Application Note

Document No.: AN1099

APM32F035_MOTOR EVAL Sensing Vector Control Scheme

Version: V1.3

Contents

1 General Introduction

1.1 Project Overview

APM32F035 is a specialized chip launched by Geehy Semiconductor Co., Ltd. for motor control. Based on APM32F035, this design provides a dual-resistance sampling sensing vector control scheme. The detailed design specifications are shown in the table below:

Table 1 Design Specifications

Control mode	Hall Sensor Field Oriented Control (FOC)
PWM modulation mode	SVPWM
Angle estimation	Hall interpolation compensation algorithm
PWM frequency	8KHz
Motor speed	400~3000RPM (2 pairs of poles)
Protection function	Overvoltage, undervoltage, software overcurrent, hardware overcurrent
Code size	11Kbytes
Development software	Keil MDK (V5.23 version and above)

1.2 APM32F035 Chip Resources

APM32F035 is a high-performance special MCU for motor control which is based on the Arm Cortex-M0+ core, integrates the mathematical operation accelerators (Cordic, Svpwm, hardware divider, etc.) commonly used in FOC algorithms, and integrates such analog peripherals as amplifiers and comparators, as well as CAN controllers.

Table 2 Functions and Peripherals of APM32F035 Series Chip

Product		**APM32F035**	
Model		**C8T7**	**K8T7**
Package		LQFP48	LQFP32
Core and maximum working frequency		Arm® 32-bit Cortex®-M0+@72MHz	
M0CP Co-processor		1	
Flash memory (KB)		64	
SRAM(KB)		10	
Timer	32-bit/16-bit universal	1/2	
	16-bit advanced	1	
	16-bit basic	2	
	24-bit counter	1	

Product		APM32F035	
Model		C8T7	K8T7
	Watchdog (WDT)	2 (1 independent watchdog +1 window watchdog)	
	Real-time clock	1	
Communication interface	USART	2	
	SPI/I2S	1/1	
	I2C	1	
	CAN	1	
12-bit ADC	Unit	1	
	External channel	16	12
	Internal channel	3	
Comparator (COMP)		2	
Operational amplifier (OPA)		4	2
GPIOs		42	27
Operating temperature		Ambient temperature: -40℃ to 105℃ Junction temperature: -40℃ to 125℃	
Working voltage		2.0~3.6V	

2 Hardware Introduction

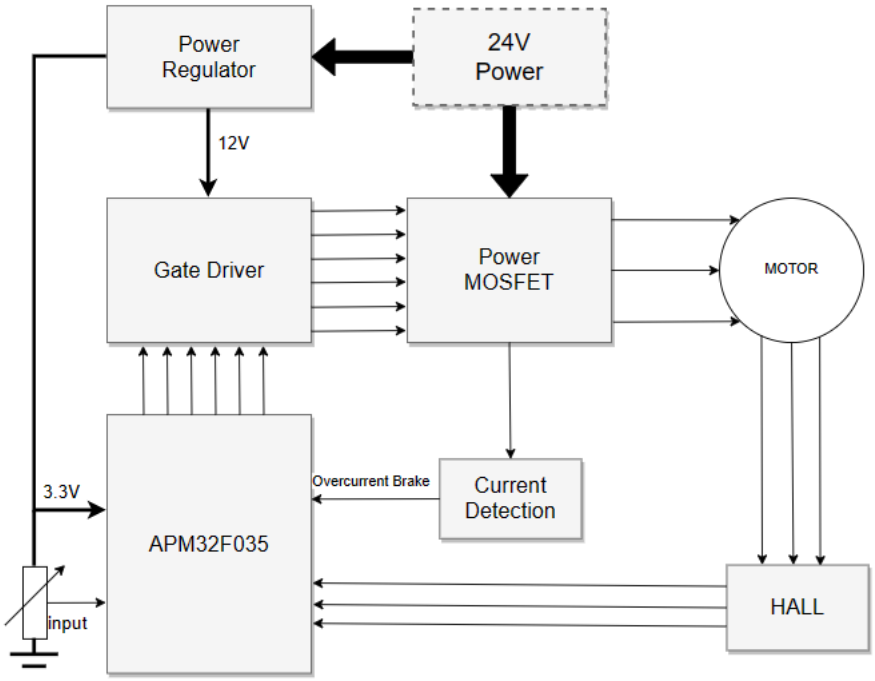
2.1 Overall Hardware Circuit

The overall hardware system is powered by an external 24V power supply and after conversion through the corresponding power step-down circuit, it outputs stable 12V, 5V, and 3.3V voltages. The 12V voltage is output to the Gate driver IC, the 3.3V voltage is output to the APM32F035 series microprocessor, and the power switch tube is directly connected to the 24V power supply. At the same time, this scheme uses a variable resistance knob to adjust the voltage input of 0~3.3V as the input end of the speed command, to adjust the motor speed. Users can directly adjust the input voltage by turning the variable resistor knob in actual use. When the input voltage value exceeds the starting threshold, the motor will start running, and when the voltage value is below the threshold, the motor will stop running.

Calculate the rotor angular velocity We through six Hall jumps. Hall is used to distinguish six sectors, and the interpolation compensation algorithm is used to estimate the rotor position and motor speed. After the motor is started, the APM32F035 processor can obtain the phase currents Iu, Iv, and Iw of three phases through the built-in operational amplifier and corresponding sampling circuit, and convert this data through the coordinate axis to control the torque current and phase of the motor. After the FOC control calculation link, adjust the TMR1 peripheral to output the corresponding three-way complementary PWM waves to control the switching components of the inverter.

The hardware block diagram is shown in the figure.

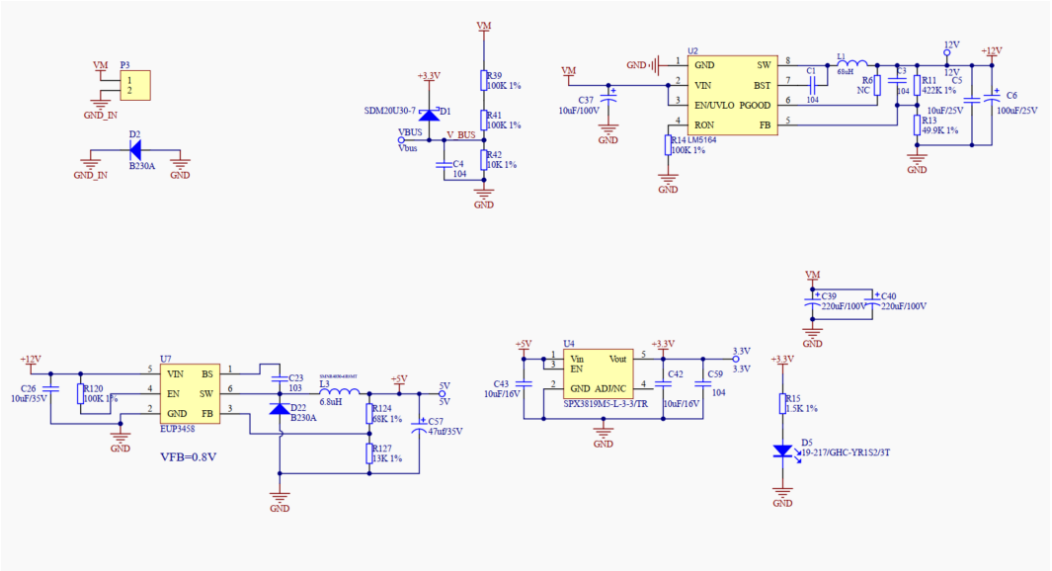
Figure 1 Hardware System Block Diagram



2.2 Interface Circuits and Settings

2.2.1 Power circuit

Figure 2 Power Circuit



As shown in the figure, supply voltage V_BUS =VM/((100K+100K+10K)/10K)=VM/21

A 12-bit ADC is adopted, and the sampling range 0-3.3V corresponds to 0-4096

Then the maximum sampling voltage corresponding to 3.3V is: VM= 3.3 *21 =69.3V

2.2.2 Phase Current Sampling Circuit

Figure 3 MOSFET Circuit

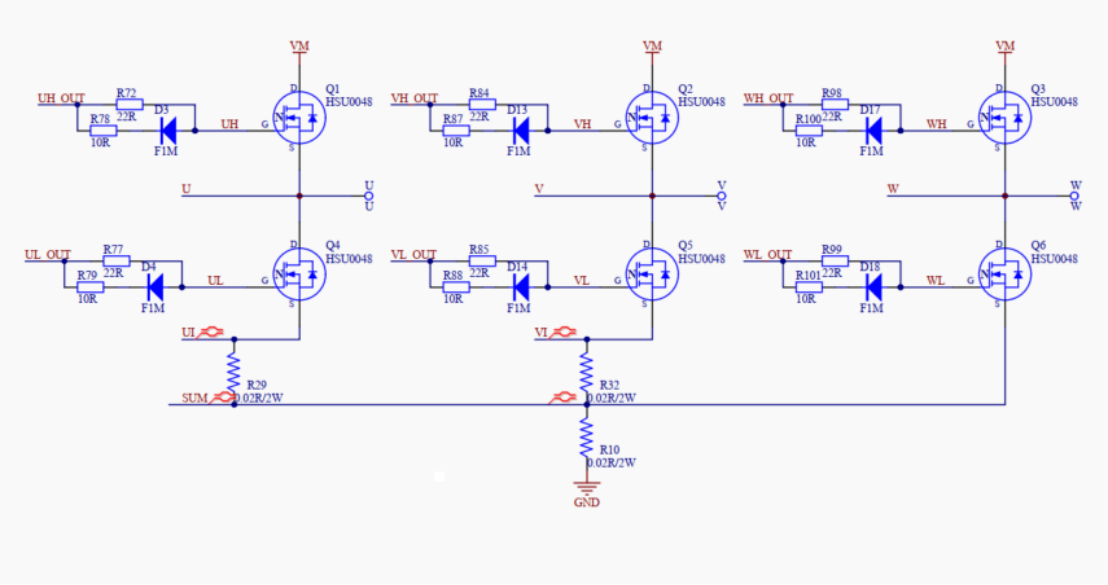
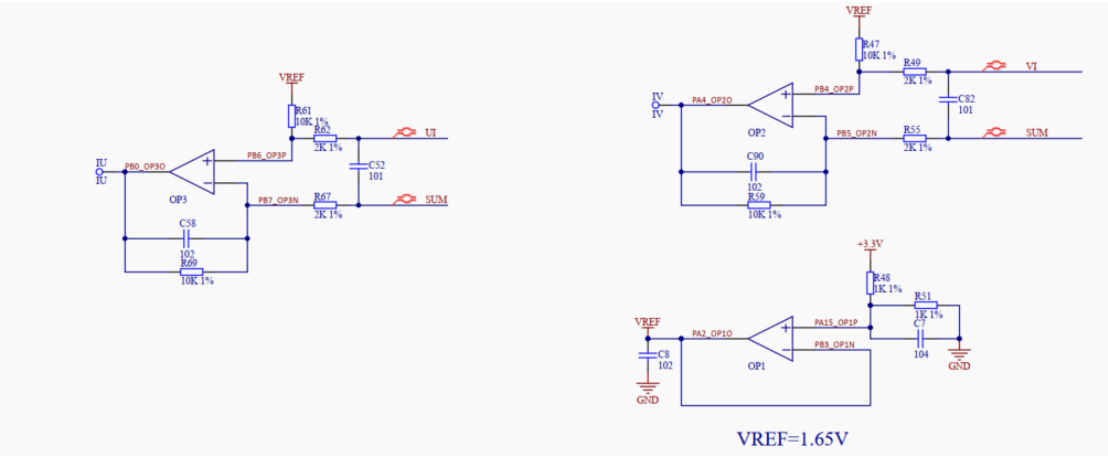


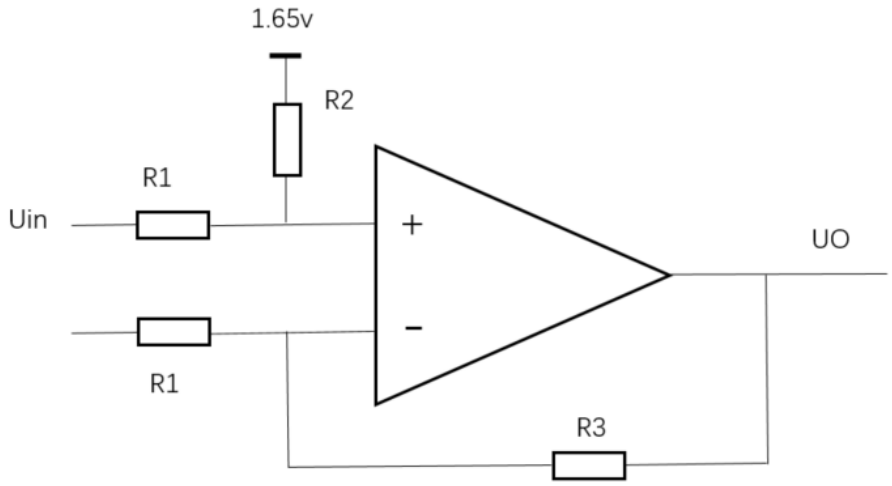
Figure 4 Current Sampling Circuit



As shown in the figure, IU = UI*4.86+1.60

Where 4.86 is the amplification factor of the operational amplifier and 1.6 is the bias voltage. The derivation process is as follows:

Figure 5 Amplifier circuit diagram



R3 is a 10K feedback resistor coupled with an internal 294K resistor, resulting in a combined resistance of 294K.

According to the virtual short concept, the equation for the positive terminal can be written as:

$$\frac{1.65 - U+}{10K} = \frac{U+ - Uin}{2K}$$, which means: $$\frac{1.65 - U+}{R2} = \frac{U+ - Uin}{R1}$$

Similarly, the negative terminal can be formulated as follows:

$$\frac{UO - U-}{9.671K} = \frac{U- - 0}{2K}$$, which means: $$\frac{UO - U-}{R3} = \frac{U- - 0}{R1}$$

Based on the virtual short: U+ = U-, the final equation can be obtained: 1.604 + 4.86Uin = Uo

Where 1.604 is the bias voltage and 4.86 is the amplification factor.

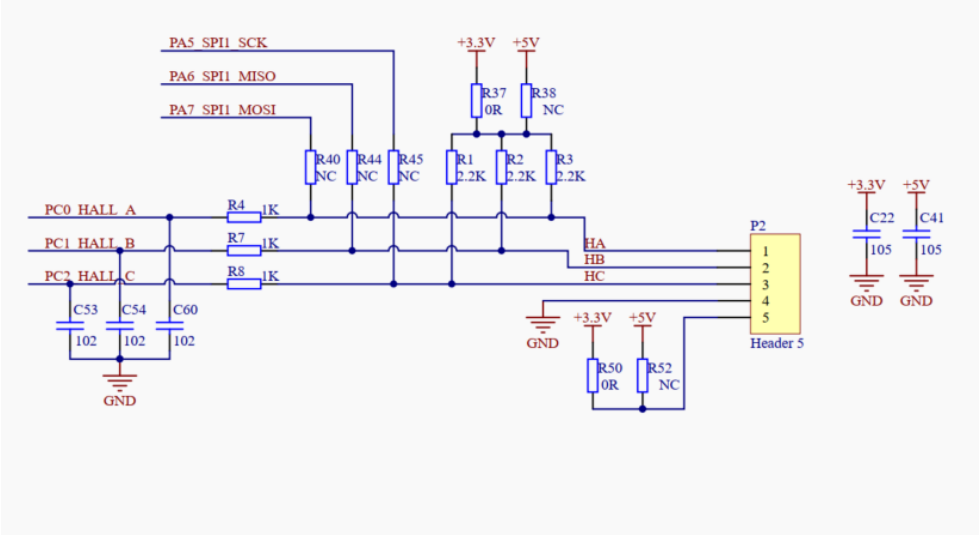
A 12-bit ADC is adopted, and the sampling range 0-3.3V corresponds to 0-4096

As shown in Figure 3, when the sampling resistance is selected as 0.02R,

Then the maximum peak-to-peak current corresponding to 3.3V is 16.46A = 1.60/4.86/0.02

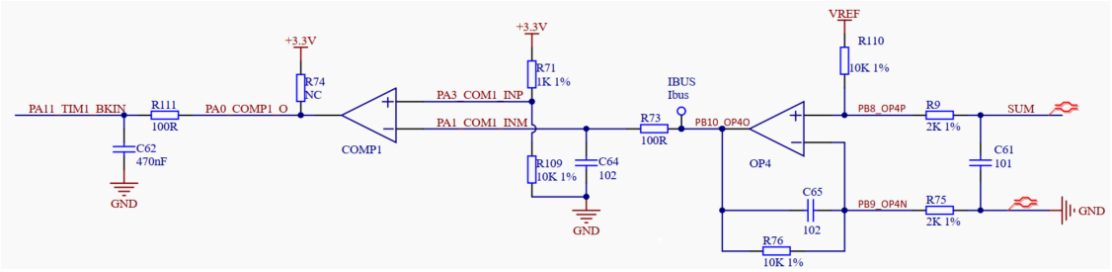
2.2.3 Hall Detection Circuit

Figure 6 Hall Detection Circuit



2.2.4 Overcurrent protection circuit

Figure 7 Overcurrent Protection Circuit



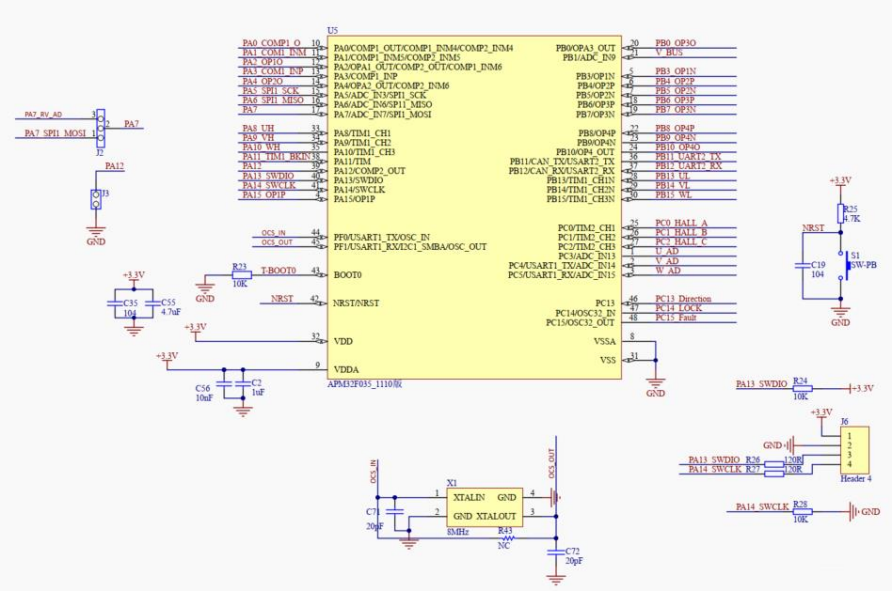
As shown in the figure, a built-in operational amplifier OPA4 is used to sample the bus current. A 12-bit ADC is adopted with a sampling range of 0-3.3V corresponding to 0-4096. From Figure 2-3, it can be seen that the sampling resistance is 0.02R,

the output end of OPA4 is used as the reverse input end of COMP1, and resistance voltage division is adopted at the forward input end. Through simple calculation, it can be concluded that when the input is 3V,

the maximum current corresponding to 3V is (3-1.65)/5/0.02=13.5A

2.2.5 Minimum system circuit

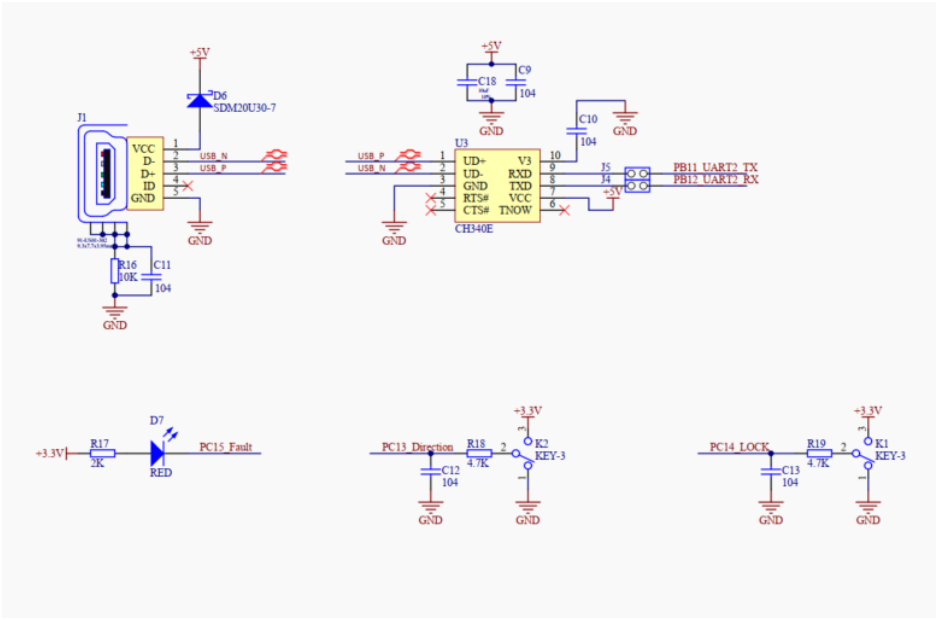
Figure 8 Minimum System Circuit



As shown in the figure, the utilization of APM32F035 MOTOR EVAL V1.0 board hardware interface resources is described in the above figure. The external crystal oscillator input of HSE is 8MHz, and the SWD burning interface is adopted for burning.

2.2.6 Communication Interface and Button Circuit

Figure 9 Communication Interface and Button Circuit



As shown in the figure, a USB-to-serial port and a fault indicator light are reserved in the

APM32F035 MOTOR EVAL V1.0 board hardware for debugging by developers; the two buttons are responsible for implementing the functions of controlling the running direction of the motor and locking.

2.3 Physical System Hardware

The picture of the system is shown in the figure, and it mainly includes the following five interfaces:

(1) Power input interface (connect to 24V; pay attention to positive and negative poles)

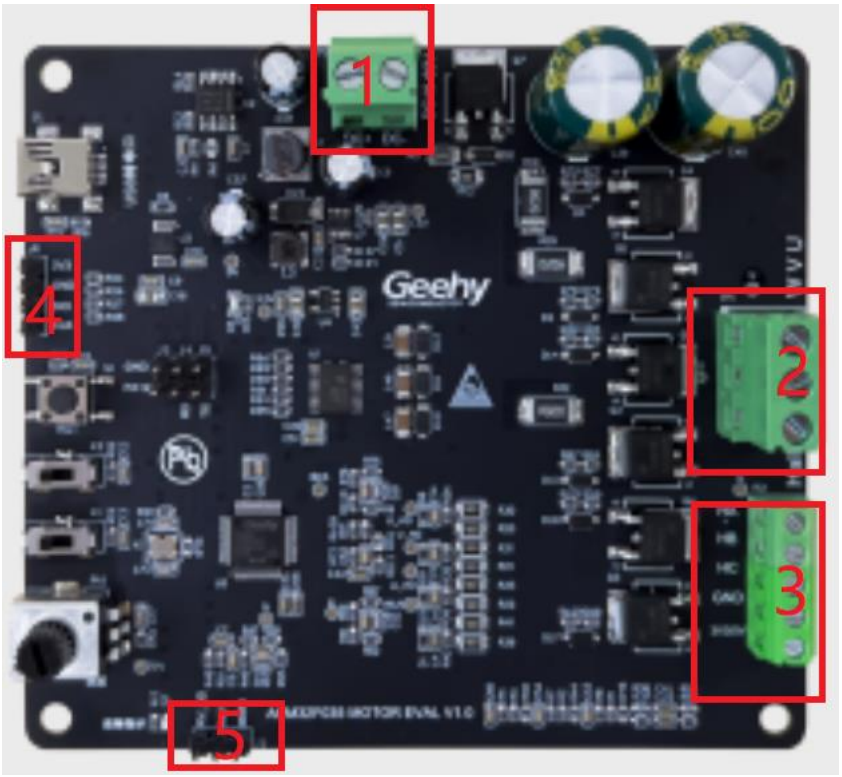
(2) Three-phase motor interface (phase sequence only affects the direction of rotation)

(3) HALL input interface

(4) SWD debugging interface

(5) The jumper cap port needs to be connected

Figure 10 Hardware Picture



3 Software Introduction

3.1 Overall Program Architecture

The overall code architecture of this project can be divided into four layers: user layer, peripheral driver layer, motor control driver layer, and motor algorithm layer. The specific functional descriptions are as follows:

3.1.1 USER Layer

main.c: The main function entry is responsible for switching motor initialization parameters, underlying peripherals, interrupt priority, while cycle, and low-speed state machine loop;

apm32f035_int.c: All interrupt handling functions, mainly including TMR1 interrupt function and ADC interrupt handler function;

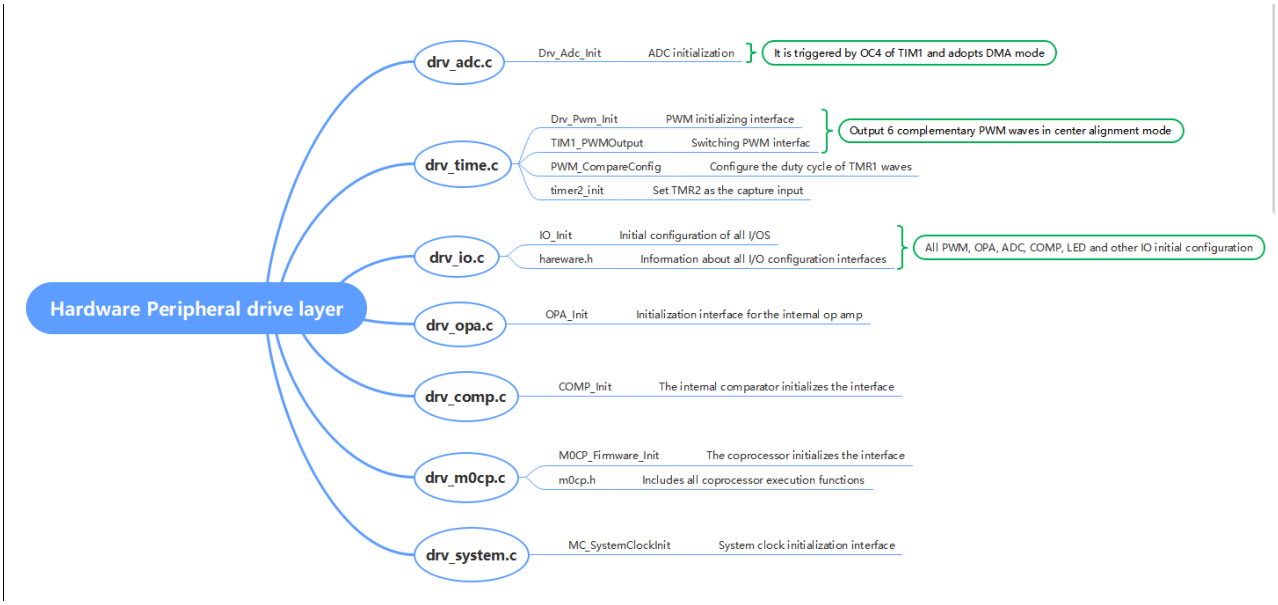
user_function.c: Includes initialization configuration, parameter reset, and other handler functions of motor parameters;

parameter.h: Includes all required configuration parameter information;

3.1.2 Peripheral Driver Layer (HARDWARE Layer)

The peripheral driver layer is mainly responsible for the peripheral driver functions and configuration of the APM32F035 chip, mainly including GPIO, PWM, ADC, OPA, COMP, and M0CP coprocessors, as shown in the following figure.

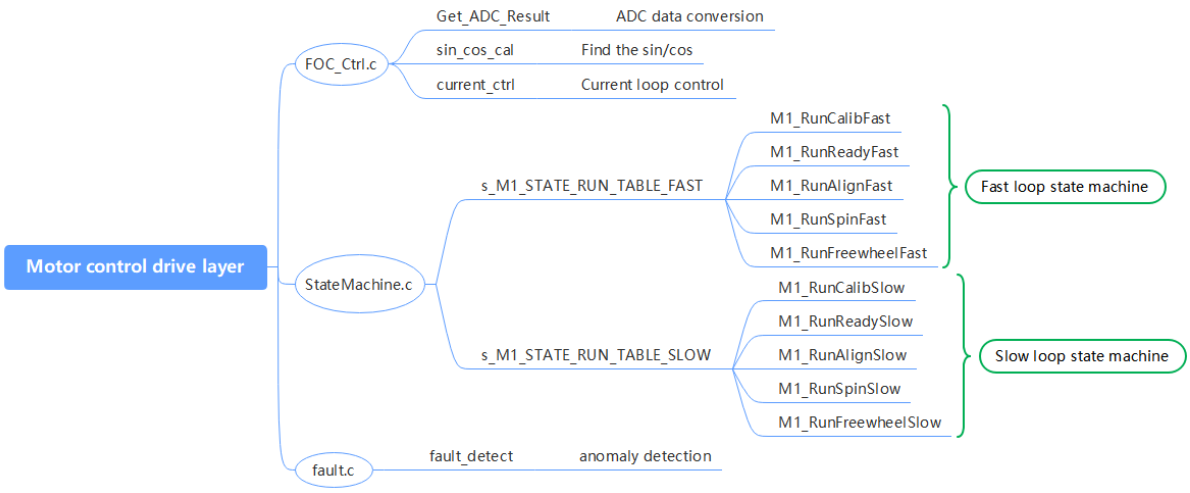
Figure 11 Peripheral Driver Layer



3.1.3 Motor Control Drive Layer (MOTOR_CONTROL Layer)

The motor control driver layer is mainly responsible for the control run logic and core processing algorithm call of the motor, as shown in the following figure.

Figure 12 Motor Control Driver Layer



3.1.4 Geehy Motor Algorithm Layer (Geehy_MCLIB Layer)

The motor algorithm layer includes coordinate transformation, vector control, and other related functions, as well as math libraries, angle estimation, and other library functions.

3.2 Introduction to State Machine

In this case, the structure of embedding the sub-state machine into the main state machine is adopted, as shown below:

Four main states: INIT, STOP, FAIL, and RUN;

The six RUN sub-states of the main state are **run calib, run-ready, run-align, run-startup, run-spin, and run-freewheel**.

The main state machine is described below:

Fault: When an error occurs in the system, it will remain in this state until the error flag bit is cleared;

Then after delay for a while, it will jump from the Fault state to the STOP state and wait for the start command

Init: This main state executes variable initialization;

Stop: The system waits for the speed command after completing initialization. In this state, the PWM output is turned off;

Run: When the system is running and there is a Stop command, the system will stop running.

Run-Calib: Executes the current bias ADC self-calibration function. After executing this state, the system will switch to the Ready state and disable PWM output.

Ready: Enables PWM output, synchronously samples current, and performs abnormal state checks.

Align: Samples the current, calls the pre-positioning algorithm, and updates PWM. Executes the state within a specified time, while sampling and filtering the DC bus voltage.

Spin: Samples the current, calls the observer to estimate the rotor speed and position, calls the corresponding algorithm, updates PWM, and the motor starts closed-loop operation.

Freewheel: Enables PWM output and uses short-circuit braking to stop the motor. Due to rotor inertia, it is necessary to wait until the motor stops running before switching to the Ready state. If an error occurs, it will enter the Fault state.

3.3 Top-layer Peripheral Configuration

3.3.1 PWM Output Configuration

void Drv_Pwm_Init(uint16_t u16_Period,uint16_t u16_DeadTime)

(1) The general configuration of PWM is as follows:

Set the PWM clock frequency division to 1, select the center-aligned mode 2, and set the repeat counter to 1, as shown in the figure below.

Figure 13 General Configuration of PWM

```
    /* Time Base configuration ,init time1 freq*/
    TIM_TimeBaseInitStructure.period          = u16_Period;
    TIM_TimeBaseInitStructure.div             = 0;
    TIM_TimeBaseInitStructure.counterMode     = TMR_COUNTER_MODE_CENTERALIGNED2;
    TIM_TimeBaseInitStructure.clockDivision   = TMR_CKD_DIV1;
    TIM_TimeBaseInitStructure.repetitionCounter = 1;
    TMR_ConfigTimeBase(TMR1, &TIM_TimeBaseInitStructure);
```

Figure 14 Center-aligned Mode Selection

Center-Aligned Mode Select

In the Center-aligned mode, the counter counts up and down alternately; otherwise, it will only count up or down. Different Center-aligned modes affect the timing of setting the output comparison interrupt flag bit of the output channel to 1; when the counter is disabled (CNTEN=0), select the Center-aligned mode.

00: Edge-alignment mode

01: Center-aligned mode 1 (the output comparison interrupt flag bit of output channel is set to 1 when counting down)

10: Center-aligned mode 2 (the output comparison interrupt flag bit of output channel is set to 1 when counting up)

11: Center-aligned mode 3 (the output comparison interrupt flag bit of output channel is set to 1 when counting up/down)

(2) PWM Output Status Configuration

Set the output status of the upper and lower tubes of PWM and enable the configuration of PWM output of the upper and lower tubes to be effective,

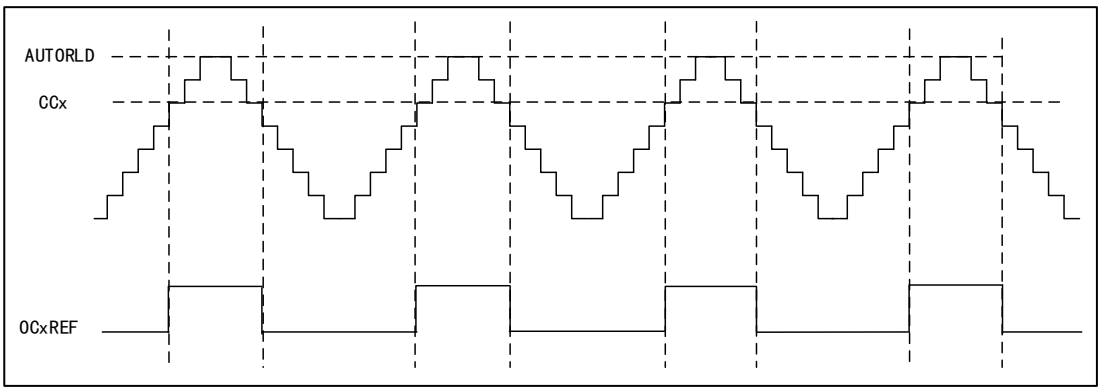
Configure the enabled brakes, configure the brake polarity, disable the automatic output, and prevent automatic PWM output in the next update event.

Figure 15 PWM Output Status Configuration

```
/* Automatic Output enable, Break, dead time and lock configuration*/
TIM_BDTRInitStructure.RMOS_State       = TMR_RMOS_STATE_ENABLE;
TIM_BDTRInitStructure.IMOS_State       = TMR_IMOS_STATE_ENABLE;
TIM_BDTRInitStructure.lockLevel        = TMR_LOCK_LEVEL_OFF;
TIM_BDTRInitStructure.deadTime         = u16_DeadTime;//
/**
  * Brake configuration: enable brake
  * Brake input polarity: active in low level
  * Auto output enable configuration: Disable MOE bit hardware control
  */
TIM_BDTRInitStructure.breakState       = TMR_BREAK_STATE_ENABLE;
TIM_BDTRInitStructure.breakPolarity    = TMR_BREAK_POLARITY_LOW;
TIM_BDTRInitStructure.automaticOutput  = TMR_AUTOMATIC_OUTPUT_DISABLE;
TMR_ConfigBDT(TMR1, &TIM_BDTRInitStructure);

/*pwm driver set, channel 1,2,3,4set pwm mode*/
TIM_OCInitStructure.OC_Mode        = TMR_OC_MODE_PWM2;
TIM_OCInitStructure.OC_OutputState = TMR_OUTPUT_STATE_ENABLE;  //TMR_OUTPUT_STATE_DISABLE;
TIM_OCInitStructure.OC_OutputNState= TMR_OUTPUT_NSTATE_ENABLE; //TMR_OUTPUT_NSTATE_DISABLE;
TIM_OCInitStructure.Pulse          = 0;
TIM_OCInitStructure.OC_Polarity    = TMR_OC_POLARITY_HIGH;
TIM_OCInitStructure.OC_NPolarity   = TMR_OC_NPOLARITY_HIGH; //
TIM_OCInitStructure.OC_Idlestate   = TMR_OCIDLESTATE_RESET;  // TMR_OCIDLESTATE_SET; //
TIM_OCInitStructure.OC_NIdlestate  = TMR_OCNIDLESTATE_RESET; // TMR_OCNIDLESTATE_SET;//
```

Figure 16 Timing Diagram of PWM2 Center-aligned Mode



In count-up mode, when TMR1_CNT<TMR1_CCR1, Channel 1 is invalid level; otherwise it is valid level;

In count-down mode, when TMR1_CNT>TMR1_CCR1, Channel 1 is a valid level; otherwise it is an invalid level.

3.3.2 ADC Configuration

void Drv_Adc_Init(void)

(1) ADC underlying configuration

DMA mode is adopted, and the quantized data of ADC is directly transported to the ADC_ConvertedValue array for storage. The ADC trigger condition uses CC4 of TMR1 as the trigger source, to enable ADC and configure ADC interrupt priority and its enable. Details are shown below:

Figure 17 ADC Underlying Configuration

```
void Drv_Adc_Init(void)
{
    ADC_Config_T    ADC_InitStructure;
    DMA_Config_T    DMA_InitStructure;
    DMA_InitStructure.peripheralAddress  = (uint32_t)&(ADC->DATA);//ADC address
    DMA_InitStructure.memoryAddress      = (uint32_t)&ADC_ConvertedValue[0]; //memory address
    DMA_InitStructure.direction          = DMA_DIR_PERIPHERAL; //Direction (from peripherals to memory)
    DMA_InitStructure.bufferSize         = TOTAL_CHANNEL;//TOTAL_CHANNEL; //The size of the transferred content -- the number of transfers
    DMA_InitStructure.peripheralInc      = DMA_PERIPHERAL_INC_DISABLE; //The peripheral address is fixed
    DMA_InitStructure.memoryInc          = DMA_MEMORY_INC_ENABLE;//DMA_MEMORY_INC_ENABLE;
    DMA_InitStructure.peripheralDataSize = DMA_PERIPHERAL_DATASIZE_HALFWORD ; //Peripheral data unit
    DMA_InitStructure.memoryDataSize     = DMA_MEMORY_DATASIZE_HALFWORD ;    //Memory data unit
    DMA_InitStructure.circular           = DMA_CIRCULAR_ENABLE  ; //DMA mode: cyclic transmission
    DMA_InitStructure.priority           = DMA_PRIORITY_LEVEL_VERYHIGH ; //Priority: High
    DMA_InitStructure.memoryTomemory     = DMA_M2M_DISABLE;    //Memory to memory transmission is disabled
    ADC_Reset();
    DMA_Config(DMA_CHANNEL_1, &DMA_InitStructure);  //Configure channel 1 for DMA
    DMA_Enable(DMA_CHANNEL_1);
    ADC_ClockMode(ADC_CLOCK_MODE_ASYNCLK);//48M/4=12mADC_CLOCK_MODE_SYNCLKDIV4
    ADC_ConfigStructInit(&ADC_InitStructure);
    ADC_InitStructure.convMode      = ADC_CONVERSION_SINGLE;
    ADC_InitStructure.scanDir       = ADC_SCAN_DIR_UPWARD;
    ADC_InitStructure.extTrigConv1  = ADC_EXT_TRIG_CONV_TRG1;   //  timer1 CC4
    ADC_InitStructure.extTrigEdge1  = ADC_EXT_TRIG_EDGE_RISING;
    ADC_InitStructure.dataAlign     = ADC_DATA_ALIGN_RIGHT;
    ADC_InitStructure.resolution    = ADC_RESOLUTION_12B;
    ADC_Config(&ADC_InitStructure);
    ADC_ConfigChannel(ADC_CHANNEL_2 | ADC_CHANNEL_8 | ADC_CHANNEL_9 | ADC_CHANNEL_7 ,ADC_SAMPLE_TIME_1_5);
    ADC->CFG1_B.OVRMAG = 1;
    ADC_EnableInterrupt(ADC_INT_CS);
//=======================ADC interrupt=====================================================
    NVIC_EnableIRQ(ADC_COMP_IRQn);
    NVIC_SetPriority(ADC_COMP_IRQn,0);
    ADC_DMARequestMode(ADC_DMA_MODE_CIRCULAR);
    ADC_EnableDMA();
    ADC_Enable();
    ADC_StartConversion();//Gotta start it up
```

3.3.3 OPA and COMP Underlying Configuration

(1) OPA underlying configuration

To configure the underlying configuration of OPA, first configure the OPA pin, DISABLE the operational amplifier OPA, configure to use an external resistor network, and then ENABLE it, as shown in the figure below;

Figure 18 OPA Underlying Configuration

```
void OPA_Init(void)
{
    OPA_Disable(OPA1);
    OPA_Disable(OPA2);
    OPA_Disable(OPA3);
    OPA_Disable(OPA4);
    OPA_SelectGainFactor(OPA1,OPA_GAIN_FACTOR_0);
    OPA_SelectGainFactor(OPA2,OPA_GAIN_FACTOR_0);
    OPA_SelectGainFactor(OPA3,OPA_GAIN_FACTOR_0);
    OPA_SelectGainFactor(OPA4,OPA_GAIN_FACTOR_0);
    OPA_Enable(OPA1);
    OPA_Enable(OPA2);
    OPA_Enable(OPA3);
    OPA_Enable(OPA4);
}
```

(2) COMP underlying configuration

COMP is used for overcurrent anomaly detection. To configure the underlying configuration of COMP, first configure the COMP pin, set the COMP output to the BKIN connected to TMR1, set

the output reverse, and trigger the BKIN of TMR1 at a low level, as shown in the following figure;

Figure 19 COMP Underlying Configuration

```
void COMP_Init(void)
{
    COMP_Config_T    compConfig;
    /* Configure COMP1 */
    COMP_ConfigStructInit(&compConfig);
    compConfig.invertingInput = COMP_INVERTING_INPUT_PA1;
    compConfig.output         = COMP_OUTPUT_TIM1BKIN;
    compConfig.outputPol      = COMP_OUTPUTPOL_NONINVERTED;
    compConfig.hysterrsis     = COMP_HYSTERRSIS_NO;
    compConfig.mode           = COMP_MODE_HIGHSPEED;
    COMP_Config(COMP_SELECT_COMP1,&compConfig);
    /* Enable COMP2 */
    COMP_Enable(COMP_SELECT_COMP1);
}
```

3.4 Calibration Standardization

3.4.1 Concept of Per Unit

There are typically two methods to define the magnitude of a variable:

Firstly, the nominal value: the value obtained from measuring the variable using instruments and meters. This employs the International System of Units (SI), voltage (V), current (A), and rotational speed (r/min).

Secondly, the per unit value: the nominal value divided by the reference value. This uses the per-unit system (p.u).

Advantages of the per-unit system include compatibility with fixed-point MCU operations and prevention of data overflow, among others.

3.4.2 Per Unitization of the Entire System Software

All are calibrated to the Q15 format.

Firstly, voltage calibration: based on the actual maximum measurable bus voltage as the reference value, it is mapped to a voltage range of 0-3.3V at the actual ADC pin. The corresponding mapping in the program is 0-32767, as shown in the figure below:

Figure 20

```
/*Voltage Sampling*/
#define UDC_MAX                 (69.0f)     //  unit:V      Max DC Voltage of Hardware
#define DCBUS_OVER              Q15(48.0f/UDC_MAX)     //  unit:V
#define DCBUS_UNDER             Q15(20.0f/UDC_MAX)     //  unit:V
```

Secondly, current calibration: based on the actual maximum measurable bus current as the reference value, the voltage mapped to the actual ADC pin is 0-3.3V, and the corresponding mapping in the program is 0-32767

Figure 21

```
/*Current Sampling*/
#define ADC_REFV                SYS_REFV
#define R_SHUNT                 (0.02f)     //  unit:ohm
#define CURRENT_OPA_GAIN        (4.86f)     //  unit:ohm Using the combination of 2K/10K resistors, combined with the internal coupling resistance calculation
#define I_MAX                   (16.46f)    //  Offset 1.6V, I MAX=1.6/Gain/R
/*  I/Ibase *Rs*OP_Gain = (V-Voffset)/32768 * 3.3     On the right side is the actual voltage
    IGAIN = 3.3/OP_Gain/Rs/Ibase                      The bias has been removed
*/
#define IGAIN_Q10                                     (2113)
```

Thirdly, angle calibration: 0~ Π= 0- 32767

Fourthly, speed calibration: based on the actual rated speed, select the appropriate speed reference value, and the corresponding mapping in the program is 0-32767

Figure 22

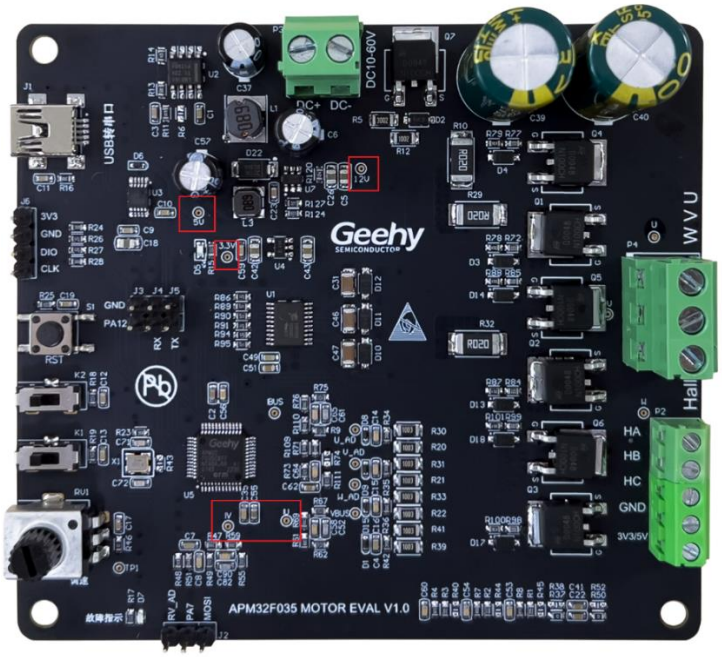
```
#define SPEED_CALIBRATION           (5000.0f)     // unit:rpm    rated speed of motor
#define MAX_SPEED                   (MAX_RPM)     // unit:rpm     max speed of motor
#define OVER_SPEED_VALUE           Q15(3500.0f/SPEED_CALIBRATION)
```

3.5 Debugging Development

3.5.1 Hardware Confirmation

Firstly, it is necessary to verify that the hardware of the board is functioning properly. After the initial power-up, use an oscilloscope to check whether the voltage points on the hardware board are stable (12V, 5V, 3.3V, etc.), and measure if points such as IV, IU, and IBUS have a DC bias voltage of 1.60V, as shown in the hardware schematic below.

Figure 23



3.5.2 Peripheral Configuration

Verify the configuration of the motor's hardware peripheral pins, such as the M0CP co-processor, the PWM wave generation configuration of TMR1, the capture configuration of TMR2, the ADC sampling port configuration, the initialization of the op-amp OPA ports, and the configuration of the comparator COMP. Moreover, implement the core feature of utilizing both COMP and TMR1 in cascade to enable the braking function (i.e., hardware overcurrent protection feature; details can be referenced in the schematic diagram). The details are as follows: (Note: The specifics of each detail are not elaborated here and should be cross-referenced with the schematic diagram and the configuration files for each peripheral.)

Figure 24

```
    /* M0CP */
    M0CP_Firmware_Init();
    RCM_EnableAHBPeriphReset(RCM_AHB_PERIPH_M0CP);
    RCM_DisableAHBPeriphReset(RCM_AHB_PERIPH_M0CP);
    RCM_EnableAHBPeriphClock(RCM_AHB_PERIPH_M0CP);
    M0CP_HardInit();
    *(volatile unsigned int*)(0x40024000+0x10) = 0;
    /* TIMER1 */
    Drv_Pwm_Init(PWM_PERIOD,DEAD_TIME);
    PWM_CompareConfig(PWM_PERIOD, PWM_PERIOD, PWM_PERIOD);
    /* TIMER2 for hall capture */
    timer2_init(TIM2_PRIOD, TIM2_PSC_LOAD);    //The overflow period is 100ms
    /* ADC */
    Drv_Adc_Init();
    /* OPA */
    OPA_Init();
    /* COMP */
    COMP_Init();
    /* GPIO */
    IO_Init();
    /* SYSTICK */
    Systick_Init(SystemCoreClock / 1000);
```

3.5.3 Motor Parameter Configuration

The first two steps mainly involve checking the hardware and peripheral configuration to ensure the stability and accuracy of the hardware foundation before starting the motor tuning process.

Open the "parameter.h" configuration file; this file is of paramount importance as most of the modifications will be done here. Emphasize this part! Firstly, it is essential to verify the motor parameters.

This section primarily introduces the development of sensor-equipped Field Oriented Control (FOC), and therefore, it is necessary to check the motor's pole pair number, rated speed, current limit (considering the motor's power), and calibration speed (evaluating the rated speed), as shown in the figure below.

(Note: It is worth mentioning that sensor-equipped FOC does not require the use of an observer, so there is no need for the motor's phase resistance, phase inductance, and other parameters. In contrast, for sensorless FOC, it is typically necessary to know the motor's phase resistance, phase inductance, and even the back electromotive force constant, etc.)

Figure 25

```
/*motor releated paremeter setting*/
/*  Motor parameter  */
#define POLEPAIRS                               2u          //2.0f          //  unit
#define SPEED_CALIBRATION                       (4000.0f)   //unit:rpm    rated speed of motor
#define MAX_SPEED                               (MAX_RPM)   //unit:rpm    max speed of motor
#define OVER_SPEED_VALUE                        Q15(3100.0f/SPEED_CALIBRATION)

#define FREQ2RPM_Q2                             (240/POLEPAIRS)

#define MAX_DUTY                                (0.95f)    //  unit
#define SPD_PI_LIMIT                            (1.5f)     //  unit:A      Speed PI output limitation
```

3.5.4 Tuning and Confirmation of PI Parameters for the Current Inner Loop

Here comes the critical part!

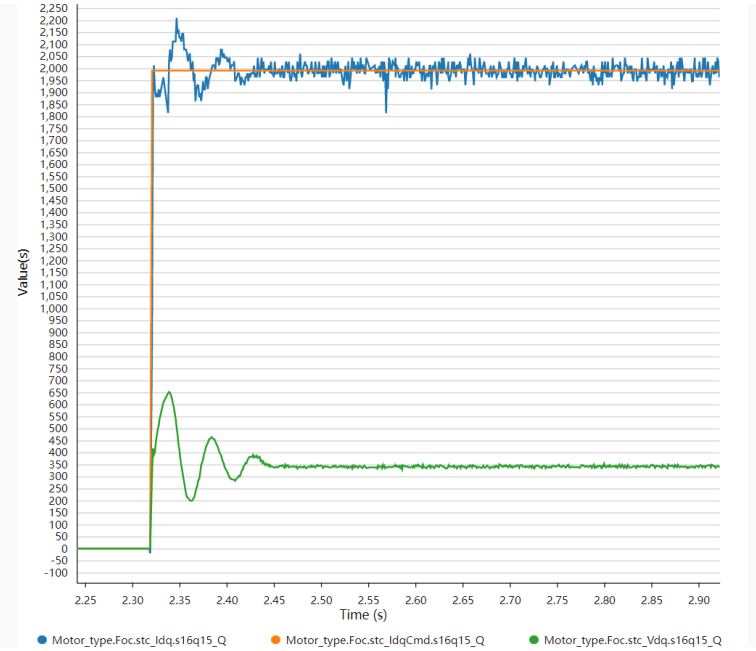
During the Align state, you should debug by adjusting the PI parameters of the current loop under the DQ axis (as shown in the figure below, generally the DQ axis uses the same PI parameters). Similar to the Q axis shown, by setting the Iq_cmd (which can be given a direct value or set with a ramp to specify a certain acceleration and value), you can determine whether the current PI parameters are appropriate by observing whether the Vq output can quickly and stably settle, and also by checking if the given Iq_cmd and the actual Iq waveform data can follow suit. An example waveform is shown in the figure.

(Note: Normally when tuning PI, you start from small to large, first adjusting KP and then adding KI for debugging. For specific methods, you can refer to online resources.)

Figure 26

```
/*  control parameter      */
/*  CURRENT LOOP PI PARAMETER   */
#define M1_IQ_KP_Q15                    (4000)      //  Q15 format KP of Q-axis Current loop
#define M1_IQ_KP_Q10                    (0)         //  Q15 format KP of Q-axis Current loop
#define M1_IQ_KI_Q15                    (400)       //  Q15 format KI of Q-axis Current loop
#define M1_IQ_KI_Q10                    (0)         //  Q15 format KI of Q-axis Current loop
#define M1_ID_KP_Q15                    (4000)      //  Q15 format KP of D-axis Current loop
#define M1_ID_KP_Q10                    (0)         //  Q15 format KP of D-axis Current loop
#define M1_ID_KI_Q15                    (400)       //  Q15 format KI of D-axis Current loop
#define M1_ID_KI_Q10                    (0)         //  Q15 format KI of D-axis Current loop
```

Figure 27



3.5.5 Self-Check of HALL Angle

When the current inner loop PI parameters are already appropriate, it indicates that the pre-positioning function can be used; therefore, the self-check function of the HALL phase sequence should be enabled.

Still in the "parameter.h" configuration file, enable the "HALL_PHASE_TEST" macro to activate the self-check function for the HALL phase sequence. Let's introduce the role of the HALL phase sequence: the difference in HALL phase angle is 60°, meaning it can output six angle points, similar to the sequence of HALL values output in a clockwise (CW) direction, such as 2-6-4-5-1-3. (Note: You can first test the HALL sequence order to see if it is consistent with the sequence given in the program by manually rotating the motor one turn to observe the actual HALL values read. Confirm the sequence of HALL values for both CW and counterclockwise (CCW) rotations, and verify the corresponding HALL phase sequence. For example, if the sequence read is 6-4-5-1-3-2, which matches the preset sequence, then there is no need for modification. However, if a sequence like 6-4-1-5-2-3 is observed, which does not match the sequence in the program, enter this sequence into the arrays u8CW_Hall_Value and u8CCW_Hall_Value in the user_function.c file. Also, correspondingly fill in the CW_hall_table

and CCW_hall_table arrays.

u8CW_Hall_Value[6]: Represents the actual Hall sequence; note the relationship between the Hall sequence and the angle sequence.

CW_hall_table[8]: The sequence of Hall values at the prior position. For example, in the program, the prior position to Hall value 1 is Hall value 5; thus, CW_hall_table[1] = 5, and the prior position to Hall value 2 is Hall value 3; thus, CW_hall_table[2] = 3.

And so on: u8CW_Hall_Value[6] = {1, 3, 2, 6, 4, 5}, corresponding to CW_hall_table[8] = {0, 5, 3, 1, 6, 4, 2, 0}; u8CW_Hall_Value[6] = {2, 3, 1, 6, 4, 5}, corresponding to CW_hall_table[8] = {0, 3, 5, 2, 6, 4, 1, 0}).

Figure 28

```
/* The following is the normal HALL phase sequence, which can be modified according to the actual motor situation */
uint8_t u8CW_Hall_Value[6] = {1,3,2,6,4,5};
uint8_t u8CCW_Hall_Value[6] = {1,5,4,6,2,3};

uint8_t CCW_hall_table[8] = {0,3,6,2,5,1,4,0};
uint8_t CW_hall_table[8] = {0,5,3,1,6,4,2,0};
```

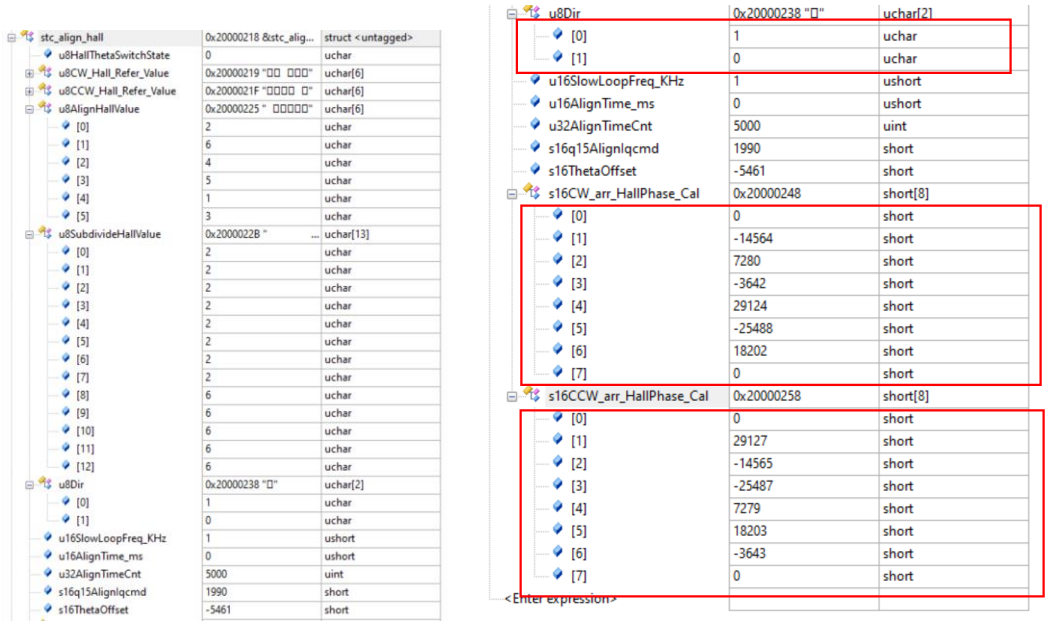
Furthermore, the focus comes again! There is indeed an installation angle deviation between the actual position of the HALL sensor and the corresponding mechanical angle of the motor, which is the machine angle deviation angle. The self-check function of the HALL sequence is designed to calculate this deviation angle and match the actual angle corresponding to the HALL value accordingly. As shown in the figure, after enabling the macro "HALL_PHASE_TEST," enter the debug mode and observe the related member variables of the structure "stc_align_hall." Implement the configuration functions "Align_HallCal_Phase_Init" and "Align_HallCal_Phase." The main modifications to the core parameters involve setting the positioning time and positioning Iq current, as well as setting the offset angle (if the detected angle is abnormal, the value can be adjusted appropriately. The recommended offset angle is 0 " 0° " or 5461 " 30° "). For example, in "stc_align_hall," "u8Dir" indicates direction, and "s16CW_arr_HallPhase_Cal" and "s16CCW_arr_HallPhase_Cal" are used to generate the calculated HALL phase angle sequence table, which must be filled into the array lists "s16CW_arr_HallPhase" and "s16CCW_arr_HallPhase" in the file "user_function.c." As shown in the figure.

Figure 29

```
#define HALL_PHASE_TEST                         1       //1: Enable the HALL phase sequence detection function
#define HALL_PHASE_ALIGN_TIME_MS                5000    //hall phase sequence positioning time: time = HALL_PHASE_ALIGN_TIME * SLOWLOOP_FREQ_KHZ
#define HALL_PHASE_ALIGN_IQ_CMD                 Q15(1.0/I_MAX) //Locating Iq current
#define HALL_PHASE_THETA_OFFSET                 (-5461) //The offset Angle used here can be combined with the actual motor, recommended 0 or 5461,
    /*******************************************************************************************************************
**Func name: Align_HallCal_Phase_Init()
**Param: Parameters: 1, stc_align_hall_para_t structure 2, slow loop frequency 3, each positioning time 4, positioning Iq current 5, offset Angle
                    6, CW reference array 7, CCW reference array

**Return: Returns 1 if initialization is normal, and others if initialization fails
**Description: HALL phase Angle calculation initialization
*******************************************************************************************************************/
#if HALL_PHASE_TEST
        /*for get hall phase*/
        Align_HallCal_Phase_Init(&stc_align_hall,SLOWLOOP_FREQ_KHZ,HALL_PHASE_ALIGN_TIME_MS,HALL_PHASE_ALIGN_IQ_CMD,HALL_PHASE_THETA_OFFSET,u8CW_Hall_Value,u8CCW_Hall_Value);
#endif

}
```



stc_align_hall	0x20000218 &stc_alig...	struct <untagged>
u8HallThetaSwitchState	0	uchar
u8CW_Hall_Refer_Value	0x20000219 "□□ □□□"	uchar[6]
u8CCW_Hall_Refer_Value	0x2000021F "□□□□ □"	uchar[6]
u8AlignHallValue	0x20000225 " □□□□□"	uchar[6]
[0]	2	uchar
[1]	6	uchar
[2]	4	uchar
[3]	5	uchar
[4]	1	uchar
[5]	3	uchar
u8SubdivideHallValue	0x2000022B "	uchar[13]
[0]	2	uchar
[1]	2	uchar
[2]	2	uchar
[3]	2	uchar
[4]	2	uchar
[5]	2	uchar
[6]	2	uchar
[7]	2	uchar
[8]	6	uchar
[9]	6	uchar
[10]	6	uchar
[11]	6	uchar
[12]	6	uchar
u8Dir	0x20000238 "□"	uchar[2]
[0]	1	uchar
[1]	0	uchar
u16SlowLoopFreq_KHz	1	ushort
u16AlignTime_ms	0	ushort
u32AlignTimeCnt	5000	uint
s16q15Alignlqcmd	1990	short
s16ThetaOffset	-5461	short

u8Dir	0x20000238 "□"	uchar[2]
[0]	1	uchar
[1]	0	uchar
u16SlowLoopFreq_KHz	1	ushort
u16AlignTime_ms	0	ushort
u32AlignTimeCnt	5000	uint
s16q15Alignlqcmd	1990	short
s16ThetaOffset	-5461	short
s16CW_arr_HallPhase_Cal	0x20000248	short[8]
[0]	0	short
[1]	-14564	short
[2]	7280	short
[3]	-3642	short
[4]	29124	short
[5]	-25488	short
[6]	18202	short
[7]	0	short
s16CCW_arr_HallPhase_Cal	0x20000258	short[8]
[0]	0	short
[1]	29127	short
[2]	-14565	short
[3]	-25487	short
[4]	7279	short
[5]	18203	short
[6]	-3643	short
[7]	0	short
<Enter expression>		

```
int16_t s16CW_arr_HallPhase[8] = {0,-14564,7280,-3642,29124,-25488,18202,0};
int16_t s16CCW_arr_HallPhase[8] = {0,28218,-15474,-26396,6370,17294,-4552,0};
```

3.5.6 Speed Loop Parameter Tuning

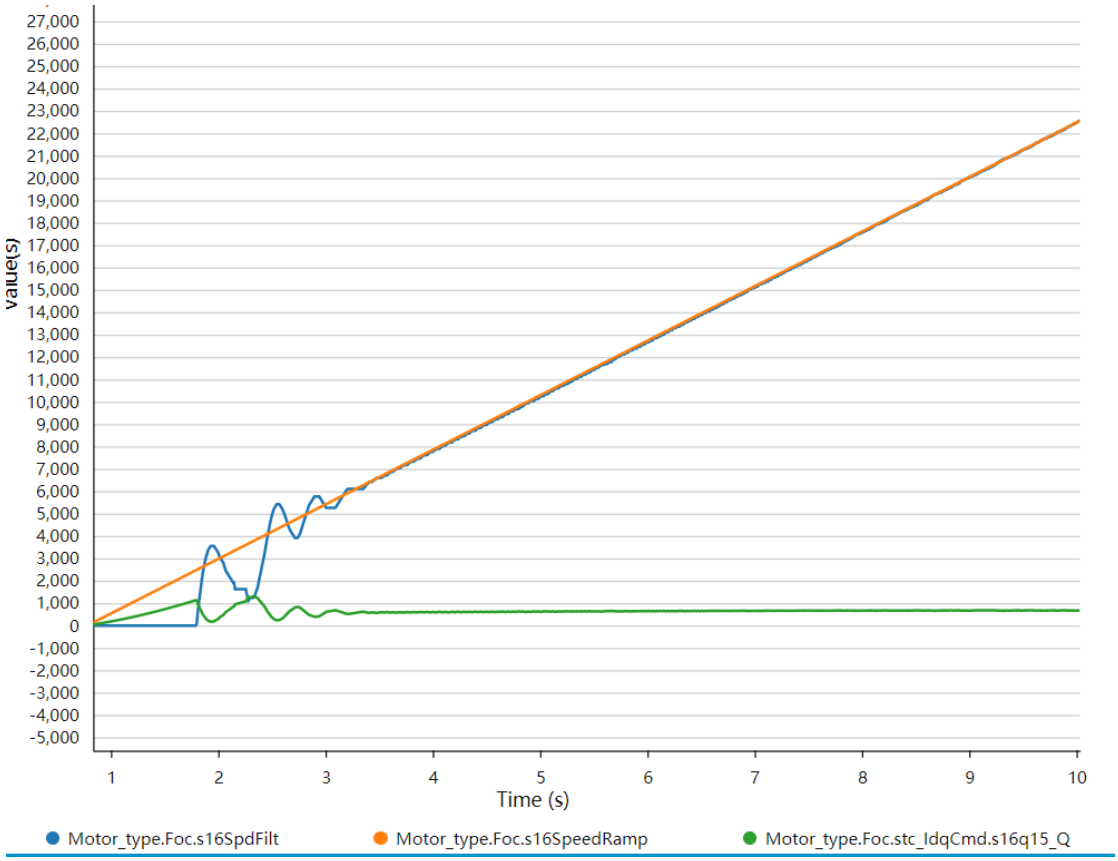
Once the HALL phase sequence angle table has been acquired and entered, you need to proceed to the "parameter.h" configuration file and first turn off the "HALL_PHASE_TEST" macro parameter, disabling the HALL self-check mode. At this point, you're ready to try and get the motor running for a test!

Adjust the speed knob on the hardware board, and under normal circumstances, the motor should start running. If it does not start, you'll need to backtrack and verify the previous steps to identify any anomalies. If the motor runs but the speed is unstable, then you've reached a crucial point! Start taking notes! When the speed is unstable, you will need to further adjust the speed loop's PI parameters and verify whether the acceleration set for the motor is reasonable. The adjustment of the PI is similar to the mode used previously for tuning the current loop PI: start with tuning Kp before introducing Ki, and progressively validate through testing with increasing values. The parameter's appropriateness is judged by observing the stability trend of the actual output Iq_CMD, and by comparing the given speed ramp spd_ramp with the actual filtered speed spd_filt through their following curves, as shown in the diagram below.

Figure 30

```
/*  SPEED LOOP PI PARAMETER  */
#define M1_SPEED_KP_Q15                (10000)          //  Q15 format KP of speed loop
#define M1_SPEED_KP_Q10                (4048)           //  Q10 format KP of speed loop
#define M1_SPEED_KI_Q15                (10)             //  Q15 format KI of speed loop
#define M1_SPEED_KI_Q10                (9)              //  Q10 format KI of speed loop

/*run state */
#define SPIN_SPD_INC                   1000.0f     //  RPM/s ramp of speed command increase
#define SPD_DEC                        1000.0f     //  RPM/s ramp of speed command decrease
```



3.5.7 Custom Motor Development

From now on, the motor has been successfully operated by you. What follows is to add application layer logic and other customization developments according to your actual development needs!!!

3.6 Settings of Key Parameters

All parameters in this system are configured in parameter.h of the user layer, mainly including system parameters, related parameters of a backplane, related parameters of a state machine, and related parameters of a motor, as follows:

3.6.1 System Parameters

Table 3 System Parameters

Parameter name	Parameter description	Set value
SYS_REFV	Supply voltage of the system	3.3 (V)
SYSCLK_HSE_72MHz	Main frequency of the system	72000000 (Hz)
PWMFREQ	PWM frequency	8000 (Hz)

Parameter name	Parameter description	Set value
DEAD_TIME	PWM dead band time	1.0 (μs)
SLOWLOOP_FREQ	Control frequency of slow loop	1000 (Hz)

3.6.2 Backplane Hardware Parameters

Table 4 Parameters of Backplane Hardware

Parameter name	Parameter description	Set value
ADC_REFV	ADC reference voltage	3.3 (V)
R_SHUNT	Sampling resistance value	0.02 (Ω)
CURRENT_OPA_GAIN	Amplification factor of operational amplifier	5.0
I_MAX	Current standardization reference value	16.5 (A)
UDC_MAX	Voltage standardization reference value	69.0 (V)
U_MAX	Phase voltage standardization reference value	39.83 (V)

3.6.3 Motor Related Parameters

Table 5 Motor-Related Parameters

Parameter name	Parameter description	Set value
Rs	Phase resistance of motor	0.15 (ohm)
Ls	Phase inductance of motor	0.00037 (H)
POLEPAIRS	Number of motor pole-pairs	2 (unit)
M1_IQ_KP_Q15	Q-axis current loop KP parameter Q15 format	25000
M1_IQ_KI_Q15	Q-axis current loop KI parameter Q15 format	8
M1_ID_KP_Q15	D-axis current loop KP parameter Q15 format	25000
M1_ID_KI_Q15	D-axis current loop KI parameter Q15 format	8
M1_SPEED_KP_Q15	Speed loop KP parameter Q15 format	16384
M1_SPEED_KI_Q15	Speed loop KI parameter Q15 format	163

3.7 Precautions

(1) Check hardware connections: Check if the connections between the motor, motor driver, and controller are correct, and if the power supply is stable, and ensure that all interfaces are inserted and tightened and free of damage or short circuits.

(2) Configuration parameters: Configure the parameters of the controller according to the specifications of the motor and driver, such as the rated current of the motor, the motor

parameters, the number of pole pairs of the motor, and the maximum speed of the motor. Ensure that all parameters are set correctly.

(3) Conduct a no-load test: Conduct a load test with no motor load installed, to check whether the motor can start and rotate normally and whether the speed meets the requirements. Note: It is necessary to verify whether the HALL operation sequence of the motor is correct. An oscilloscope is used to collect the HALL signal port. Manually rotate the motor for one turn to confirm the forward and reverse operation sequence of the HALL.

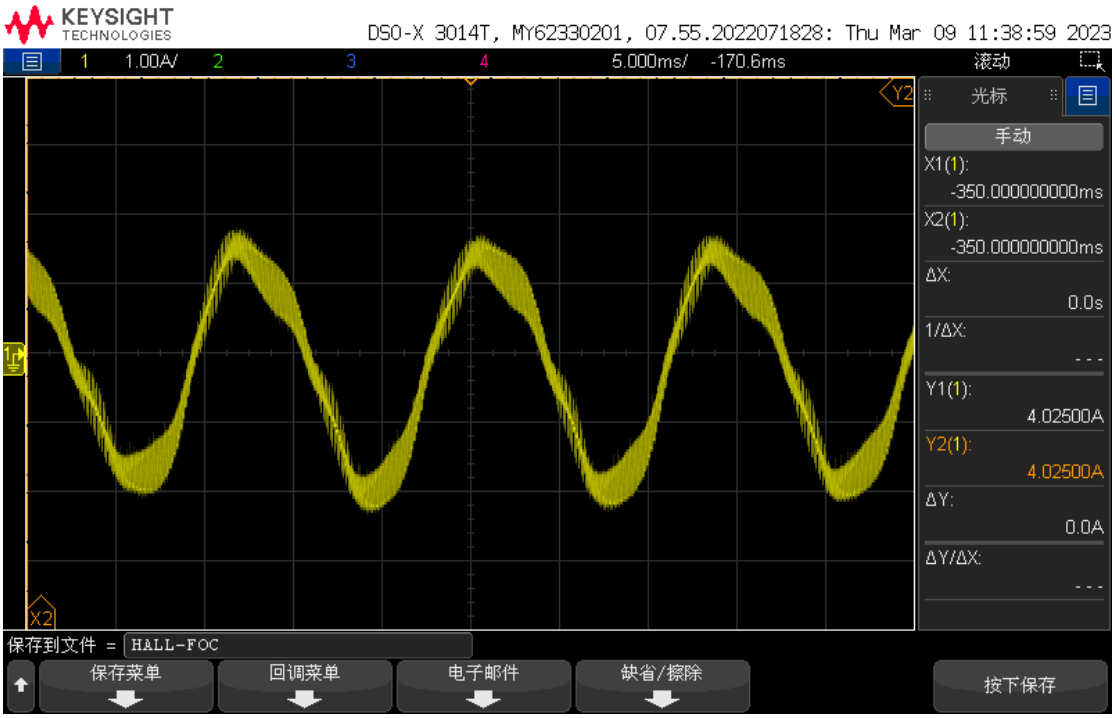
(4) Conduct load test: Conduct a load test with the motor load installed, to check the performance of the motor under load, such as speed and torque.

(5) Conduct speed PID debugging: Use a PID regulator to control the response speed of the motor. The response and stability of the motor can be optimized by changing the PID parameters (note: all parameters are modified in parameter.h). During debugging, the experimental results should be recorded for future reference.

(6) Conduct performance tests: After the above steps are completed, some performance tests can be conducted, such as measuring the maximum speed, maximum torque, and efficiency of the motor. Some testing equipment can be used for measurement, such as a tachometer, load tester, and power meter.

4 Actual test waveform

Figure 31 Actual Test Waveform



5 Revision History

Table 6 Document Revision History

Date	Revision	Revision History
July 26, 2023	1.0	New
August 14, 2023	1.1	(1) Modified the production information form (2) Modified the format
October 19, 2023	1.2	Version iteration supplement
January 16, 2024	1.3	(1) Modified the amplification derivation process of the operational amplifier. (2) Made some revisions to the wording and details in the description.

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as "Geehy"). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Please read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the "users") have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The "极海" or "Geehy" words or graphics with "®" or "™" in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party's products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party's products, services or intellectual property, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in thesales order or the

sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTIRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT

DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDE THE DOCUMENT "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES).

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.