Application Note

Debugging Manual for Sensorless Square-wave Driver of Electric Drill

Three-phase Motor Control
EU6861-Q MCU

Fortior Technology Co., Ltd.

Table of Content

1 Overview

This Debugging Manual is intended to guide you through the use of Fortior EU6861Q chip to implement Hall sensorless square-wave drive control for BLDC motor of electric drill on a DEMO board dedicated for electric tools. When reading the Manual, you may firstly browse through the hardware schematic diagrams in Section 2 and software schematic diagrams in Section 3, and then carefully read the debugging procedures explained in Section 4 and protection function test in Section 5 .

Relevant software/hardware

Software/hardware modules	Designation	Section	Remarks
Software		All	Debugging must be done by running this engineering software
Hardware		All	Debugging must be done on this hardware

2 Hardware

2.1 Hardware Mechanism

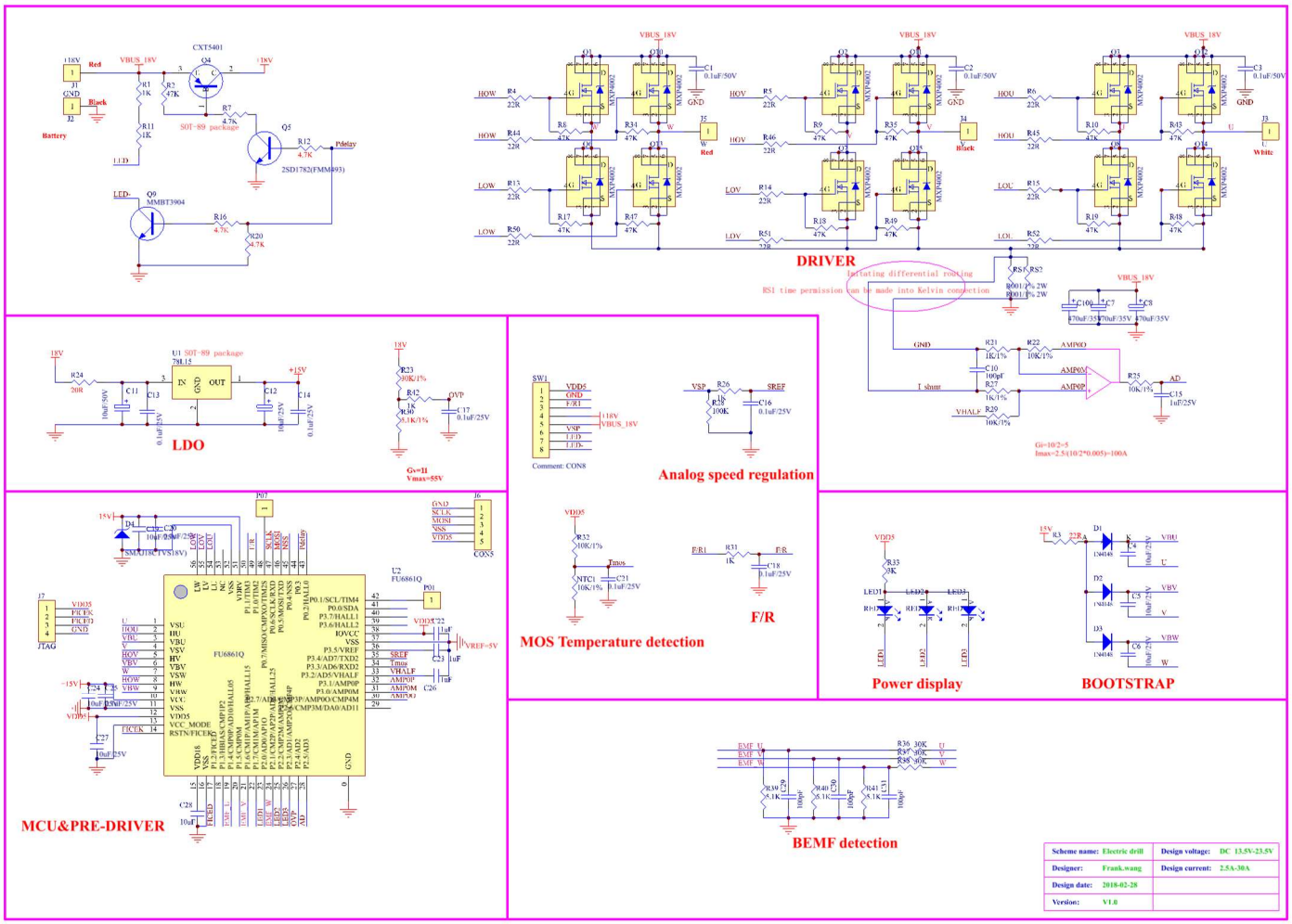


Figure 2- 1 Hardware Schematic Diagram

Directions for use:

It is a DEMO board dedicated for electric drill applications, which can be operational after power on.

Instructions:

The DC bus voltage divider ratio, Op-amp gain, shunt resistance and voltage divider ratio of BEMF (back electromotive force) sampling circuit should be configured properly, depending on the specific motor voltage and current.

2.1.1 Power Circuit

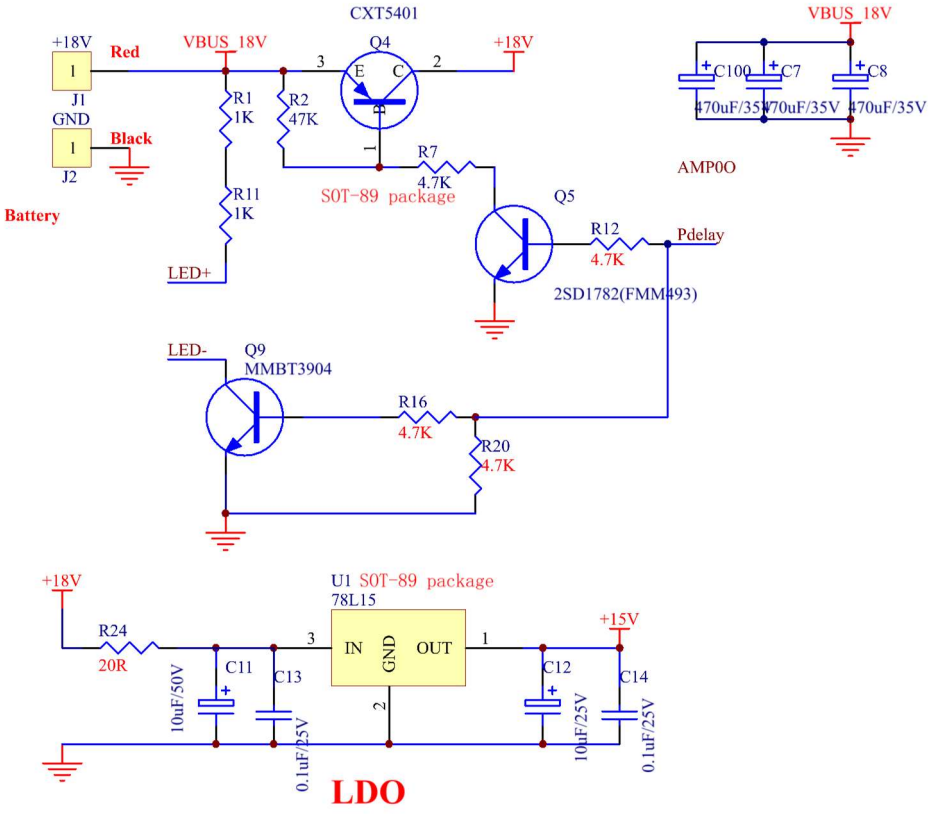


Figure 2-2 Power Supply Schematic

Directions for use:

Connect DC source positive cable to connector J1 and negative cable to connector J2

2.1.2 Chip Circuit

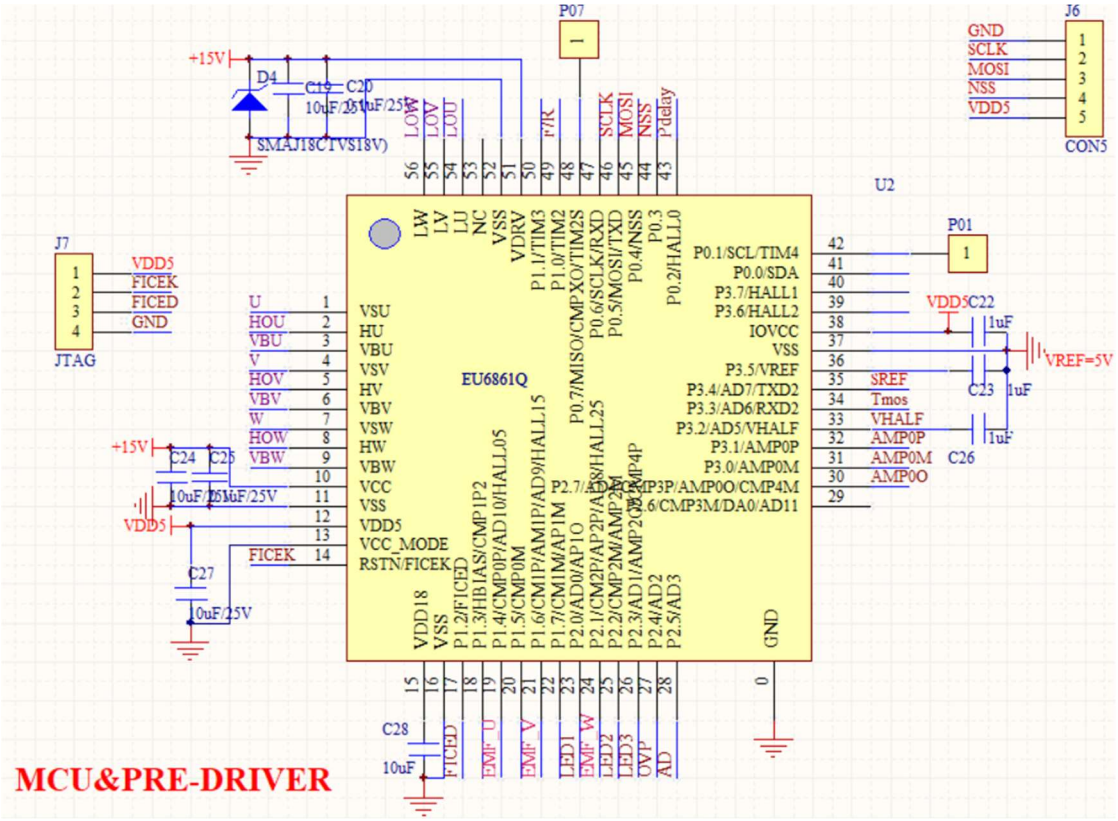


Figure 2-3 Chip Body Schematic

Directions for use:

EU6861Q chip is used for LV/MV 6-NMOSFET driver applications. J7 is a programming cable interface, and J6 is a SPI interface.

2.1.3 BEMF Detection Circuit

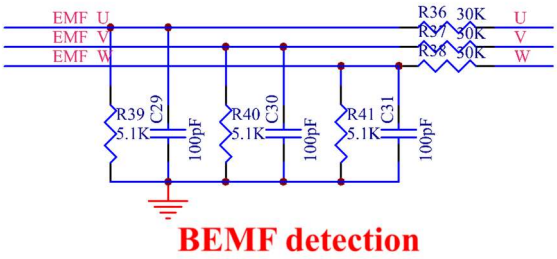


Figure 2- 4 BEMF Detection Circuit Schematic

The above circuit is designed for BEMF detection.

2.1.4 Power Driver Circuit

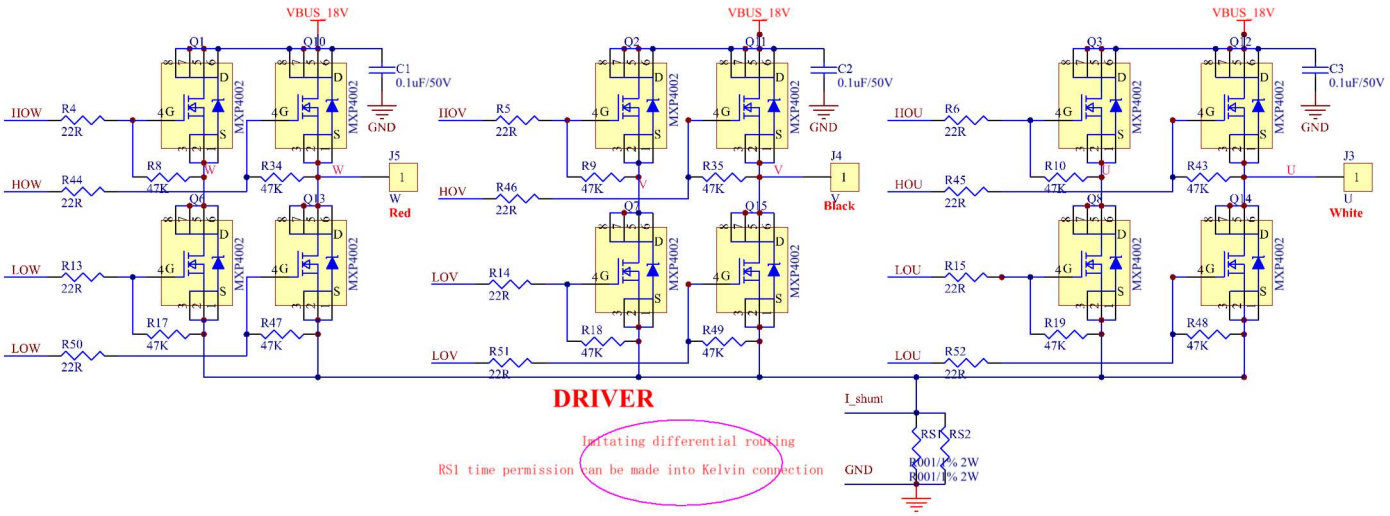


Figure 2-5 Power Driver Schematic

Instructions:

I shunt and GND should form pseudo-differential wiring, and Resistor RS1 wiring must be of kelvin connection type.

2.1.5 Op-amp Configuration Circuit

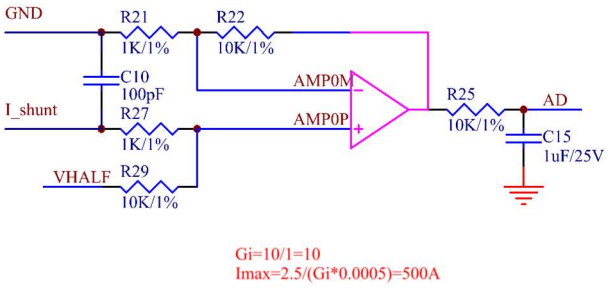


Figure 2-6 Op-amp Configuration Circuit Schematic

Instructions:

1. ±1% tolerance resistors are required for R21, R22 and R27 and R29;
2. Here, AD is used for sampling the average bus voltage;
3. Gain = R22/R21 = R29/R27;
4. Maximum sampling current = (VREF - VHALF)/gain/shunt resistor value (Imax=2.5/(10/2*0.005)=100A);

2.1.6 DC Bus Voltage Sampling Circuit

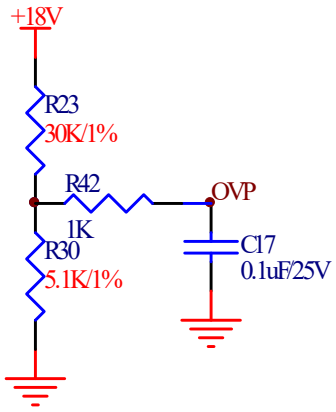


Figure 2-7 DC Bus Voltage Sampling Circuit Schematic

Instructions:

1. Parameters R23 and C30 are not adjustable;
2. ±1% tolerance resistors are required for R23 and R30;
3. Maximum sampling voltage = (R23 + R30)/(R30)*VREF;
4. Maximum sampling voltage selected is typically 2 times the maximum applied voltage, where the OVP threshold must be lower than 0.8*VREF.

3 Software Architecture

3.1 Motor State Machine Flowchart

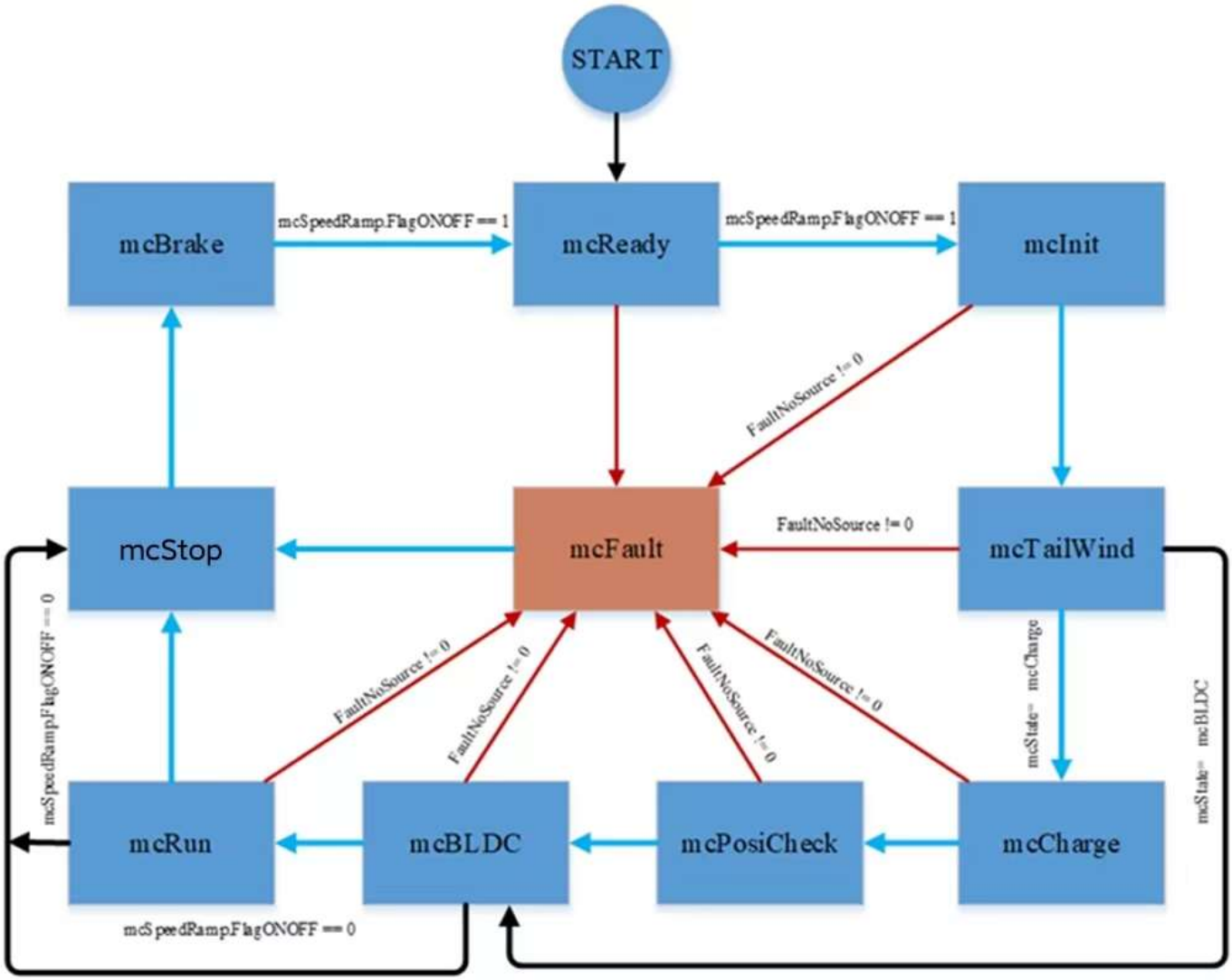


Figure 3-1 Motor State Machine Flowchart

As shown in the above figure, the motor state machine has three state transition paths:

1. Run: mcReady -> mcInit -> mcTailWind -> mcCharge -> mcPosiCheck -> mcBLDC -> mcRun;
2. Stop: The state machine will switch to mcStop state from mcBLDC and mcRun states to slow down motor till it comes to a stop if a Stop signal is detected;
3. Fault: The state machine will switch to mcFault state from all states in case of fault. In this state, no more fault detection will be performed, so multiple faults will not be reported simultaneously.

Description:

1. mcReady: Ready state, waiting for a "Start" command; the state machine will switch to mcInit state

if mcStart is enabled;

2. mcInit: Initialize driver variables and detect motor desired rotation before shifting to the next state;
3. mcTailWind: Tailwind/headwind detection state; when tailwind is detected, the state machine will shift to mcBLDC state directly; when windlessness is detected, motor will actuate as programmed.
4. mcCharge: Pre-charge state, where bootstrap capacitor is charged for subsequent MOSFET driver; when the charging is completed, motor will actuate as programmed;
5. mcPosiCheck: Initial position check state, where the position and angle of motor is detected. The state machine will switch to mcBLDC state upon completion of alignment.
6. mcBLDC: Software running state, where motor runs at a low speed and shifts to the next state mcRun after the speed builds up to the preset level;
7. mcRun: Hardware running state, where motor runs at a high speed;
8. mcStop: Stop state, which is used to stop the motor. Under this state, motor will slow down before shifting to mcBrake state;
9. mcBrake: Brake state, where motor will shift to mcReady state if mcStart is enabled;
10. mcFault: Fault state; when a protection event is detected, the program will record the fault source and the state machine will switch to mcFault state to shut down the motor for protection purpose; when the fault source is eliminated, it will shift to mcBrake state, waiting for the next "Start" command.

Instructions:

1. The state machine description contains 10 states , and transitions between states happen only in the intended direction. For example, motor can switch from mcReady state to mcInit and mcFault states only.

3.2 Program Flowchart

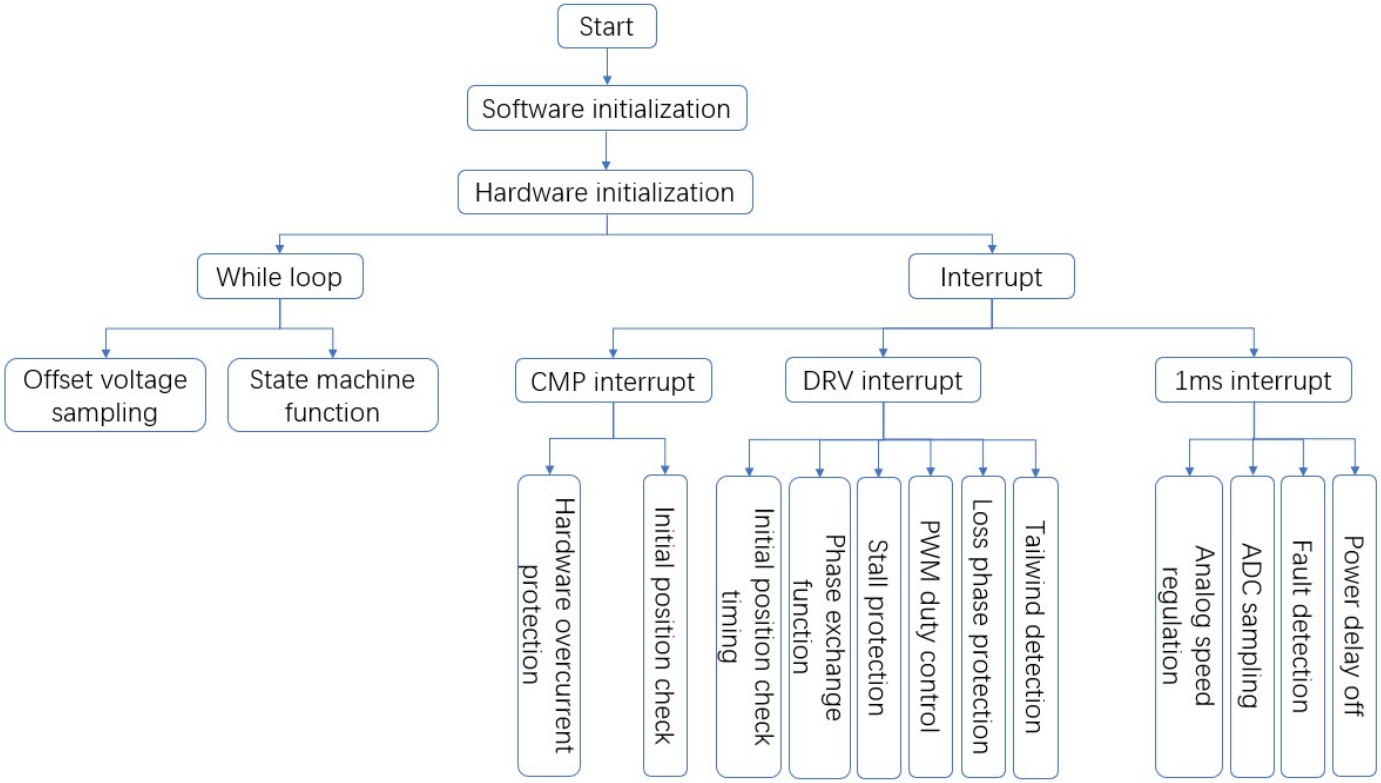


Figure 3-2 Program Execution Flowchart

3.3 Program Specification

3.3.1 Main Function:

Program initialization -> bias voltage sampling GetCurrentOffset() + motor running control MC_Control().

Program initialization is broken into SoftwareInit() and HardwareInit(), the former of which is to initialize software variables, and the latter includes GPIO_Init(), CMP3_Init(), ADC_Init(), AMP_Init(), MDU_16MUL16_INT(15, 0), CMP3_Inter_Init(), Driver_Init() and TIM_1ms_Init().

3.3.2 Comparator Interrupt

1. Hardware comparator OCP

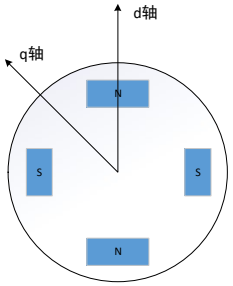
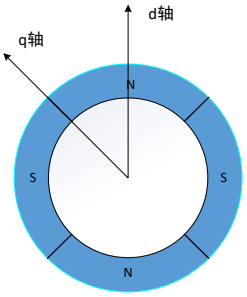
In response to CMP3 interrupt event, the FaultProcess() function will be invoked if the state machine function does not execute initial position check mcState != mcPosiCheck, which will turn off the MOSFET output and trigger hardware OCP. The state machine is in mcFault and the brake operation is not executed.

```
24  void CMP_ISR(void) interrupt 7
25  {
26      if (ReadBit(CMP_SR, CMP3IF))
27      {
28          if ( mcState != mcPosiCheck )
29          {
30              FaultProcess();                                  // close output
31              mcFaultSource          = FaultHardOVCurrent;     // OCP
32              mcState                = mcFault;
33          }
```

Figure 3-3 Hardware OCP

2. Initial position check

If the state machine function executes initial position check, the corresponding logic judgment will be made.

Salient pole motor	Non-salient pole motor
q-axis inductance ≠ d-axis inductance	q-axis inductance = d-axis inductance
	
d-axis and q axis magnetic circuits are asymmetrical, producing a salient pole design.	d-axis and q axis magnetic circuits are symmetrical, producing a salient pole design.
Mostly MTP+A field weakening control	Mostly id=0 vector control

This program is specific to salient pole motor with its characterization summarized in the above table. Given the salient pole of motor, the U, V and W phases have different inductances, so initial position sensing is executed for different on-states of U, V and W phases.

```
36          if (IPD.Step == 1)
37          {
38              IPD.Current_cnt[IPD_phase] = TIM2__CNTR;
39
40              MOE     = 0;
41              DRV_CMR = 0;
42              TIM2__CNTR = 0;
43              ClrBit(TIM2_CR1, T2EN);
44              if ( (uint8)IPD_phase < 7 )
45              {
46                  IPD_phase++;
47                  if( (uint8)IPD_phase == 6 )
48                  {
49                      DAC_DR = DAC_NS_CurrentValue;
50                      if(    ((IPD.Current_cnt[5] >= IPD.Current_cnt[1] ) && ( IPD.Current_cnt[5] > IPD.Current_cnt[3] )
51                          || ((IPD.Current_cnt[5] > IPD.Current_cnt[1] ) && ( IPD.Current_cnt[5] >= IPD.Current_cnt[3] )
52                      )         //5 or  2
53                      {
54                           IPD.NS_CMR[0] = WH_ULA1;
55                           IPD.NS_CMR[1] = UH_WLA1;
56                      }
57                      else if(   ( ( IPD.Current_cnt[3] >= IPD.Current_cnt[5] ) && ( IPD.Current_cnt[3] > IPD.Current_cnt
58                              || ( ( IPD.Current_cnt[3] > IPD.Current_cnt[5] ) && ( IPD.Current_cnt[3] >= IPD.Current_cnt
59                      )  //1or6
60                      {
61                           IPD.NS_CMR[0] = WH_VLA1;
62                           IPD.NS_CMR[1] = VH_WLA1;
63                      }
64                      else if(   (( IPD.Current_cnt[1] >= IPD.Current_cnt[3] ) && ( IPD.Current_cnt[1] > IPD.Current_cnt[
65                              || (( IPD.Current_cnt[1] > IPD.Current_cnt[3] ) && ( IPD.Current_cnt[1] >= IPD.Current_cnt[
66                      )  //3or4
67                      {
68                           IPD.NS_CMR[0] = UH_VLA1;
```

Figure 3-4 Initial Position Check

3.3.3 DRV Interrupt

1. Tailwind/headwind detection

   WindFR_DetermineFunction(); tailwind/headwind detection function

If the state machine function executes tailwind/headwind detection and the tailwind/headwind config. flag bit is set to 1, then WindFR_DetermineFunction() is executed.

The tailwind/headwind detection feature of sensorless square-wave driver senses the direction of motor rotation according to the Hall signal. The switch-on sequence of sensorless square-wave driver when motor rotates forward is as follows: 1. U+V – 2. U+W - 3. V+W - 4. V+U - 5. W+U - 6. W+V -, HALL signal changes to 4 - 5 - 1 - 3 - 2 - 6 - 4 when the motor rotates forward. If motor is in forward rotation and the current Hall signal reading is 5, then the corresponding Hall signal interval for commutation to the next step should be 1, and the Hall signal value of the previous step is 4; if motor is in reverse rotation and the current Hall signal reading is 5, then the corresponding Hall signal interval for commutation to the next step should be 4, and the Hall signal value of the previous step is 1.

```
144         if ((mcState == mcTailWind) && (McStaSet.SetFlag.TailWindSetFlag == 1))
145         {
146             WindFR_DetermineFunction();
147         }
```

Figure 3- 5 Twailwind/headwind Detection

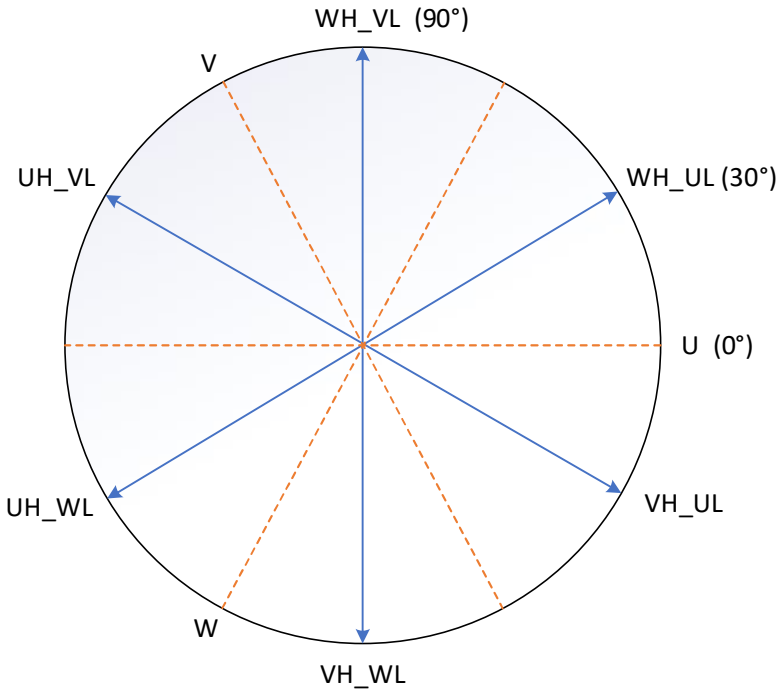
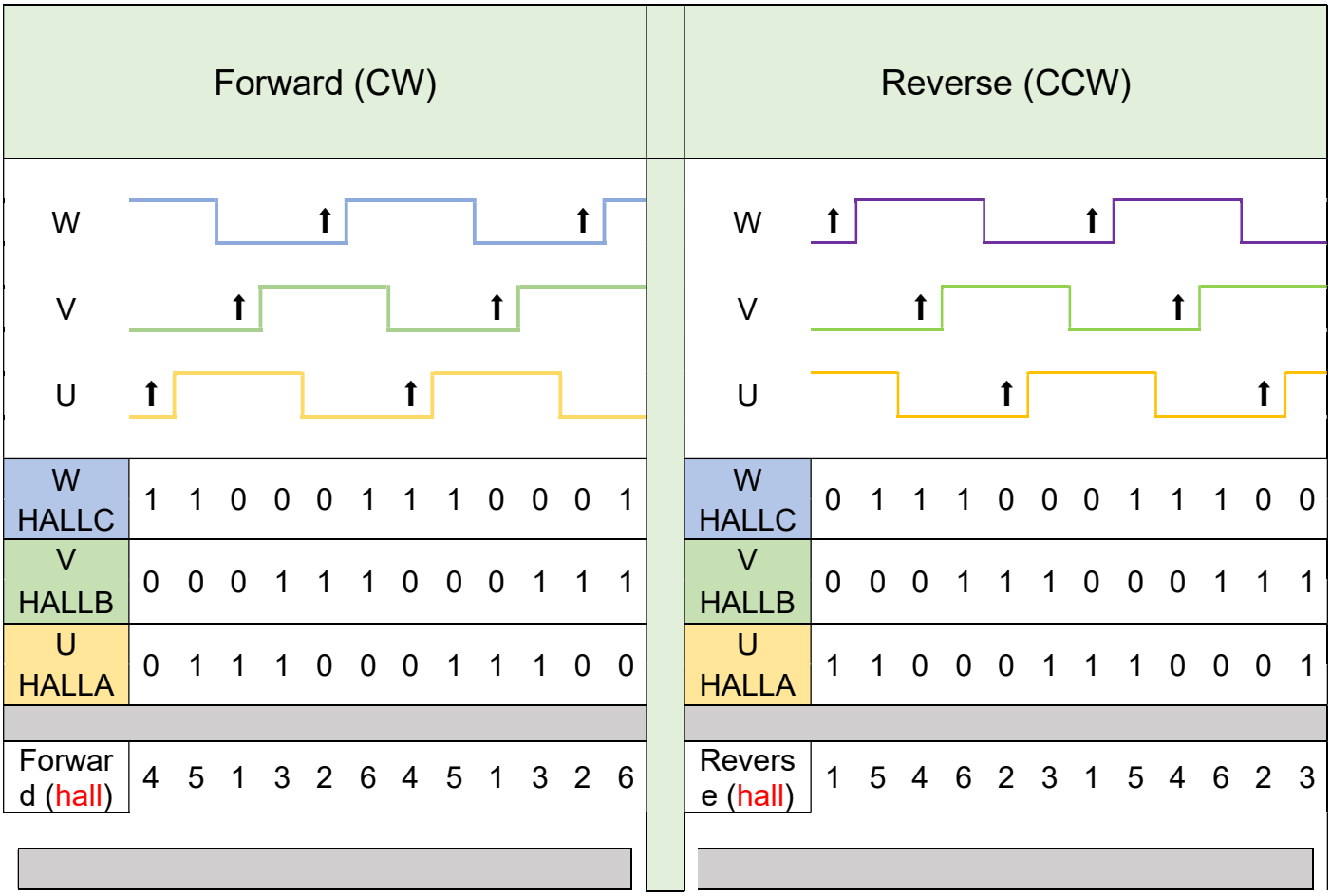


Figure 3-6 Motor Angle for Six On-states



2. Initial position sensing time measurement

If the state machine function executes initial position sensing and a timeout occurs, the phase loss protection will be triggered to call fault detection function.

```
148         else if (mcState == mcPosiCheck)
149         {
150             if (ReadBit(TIM2_CR1, T2IF))
151             {
152                 ClrBit(TIM2_CR1, T2IF);
153                 if (IPD.Step == 1)
154                 {
155                     ClrBit(PI_LPF_CR, T2RPD);                    // close IPD function
156                     TIM2_CR0 = 0x00;                             // close TIM2
157                     TIM2_CR1 = 0x00;
158                     mcFaultSource = FaultLossPhase;              // loss phase protect
159                     DAC_DR   = DAC_OvercurrentValue;             // recovery OCP current value
160                     FaultProcess();
161                     mcState  = mcFault;
162                 }
163             }
164 
165             if (IPD.pwm_cnt > 0)
166             {
167                 IPD.pwm_cnt--;
168             }
169         }
```

Figure 3-7 Initial Position Sensing Time Measurement

3. Commutation function

It is used to control the commutation operation during motor operation, which is divided into two parts: software operation and hardware operation. It is software operation at low speed and switch to hardware operation at high speed. This commutation function is applicable to salient pole motors. If the inductance of D and Q axes of salient pole motors are different, the rotor at different positions will cause changes in the phase inductance. The time of commutation is determined by reading the voltage value of the suspended phase and reflecting the position of the rotor.

4. Motor lock protection (MLP)

Under the software operation mode, if motor lock occurs, the low-speed MLP will be triggered; under the hardware operation mode, the high-speed MLP will be triggered in case of motor lock.

```
170         else if ( (mcState == mcBLDC) || (mcState == mcRun) )
171         {
172             if(mcState == mcBLDC)
173             {
174                 if(IPD.start_Cnt <= 0)
175                 {
176                     Zero_BLDC();                    // software phase exchange control
177                 }
178             }
179             else if(mcState == mcRun)
180             {
181                 RUN_Function();
182             }
183 
184             Duty_Function();                        // PWM duty control
185 
186             #if (StallProtect == Enable)
187             {
188                 if( mcSpeedRamp.FlagONOFF == 1 )
189                 {
190                     BLDC_StallProtect();            // stall protect
191                 }
192             }
193             #endif
194         }
```

Figure 3-8 High and Low Speed Motor Lock

5. Duty cycle function

The DRV_ DR value is obtained according to the VSP signal read in VSPSample(), corresponding to PWM duty cycle.

3.3.4 1ms Timer Interrupt

The potentiometer speed regulation, fault detection, MOS temperature detection, bus current and voltage sampling, off delay and other features are all called with 1ms Timer Interrupt as follows:

1. VSPSample() -- Potentiometer speed regulation function: mcSpeedRamp.BLDC_Value can be calculated according to the VSP voltage and then converted to duty cycle using Duty_Function() to control motor speed.
2. Fault_ Detection() -- fault detection function: It is used to detect faults and trigger protection, including software OCP, OVLO/UVLO, MOS TSD and battery pack TSD.
3. ADC data sampling program (as shown in the figure below), bus current and voltage sampling, MOS temperature and trigger signal VSP.

```
93  mcFocCtrl.mcDcbusFlt = LPFFunction(ADC2_DR << 3, mcFocCtrl.mcDcbusFlt, 10);
94
95  Power_Currt_temp1 = (ADC3_DR << 3);
96  Power_Currt1      = (Power_Currt_temp1 > mcCurOffset.Iw_busOffset) ? (Power_Currt_temp1 - mcCurOffset.Iw_busOf
97
98  mcFocCtrl.MosTempDecFlt = LPFFunction(ADC6_DR << 3, mcFocCtrl.MosTempDecFlt, 10);
99
100 VSP = ADC7_DR << 3;               // VSP signal collection
```

Figure 3-9 ADC Data Sampling

4. Off Delay program (as shown in the figure below): the connection between the mains and the chip power supply circuit is controlled by the Pdelay signal. Power-off delay is triggered when mcSpeedRamp.FlagONOFF is set to 0. The specific time of delay (Pdelay_Vaule) can be set as needed.

```
730 void Pdelay_Function(void)
731 {
732     if ((mcSpeedRamp.FlagONOFF  == 0) || (mcState == mcFault))
733     {
734         if (Pdelay_time++ >= PDELAY_Vaule)
735         {
736             Pdelay_time = PDELAY_Vaule;
737             Pdelay_OFF ;
738         }
739     }
740     else if (mcSpeedRamp.FlagONOFF  == 1)
741     {
742         Pdelay_time = 0;
743         Pdelay_ON;
744     }
745 }
```

Figure 3-10 Off Delay Settings

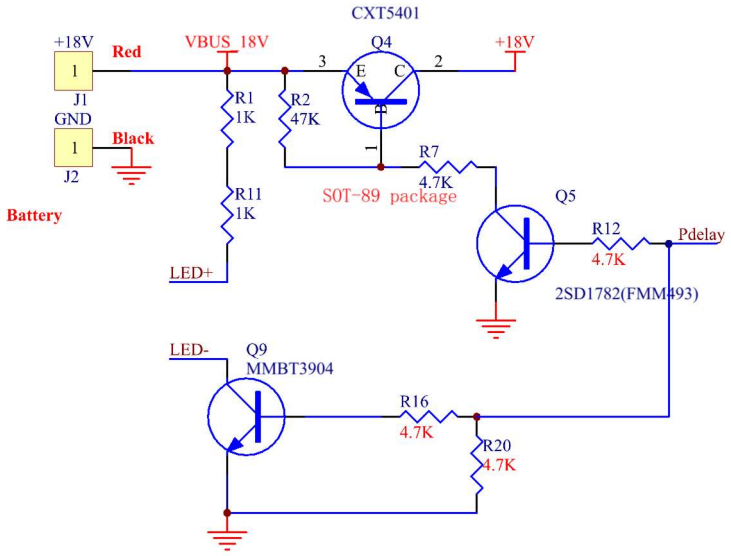


Figure 3-11 Off Delay Schematic

4 Debugging Procedures

4.1 Chip Intrinsic Parameters Configuration

```
/*MCU parameter*/
#define MCU_CLOCK                       (24.0)              // (MHz) Main frequency
#define TIM1_FRE                        (6000000.0)         // (Hz)  TIM1 counting frequency
#define PWM_DEADTIME                    (0.0)               // (us)  Dead time
#define PWM_FREQUENCY_BLDC              (20.0)              // (kHz) Carrier frequency
```

Figure 4-1 Chip Parameters Configuration

Instructions:

1. Carrier frequency must be selected properly before debugging since it will affect motor startup, MOSFET temperature rise, etc. The electric drill can be debugged at 20K by default;
2. The dead time is set to prevent shoot-through risk, which is generally less than 1/16 of the carrier period.

4.2 Motor Parameters Configuration

4.2.1 Motor Parameters

1. Pole_Pairs;
2. MOTOR_SPEED_BASE = 2*motor rated speed.

```
/*Motor parameter*/
#define Pole_Pairs                      (2.0)               //  pole pairs
#define MOTOR_SPEED_BASE                (60000.0)           //  (RPM) speed base
```

Figure 4- 2 Motor Parameters

4.2.2 Motor Parameters Measurement

1. Pole_Pairs: A parameter to be assigned in motor design.
2. The commutation function of this program is based on the salient polarity principle of motors, which is not applicable to non-salient pole motors (such as surface-mount motors), so you need to test the motor before using this program to confirm that it is a salient pole motor. The testing procedures are as follows:
   (1) Use LCR meter to measure the phase inductance at 1KHZ by connecting two phase lines of the motor (see Figure 4-1);
   (2) Rotate the motor manually and observe the variation of phase inductance (see Figure 4-1). If a significant variation is observed, then it is a salient pole motor.

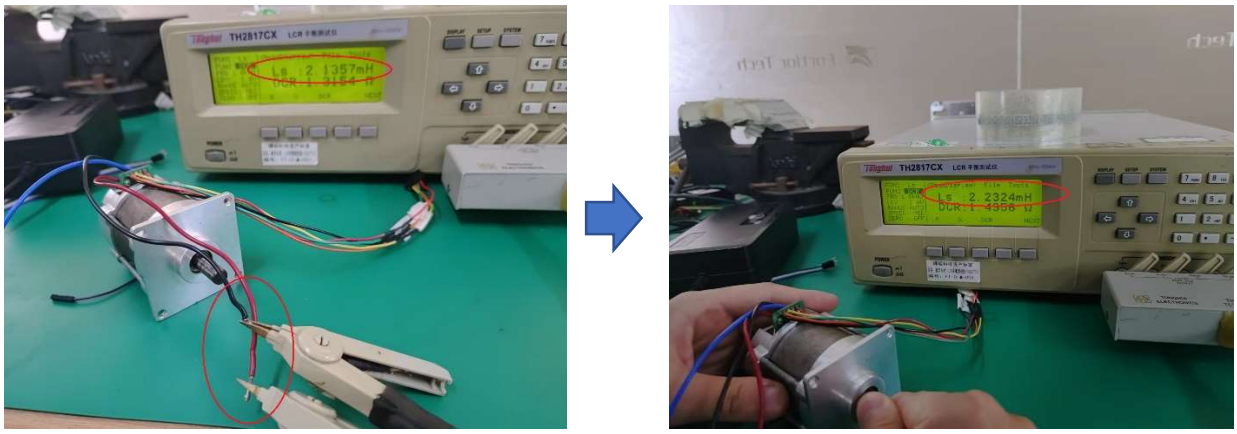


Figure 4-3 Motor Saliency Test

3. MOTOR_SPEED_BASE: The speed base is typically set to about 2 times of the motor maximum speed. This parameter will affect the motor's startup and other performances, which must be set in advance and preferably remain unchanged.

4.3 Hardware Parameters Configuration

4.3.1 Drive Level

Drive level is selected according to the drive integrated in the chip or the drive connected to the peripheral circuit of the chip. EU6831 has a built-in 3P3N drive of High (or “1”)-Low (or “0”) level configuration; EU6818 integrates a 6N drive of High (or “1”)-High (or “1”) level configuration; and EU6861 incorporates a 6N drive of High (or “1”)-Low (or “0”) level configuration. If you use EU6811 or EU6812, you need to set the drive level according to the drive connected to the peripheral circuit.

```
/*Hardware parameter*/
/*HVIC effective level
    All_High_Level -- all high effective
    All_Low_Level  -- all low effective
    UP_H_DOWN_L    -- up bridge high down bridge low effective
    UP_L_DOWN_H    -- up bridge low down bridge high effective
*/
#define PWM_Level_Mode                          (UP_H_DOWN_L)
```

Figure 4-4 Drive Level Configuration

4.3. 2 Hardware Parameters

1. The DC bus voltage divider ratio, shunt resistance and (amplifier) gain are a function of the motor voltage range and power range.

2. Rules for selection of shunt resistance and gain:

   1) DC bus voltage divider ratio:

      - Voltage divider ratio should not be too low: In general, recommended maximum shunt voltage is 0.8*VREF. Assuming that the maximum voltage of a motor is 30V, and the ADC VREF value is 4.5V, then the recommended minimum voltage divider ratio should be: 30/0.8/4.5 = 8.33; if the voltage divider ratio is too low, ratio = 5 for instance, then the voltage applied on the AD port after scaling down the maximum voltage 30V will be 6V, which is beyond the ADC full scale.
      - Voltage divider ratio should not be too high: If the voltage divider ratio is too high, the AD accuracy will be compromised. Assuming that the maximum voltage of a motor is 30V and the voltage divider ratio is 40, then the voltage applied on the AD port will be 30V/40V = 0.75v; when the voltage is 28V, the voltage applied on the AD port will be 0.7V, creating a less accurate measurement; besides, the AD has a margin of 4.5 - 0.75 = 3.75V relative to its full scale.

   2) Shunt resistance and gain:

      Maximum sampling current = VREF/HW_RSHUNT/HW_AMPGAIN; it should be noted that the maximum sampling current is the current flowing through the shunt resistor instead of the current shown on the power source (the power source is filtered).

      - The shunt resistance should not be too high: If the shunt resistance is too high, the voltage pulled down may go beyond the ADC full scale, or the shunt resistor may exceed its power rating; shunt resistors in a 2512 package have a typical power rating of 1W or 2W, and those in a 1206 package have a typical power rating of 1W or 4W. When choosing a shunt resister, make sure that the power applied on the shunt resistor I^2R does not exceed the above power ratings.
      - The shunt resistance should not be too low: If the shunt resistance is too low, the accuracy will be compromised.
      - The gain must be adjusted against the shunt resistance. That means, you should firstly choose a proper shunt resistance, and then adjust the gain.

   HW_RSHUNT represents shunt resistance, and HW_AMPGAIN represents gain.

3. The DC bus voltage divider ratio, shunt resistance and gain must be written into the program (in the Customer.h file).

```
#define HW_RSHUNT                               (0.001)                 // (Ω)Sampling resistance
#define HW_AMPGAIN                              (10.0)                  // Operational amplifier magni
/*VREF pin is not connected with 1uf capacitance or Offsetcurrent is obtained by VDD5 partial pressure, On
#define HW_ADC_VREF                             (VREF4_5)               // (V)ADC reference voltage VR

/*VHALF parameter*/
#define VHALF_OUT_EN                            (1)                     // VHALF enable

#define Outside_VHALF                           (0)                     // Output partial voltage thro
#define Inside_VHALF                            (1)                     // The VHALF pin output is use
#define VHALF_MODE                              (Inside_VHALF)

#define RV1                                     (0.0)                   // (kΩ) Bus voltage divider1
#define RV2                                     (30.0)                  // (kΩ) Bus voltage divider2
#define RV3                                     (5.1)                   // (kΩ) Bus voltage divider3
#define RV                                      ((RV1 + RV2 + RV3) / RV3)   // Partial pressure ratio
```

Figure 4-5 Hardware Parameters Configuration

4.4 VREF Calibration

VREF calibration is used in the GetCurrentOffset() function, where the calibration times can be adjusted to improve the accuracy of offset voltage extraction.

```
/*Base voltage calibration*/
#define Calib_Time                          (1000)                  // calibration times,Fixed 1000 time
```

Figure 4-6 Calibration Times Setting

4.5 ON/OFF Parameters

1. Motor has three speed control modes: speed control with analog voltage, PWM speed control and constant speed control. Among them, speed control with analog voltage is selected in most cases.
2. OFF_Duty ~ Max_Duty indicates the level corresponding to the duty cycle in ON/OFF state and the level corresponding to the duty cycle at maximum and minimum speeds. The default value can be used here.
3. PDELAY_Value is the time of power-off delay, generally set as 10000ms.

```
/*Trigger Switch parameter*/
#define OFF_Duty                            _Q15(0.40/HW_ADC_REF)       // Shutdown speed regulation voltag
#define ON_Duty                             _Q15(0.7/HW_ADC_REF)        // start up speed regulation voltag
#define MIN_Duty                            _Q15(1.0/HW_ADC_REF)        // Min speed regulation voltage on
#define MAX_Duty                            _Q15(4.30/HW_ADC_REF)       // Max speed regulation voltage on

#define PDELAY                              (GP01)
#define PDELAY_Vaule                        (10000)                     // pdelay time, unit: ms
#define Pdelay_ON                           (GP01 = 1)
#define Pdelay_OFF                          (GP01 = 0)
```

Figure 4-7 ON/OFF Parameters

4.6 Pre-charge

1. IPMState can be set to two modes:

   1) NormalRun: In this mode, motor operates normally and enters the pre-charge state. Pre-charge is implemented in three steps. Power on the bootstrap capacitors of U, V and W phase upper bridges respectively for charging. The default values of Charge_Time and Charge_Duty can be used here.

2) IPMtest: Test mode, where U, V and W phases output PWM with fixed duty cycle. The set duty cycle can be used to test whether the three-phase outputs are correct. It should be noted that the motor is disconnected when debugging in the test mode.

```
/*mcCharge set*/
/*MOS output check:  IPMtest--IPM test mode   NormalRun--motor normal run mode*/
#define IPMState                              (NormalRun)
#define PWM_DEADTIME_TEST                     (1.5 * MCU_CLOCK)          // IPM test mode must have dead

#define Charge_Time                           (30)                       // (ms) charge time, unit: ms
#define Charge_Duty                           (0.5)                      // charge PWM duty
```

Figure 4-8 Pre-charge Mode Setting

```
void Motor_Charge(void)
{
    if (McStaSet.SetFlag.ChargeSetFlag == 0)
    {
        McStaSet.SetFlag.ChargeSetFlag = 1;
        #if (IPMState == IPMtest)                                        // IPM test
        {
            DRV_DTR = PWM_DEADTIME_TEST;
            DRV_DR  = 0.3 * DRV_ARR;
        }
        #elif (IPMState == NormalRun)                                    // Normal run
        {
            DRV_DR = Charge_Duty * DRV_ARR;
        }
        #endif
        mcFocCtrl.ChargeStep = 0;
```

Figure 4-9 Pre-charge Program

4.7 Initial Position Check

```
/*Initial position check parameters*/
#define RPD_Time                          (5)                    // Number of injected pulse interv
#define Phase_CurrentValue                (30.0)                 // (A) Distinguish the current val
#define NS_CurrentValue                   (60.0)                 // (A) Distinguish the current val
```

Figure 4-10 Initial Position Check Parameters

When the motor cannot be started, you may reduce NS_CurrentValue appropriately for testing.

4.8 Motor Startup

```
/*Phase exchange point parameter*/
#define FIX_KR                              (0)                         // KR value fixed
#define GRADIENT_KR                         (1)                         // KR value increase
#define KR_MODE                             (GRADIENT_KR)

#define BLDC_KR_MIN                         _Q15(0.60)                  // Phase exchange point at rising
#define KRKF_Sum                            (uint16)(32767*(1.1))
#define BLDC_KF_MIN                         (uint16)( KRKF_Sum - BLDC_KR_MIN )  // Phase exchange point at


/*Start up PWM duty parameter*/
#define samr                                (0xaa)                      // sampling delay
#define Max_Str_Duty                        (0.2)                       // start up max PWM duty
#define Min_Duty                            (0.1)                       // VSP start min PWM duty
#define Max_Duty                            (1.0)                       // VSP end max PWM duty
#define DUTY_Updata_ADD_Num                 (uint16)(2)                 // DUTY(+)Number of response carri
#define DUTY_Updata_SUB_Num                 (uint16)(2)                 // DUTY(-)Number of response carri
```

Figure 4- 11 Motor Startup Parameters

1. You can modify KR and KF values by adjusting BLDC_KR_MAX and BLDC_KR_MIN values; wherein, KF = 1 - KR, so KF value changes with KR value. Adjusting KR and KF values is actually to adjust the commutation point, which can be set according to the salient polarity of the motor, that is, the variation range of phase inductance. KR is the commutation point of the upward hypotenuse, such as commutation point 1; KF is the commutation point of the downward hypotenuse, such as commutation point 2. Generally, the KR set value falls within 0.5~0.95, so the range of KF value is 0.05~0.5. When the KR and KF values change, the corresponding commutation points will also move up or down. If motor stalls during startup, you may adjust the KR and KF values.

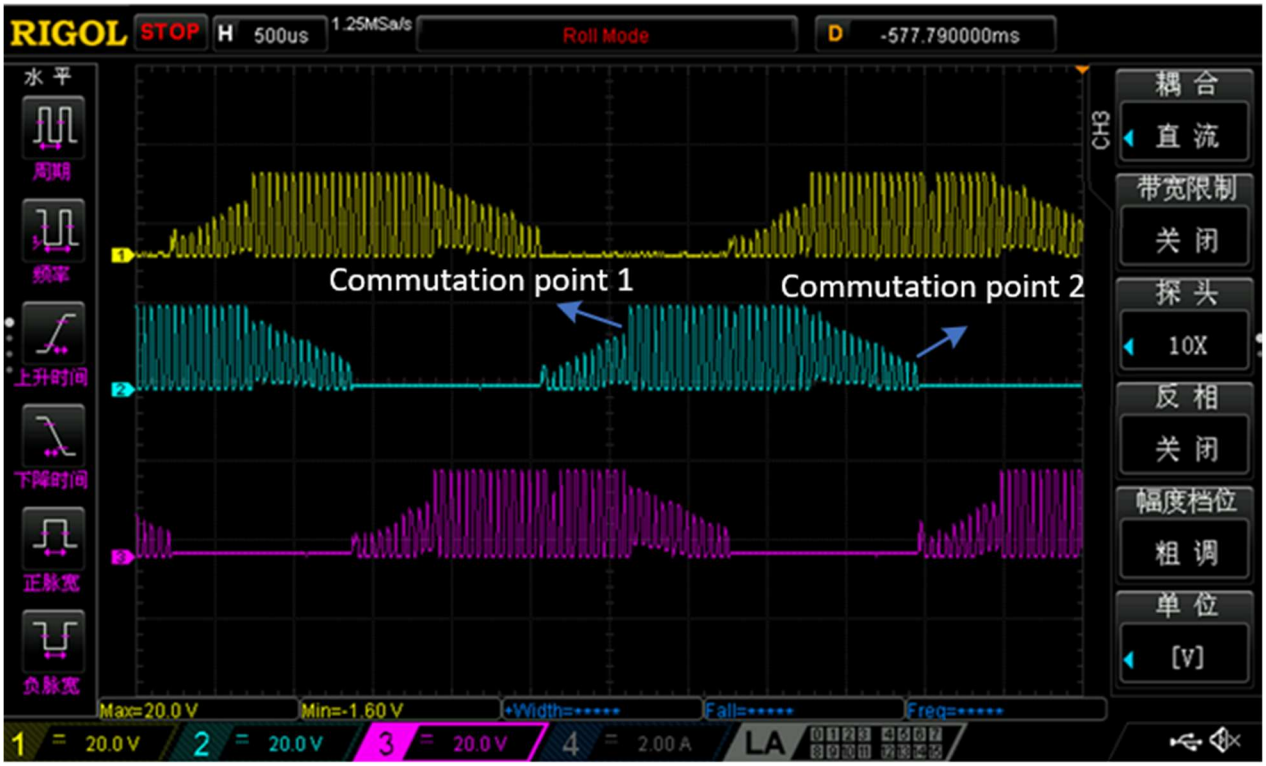


Figure 4-12 Motor Three-phase Waveform

2. Set the freewheeling shielding angle MaskAngle (uint8)(30) which typically falls within 1 °~59 °, generally 30 °.
3. Min_BLDC_Duty (0.20) // Duty cycle corresponding to the VSP start point
   Max_BLDC_Duty (1.00) // Duty cycle corresponding to VSP end point
4. STH_RPM (8000.0) is the speed at which the motor switches from low-speed software operation to high-speed hardware operation; HTS_RPM (4000.0) is the speed at which the motor switches from high-speed hardware operation to low-speed software operation. Generally, no change is required for them.

5 Function Description

A total of 10 protection features are provided in the sensorless square-wave driver. Different motors and boards vary in their OVLO and OCP thresholds. Those protection functions should be configured as needed to avoid false triggering and achieve timely protection.

```
typedef enum
{
      FaultNoSource              = 0,      // No fault
      FaultSoftOVCurrent         = 1,      // Primary overcurrent protection
      FaultSoftOVCurrent1        = 2,      // Secondary overcurrent protection
      FaultSoftOVCurrent2        = 3,      // Three stage overcurrent protection
      FaultHardOVCurrent         = 4,      // OCP
      FaultUnderVoltage          = 5,      // Undervoltage protection
      FaultOverVoltage           = 6,      // OverVoltage protection
      FaultMosOverTemperature    = 7,      // Mos Over Temperature protection
      FaultBatOverTemperature    = 8,      // Bat Over Temperature protection
      FaultLossPhase             = 9,      // Loss phase protection
      FaultStall                 = 10,     // Stall protection
      FaultLostStep              = 11,     // LostStep protection
      FaultLVW                   = 12,     // LVW protection
      FaultOffSetCurrt           = 13,     // Offset current protection
} FaultStateType;
```

Figure 5-1 Protection Settings

5.1 Overcurrent Protection (OCP)

5.1.1 Introduction

I. Hardware OCP

Hardware OCP means hardware is implemented to disable PWM output in case of overcurrent that is determined by comparison between MCU comparator and op-amp outputs. This feature can stop the motor in case of hardware overcurrent for protection purpose. The hardware OCP current threshold is usually selected based on the maximum current that the hardware MOSFET can withstand, which can be adjusted by changing OverHardcurrentValue.

```
/*Hardware OCP*/
 #define Hardware_FO_Protect              (1)                          // FO protection
 #define Hardware_CMP_Protect             (2)                          // CMP protection
 #define Hardware_FO_CMP_Protect          (3)                          // FO & CMP protection
 #define Hardware_Protect_Disable         (4)                          // Disable OCP
 #define HardwareCurrent_Protect          (Hardware_CMP_Protect)
/*Hardware OCP comparison value mode*/
 #define Compare_DAC                      (0)                          // DAC
 #define Compare_Hardware                 (1)                          // External circuit
 #define Compare_Mode                     (Compare_DAC)
/*Hardware OCP current value*/
 #define OverHardcurrentValue             (80.0)                       // Hardware overcurrent value
```

Figure 5-2 Hardware OCP Settings

II. Software OCP

Software OCP feature has three-order overcurrent thresholds. Take the first-order software OCP threshold as an example. Firstly, you need to enable this protection by setting OverSoftCurrentProtect to 1, and then change OverSoftCurrent and OverSoftCurrentTime. Make one count when the maximum current exceeds the set OverSoftCurrent, and this protection is triggered when the counts exceed OverSoftCurrentTime. The parameters should be set specific to the actual project.

```
/*Software overcurrent protection*/
#define    OverSoftCurrentProtect          (1)                          // Primary overcurrent protec
#define    OverSoftCurrent                 I_Limt_Value(20.0)           // Primary overcurrent value,
#define    OverSoftCurrentTime             (1500)                       // Primary overcurrent time,

#define    OverSoftCurrentProtect1         (1)                          // Secondary overcurrent prot
#define    OverSoftCurrent1                I_Limt_Value(30.0)           // Secondary overcurrent valu
#define    OverSoftCurrentTime1            (600)                        // Secondary overcurrent time

#define    OverSoftCurrentProtect2         (1)                          // Three stage overcurrent pr
#define    OverSoftCurrent2                I_Limt_Value(50.0)           // Three stage overcurrent va
#define    OverSoftCurrentTime2            (50)                         // Three stage overcurrent ti
```

Figure 5-3 Software OCP Settings

5.1.2 OCP Testing

I. Hardware OCP testing

1. Reduce the set OverHardcurrentValue in the program;
2. Start the electric drill and increase the load (drill nails) to see if the hardware OCP can be triggered.

II. Software OCP testing

1. Enable software OCP;
2. Reduce the set OverSoftcurrent of each order in the program;
3. Start the electric drill and increase the load (drill nails) to see if the software OCP can be triggered;
4. Sometimes the software OCP cannot be triggered by reducing the set OverSoftcurrent probably because OverSoftCurrentTime selected for each order is too high. You may adjust this value as needed and test again.

5.2 Overvoltage/Undervoltage Lockout (OVLO/UVLO)

5.2.1 Introduction

OVLO is triggered when the sensed voltage goes above the setpoint; and UVLO is triggered when the sensed voltage is below the setpoint. Firstly, enable this protection by setting VoltageProtect to 1, and then set Over_Protect_Voltage and other parameter values.

```
/*Over/Under Voltage protection*/
#define VoltageProtect                  (1)                          // protection enable/disable
#define Over_Protect_Voltage            (24.0)                       // Over DC Voltage protect v
#define Over_Recover_Vlotage            (22.0)                       // Over DC Voltage recover v
#define Under_Protect_Voltage           (13.0)                       // Under DC Voltage protect
#define Under_Recover_Vlotage           (14.0)                       // Under DC Voltage recover
```

Figure 5-4 OVLO/UVLO Settings

5.2.2 OVLO/UVLO Testing

1. Turn on the electric drill, and increase the supply voltage gradually to the OVLO threshold to see if the actual OVLO threshold is consistent with the program set value and if the motor stops when the set OVLO threshold is exceeded.
2. Turn on the electric drill, and reduce the supply voltage gradually to the UVLO threshold to see if the actual UVLO threshold is consistent with the program set value and if the motor stops when the set UVLO threshold is exceeded.

5.3 Thermal Shutdown (TSD)

5.3.1 Introduction

The figure below provides a schematic representation of TSD, where the voltage divider is used with a NTC thermistor, the resistance of which decreases gradually with increasing temperature. A resistance value corresponds to each given temperature. You may consult corresponding NTC curves in the manual for each thermistor model. Connect Tmos to AD6 port of the chip. The program senses the voltage applied to the AD port. When the voltage becomes less than the voltage at the given temperature, it indicates that the NTC resistor temperature exceeds the setpoint and thus TSD is triggered.

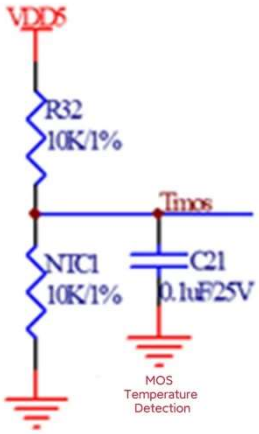


Figure 5-5 TSD Schematic Diagram

Firstly, enable TSD by setting Mos_TemperatureProtectEnable to 1. And then, set the TSD threshold OVER_Mos_Temperature, Tempera_Value() in the program according to the NTC curves of different thermistors. 10.0 in Tempera_Value(NTC_Value) is pull-up resistor value which should be set according to the actual circuit. When the thermistor temperature rises, TSD is triggered to stop the motor. For different hardware, you need to reset the TSD threshold when implementing TSD program.

```
/*Mos over temperature protection*/
#define Mos_TemperatureProtectEnable       (0)                                  // protection enable/disable
#define Tempera_Value(NTC_Value)           _Q15((5.0*NTC_Value/(10.0+NTC_Value))/HW_ADC_REF)
#define OVER_Mos_Temperature               Tempera_Value(0.966)                 // MOS over temperature prote
#define UNDER_Mos_Temperature              Tempera_Value(1.07)                  // MOS over temperature recov
#define Over_Mos_TemperRecoverTime         (1000)                               // MOS over temperature recov
```

Figure 5-6 TSD Settings

Battery pack TSD may be enabled where necessary, and the over-temperature threshold can be set according to the NTC curve.

```
/*Bat over temperature protection*/
#define Bat_TemperatureProtectEnable       (0)                                  // protection enable/disable:
#define OVER_Bat_Temperature               Tempera_Value(1.117)                 // BAT over temperature prote
#define UNDER_Bat_Temperature              Tempera_Value(1.275)                 // BAT over temperature recov
#define Over_Bat_TemperRecoverTime         (1000)                               // BAT over temperature recov
```

Figure 5-7 Battery Pack TSD Settings

5.3.2 TSD Testing

1. Enable TSD
2. Reduce the temperature that triggers TSD appropriately, and write the corresponding thresholds shown in the NTC curve into the program.
3. If the test board is provided with a radiator, you will observe minor temperature rise during normal operation, so you can use a heat gun or other similar tools to heat up the thermistor to simulate the over-temperature condition, and observe whether TSD is triggered.

5.4 Phase Loss Protection (PLP)

5.4.1 Introduction

PLP performs detection in the initial position sensing stage where pulses are applied to each phase. TIM2 outputs the time for current rise to the specified value (NS_CurrentValue). PLP is triggered if that time goes beyond the TIM2 full scale. Generally, it is caused by phase loss, that is, the motor phase lines are connected in a faulty manner, resulting in no phase current. Alternatively, when the motor phase line connection is correct, PLP will also be triggered if the set CurrentValue is too high.

```
/*Initial position check parameters*/
#define RPD_Time                          (5)                          // Number of injected pulse interva
#define Phase_CurrentValue                (30.0)                       // (A) Distinguish the current valu
#define NS_CurrentValue                   (60.0)                       // (A) Distinguish the current valu

#if (VHALF_OUT_EN == 1)
    #define DAC_Phase_CurrentValue        _Q7(I_ValueX(Phase_CurrentValue*2.0))+0x7F
    #define DAC_NS_CurrentValue           _Q7(I_ValueX(NS_CurrentValue*2.0))   +0x7F
#else
    #define DAC_Phase_CurrentValue        _Q7(I_ValueX(Phase_CurrentValue*2.0))
    #define DAC_NS_CurrentValue           _Q7(I_ValueX(NS_CurrentValue*2.0))
#endif
```

Figure 5-8 Initial Position Check Parameters

```
150             if (ReadBit(TIM2_CR1, T2IF))
151             {
152                 ClrBit(TIM2_CR1, T2IF);
153                 if (IPD.Step == 1)
154                 {
155                     ClrBit(PI_LPF_CR, T2RPD);                    // close IPD function
156                     TIM2_CR0 = 0x00;                             // close TIM2
157                     TIM2_CR1 = 0x00;
158                     mcFaultSource = FaultLossPhase;              // loss phase protect
159                     DAC_DR   = DAC_OvercurrentValue;             // recovery OCP current value
160                     FaultProcess();
161                     mcState  = mcFault;
162                 }
163             }
```

Figure 5-9 PLP Settings

5.4.2 PLP Testing

Debugging is required to check that the PLP feature functions properly:

1. Set an appropriate NS_CurrentValue according to the runtime current to ensure that the motor can start up normally;
2. Disconnect one of the phase lines of the motor to see if PLP is triggered normally after startup.

5.5 Motor Lock Protection (MLP)

5.5.1 Introduction

MLP is implemented into two ways: One is to detect the stall time, where MLP is triggered if motor does not commutate after the set stall time. The set stall time should be two different values under software operation and hardware running. The other is to detect the motor speed, where MLP is triggered to disable driver output if the speed is higher than the out-of-step speed. The set out-of-step speed is typically 2~3 times the maximum motor speed.

```
61  /*Stall protection*/
62   #define StallProtect                    (1)                          // protection enable/disab
63   #define BLDC_STALL_COUNT                TIM_FREQUENCY_BLDC(3000.0)   // software control time,
64   #define RUN_STALL_COUNT                 TIM_FREQUENCY_BLDC(80.0)     // hardware control time,
65   #define RUN_STALLRPM_COUNT              RPM_TO_BCOR(40000)
```

Figure 5- 10 MLP Parameters

5.5.2 MLP Testing

Debugging is required to check that the MLP feature functions properly:

1. Enable MLP;
2. Reduce the stall time appropriately and the motor speed as needed for actual motor operation, increase the load (such as drilling on a board or holding the motor while wearing protective gloves) to see if MLP can be triggered normally in case of motor lock.

3. Reduce the out-of-step speed gradually to less than the maximum running speed of the motor as needed for actual motor operation, and test whether MLP can be triggered normally when the motor rotates at the maximum speed.

5.6 Other Protection Features

Other protection features may be added as needed according to customer's requirements.

6 Debugging Problems & Solutions

Debugging at constant power	
Problem	**Solution**
Startup failure	After the software fault is cleared, check if there is any problem with hardware sampling wiring.
Motor fails to start or rotates in reverse	Reduce NS_CurrentValue of initial position check Customer.h
Motor jitters or stalls at startup	Change the KR and KF values, and adjust the hypotenuse of phase voltage waveform at startup

7 Revision History

Rev.: 1st digit - Schematic Diagram 2nd digit - Module 3rd & 4th digits - Details

Rev.	Changes	Effective Date	Revised by	Approved by
V1.0.00 MCU	First release	2022-08-25	Liu Jiaxin/ Lin Yingpeng	
V1.0.01	Adjusted the format of the table of contents	2022-12-12	Liu Jiaxin/ Lin Yingpeng	

8 Copyright Notice

Fortior Technology Co.,Ltd.
(Singapore)：1003 Bukit Merah Central, #04-22, INNO Center,(s)159836
Customer service:info@fortiortech.com
URL: http://www.fortiortech.com/global/