_type *startup_handler, voltage_type *volt_handler)

It is the angle detection function for motor sensorless startup, which is used to detect initial angle and then set the startup voltage, maximum speed and acceleration at startup. After the initial angle is obtained, start the motor by means of open loop startup to avoid inverse motor rotation. It returns the motor startup complete flag, where “SET” indicates successful startup and “RESET” indicates startup failure.

- void foc_sensorless_angle_init(foc_angle_init_type *angle_detect_handler, current_type *curr_handler)

It is the initial angle detection function for motor sensorless startup, which is mainly used to detect the initial angle of motor to avoid inverse rotation at startup.

- void current_angle_init_3shunt(foc_angle_init_type *angle_detect_handler, current_type *curr_handler)

This function is used to sample three-phase currents when detecting the motor initial angle. Then the motor initial angle is estimated based on the sampled current vectors

- void current_angle_init_2_1shunt(foc_angle_init_type *angle_detect_handler, current_type *curr_handler)

It is the single-shunt/two-shunt current sampling function for motor initial angle detection. The motor

initial angle is estimated based on the sampled current vectors. This function is only used for the drive hardware with a bus current shunt.

- alphabeta_type motor_volt_calc(motor_volt_type *motor_volt_handler)

This function calculates the output voltage vector of drive according the output voltage command and detected input DC voltage. It returns two-phase orthogonal voltage vector.

- alphabeta_type motor_volt_read(motor_volt_type *motor_volt_handler)

This function reads the driver voltage output according detected input DC voltage and PWM output voltage. It returns two-phase orthogonal voltage vector.

- int16_t obs_pll_execute(state_observer_type *state_obs_handler, int16_t hBemf_alfa_est, int16_t hBemf_beta_est)

This function is used by the quadrature phase-lock-loop observer in sensorless control mode to estimate the motor speed based on the estimated BEMF signals. It returns the estimated rotor speed.

- int16_t rotor_angle_sensorless(state_observer_type *state_obs_handler, motor_volt_type *motor_volt_handler)

This function is used by the rotor angle estimator in sensorless mode, to estimate the motor BEMF and calls the “obs_pll_execute” function to obtain the rotor speed, and then estimates the rotor angle by an integrator. It returns the estimated rotor angle.

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

Functions related to six-step square wave control

- void bldc_output_config(uint8_t hall_states)

Configure the timer for PWM output.

Parameter: “hall_state” represents Hall state, with a range of 1-6 that respectively correspond to the six phases in six-step square wave control mode.

Return value: None.

- void disable_mosfet(tmr_type *tmr_x)

Disable PWM output of timer to turn off MOSFET.

Parameter:“tmr_x” represents the timer code type variable. The timer is used for output PWM.

Return value: None.

- void adc_sample_point_set(adc_sample_type *adc_sample,int16_t pwm_duty)

Calculation of ADC sampling points, including the current sampling point and BEMF (Note [4]) sampling point.

Parameter 1: “adc_sample” type variable includes the required parameters and peripheral for ADC sampling.

Parameter2: “pwm_duty” represents the output PWM duty currently.

Return value: None.

Note [4]: The BEMF sampling point is only used in sensorless mode.

Functions related to sensorless six-step square wave control

- void bldc_sensorless_angle_init(tmr_type *tmr_x,angle_init_type *angle_init,int16_t step_count)

It is the function of detecting initial motor angle. This function is used together with the *angle_init_estimation* function.

Parameter1: “tmr_x” represents the timer code type variable. The timer is used for output PWM.

Parameter2: “angle_init” type variable includes the initial angle detection related parameters.

Parameter3: “step_count” is the counts of the detection steps (7 steps in total).

Return value: None

- uint8_t angle_init_estimation(angle_init_type *angle_init)

Estimation of the initial rotor angle. This function is called after the *bldc_sensorless_angle_init* function is executed.

Parameter: “angle_init” type variable includes the initial angle detection related parameters.

Return value: return the control phase corresponding to the detected initial angle in six-step square wave control mode, with a range of 1-6.

- void bldc_sensorless_detectEMF_config(adc_sample_type *adc_sample)

Configuration of ADC peripheral for the BEMF zero crossing point detection.

Parameter: “adc_sample” type variable includes the required parameters and peripheral for ADC sampling.

Return value: None.

- void bldc_emf_comp_read(hall_sensor_type *hall_handler,adc_sample_type *adc_sample,speed_type *rotor_speed)

It is used for detecting BEMF zero crossing point, calculating motor speed and trigger phase commutation interrupt (this function is only used for the comparator detection method).

Parameter1: “hall_handler” includes relevant parameters for BEMF comparison signals configuration and six-step commutation stages.

Parameter2: “adc_sample” type variable includes the required parameters and peripheral for ADC sampling.

Parameter3 “rotor_speed” type variable represents the speed-related parameter.

Return value: None.

- void bldc_emf_adc_read(hall_sensor_type *hall_handler,adc_sample_type *adc_sample,speed_type *rotor_speed)

It is used for detecting BEMF zero crossing point, calculating motor speed and trigger phase commutation interrupt (this function is only used for the ADC detection method).

Parameter1: “hall_handler” includes relevant parameters for BEMF comparison signals configuration and six-step commutation stages.

Parameter2: “adc_sample” type variable includes the required parameters and peripheral for ADC sampling.

Parameter3 “rotor_speed” type variable represents the speed-related parameters.

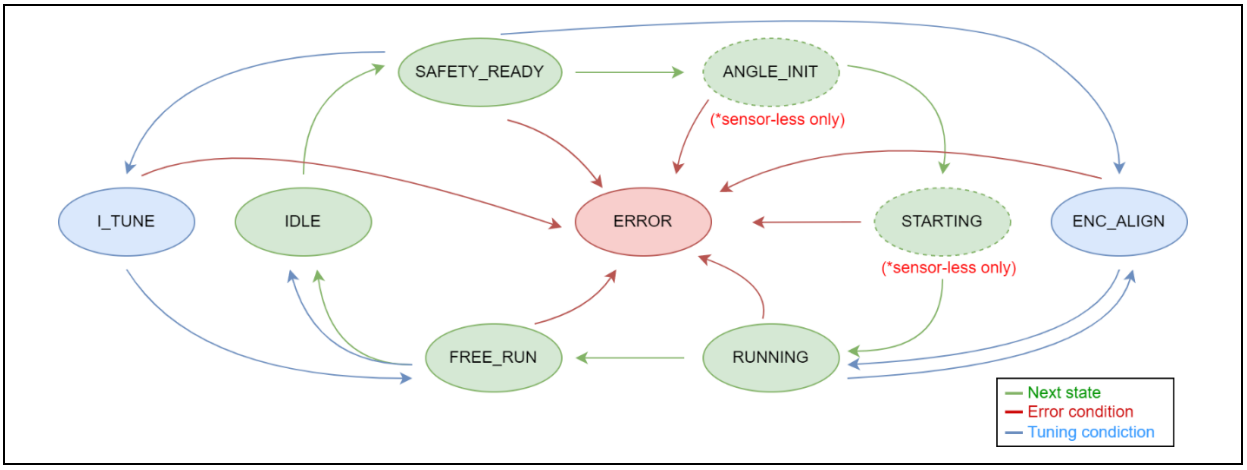
Return value: None.

5 Program structure of motor control library application examples

5.1 State machine

Figure 4 shows the state machine of the application example of motor control library. It includes Idle, Safety ready, Angle init, Starting, Running, Free run, I_tune, Enc_align and Error states.

Figure 4. State machine process flow



5.1.1 Description of state

Idle*

It is the initial state of state machine, and motor idles in this state. The state machine returns to “Idle” state when the error condition is released or the motor stops.

Safety ready*

This state indicates that the motor can be started up safely after all parameters are set and checked the current offset is obtained in the “Idle” state.

Angle init

In the sensorless control mode, the program enters this state for detecting the initial angle before starting the motor, and then jumps to the "Starting" state after obtaining the initial angle. It is the exclusive state of the sensorless control mode. It can be omitted if the detetion of motor initial angle is not required.

Starting

In the sensorless control mode, enter this state after detecting the initial angle, set the operation mode parameters, and peripheral configuration in this state. In addition, set the conditions for switching to closed-loop control to improve system stability.

Running*

The motor is running in this state. Through the UI software, users can on-line adjust parameters (target speed, target current, etc.) or give a command to stop the motor.

Free run

In this state, the drive stops voltage output, and the motor speed will slowly down to zero. Program remains in this state until the motor stops completely, and then returns to the initial state (Idle).

I_tune*

This state is used for adjusting PID controller parameters. In this state, users can adjust the target current, current loop KP and KI values through UI software. A step current is generated in this state, based on which parameters are adjusted.

Enc_align

This is state is for the zero calibration of the encoder. After the driver is restarted, the zero calibration will be performed automatically before starting the motor, or user can start the zero calibration function on the UI software. Skip this state if the motor does not have an encoder.

Error*

Jump to this state in case of any error.

Note: The states marked with "" are executed continuously until other event is generated (operation by users or any fault).*

5.1.2 State machine process

The state machine of motor control program is shown in Figure 4 in previous section, states in green circles are the general states in normal conditions, and states in blue circles are in special conditions. 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.

6 Revision history

Table 9. Document revision history

Date	Version	Revision note
2022.12.23	2.0.1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services, and ARTERY assumes no liability whatsoever relating to the choice, selection or use of the ARTERY products and services described herein.

No license, express or implied, to any intellectual property rights is granted under this document. If any part of this document deals with any third party products or services, it shall not be deemed a license grant by ARTERY for the use of such third party products or services, or any intellectual property contained therein, or considered as a warranty regarding the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

Unless otherwise specified in ARTERY's terms and conditions of sale, ARTERY provides no warranties, express or implied, regarding the use and/or sale of ARTERY products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose (and their equivalents under the laws of any jurisdiction), or infringement of any patent, copyright or other intellectual property right.

Purchasers hereby agrees that ARTERY's products are not designed or authorized for use in: (A) any application with special requirements of safety such as life support and active implantable device, or system with functional safety requirements; (B) any air craft application; (C) any automotive application or environment; (D) any space application or environment, and/or (E) any weapon application. Purchasers' unauthorized use of them in the aforementioned applications, even if with a written notice, is solely at purchasers' risk, and is solely responsible for meeting all legal and regulatory requirement in such use.

Resale of ARTERY products with provisions different from the statements and/or technical features stated in this document shall immediately void any warranty grant by ARTERY for ARTERY products or services described herein and shall not create or expand in any manner whatsoever, any liability of ARTERY.