Sensorless FOC Application Note

Three-phase Motor Control
EU6861-Q2 MCU

Fortior Technology Co., Ltd.

Table of Contents

1 Overview

1.1 Introduction

This Debugging Manual illustrates the debugger for implementation of sensorless FOC with EU68XX Series MCU for BLDC motors. Sections 2~6 are developed for theories, Sections 7~8 for debugging procedures, and Section 9 for protection functionality.

1.2 Relevant Software/Hardware

The software code referenced herein is the EU6861Q2 Specification for Vacuum Cleaner Application, and the hardware referenced is the EU6861 MV Motor Control DEMO Board.

2 Field Oriented Control (FOC) Theories

Field oriented control (FOC), also known as vector control, is a variable frequency drive (VFD) control technique used to control three-phase BLDC motors by controlling the amplitude and frequency of the inverter's output voltage.

FOC essentially converts motor phase currents in a three-phase stationary coordinate system into those in a rotating coordinate system that is stationary in relative to the rotor field axis, aiming to control the motor by controlling the magnitude and direction of the vectors in the rotating coordinate system. The voltage, current and electromotive force in the stator reference system are AC variables that rotate in space at synchronous speeds, making it difficult to realize control using the control algorithm. After the coordinate transformation, the rotating synchronous vectors are converted into stationary vectors, and both voltage and current are changed into DC variables. Then, the torque equation is used to find out the relation between the torque and controlled variables in the rotating coordinate system, and DC setpoints required for generation of the torque are calculated and regulated in real time, thus indirectly controlling the motor to deliver its desired performance. With the aid of vector control block diagram, the vector control process can be summarized as follows:

1. The three-phase stator currents are measured to get I_a and I_b. The three-phase currents are converted through Clark Transform to two-phase currents I_α and I_β which are orthogonal, time-variant current values;
2. With the motor angle calculated at the last iteration of the control loop, the two-phase currents in a stationary reference system are converted using Park Transform to orthogonal currents I_d and I_q in a rotating coordinate system. For steady state conditions, I_d and I_q are constant.
3. The I_d reference controls the rotor flux, the I_q reference controls the motor's torque output, and the measured and reference values of these two constants are compared to get inputs to PI controllers of the current loop. The PI controller parameters are tunned to output V_d and V_q which are voltage vectors that will be applied to the motor.
4. A new motor angle and speed are estimated from V_α, V_β, I_α and I_β using the rotor position estimation algorithm. This new angle guides the FOC algorithm as to where to place the next voltage vector. The calculated speed is compared with the setpoint to get an error which is then input to PI controllers of the speed loop to generate I_q reference;
5. The V_d and V_q outputs from the PI controllers are rotated back through Park Transform to the stationary two-axis reference frame using the new angle. This calculation provides the next quadrature voltage values V_α and V_β. The SVPWM algorithm is then employed to find out the sector where the voltage vectors synthesized are located and calculate the switch-on time of each leg of the three-phase switch tube. Finally, the desired three-phase voltage outputs are generated by the three-phase inverter.

The entire process of coordinate transformation, PI iteration, rotor position estimation and PWM generation is illustrated inFigure 2-1.

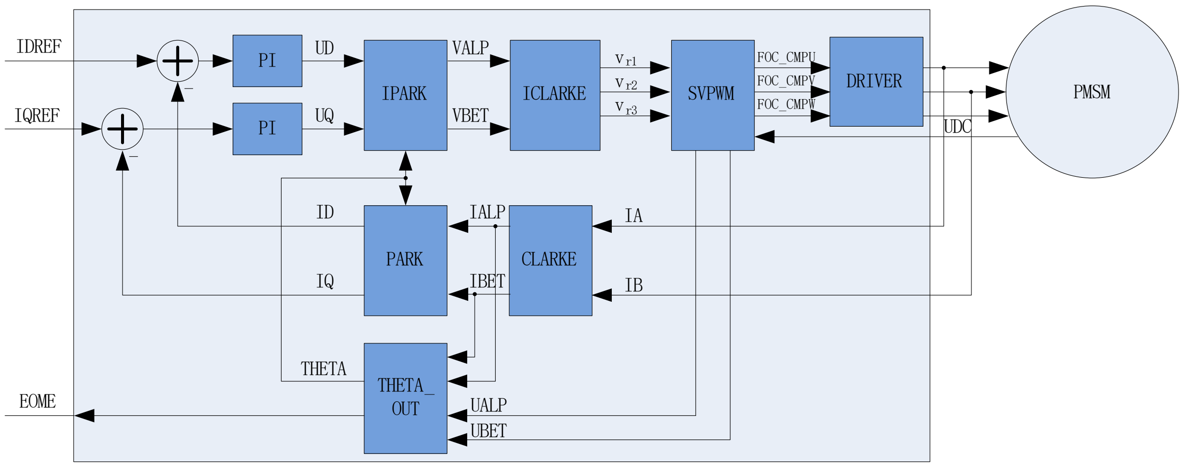


Figure 2-1 Vector Control Block Diagram

2.1 Coordinate Transforms

2.1.1 Clark Transform

For calculation purposes, the Clark Transform projects the three-phase coordinate system, referenced to the stator, into a stationary two-phase coordinate system (seeFigure 2-2). The transformation formula is presented as Equation 2−1.

$$I_a + I_b + I_c = 0$$
$$I_\alpha = I_a$$
$$I_\beta = (I_a + 2I_b)/\sqrt{3}$$

Equation 2-1

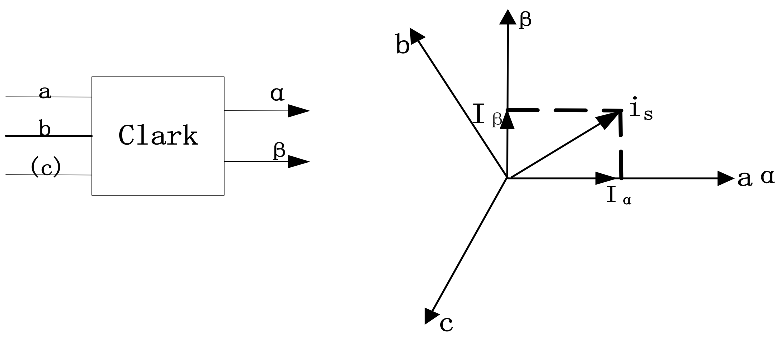


Figure 2-2 Clark Transform

Notes: Clark Transform keeps the amplitude invariant.

2.1.2 Park Transform

Park Transform moves a stationary two-phase reference frame onto a rotating two-phase d-q reference frame. This d-q reference frame, also called "rotor field oriented coordinate system", is rotating with the rotor at the same angle, as illustrated in Figure 2-3. The transformation formula is presented as Equation 2-2, where θ represents the rotor angle.

$$I_d = I_\alpha \cos\theta + I_\beta \sin\theta$$
$$I_q = -I_a \sin\theta + I_\beta \cos\theta$$

Equation 2-2

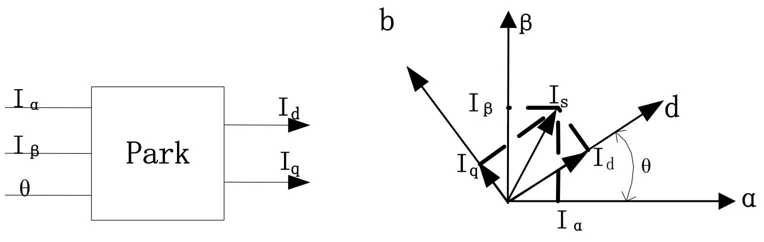


Figure 2-3 Park Transform

2.1.3 Inverse Park Transform

After the PI iteration of the d-q axis current loop, you can get two voltage component vectors in the rotating d-q coordinate system. The user will need to transform back to the three-phase motor voltages through inverse transformation. Firstly, the rotating two-phase reference frame is transformed to a stationary two-phase reference frame through Inverse Park Transform using Equation 2-3:

$$U_\alpha = U_d \cos\theta - U_q \sin\theta$$
$$U_\beta = U_d \sin\theta + U_q \cos\theta$$

Equation 2-3

2.2 PI/PID Controller

PI/PID controller is a linear controller which generates controlled quantity by linear combination of error proportional (P), integral (I) and differential (D) actions for controlling the target parameters with actuator. It is used to control motor speed and position in the motor control system.

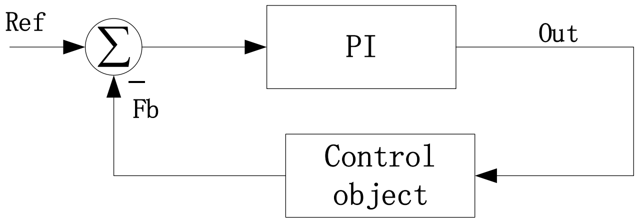


Figure2-4 PI Controller

PI algorithm:

$$U_k = U_{k-1} + Kp \times (E_k - E_{k-1}) + Ki \times E_k$$

PID algorithm:

$$U_k = U_{k-1} + Kp \times (E_k - E_{k-1}) + Ki \times E_k + Kd \times (E_k - 2 \times E_{k-1} + E_{k-2})$$

Where,

U_k: controlled quantity generated by computation k

U_{k-1}: controlled quantity generated by computation k-1

E_k: Error input at time k

E_{k-1}, E_{k-2}: Errors input at time k - 1 and time k - 2

Kp, *Ki*, *Kd*: proportional (P), integral (I) and differential (D) terms to adjust controlled quantity:

The maximum and minimum U_k are represented as PIx_UKMAX (x=0 ~ 3) and PIx_UKMIN.

In the entire control process, the rotor speed, flux and torque are each controlled by a separate PI control loop. Given that a significant input error tends to cause windup of the PI controller, the controller output is limited under a clamp value, and the relevant register output is constrained within the bounds PIx_UKMAX and PIx_UKMIN.

2.3 Rotor Position Estimation

2.3.1 Motor Model

The motor position can be estimated with the aid of a DC motor model represented by winding resistance, inductance and back electromotive force (BEMF), as shown in Figure 2-5:

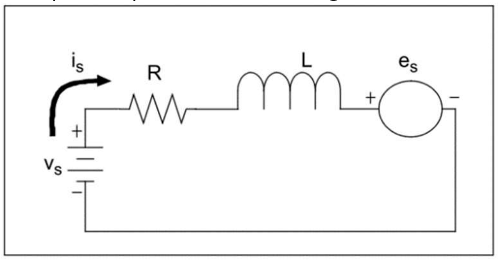


Figure 2-5 Motor Model

The sensorless control technology implements the FOC algorithm by estimating the motor position used to calculate the commutation angle needed for FOC. The motor position and speed can be generally estimated from the measured currents and calculated voltages. The SMO (sliding-mode observer)-based algorithm is one of simple and best-performing algorithms designed for estimating the position of a BLDC motor. It develops a simplified mathematical model of BLDC motor that is represented by winding resistance, inductance and BEMF without compromising the motor's control functionality, as shown in $V_s = Ri_s + L\frac{d}{dt}i_s + e_s$ Equation 2-4.

$$V_s = Ri_s + L\frac{d}{dt}i_s + e_s \qquad \text{Equation 2-4}$$

Where, i_s is motor current vector, e_s BEMF vector, V_s input voltage vector, L winding inductance, and R winding resistance.

In the digital domain, this equation becomes:

$$i_s(n+1) = (1 - T_s * \frac{R}{L}) i_s(n) + \frac{Ts}{L}(V_s(n) - e_s(n))$$ Equation 2-5

The above equation represents a digitalized model of motor known as current observer. The motor position and speed estimator is based on a current observer. This mathematical model provides a software representation of the hardware. In order to match measured current and estimated current, the digitalized motor model must be corrected using closed loop, as shown in the figure below.

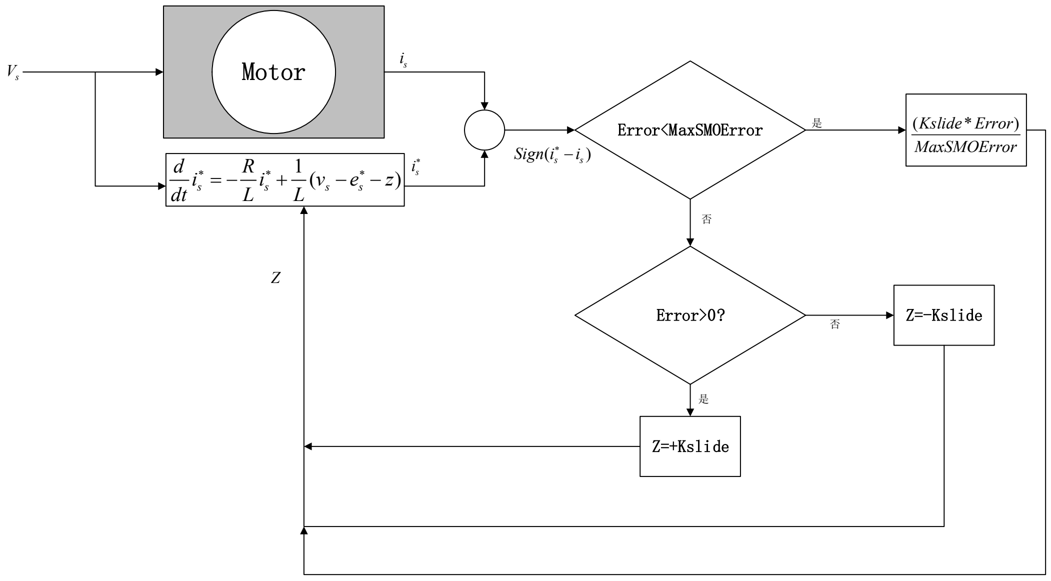

V_s
Motor
i_s
$\frac{d}{dt}i_s^* = -\frac{R}{L}i_s^* + \frac{1}{L}(v_s - e_s^* - z)$
i_s^*
$Sign(i_s^* - i_s)$
Error<MaxSMOError
是
$\frac{(Kslide * Error)}{MaxSMOError}$
否
Z
Error>0?
否
Z=-Kslide
是
Z=+Kslide

Figure 2-6 Current Observer Block Diagram

Considering two motor representations, one is the physical motor (shaded area) and the other is a digitalized motor system simulated by software, with the same input voltage (V_s) fed into both systems, and matching the estimated current (i_s^*) from the model with measured current (i_s), we can presume that BEMF (e_s^*) from the digitized model is the same as the BEMF (e_s) from the motor.

After compensating the digitized model, you have a motor model with the same variable values for the input voltage and for current. And the next step is to estimate BEMF (e_s^*) by filtering the correction factor (Z). The estimated BEMF (e_s^*) is updated in each control cycle and fed back to the digital model. e_α and e_β are the vector components of e_s, which can be used to estimate θ^*. Arc cotangent is computed on the BEMF vector components to calculate theta, as illustrated in Equation 2-6:

$$\theta = arc\tan(e_\alpha, e_\beta)$$ Equation 2-6

Since the filtering function is applied in theta calculation, some phase compensation is needed before the calculated angle is used to energize the motor windings. The amount of theta compensation depends on the rate of change of theta, or the motor speed. The theta compensation is performed in two steps:

1. The motor speed is calculated based on the uncompensated theta calculation.
2. The calculated speed is filtered and used to calculate the amount of compensation, as shown in Figure 2-7.

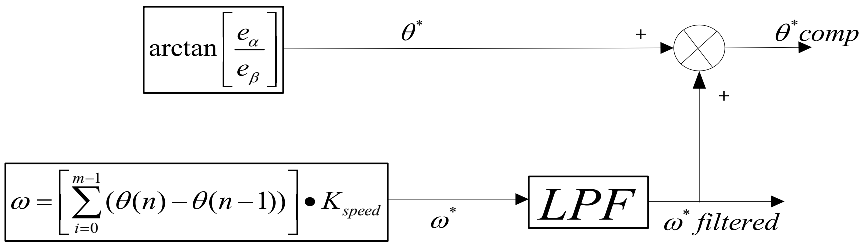

$\arctan\left[\frac{e_\alpha}{e_\beta}\right]$
θ^*
+
$\theta^* comp$
+
$\omega = \left[\sum_{i=0}^{m-1}(\theta(n)-\theta(n-1))\right] \bullet K_{speed}$
ω^*
LPF
ω^* filtered

Figure 2-7 Speed Calculation Block Diagram

3 Current Sampling Theories

3.1 Two-/Three-Shunt Sampling

3.1.1 Sampling Mode Setup and Theories

3.1.1.1 Theories and Configuration of Sequential Sampling Mode

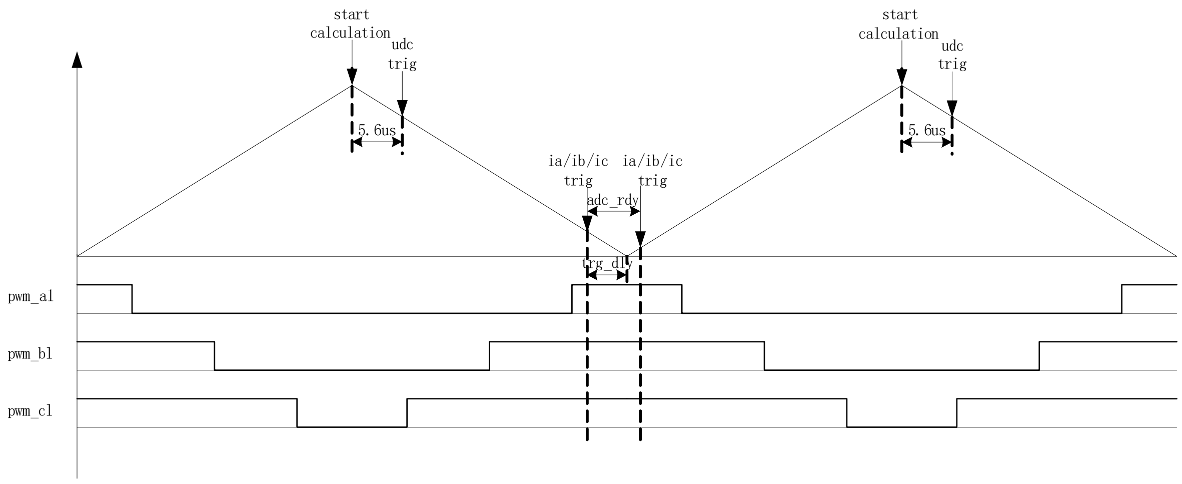


Figure 3-1 Two-/Three-Shunt Sequential Current Sampling Mode

Configure FOC_CR1[CSM] = 10/11 and FOC_CR2[DSS] = 0, and select the sequential sampling mode of double/triple resistance current. In the three-resistance current sequential sampling mode, the sampling opportunity of one phase current (ia/ib/ic is determined according to the sector) of the three-phase current is set through the FOC_TRGDLY register, and the other phase is sampled quickly after the sampling. In the double-resistance mode, adjust the sampling timing of ia by setting the FOC_TRGDLY register, and sample ib quickly after sampling. Note that the current sampling timing should be so configured that the sampling points of three phase currents are all within the vector 000 range. Example: FOC_TRGDLY = 0xb2, then when the FOC counter counts down, the ia/ib/ic is sampled 50*T = 2.08μs before the underflow event, and the other phase ia/ib/ic is sampled after the sampling.

3.1.1.2 Theories and Configuration of Alternate Sampling Mode

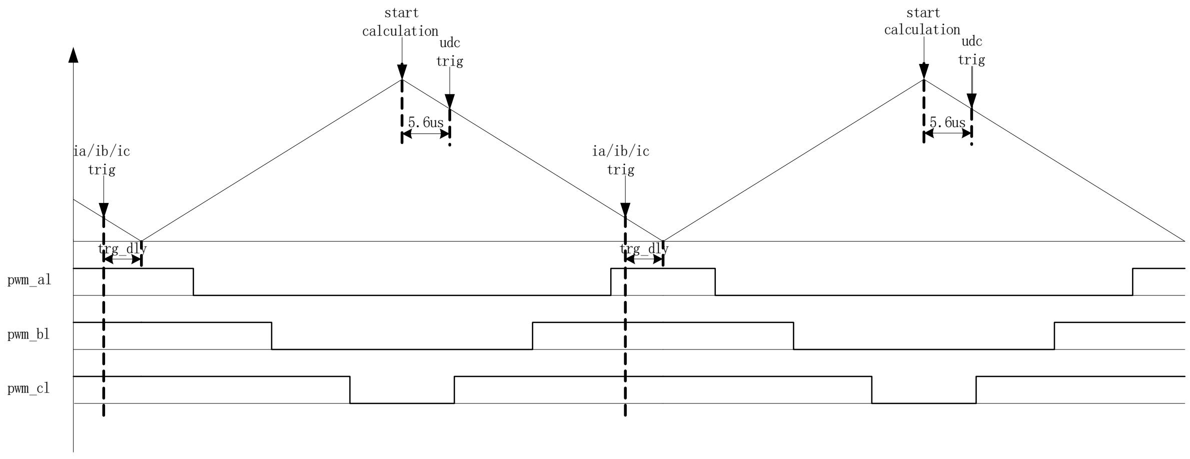
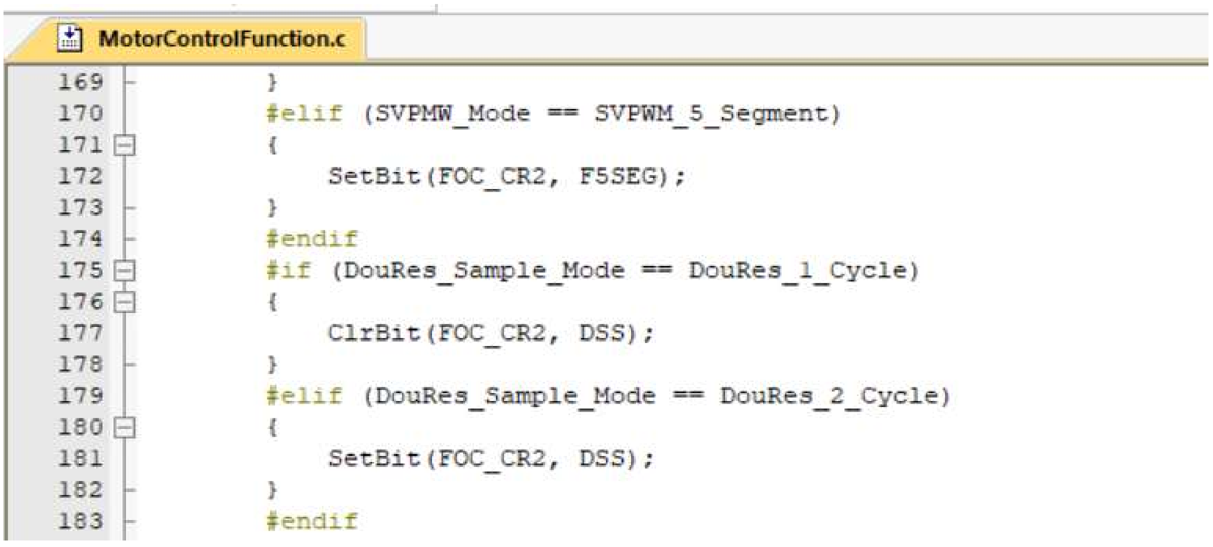


Figure 3-2 Alternating sampling mode of double/triple resistance current

Configure FOC_CR1[CSM] = 10/11 and FOC_CR2[DSS] = 1, that is, choose the alternate sampling mode of double/triple resistance current. In the double/triple resistance current alternating sampling mode, the FOC module operates once in a carrier cycle, but only samples one phase current (ia/ib/ic according to the sector). One phase of ia/ib/ic is collected in the previous carrier cycle, and the phase current of the other phase is collected in the next carrier cycle, so that the current of two of the three phases is sampled alternately. Set the sampling timing of current ia (channel 0), ib (channel 1) and ic (channel 4) through the FOC_TRGDLY register. It should be noted that the sampling timing of current should be set so that the sampling points of ia/ib/ic are all in the vector 000 range. Example: FOC_TRGDLY = 0xb2, then when the FOC counter counts down, the current will be sampled 50*T = 2.08μs before the underflow event.

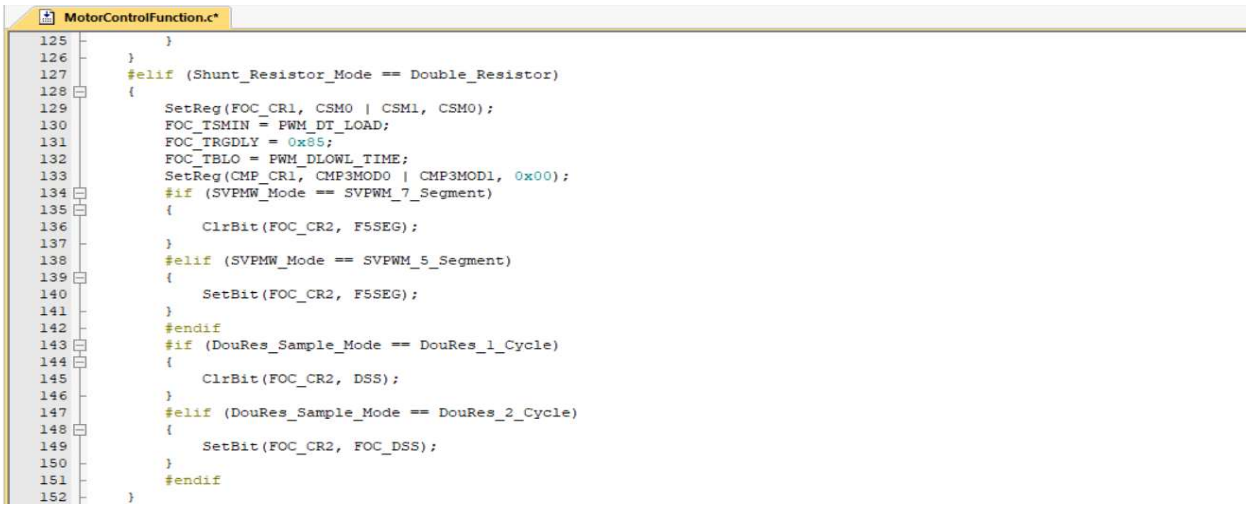
The double/triple resistance current sampling mode samples the DC bus voltage after the Driver counter counts down and the FOC module operation is completed.

3.1.1.3 Sampling mode configuration code


MotorControlFunction.c

```
        }
        #elif (SVPMW_Mode == SVPWM_5_Segment)
        {
            SetBit(FOC_CR2, F5SEG);
        }
        #endif
        #if (DouRes_Sample_Mode == DouRes_1_Cycle)
        {
            ClrBit(FOC_CR2, DSS);
        }
        #elif (DouRes_Sample_Mode == DouRes_2_Cycle)
        {
            SetBit(FOC_CR2, DSS);
        }
        #endif
```

3.1.2 Code description



MotorControlFunction.c*

```
125             }
126         }
127         #elif (Shunt_Resistor_Mode == Double_Resistor)
128         {
129             SetReg(FOC_CR1, CSM0 | CSM1, CSM0);
130             FOC_TSMIN = PWM_DT_LOAD;
131             FOC_TRGDLY = 0x85;
132             FOC_TBLO = PWM_DLOWL_TIME;
133             SetReg(CMP_CR1, CMP3MOD0 | CMP3MOD1, 0x00);
134             #if (SVPMW_Mode == SVPWM_7_Segment)
135             {
136                 ClrBit(FOC_CR2, F5SEG);
137             }
138             #elif (SVPMW_Mode == SVPWM_5_Segment)
139             {
140                 SetBit(FOC_CR2, F5SEG);
141             }
142             #endif
143             #if (DouRes_Sample_Mode == DouRes_1_Cycle)
144             {
145                 ClrBit(FOC_CR2, DSS);
146             }
147             #elif (DouRes_Sample_Mode == DouRes_2_Cycle)
148             {
149                 SetBit(FOC_CR2, FOC_DSS);
150             }
151             #endif
152         }
```

- Line 130 sets the dead-time compensation value. Pay special attention to the fact that the dead-time compensation can only be used in the double/triple resistance current sampling mode. In the single resistance mode, the dead-time compensation needs to be set to 0, which can effectively improve the sine of the current waveform at low speed.
- Line 132 sets the minimum pulse width of the lower bridge, FOC_TBLO, which is used in the three-resistance current sampling mode. When the lower bridge is turned on for less than FOC_TBLO, the current of this phase is not sampled, and the current is obtained by special treatment.
- In high-voltage application, the upper bridge of the high-voltage Pre-driver has the requirement that the minimum conduction pulse must be greater than a certain value. Set the upper bridge conduction narrow pulse elimination check FOC_TGLI. When the conduction pulse is less than the set value, it will not conduct, which will improve the noise to some extent.

3.1.3 Deadtime Compensation

Due to the delay of the actual power tube opening and closing, the dead time must be inserted to avoid the same bridge arm through. However, the dead time causes the output voltage loss, and the dead time compensation is to compensate the output voltage loss of H-bridge inverter caused by the dead time.

Dead-time effect brings additional problems: for power supply control, output voltage distortion, harmonic current, THD, the total harmonic distortion index, rises, especially for low-frequency operation; For motor control, the introduction of 6-fold torque ripple will affect the low-speed performance and hinder the application of position-free algorithm and high-performance frequency converter. Therefore, it is generally necessary to compensate the dead zone for applications with high requirements on low-speed noise, vibration and current sine.

As shown in Figure 3-1, before the dead-time compensation, there is an obvious platform when the dead-time effect causes the current to cross zero, which will introduce torque ripple and noise;

As shown in Figure 3-2, after the dead-time compensation, the sine of current is obviously improved, and the total harmonic distortion index THD is obviously improved.

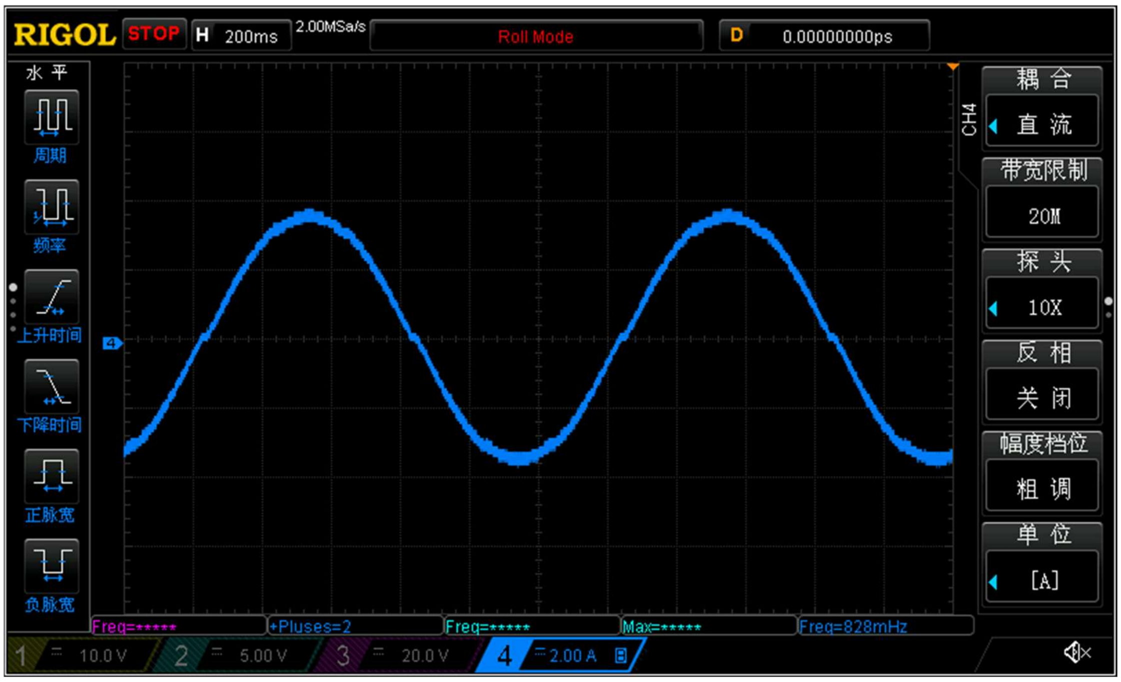


Figure 3-3 Before Deadtime Compensation

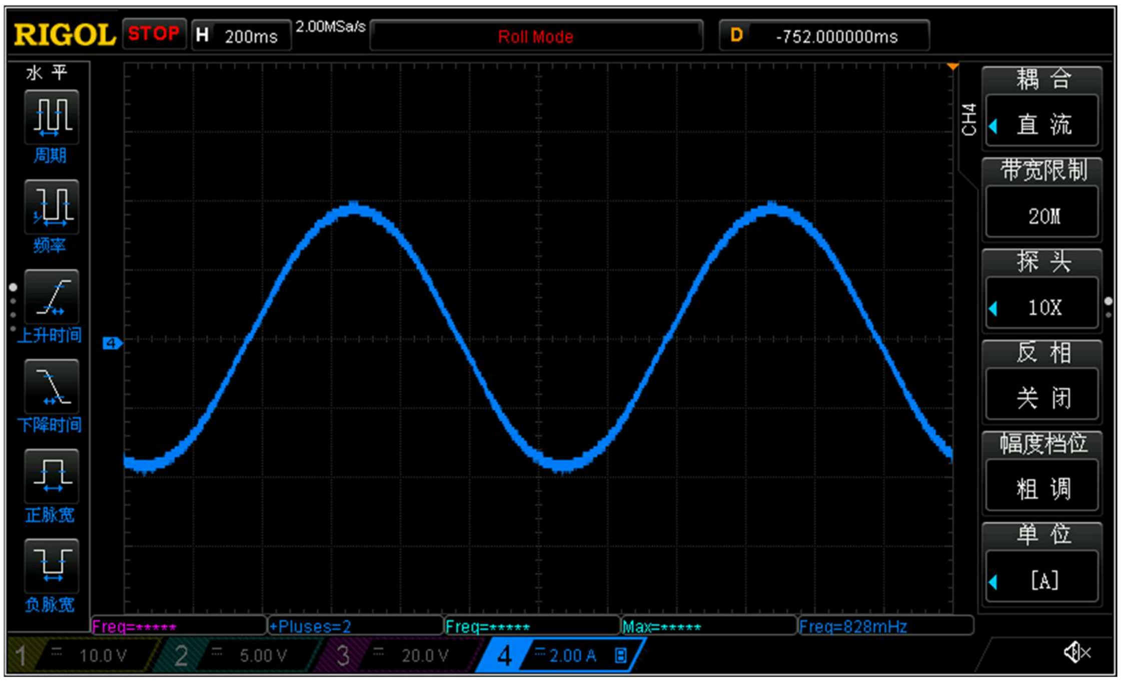


Figure 3-4 After Deadtime Compensation

3.2 Single Shunt Sampling

3.2.1 Theories and Configuration

Configure FOC_CR1[CSM] = 00, and select the single resistance current sampling mode. In the single resistance current sampling mode, the FOC module samples the bus current itrip (channel 4) twice in the up-counting interval of the Driver counter, and samples the DC bus voltage in the down-counting interval of the Driver counter after the operation of the FOC module is completed.

Sampling in the dead time will affect the accuracy of current sampling. The FOC module needs to sample in the effective vector application time T1′ and T2′ to remove the dead time. By configuring FOC_TRGDLY to advance or delay the sampling time, the sampling can be completed in T1' and T2'. Example: FOC_TRGDLY = 5, the delay is 5 * t = 208ns; FOC_TRGDLY = 0xfb (-5), it will be advanced by 208ns.

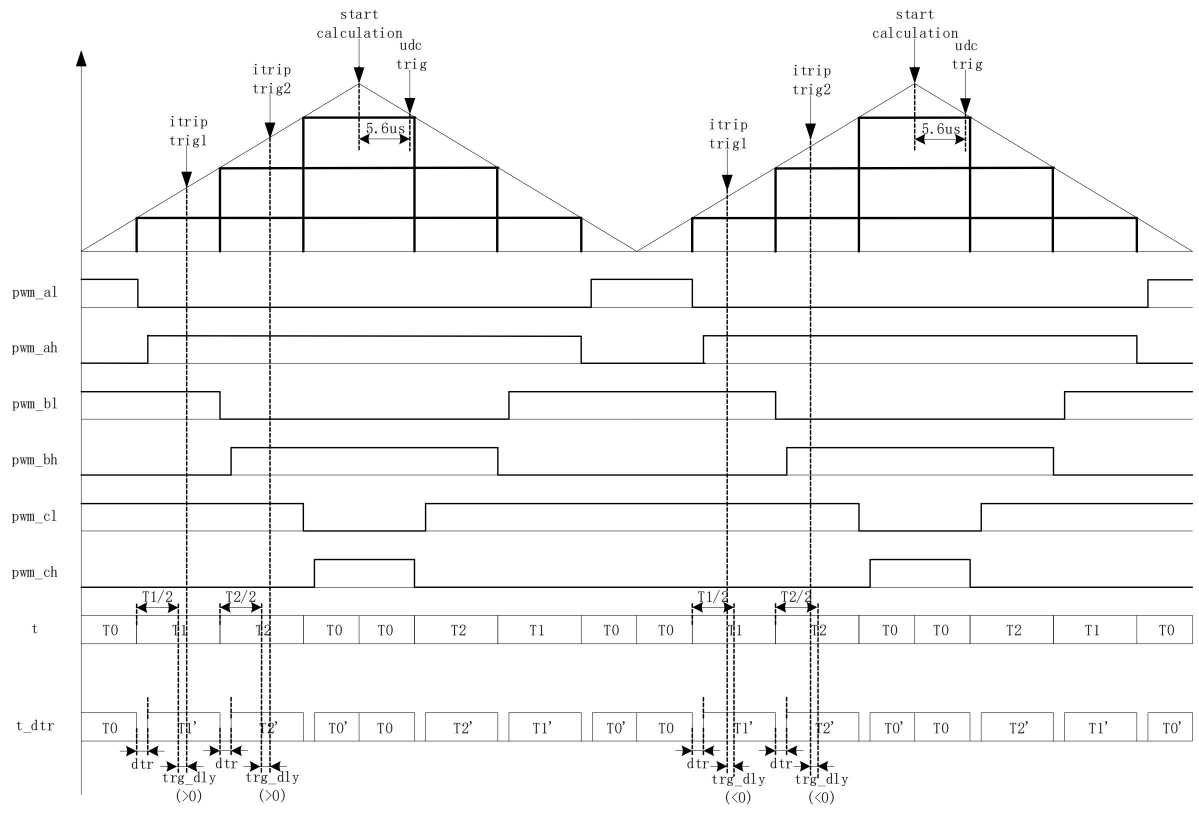


Figure3-5 Single-shunt Sampling Sequence

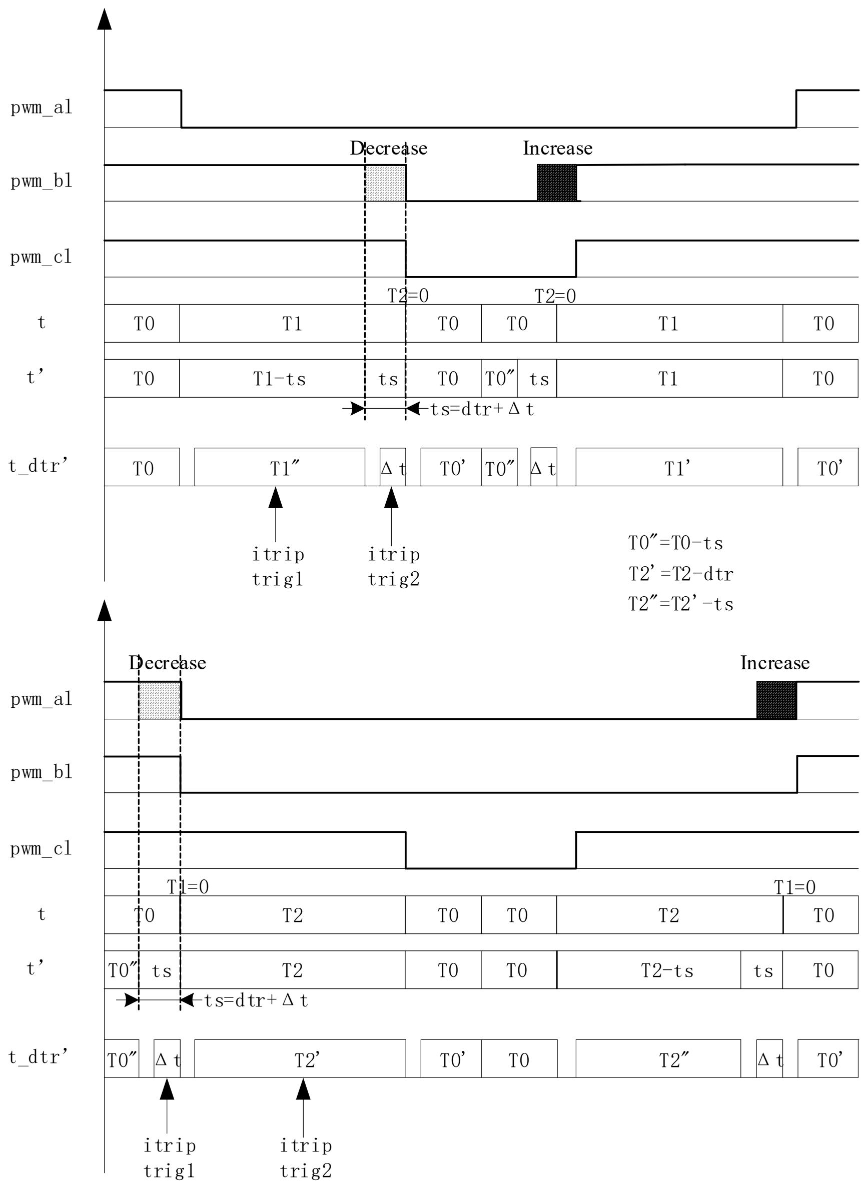
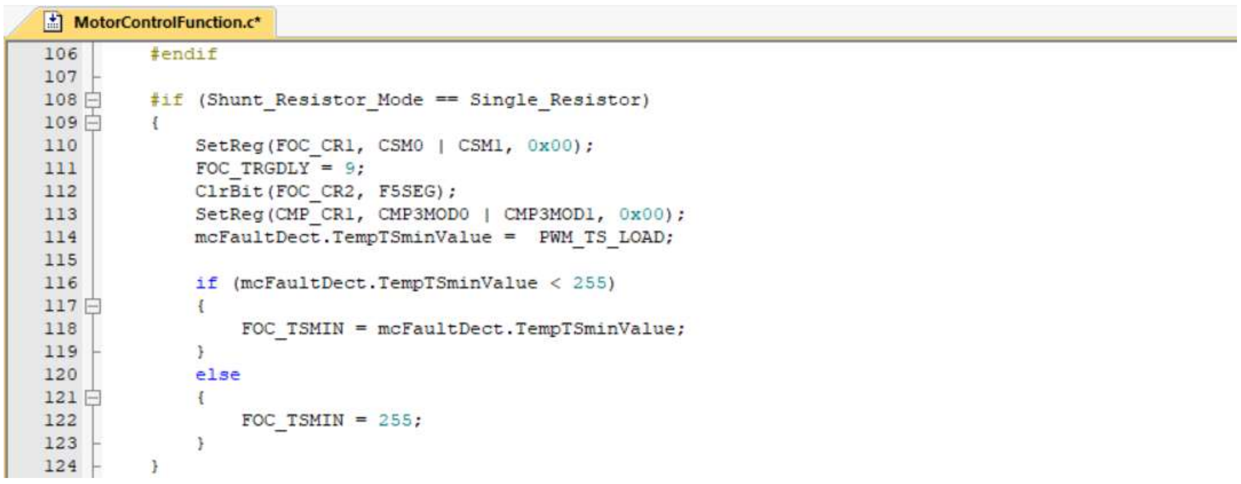


Figure 3-6 Single-shunt Sampling Time Compensation

The sampling window of single resistor sampling is not wide enough in low-key system and sector switching. At this time, it is necessary to adjust the output waveform to ensure the minimum sampling window required for sampling. By setting FOC_TSMIN (FOC_TSMIN = minimum sampling window time + dead time), the FOC module will automatically phase shift the PWM waveform.

3.2.2 Code description


MotorControlFunction.c*

```
106     #endif
107
108     #if (Shunt_Resistor_Mode == Single_Resistor)
109     {
110         SetReg(FOC_CR1, CSM0 | CSM1, 0x00);
111         FOC_TRGDLY = 9;
112         ClrBit(FOC_CR2, F5SEG);
113         SetReg(CMP_CR1, CMP3MOD0 | CMP3MOD1, 0x00);
114         mcFaultDect.TempTSminValue =  PWM_TS_LOAD;
115
116         if (mcFaultDect.TempTSminValue < 255)
117         {
118             FOC_TSMIN = mcFaultDect.TempTSminValue;
119         }
120         else
121         {
122             FOC_TSMIN = 255;
123         }
124     }
```

- The code on line 110 is set to single resistor sampling mode.
- Line 111 sets the sampling delay or lag, with the unit of 41.67ns；; When it is set to a positive number, the sampling time is delayed at this time, otherwise it is advanced, and the typical value is generally 2 ~ 12 units.
- Line 112 sets 5-segment mode and 7-segment mode
- Lines 114 ~ 123 set the minimum sampling window value, paying special attention to FOC_TSMIN < carrier period *1/16 (unit μs)

4 Five/Seven Segment SVPWM

4.1 SVPWM Theories

SVPWM is short for Space Vector Pulse Width Modulation. The theory is based on the average equivalent principle, that is, in a switching cycle by combining the basic voltage vector, so that the average value and the given voltage vector is equal. At some point, the voltage vector rotates into a region, which can be obtained by the different combinations of two adjacent non-zero vectors and zero vectors in time. The action time of the two vectors is applied several times in a sampling period, so that the action time of each voltage vector is controlled, and the voltage space vector nearly rotates according to the circular track, the ideal flux circle is approximated by the actual flux generated by different switching states of the inverter, and the switching states of the inverter are determined by the results of the comparison between the two, thus the PWM waveform is formed.

Take the first sector, for example:

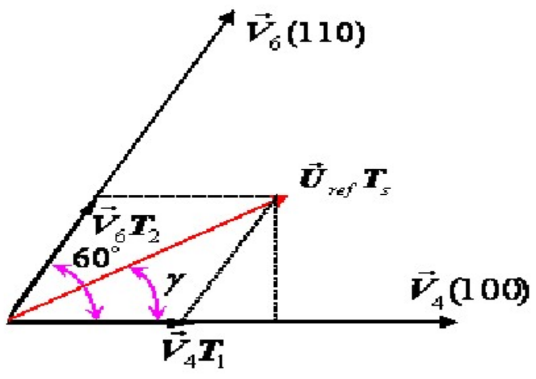


Figure 4-1 Sector I Vectors

According to the diagram, it can be concluded that:

$$\begin{cases} \vec{U}_{ref}T_S = \vec{V}_4T_1 + \vec{V}_6T_2 \\ T_0 = T_s - T_1 - T_2 \end{cases}$$

$$\Rightarrow \begin{cases} \vec{u}_{ref\alpha}T_S = T_1\vec{V}_4 + T_2\vec{V}_6\cos 60^\circ \\ \vec{u}_{ref\beta}T_S = T_2\vec{V}_6\sin 60^\circ \end{cases} \Rightarrow \begin{cases} T_1 = \frac{1}{2}(\sqrt{3}u_{ref\alpha} - u_{ref\beta})\frac{\sqrt{3}T_s}{V_{dc}} \\ T_2 = u_{ref\beta}\frac{\sqrt{3}T_s}{V_{dc}} \end{cases}$$

$T_0 = T - T_1 - T_2$

4.2 Comparison of merits and demerits

	Merits	Demerits
Five-segment SVPWM	■ The number of PWM cycle switching is reduced by 1/3, and the heating of power device is reduced ■ The maximum duty cycle of PWM output is higher than seven degrees, the maximum can reach 97%	■ The harmonic interference of the current is greater
Seven-segment SVPWM	■ The current waveform is more close to the circular, less harmonic interference	■ The number of switches, switch damage ■ The highest duty cycle of 7-segment PWM output is 93% ~ 95%

4.3 Seven-segment SVPWM

With the goal of reducing the number of switches, the basic vector action sequence assignment principle is selected as follows: In each switch state transition, only one phase of the change, the switch state. And the zero vector is distributed evenly in time to make the generated PWM symmetrical, so that the harmonic component of PWM can be reduced effectively.

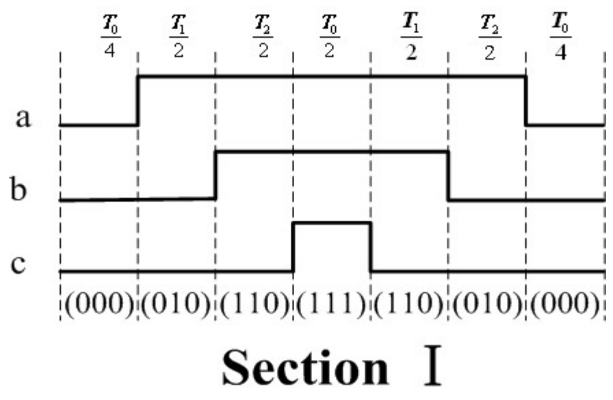


Figure 4-2 Section I seven-segment three-phase PWM wave

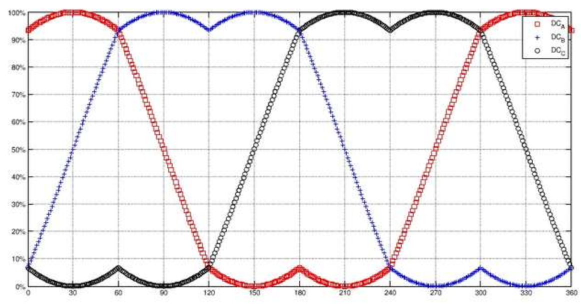


Figure 4-3 Seven-segment SVPWM phase voltage waveform

4.4 Five-Segment SVPWM

For seven-segment SVPWM, the generating wave is symmetric and the harmonic content is small, but there are 6 switching times per switching cycle, in order to further reduce the switching times, each phase switch keeps the same sequence arrangement in each sector state, so there are only 3 switching times in each switch cycle, but the harmonic content is increased. The specific sequence arrangement is as follows:

- In five-segment SVPWM, the phase voltage of one phase can be constant zero (duty cycle output is 0%) or constant 1(duty cycle is 100%) without switching during a PWM cycle. EU68XX series chip adopts constant-zero mode.

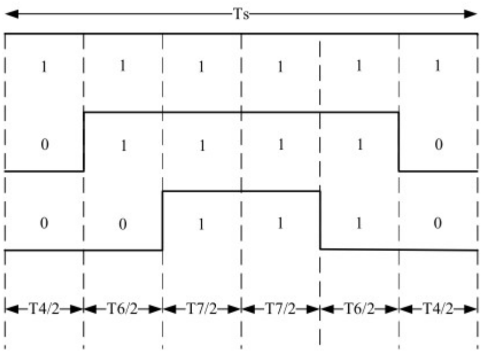


Figure 4-4 Five-segment Three-phase PWM Waveform in Sector I

- Five-segment SVPWM also has different waveform modes in a PWM cycle, such as edge-aligned and center-aligned two modes, EU68XX series chips use center-aligned mode.
- When the SVPWM waveform is symmetrical, the PWM duration is divided into 5 segments, which is the source of the five-segment PWM name.

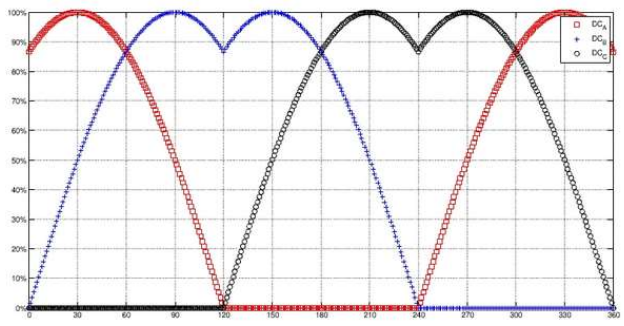
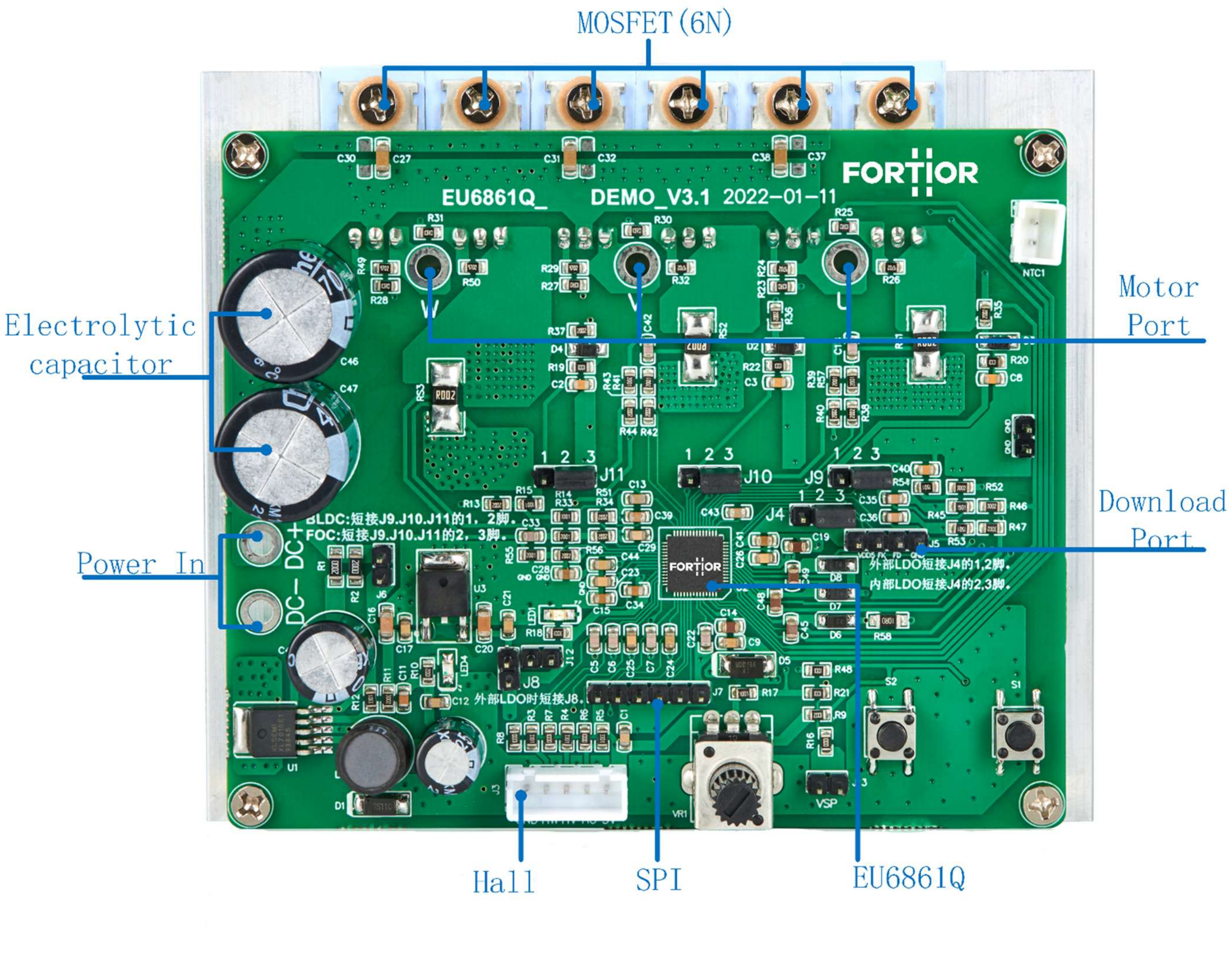


Figure 4-5 Five-segment SVPWM Phase Voltage Waveform

5 Hardware Mechanism

This chapter introduces the hardware content of Senseless FOC debugging based on EU6861Q- medium voltage DEMO.

5.1 Physical Drawing



5.2 Schematic

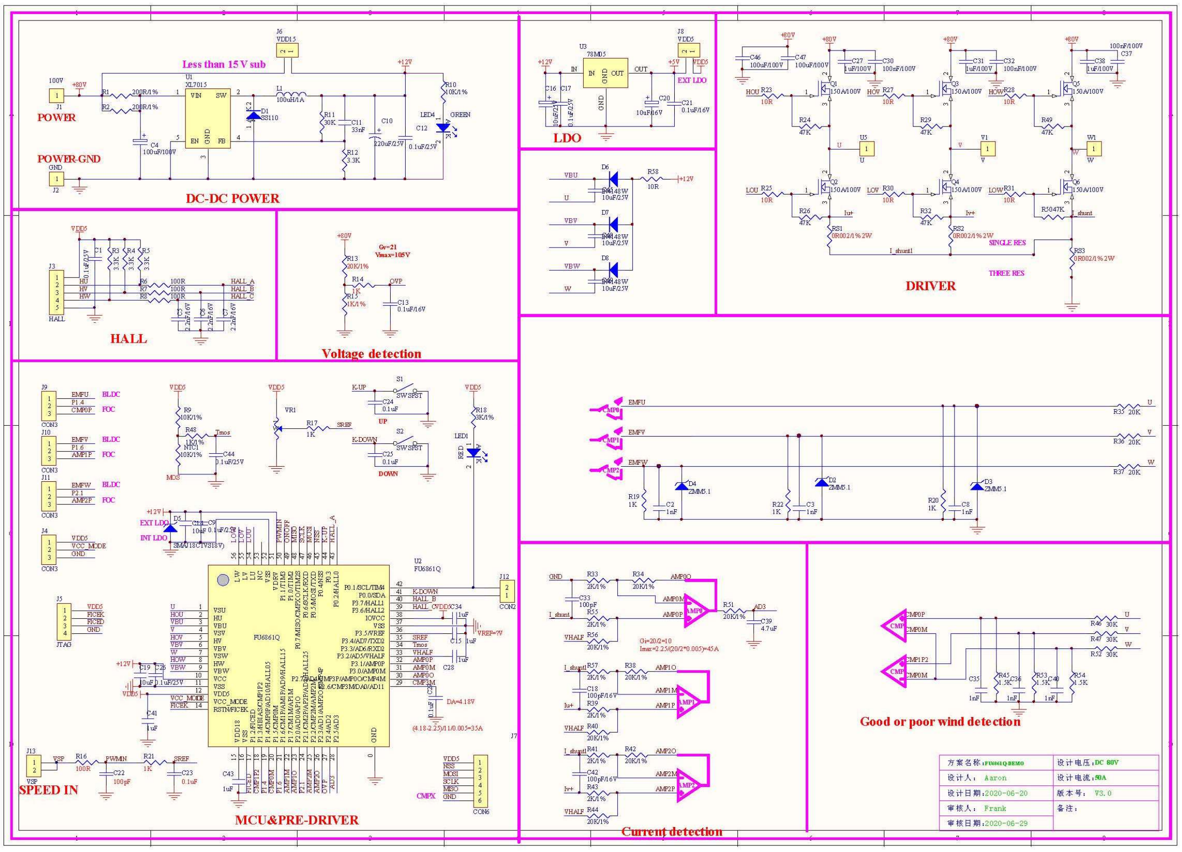


Figure 5-1 Demo Board Schematic

5.2.1 Directions for Use

- If the BEMF detection method is used for upstream and downstream detection, it is necessary to select the short circuit of pins 1 and 2 of BLDC, J9, J10 and J11 ports.
- When external LDO is selected, J4 port and pins 1 and 2 are shorted to select external LDO, and J8 also needs to be shorted at this time.
- When internal LDO is selected, J4 port and 2 and 3 pins are shorted to select internal LDO, so J8 does not need to be shorted.

5.2.2 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.

5.3 Power Supply

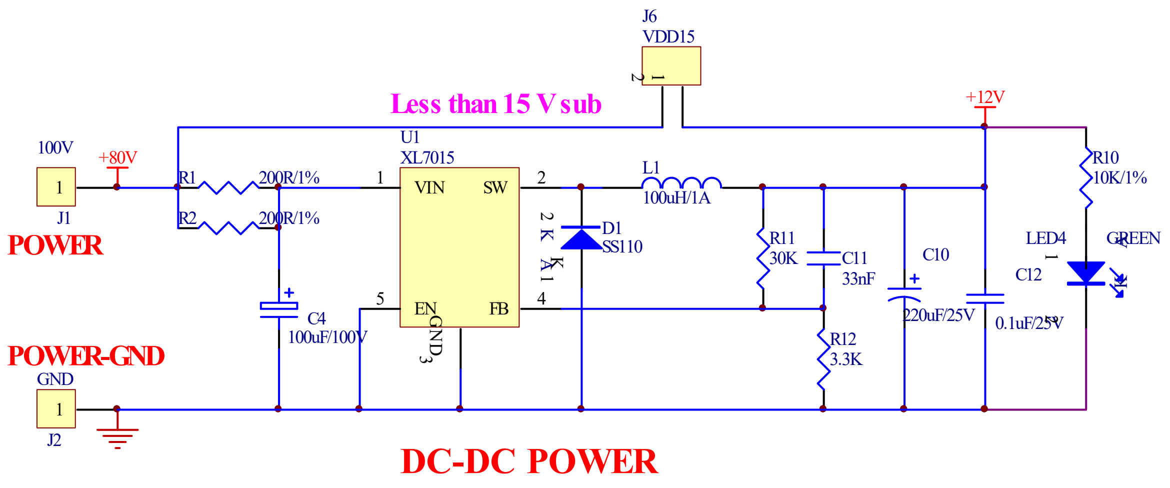


Figure 5-2 Power Supply Schematic

5.3.1 Directions for use

- When the DC bus voltage is lower than 15V, J6 is shorted by jumping cap, and the external power supply directly supplies power to the chip.
- As the VCC voltage of the chip is not recommended to be higher than 18V, in order to make the DEMO board compatible with wide voltage, turn off J6, and the external power supply will supply power to the VCC of the chip through the switching DC-DC conversion chip.
- The principle of switching power supply is not explained in detail in this manual. Pay special attention to the fact that the output voltage range is 1.25V ~ 20V, which is adjustable and determined by the voltage dividing ratio of R11 and R12.

5.3.2 Instructions

- When applying low voltage to J6 short circuit, be careful that the maximum voltage should not exceed 18V, otherwise it is easy to burn the chip.

5.4 Chip Body

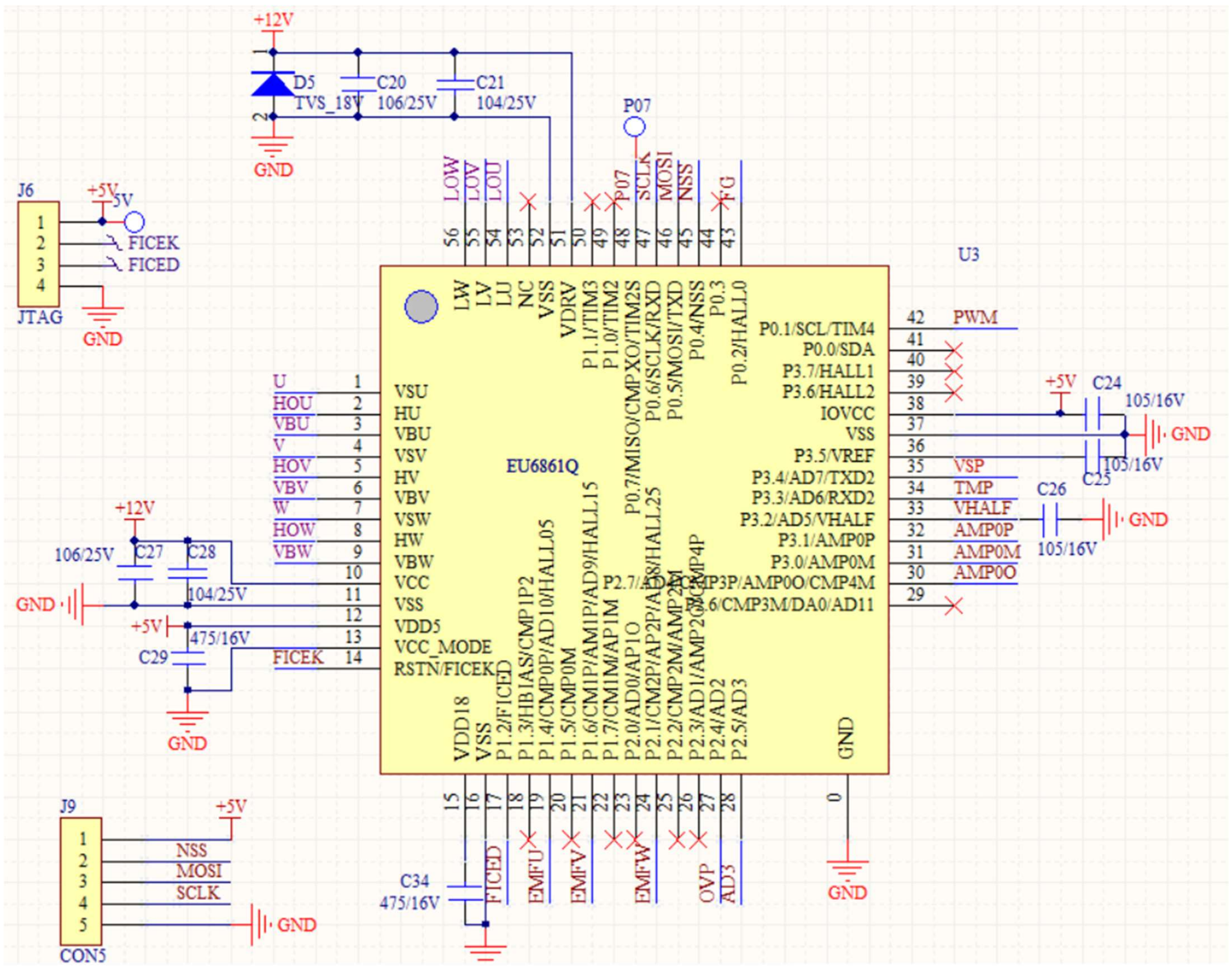


Figure 5-3 Chip Body Schematic

5.4.1 Directions for use

EU6861Q2 is used for LV/MV 6-NMOSFET driver applications. EU6861Q2 and EU6861Q are in pin-to-pin packages, allowing for direct replacement.

5.4.2 Instructions

Capacitors must be placed as close to the chip as possible.

5.5 BEMF Detection Circuit

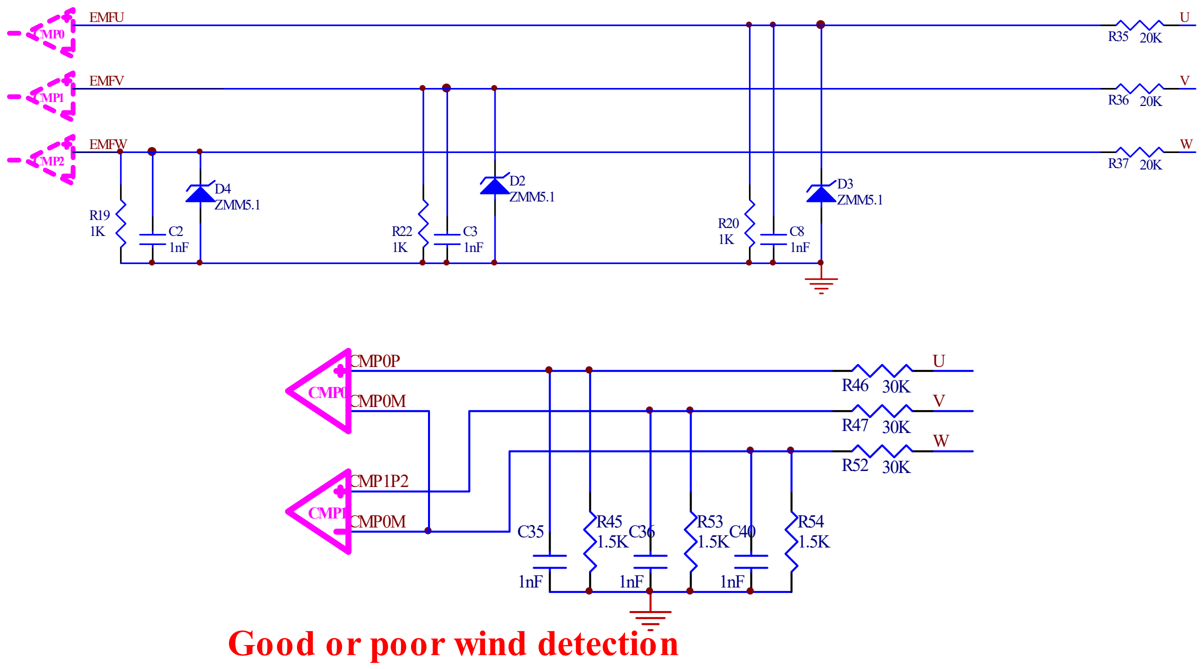


Figure 5-4 BEMF Detection Circuit

5.5.1 Instructions

1. When the 3-way comparator is idle, the BEMF mode of the 3-way comparator shown above can be used;
2. When the 2-way comparator is idle, the 2-way comparator RSD can be used as shown below;
3. Comparators with saving RSD are often used for ground brush overcurrent protection.

5.6 Power Driver

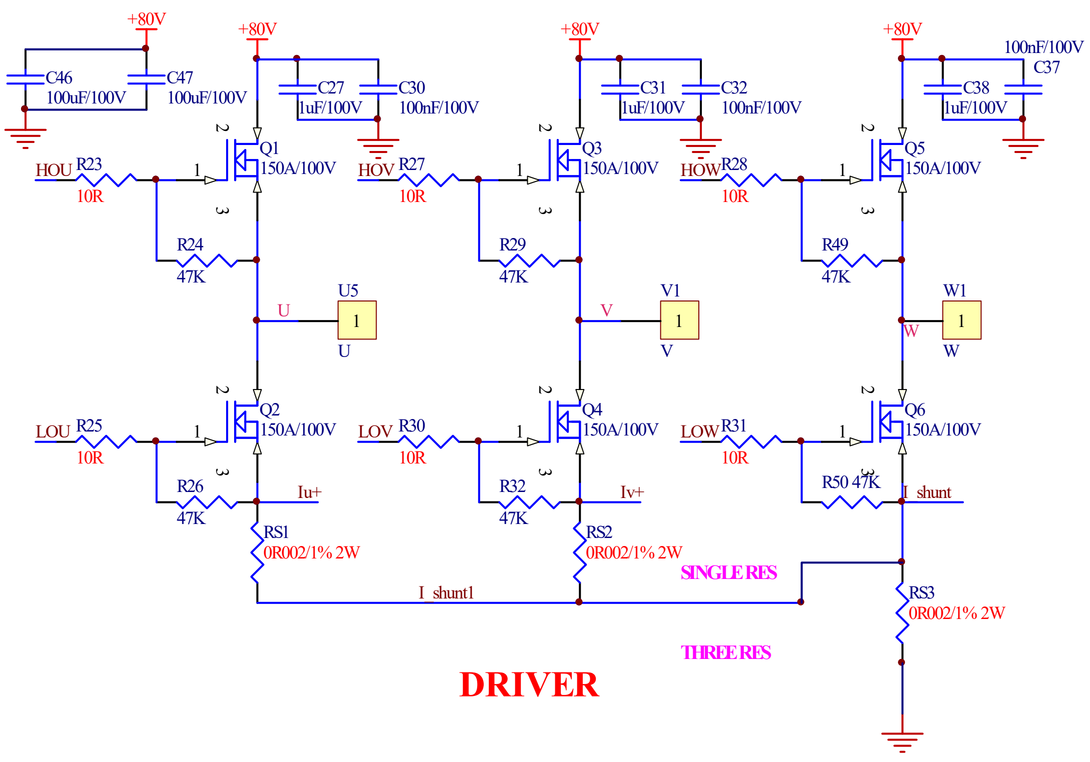


Figure 5-5 Power Driver Circuit Schematics

5.6.1 Instructions

1. C27, C31 and C38 capacitors cannot be connected to the I_shunt terminal of the shunt resistor, which will cause sampling interference;
2. The power of shunt resistor may not exceed 80% of the power rating at the maximum current.
3. Single-shunt setup is employed for all vacuum cleaners, so RS1 and RS2 are directly shorted.

5.7 Op-amp Configuration Circuit

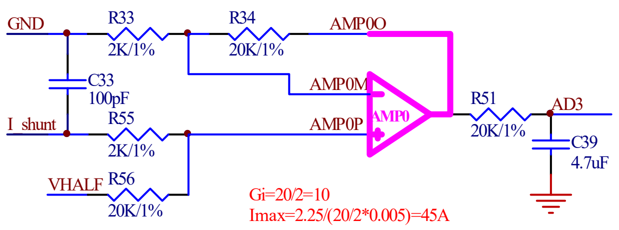


Figure 5-6 Op-amp Configuration Circuit Schematics

5.7.1 Instructions

1. C33 is a non-adjustable parameter with a required accuracy of 10%;

2. ±1% tolerance resistors are required for R34, R35 and R55 and R56;
3. Here, AD3 is used for sampling the average bus voltage, which cannot be omitted;
4. Gain = R34/R33 = R56/R55;
5. Maximum sampling current = (VREF - VHALF)/gain/shunt resistor value;
6. The maximum sampling current is generally set to about 4 times the maximum bus current.

5.8 DC Bus Voltage Sampling

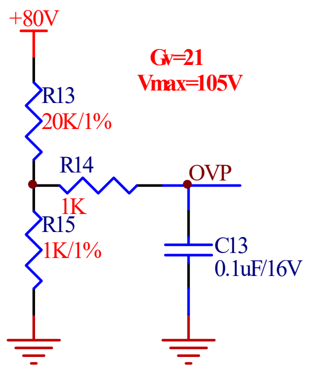


Figure 5-7 DC Bus Voltage Sampling Circuit

5.8.1 Instructions

1. Parameters R14 and C13 are not adjustable;
2. ±1% tolerance resistors are required for R13 and R15;
3. Maximum sampling voltage = (R13+R15)/(R15);
4. Maximum sampling voltage selected is typically 2 times the maximum applied voltage, where the OVP threshold must be lower than 0.8*VREF.

5.9 Bootstrap Circuit

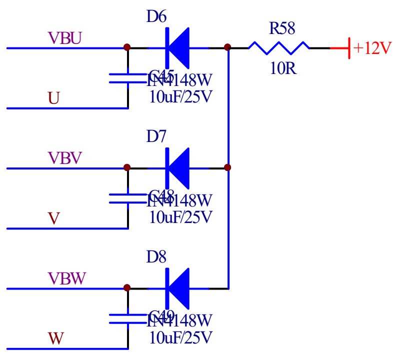


Figure 5-8 Bootstrap Circuit

5.9.1 Instructions

The RC values are referenced to the parameters in the above figure, and no adjustment is needed.

6 Motor State Machine

6.1 Flowchart

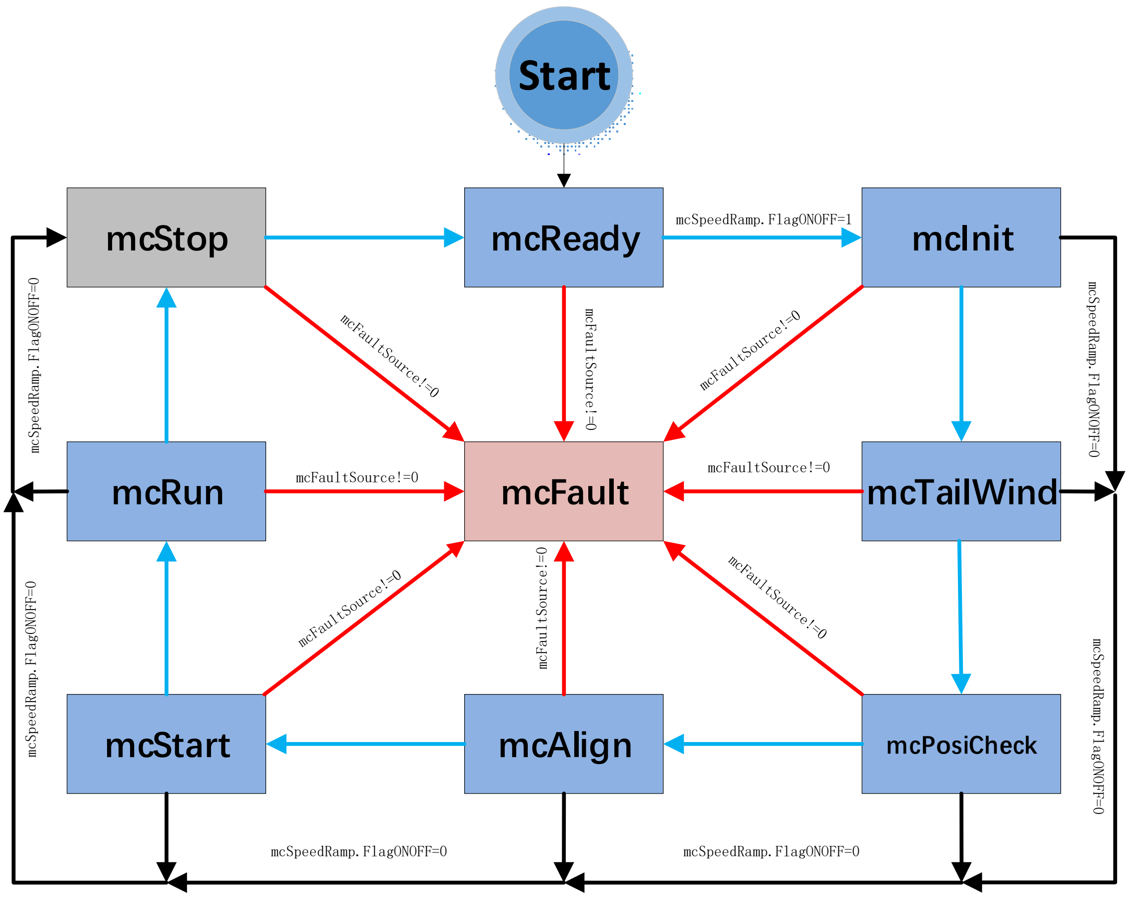


Figure 6-1Motor State Machine Flowchart

As shown in Figure Figure 6-1, the motor state machine has three state transition paths:

- Run (in blue): mcReady -> mcInit -> mcTailWind -> mcPosiCheck -> mcAlign -> mcStart -> mcRun.
- Stop (in black): The state machine will switch to mcStop state from mcInit, mcCharge, mcAlign, mcStart and mcRun states to slow down motor till it comes to a stop if a Stop signal is detected.
- Fault (in red): 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.

6.2 Program Flowchart

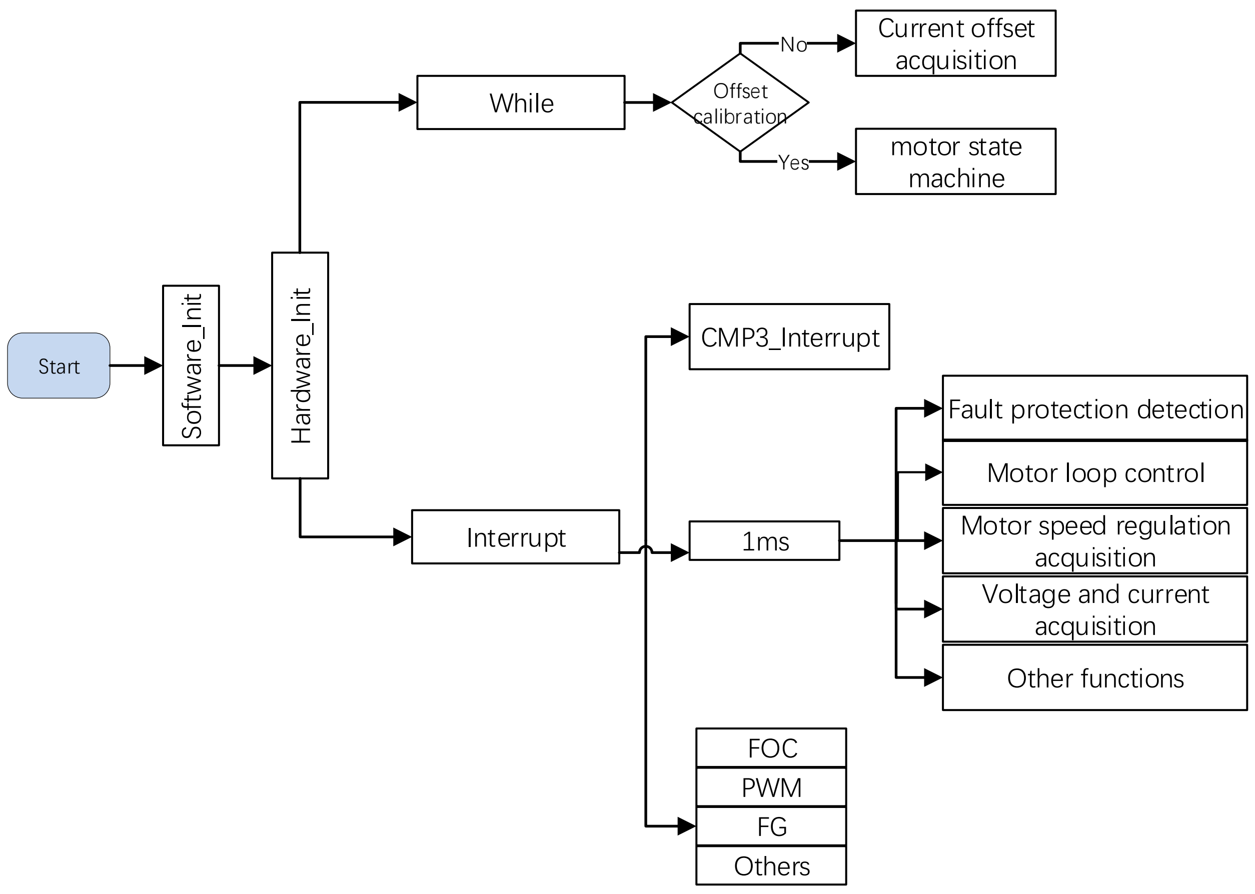
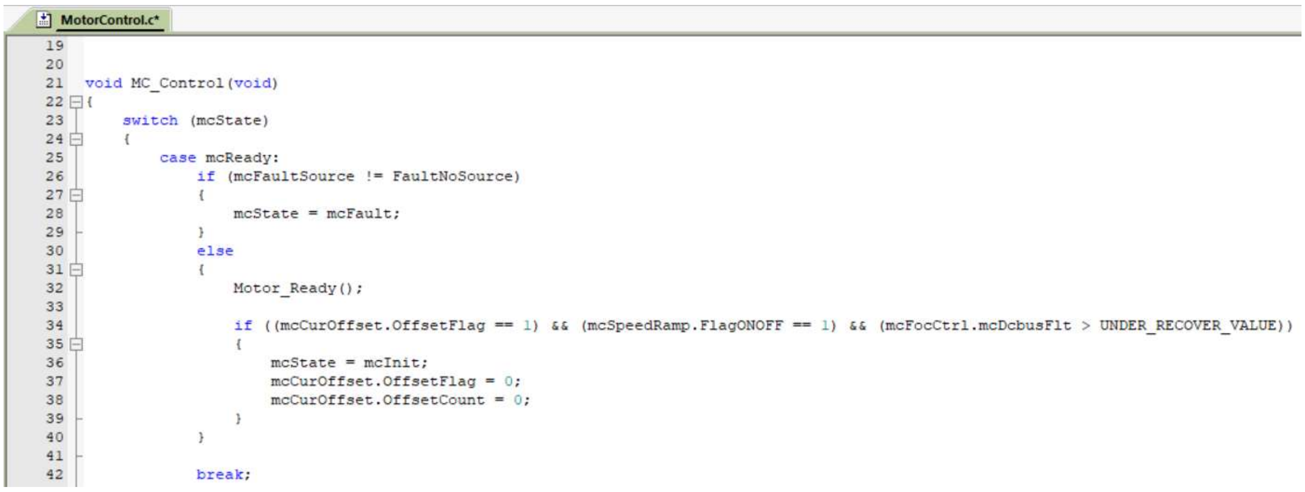


Figure 6-2 Program Execution Flowchart

6.3 mcReady

Ready state, waiting for a “Start ”command. When the bias voltage sensing is completed, the corresponding mcCurOffset.OffsetFlag is set to 1, and the mcSpeedRamp.FlagONOFF is set to 1. When the DC bus voltage detected falls within the normal startup range, the state machine will switch to mcInit state. Note that the DC bus voltage is measured here to prevent the motor from triggering UVLO immediately after voltage is pulled down at startup. Please refer to the following code implementation for details.

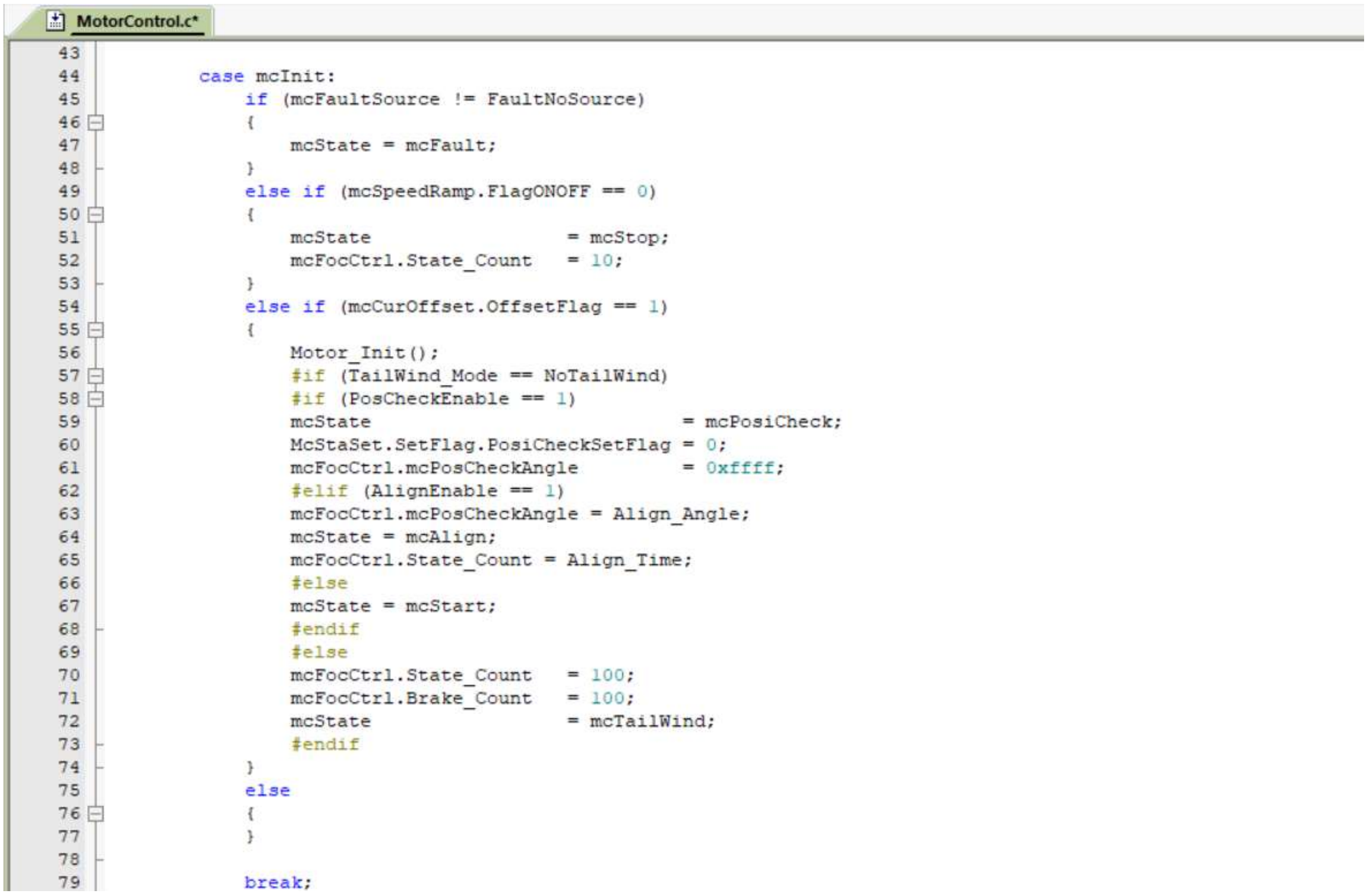


MotorControl.c*

```c


void MC_Control(void)
{
    switch (mcState)
    {
        case mcReady:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else
            {
                Motor_Ready();

                if ((mcCurOffset.OffsetFlag == 1) && (mcSpeedRamp.FlagONOFF == 1) && (mcFocCtrl.mcDcbusFlt > UNDER_RECOVER_VALUE))
                {
                    mcState = mcInit;
                    mcCurOffset.OffsetFlag = 0;
                    mcCurOffset.OffsetCount = 0;
                }
            }

            break;
```

6.4 mcInit

Initialization state, which is mainly designed to initialize the related variables and PI. In this state, motor is switched off and the external ADC is triggered for current sampling. Note that the bias voltage is measured again and mcCurOffset.OffsetFlag is set to 1 before shifting to the next state, so that the motor will not switch to the next state before bias voltage sensing is completed at next startup, which will result in biasing deviation that will affect motor startup. Please refer to the following code implementation for details.



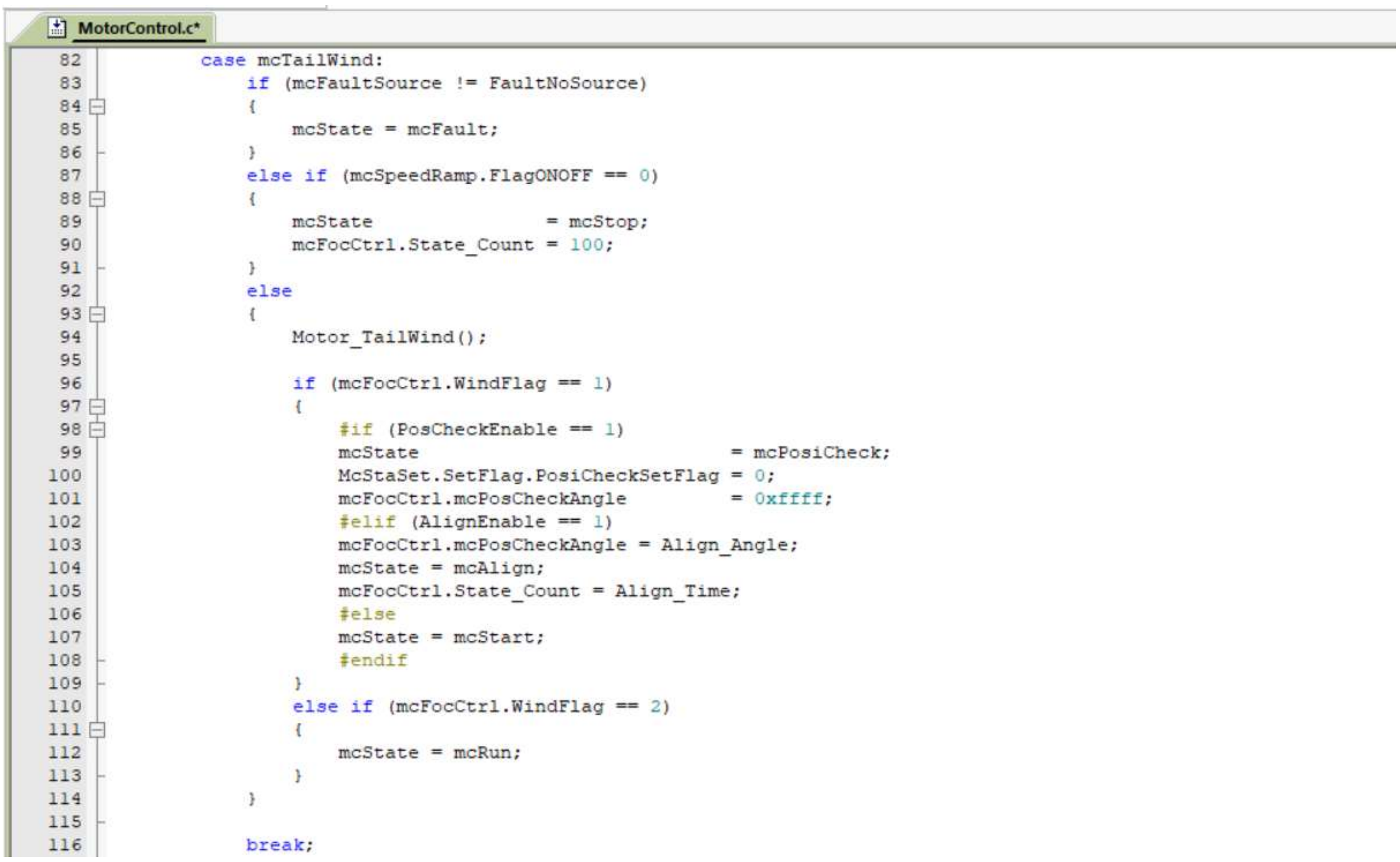
MotorControl.c*

```c

        case mcInit:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else if (mcSpeedRamp.FlagONOFF == 0)
            {
                mcState                 = mcStop;
                mcFocCtrl.State_Count   = 10;
            }
            else if (mcCurOffset.OffsetFlag == 1)
            {
                Motor_Init();
                #if (TailWind_Mode == NoTailWind)
                #if (PosCheckEnable == 1)
                mcState                           = mcPosiCheck;
                McStaSet.SetFlag.PosiCheckSetFlag = 0;
                mcFocCtrl.mcPosCheckAngle         = 0xffff;
                #elif (AlignEnable == 1)
                mcFocCtrl.mcPosCheckAngle = Align_Angle;
                mcState = mcAlign;
                mcFocCtrl.State_Count = Align_Time;
                #else
                mcState = mcStart;
                #endif
                #else
                mcFocCtrl.State_Count   = 100;
                mcFocCtrl.Brake_Count   = 100;
                mcState                 = mcTailWind;
                #endif
            }
            else
            {
            }

            break;
```

6.5 mcTailWind

Tailwind/headwind detection state; when the motor is started, the running speed (BEMFDetect.BEMFSpeed) is calculated by BEMF detection. If the speed is greater than the set tailwind startup speed threshold, it shows that the speed is sufficient for the estimator to estimate the accurate angle, so the motor will directly shift to mcRun state; when headwind is detected, motor will brake or feed negative current before actuating as programmed. Note that headwind is absent in some applications such as vacuum cleaner.

Note that in , the counter-electromotive force detection method is mainly divided into FOC detection method and counter-electromotive force detection method. Because FOC detection method can impulse voltage, it is only suitable for applications with low current. For example, the counter-electromotive force detection method is generally used in high-speed and high-power applications of vacuum cleaners. Counter-electromotive force detection is generally divided into RSD (double comparator) and BEMF (triple comparator). Please refer to the following code for details.



```c
case mcTailWind:
    if (mcFaultSource != FaultNoSource)
    {
        mcState = mcFault;
    }
    else if (mcSpeedRamp.FlagONOFF == 0)
    {
        mcState                 = mcStop;
        mcFocCtrl.State_Count = 100;
    }
    else
    {
        Motor_TailWind();

        if (mcFocCtrl.WindFlag == 1)
        {
            #if (PosCheckEnable == 1)
            mcState                             = mcPosiCheck;
            McStaSet.SetFlag.PosiCheckSetFlag = 0;
            mcFocCtrl.mcPosCheckAngle          = 0xffff;
            #elif (AlignEnable == 1)
            mcFocCtrl.mcPosCheckAngle = Align_Angle;
            mcState = mcAlign;
            mcFocCtrl.State_Count = Align_Time;
            #else
            mcState = mcStart;
            #endif
        }
        else if (mcFocCtrl.WindFlag == 2)
        {
            mcState = mcRun;
        }
    }

    break;
```

6.5.1 FOC Estimation Mode

This function is added before the pre positioning state. When the pre charging is completed, the FOC estimation mode estimates the current speed and angle by setting the given current to 0, and the FOC collects current and voltage information. The program can judge the forward and reverse rotation of the motor according to the current estimated speed, or according to the estimated angle. The former is easy to judge, and the calculated speed is a per unit value.

When some motors are at low speed or at standstill, the response is poor, and it is easy to mistake the estimated speed. The latter is relatively troublesome. It handles the angle change trend through FOC interruption, and divides the angle into three sections, less than - 170 °, more than 170 °, and between - 10 ° and 10 °. When the angle swings back and forth below - 170 ° and above 170 ° without going through - 10 ° to 10 °, it is considered as static. In the angle of - 10 ° to 10 °, calculate the speed, judge the positive and negative rotation, and clear the jitter times. When the motor is in forward rotation, the speed is counted from - 180 ° to - 10 °, and when the motor is in reverse rotation, the speed is counted from 180 ° to 10 °. Therefore, when calculating the speed, only half of the electrical cycle is counted. The speed unit calculated in this way is rpm/min.

In FOC estimation mode, SPI is required to observe whether the estimation angle is normal. For example, in forward rotation, the angle is a change of ascending slope, while in reverse rotation, the angle is a change of descending slope. In static rotation, the angle may not change, or it may be a form of up and down level jump. If the estimated angle direction is incorrect or the response is not fast enough, the following three parameters need to be adjusted: Observer bandwidth filter value ATO_ BW_ Wind, speed bandwidth filtering value SPD_ BW_ Wind, current ring Kp, Ki. In order to speed up the estimation response, the current loop Kp and Ki will be larger than the current loop in actual operation when judging the wind direction. After the estimated angle direction is correct, check whether the calculated speed is correct.

After the judgment of upwind and downwind, handle according to the current speed and direction. When the motor rotates forward and is greater than a certain value, speed tracking is adopted to directly switch the state to the operating state. If Omega is used to start the fan, different FOC can be set according to the forward rotation speed_ EKP、FOC_ EKI to improve the reliability of closing loop. When judging reverse rotation, adopt braking mode, and set different braking time according to different speeds. When the braking exceeds a certain number of times, a forced start can be set. In this case, the starting current should be increased appropriately. When the motor is stationary or the forward rotation speed is too small, it will jump to the next state directly.

Because the output duty cycle of FOC estimation mode is very small when judging the upwind and downwind, in low-voltage applications, when the upwind and downwind speed is too high, the back electromotive force is relatively large, and the voltage overshoot is easy to occur. Therefore, in the application of low pressure fan, in order to avoid the voltage overshoot caused by the high speed of forward and reverse wind, it is recommended to brake before the judgment of forward and reverse wind.

6.5.2 BEMF Mode

In BEMF mode, the controller turns off the output, compares the three-phase BEMF with the zero crossing point through the comparator, and generates the corresponding jump edge interrupt. In the comparator interrupt, the program processes the direction and speed according to the output level of the comparator and the timer count. The interrupt is handled as follows:

1. Detect the state of three BEMFs compared with zero crossing point;
2. According to the state change of the comparison between the BEMF and the zero crossing point, the forward and reverse information is obtained;
3. Speed detection is carried out according to the time interval between the two jump edges, that is, the count value of the timer is read;
4. Calculate the current speed according to the timer count value;
5. When it is detected that the motor is in forward rotation and the speed is relatively high, it will directly cut into the closed loop operation.

During the comparator interrupt, the current BEMF state is judged according to the high and low levels of the comparator output, and the forward and reverse directions are judged according to the state changes. At the same time, the speed is processed through the count value of the timer. When the forward rotation speed of the motor is relatively high, it starts at the rising edge of U-phase BEMF. When the motor reverse rotation is detected, the program will directly enter the braking mode even though the detection time has not ended. Also turn off the comparator interrupt. The program brakes at different speeds for different times. After the braking is completed, start the judgment again. When the upwind and downwind detection time is over, if the current forward speed is low or the motor is stationary, the program will jump to the next state. When adverse wind has ever occurred, the starting current can be set slightly higher than the current during static starting according to the speed of adverse wind to improve the reliability of starting.

Because the BEMF values of different motors are different, BEMF mode needs to divide the BEMF voltage when collecting the BEMF. The voltage division coefficient can be equal to that of the bus voltage. It requires three comparators and timers, so it is necessary to initialize the comparators and timers. Because the pin of the comparator and the pin of the op amp are shared, their applications are relatively limited. The 68 series is not suitable for dual resistance FOC.

6.5.3 RSD Mode

The RSD sampling double comparator method realizes the BEMF detection, which has the advantage of saving hardware resources. Because the comparison signal is the detection signal obtained by subtracting the two BEMFs, the harmonic interference on the BEMF has obvious effect all the time. The ground wire interference signal will not be introduced, and the robustness will be better than the traditional three comparator method.

Figure 6-1 below is the hardware schematic diagram of RSD detection of BEMF. Pay special attention to the fact that the three filter capacitors C35, C36 and C40 must be consistent in capacitance and not too large to avoid detection sector deviation caused by excessive signal delay and voltage and current overshoot during downwind switching.

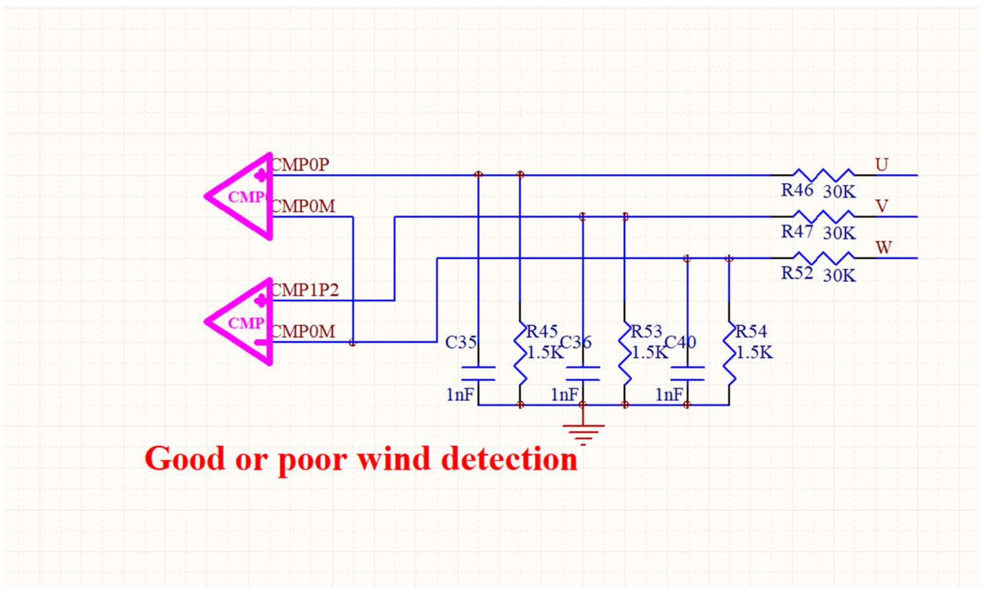


Figure 6-2 RSD BEMF Detection Circuit

Figure 6-3 is the waveform of U and V phase BEMF with W phase as reference. It can be seen that the steering direction and speed can be determined by judging the time sequence of zero crossing and the time of two zero crossing.

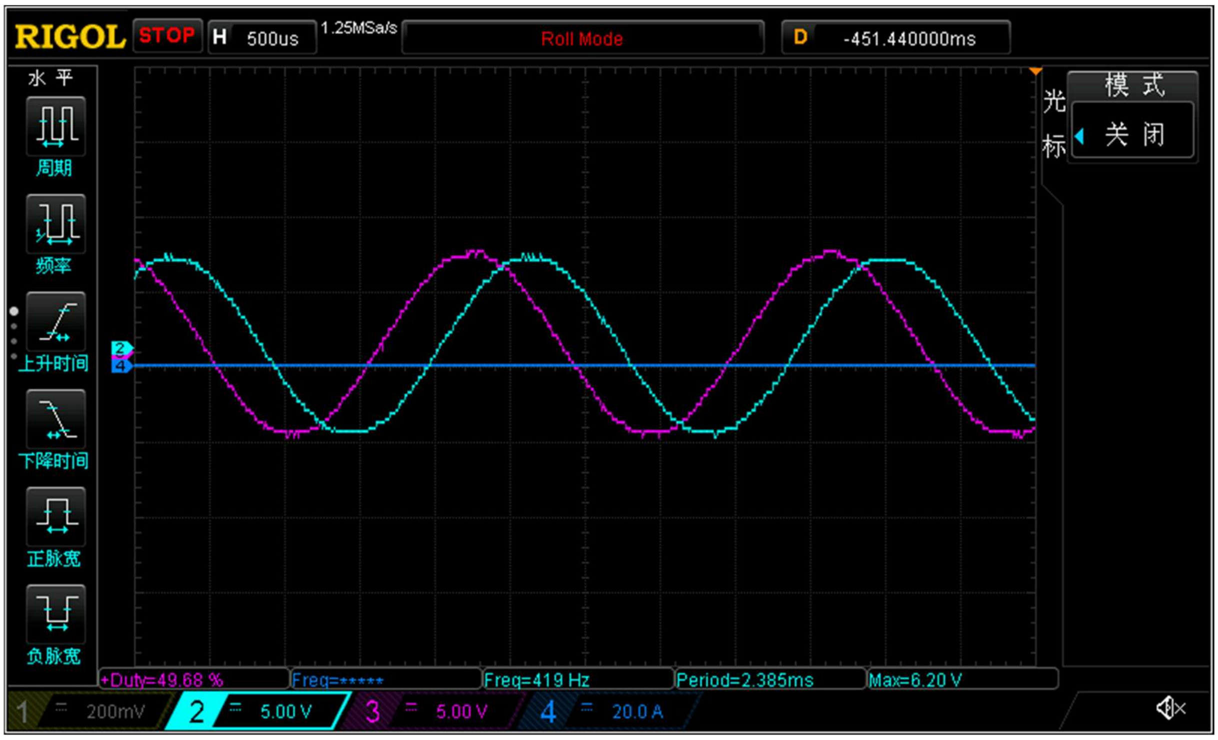


Figure 6-3 Voltage Waveform of U and V Phase BEMF with W Phase as Reference

In the following formula, assuming that eU, eV and eW are three-phase back electromotive force with phase difference of 120 °, the phase relationship of eα and eβ can be calculated. It can be seen that the waveform is consistent with the measured waveform shown in Figure 6-3. The phase difference between the two zero crossing points is 60 °

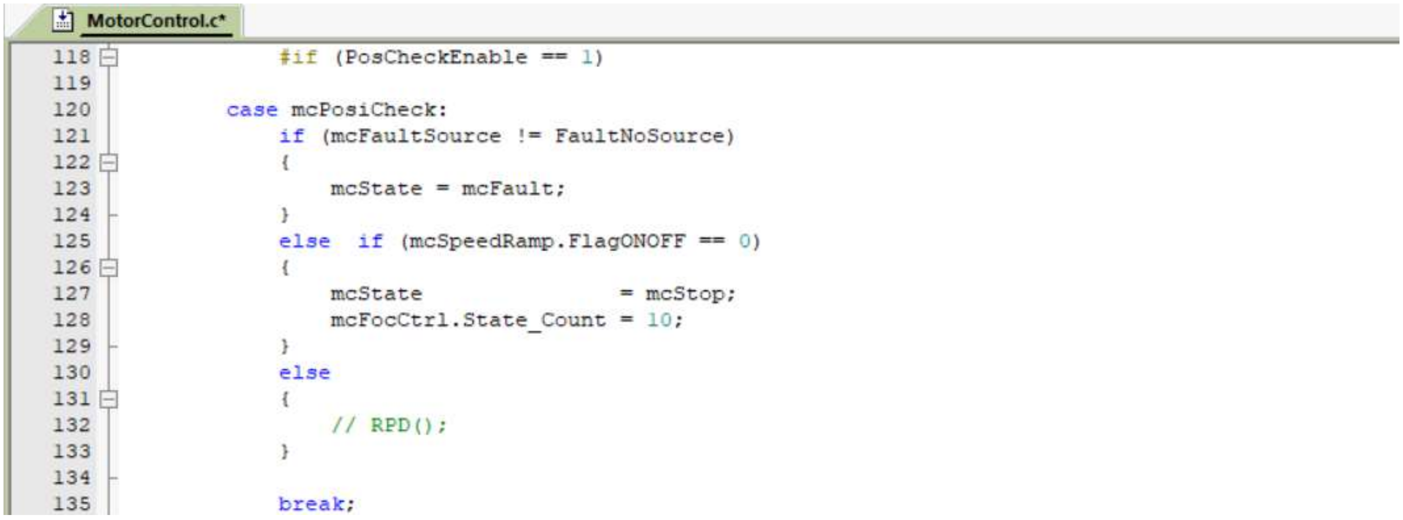
and 120 °. Therefore, the angle and speed values of switching to the closed loop can be obtained through the detection of sectors and the time of the two zero crossing points to achieve downwind startup.

$$e_u = U_m \cos\theta$$
$$e_v = U_m \cos(\theta + \frac{2}{3}pi)$$
$$e_w = U_m \cos(\theta + \frac{4}{3}pi)$$

$$e_\alpha = e_u - e_w = \sqrt{3}\ U_m \cos(\theta + \frac{1}{6}pi)$$
$$e_\beta = e_v - e_w = \sqrt{3}\ U_m \cos(\theta + \frac{1}{2}pi)$$

6.6 mcPosiCheck

Initial position check state, which mainly uses the salient pole effect of the motor to detect the rotor position. The difference in saturation degree of d and q axis magnetic circuits of the motor caused by high-frequency signal injection is used to realize the detection of rotor position to detect the initial position of the motor, confirm the initial starting angle, and improve the success rate of starting. Note that the initial position detection will introduce electromagnetic noise. The customer determines whether this function is required according to the actual application scenario and needs. Generally, the noise requirements are small Initial position detection is generally not used in applications where the reverse deflection angle needs to be small. If you do not need this function, you can disable it by setting PosCheckEnable to 0. Please refer to the following code implementation for details.


MotorControl.c*

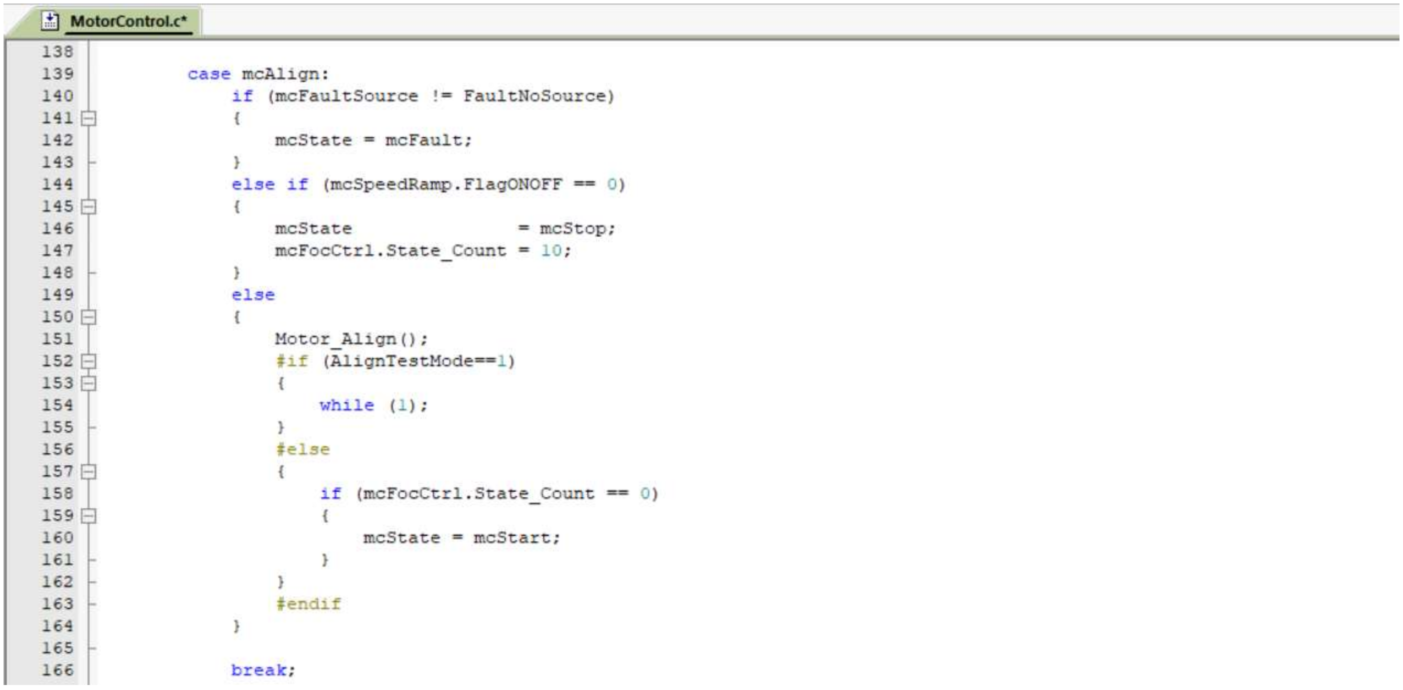
```
            #if (PosCheckEnable == 1)

        case mcPosiCheck:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else  if (mcSpeedRamp.FlagONOFF == 0)
            {
                mcState                 = mcStop;
                mcFocCtrl.State_Count = 10;
            }
            else
            {
                // RPD();
            }

            break;
```

6.7 mcAlign

Alignment state, under which the controller outputs a constant current ID or duty cycle UD to drag the motor forcibly to a fixed angular position. Its purpose is to improve the reliability of motor starting, but it will lead to longer starting time. For applications requiring starting time, pre positioning is generally not used. You can choose whether to enable this functionality through AlignEnable. The state machine will switch to mcStart state upon completion of alignment.

The advantage of ID pre positioning is that the positioning torque will not change with the change of voltage, which is equivalent to constant torque positioning. The wide voltage effect is good, but because ID is adjusted through the internal PI closed-loop, electromagnetic noise will appear.

The advantage of UD positioning is that there is no current regulation and positioning and there is almost no noise. The disadvantage is that the positioning torque of different voltages is different. Attention should be paid to the wide voltage application. UD can be set to the associated bus voltage in the program, and UD can be compensated by the current bus voltage to achieve the stability of the wide voltage positioning torque. For details, refer to the following code.

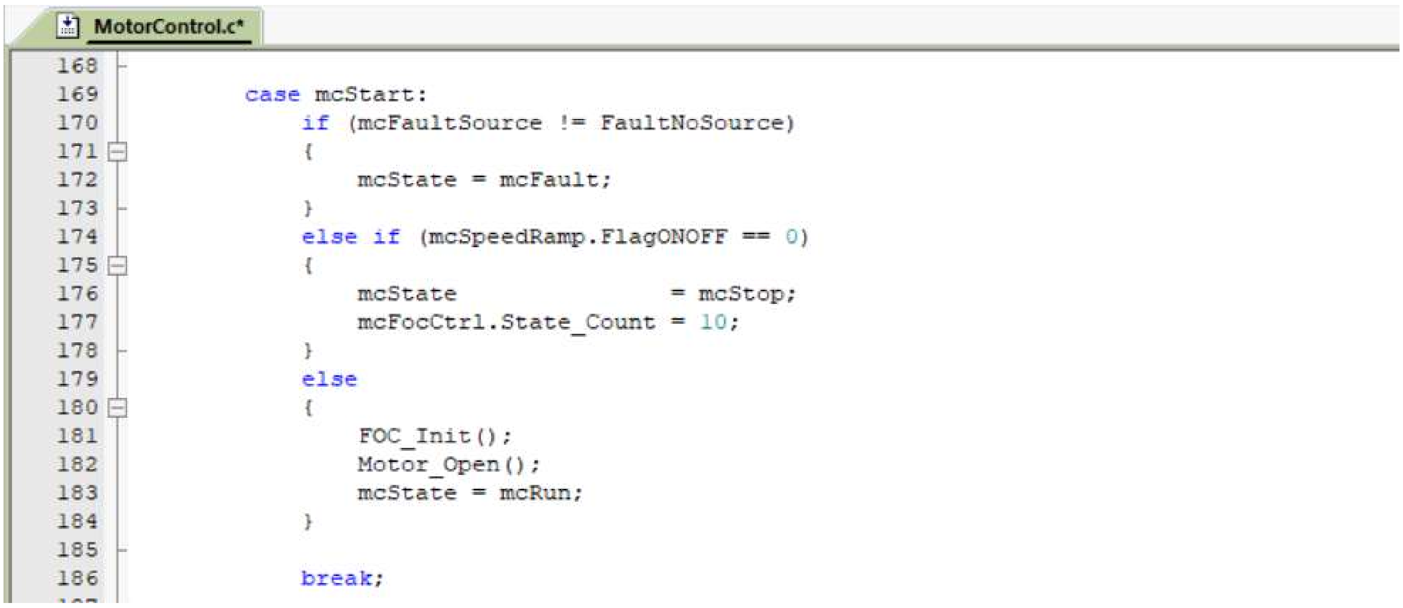


MotorControl.c*

```
        case mcAlign:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else if (mcSpeedRamp.FlagONOFF == 0)
            {
                mcState                = mcStop;
                mcFocCtrl.State_Count = 10;
            }
            else
            {
                Motor_Align();
                #if (AlignTestMode==1)
                {
                    while (1);
                }
                #else
                {
                    if (mcFocCtrl.State_Count == 0)
                    {
                        mcState = mcStart;
                    }
                }
                #endif
            }

            break;
```

6.8 mcStart

Start state, which is mainly used to configure the startup code of motor. After configuring the relevant register code and variables, the state machine will shift to the next state mcRun. The motor startup process is implemented by the ME kernel. Please refer to the following code implementation for details.

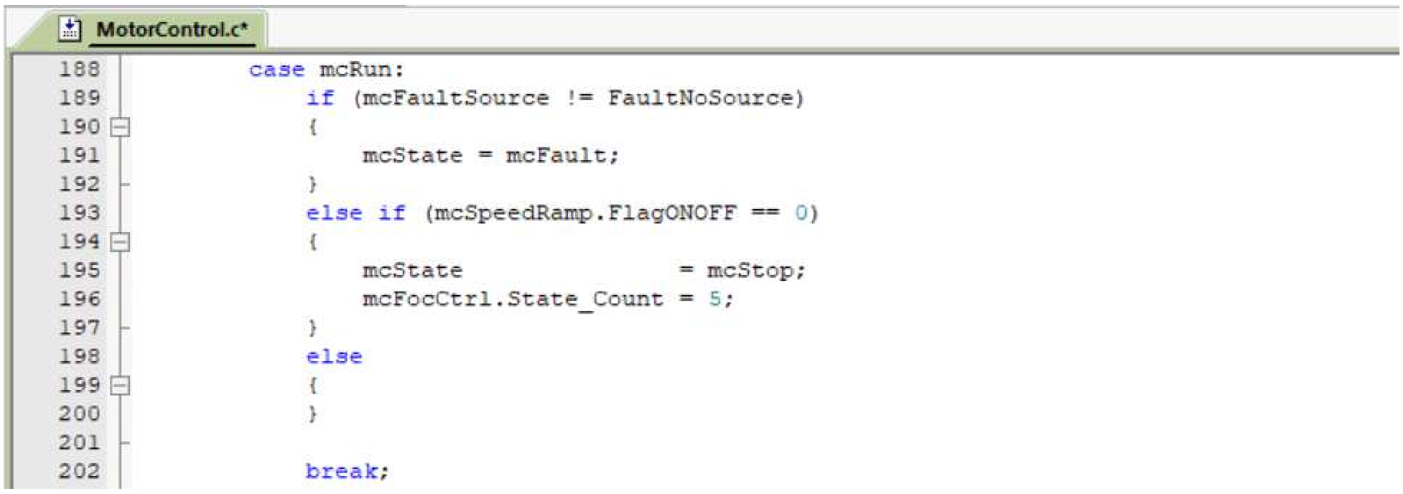


MotorControl.c*

```
        case mcStart:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else if (mcSpeedRamp.FlagONOFF == 0)
            {
                mcState                = mcStop;
                mcFocCtrl.State_Count = 10;
            }
            else
            {
                FOC_Init();
                Motor_Open();
                mcState = mcRun;
            }

            break;
```

6.9 mcRun

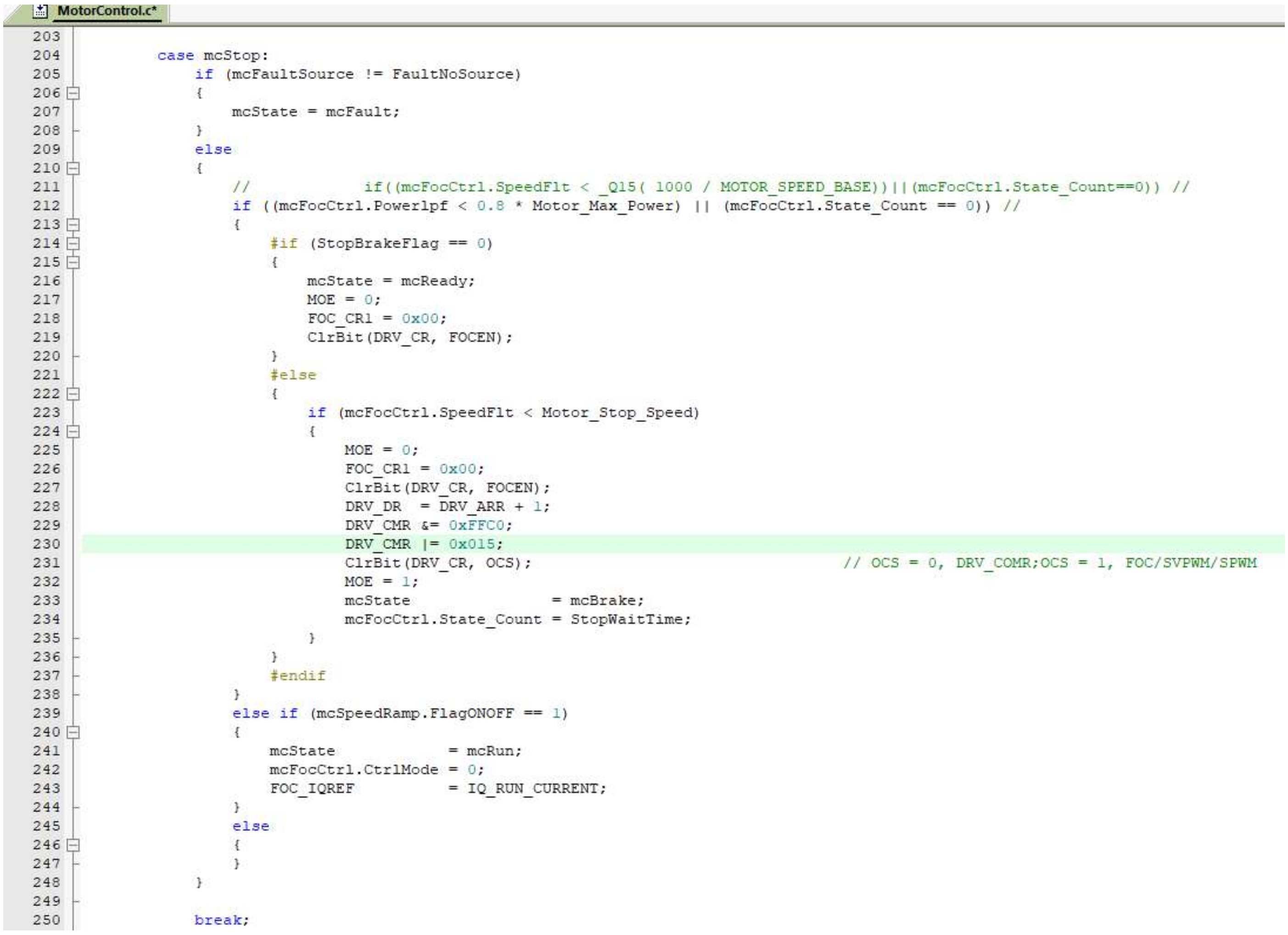
Running state, under which, the program will perform closed-loop PI control in the start and running phases using the Speed_response() function. Please refer to the following code implementation for details.


MotorControl.c*

```
        case mcRun:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else if (mcSpeedRamp.FlagONOFF == 0)
            {
                mcState                  = mcStop;
                mcFocCtrl.State_Count = 5;
            }
            else
            {
            }

            break;
```

6.10 mcStop

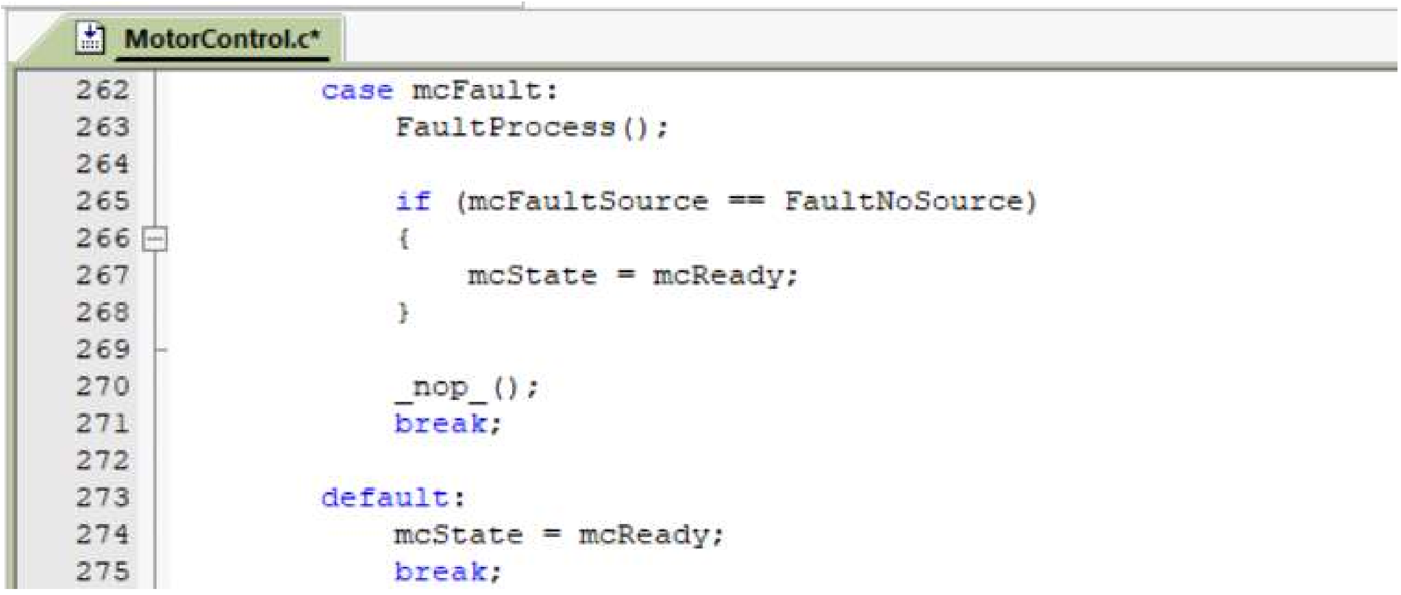
Stop state is used to stop the motor. Users can decide whether quick shutdown is required by enabling or disabling the Stop & Brake function. Generally, the low side is fully opened for braking. Pay attention to this braking mode. Due to the large reverse electromotive force of the motor at high speed, it will generate a large induced current, which is easy to burn power devices. The program will generally be set lower than the shutdown speed Motor_ Stop_ Speed will start braking. After braking, motor will shift to mcReady state, waiting for the next "Start" command.



```
        case mcStop:
            if (mcFaultSource != FaultNoSource)
            {
                mcState = mcFault;
            }
            else
            {
                //              if((mcFocCtrl.SpeedFlt < _Q15( 1000 / MOTOR_SPEED_BASE))||(mcFocCtrl.State_Count==0)) //
                if ((mcFocCtrl.Powerlpf < 0.8 * Motor_Max_Power) || (mcFocCtrl.State_Count == 0)) //
                {
                    #if (StopBrakeFlag == 0)
                    {
                        mcState = mcReady;
                        MOE = 0;
                        FOC_CR1 = 0x00;
                        ClrBit(DRV_CR, FOCEN);
                    }
                    #else
                    {
                        if (mcFocCtrl.SpeedFlt < Motor_Stop_Speed)
                        {
                            MOE = 0;
                            FOC_CR1 = 0x00;
                            ClrBit(DRV_CR, FOCEN);
                            DRV_DR  = DRV_ARR + 1;
                            DRV_CMR &= 0xFFC0;
                            DRV_CMR |= 0x015;
                            ClrBit(DRV_CR, OCS);                                        // OCS = 0, DRV_COMR;OCS = 1, FOC/SVPWM/SPWM
                            MOE = 1;
                            mcState                 = mcBrake;
                            mcFocCtrl.State_Count = StopWaitTime;
                        }
                    }
                    #endif
                }
                else if (mcSpeedRamp.FlagONOFF == 1)
                {
                    mcState             = mcRun;
                    mcFocCtrl.CtrlMode = 0;
                    FOC_IQREF           = IQ_RUN_CURRENT;
                }
                else
                {
                }
            }

            break;
```

6.11 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 mcReady state, waiting for the next "Start" command. Please refer to the following code implementation for details.


MotorControl.c*

```
        case mcFault:
            FaultProcess();

            if (mcFaultSource == FaultNoSource)
            {
                mcState = mcReady;
            }

            _nop_();
            break;

        default:
            mcState = mcReady;
            break;
```

7 Startup Modes

There are three common starting modes for the motor: ramp starting, Omega starting using position estimation algorithm, and ramp starting followed by Omega starting. Ramp starting is suitable for heavy load and slow start applications; Omega starting is suitable for applications with fast startup and smooth switching; the third startup mode is suitable for applications where initial position check function is enabled.

7.1 Ramp Starting

7.1.1 Theories

The principle is that given the starting current (IQREF is non-zero), the electrical frequency of the motor is slowly accumulated by ramp, the angle information of coordinate transformation is provided by the ramp angle, and the motor is driven to rotate. When the speed of the driving motor reaches a certain value, the starting is completed, and the angle information is gradually cut in to be provided by the position estimation algorithm.

In this mode, the ramp angle consists of angle THETA, speed RTHESTEP, acceleration RTHEACC and ramp counter RTHECNT. Below is the ramp formula:

RTHESSTEP (32bit) = RTHESSTEP (32bit) + RTHEACC

THETA (16bit) = THETA (16bit) + RTHESSTEP (high 16bit)

In every cycle, ramp module makes a ramp operation and ramp counter adds 1. When the value of the counter reaches RTHECNT, RFAE is cleared by hardware, and then the ramp is over. The estimator also estimates the angle in the ramp. At the end of the ramp, it switches to the estimator mode. Because there is a deviation between the estimated angle and the ramp angle, smooth switching is required. The correction value FOC can be switched by adjusting the angle_ The value of THECOR reduces the switching jitter and achieves smooth switching.

7.1.2 Parameter Specification

	Parameter Debugging		
	Parameter	**Range**	**Description**
Ramp Starting	Starting current	Lower than the maximum running current	1. The starting current is determined by the actual load. When debugging the fan, you can grasp the load with your hands under safe conditions and feel a stable torque output; 2. When the starting current increases, the starting torque will increase, and it is easy to start overshoot if it is too large; If the starting torque is too small, it is easy to fail or fail to start.
	Ramp increment MOTOR_OPEN_ACC	1-200	1. The ramp increment is a function of base speed, which affects the speed of starting frequency increase. The higher the motor base speed, the greater the ramp increment ; 2. The motor startup speed increases as the increment grows. If the ramp increment is too large, the actual frequency cannot align with the given frequency, causing motor goes out-of-step; if the increment is too small, the startup is too slow, and it is easy to pause.
	Initial speed of ramp starting MOTOR_OPEN_ACC_MIN		1. The initial speed affects the starting speed, and the default initial speed is 0; 2. In the case that the startup response is required to be very fast, the initial speed can be set to non-zero speed.
	Number of ramp starts MOTOR_OPEN_ACC_CNT	1-255	1. Start climbing times, the maximum value of the register is 255; 2. The starting times increase and the starting time increases. In some applications, the start time is long, and the counter can be set to be re assigned after counting. The number of assignments depends on the actual application.
	Observer bandwidth filter output ATO_ BW		1. The observer bandwidth affects the estimated speed response of the estimator; 2. During the ramp up start, the estimator is

			independent of the ramp up start, and it is only necessary to set a bandwidth filter value to enable the motor to operate normally.
	Minimum speed of SMO		1. SMO's minimum speed influence estimator's estimation; 2. The value setting is related to the speed reference, which should be lower than the minimum speed of the motor for normal operation, but not too small.

7.2 Omega Startup

7.2.1 Theories

The principle is that given the starting current (IQREF is non-zero), the position estimation algorithm estimates the current speed. When the estimated speed is less than the minimum switching speed set by the user, the estimator outputs a forced angle to drag the motor. The forced speed starts from 0, and each calculation cycle is added to the speed increment EFREQACC. At the same time, the maximum limiting is performed according to the starting limiting speed. When the estimated motor speed is greater than or equal to the minimum switching speed of starting, the starting is completed, and the angle information is calculated by the estimation algorithm.

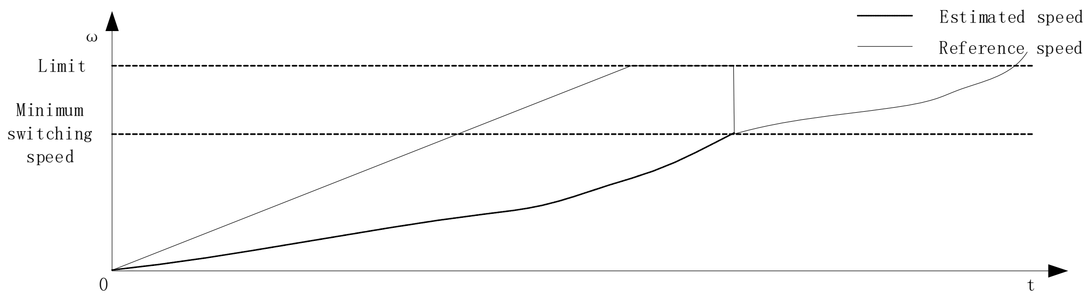


Figure 7-1 Speed Curve in Omega Startup Mode

7.2.2 Parameter Specification

	Parameter Debugging		
	Parameter	**Range**	**Description**
Omega Startup	Starting current	Lower than the maximum running current	1. The starting current is determined by the actual load. When debugging the fan, you can grasp the load with your hands under safe conditions and feel a stable torque output;

			2. If the starting current increases, the starting torque will increase, and if it is too large, it is easy to start with overshoot or noise; If the starting torque is too small, it is easy to fail or fail to start.
	Startup Increment Motor_Omega_Ramp_ACC.	1-200	1. The startup increment is a function of base speed, which affects the speed of starting frequency increase. The higher the motor base speed, the greater the ramp increment ; 2. When the increment increases, the startup output frequency increases. If the increment is too large, the actual frequency of the motor cannot keep up with the given frequency, and it is easy to lose step; The increment is too small, the startup is too slow, and it is easy to pause.
	Minimum switching speed for startup (rpm/min) MOTOR_ OMEGA_ ACC_ MIN		1. The minimum starting switching speed is related to the speed reference. The higher the motor speed reference is, the greater its value will be; 2. Its value is less than the starting limit speed; 3. If it is too small, the estimator is easy to make an error in estimation, and the motor has not been dragged to start and switch over, which is easy to lead to start failure; If it is too large, it is easy to cause the startup to fail to cut into the closed loop or the speed to overshoot.
	Starting limit speed (rpm/min) MOTOR_ OMEGA_ ACC_ END		1. The starting limit speed is related to the speed reference. The higher the motor speed reference is, the greater its value will be; 2. The value shall be less than the minimum running speed of the motor and greater than the minimum switching speed; 3. If it is too small, the estimator is easy to make an error in estimation, and the motor has not been dragged to start and switch over, which is easy to lead to start failure; If it is too large, it is easy to cause the startup to fail to cut into the closed loop or the

			speed to overshoot.
	Start dead time voltage compensation coefficient (RV3_Start = (0.5-2.0)*RV3_Actual)	0.5-2.0	1. The compensation coefficient of starting dead time voltage affects the output during starting, which is related to the bus voltage division multiple and starting current; 2. It is recommended to debug from 1.0 to both sides, mainly ranging from 0.8-1.75. When the starting current changes, its value may change.
	Runtime observer ATO_BW		1. The bandwidth of the observer affects the PI loop in the estimator to estimate the speed; 2. To prevent the estimated speed from fluctuating greatly at low speeds, the bandwidth filtering value is generally increased from small to large. The acceleration time is related to the actual startup time. The shorter the startup time is required, the shorter the acceleration interval is. During commissioning, it shall be modified according to starting current waveform and actual rotation effect; 3. If AT0_ BW is too large, it is easy to cause reverse deflection startup can not be pulled back to forward rotation, resulting in out of step; If it is too small, it is easy to cause a pause in the positive start.
	Minimum speed of SMO		1. SMO's minimum speed influence estimator's estimation; 2. The value setting is related to the speed reference and should be lower than the minimum speed of the motor for normal operation, but it should not be too small, which is greater than the starting limit speed.
	Current loop Kp, Ki		1. The specific parameter range can refer to the PI coefficient description of the operating current loop; 2. During debugging, it can be increased first, and then decreased after observing the bad current waveform.

7.3 Ramp Startup Followed by OMEGA Startup

7.3.1 Theories

Ramp startup followed by OMEGA startup is to combine the advantages of the two. First drag the motor. After the motor has a certain speed, start it with Omega to ensure reliability. It is mainly used for starting with initial position detection. For parameter settings, refer to the specific debugging parameters of the previous two types of startup.

Among the three modes, the longest startup time is open-loop startup and the shortest is Omega startup. After the motor startup parameters are configured, the motor starts to enter the running mcRun state. When the motor is running, speed closed-loop control or other closed-loop control can be added according to performance requirements.

7.3.2 Parameter Specification

8 Debugging Procedures

8.1 Motor Parameters Measurement

8.1.1 Measurement Steps

Pole pairs, resistance, inductance and BEMF vary from one motor to another. Since these parameters have a major bearing on FOC calculation, the parameters used to debug motor must be written to the program during debugging. Motor parameters are measured as follows:

1. Pole pairs (P): A parameter to be assigned in motor design.
2. Phase resistance (Rs): Measure the motor's two-phase line resistance RL with a multimeter, and then calculate phase resistance using the equation Rs = RL/2.
3. Phase inductance (Ls): Measure the two-phase line inductance LL at 1KHz frequency with an LCR meter, and then figure out phase inductance using the equation Ls = LL/2.
4. BEMF constant (Ke): Attach the oscilloscope probe to one phase of the motor and connect one of the motor's the other two phases to ground, rotate the load, and measure the BEMF waveform. Take a sine wave in the middle and measure its peak-to-peak value KeVpp and frequency Kef using the following formula:

$$\mathrm{Ke} = 1000 * \mathrm{P} * \frac{\mathrm{Vpp}}{2 * 1.732 * 60 * \mathrm{f}}$$

P denotes pole pairs.

For example, the BEMF waveform is measured as follows:

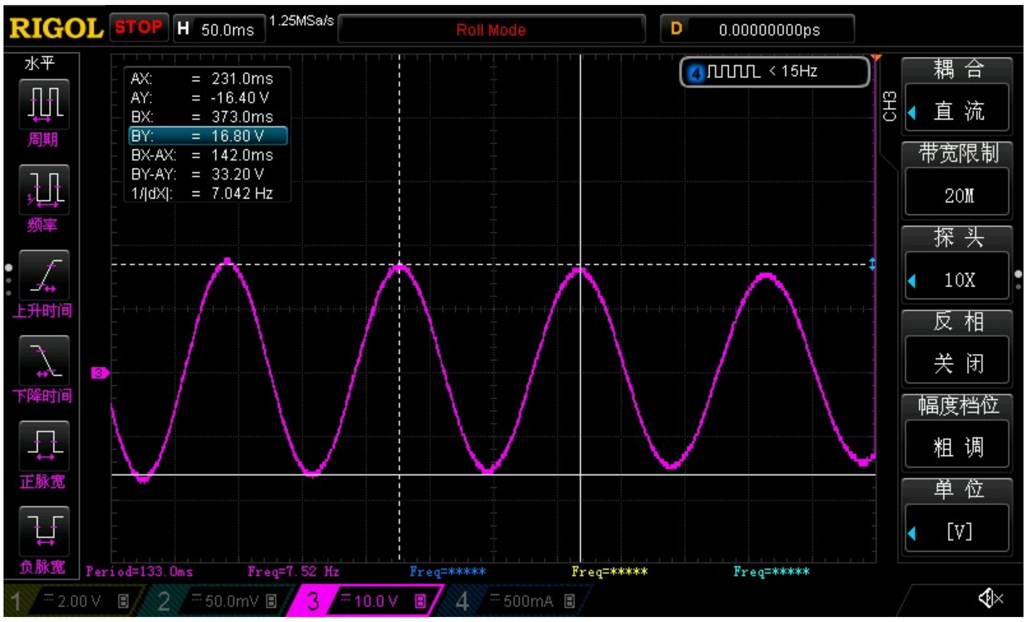


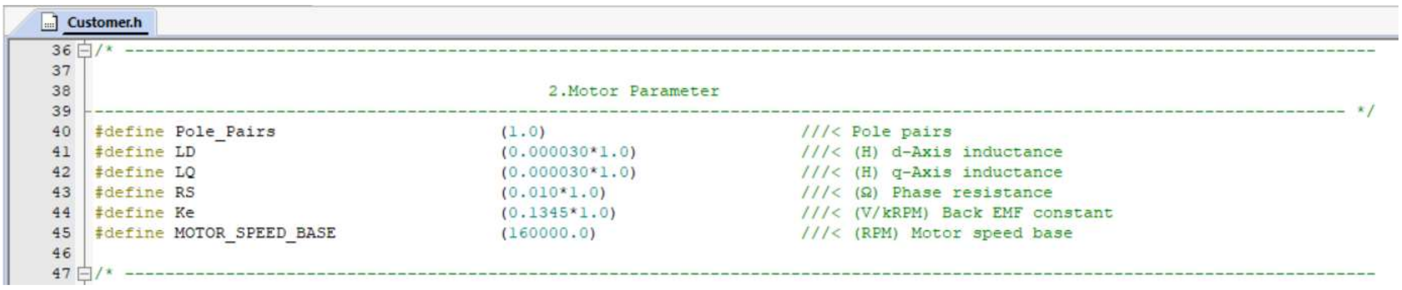
Figure 8-1 BEMF Waveform

As shown in the above figure, the measured KeVpp is 33.2V, the Kef is 7.042Hz, and the pole pairs is 4, then:

$$\text{BEMF Ke} = 1000 * 4 * \frac{33.2}{2*1.732*60*7.042} = 90.73\ \text{V/KRPM}$$

8.1.2 Code Implementation

Write the motor parameters to the following software code. In consideration of measurement error, we multiple inductance, resistance and Ke by the factor 1.0. When debugging, users can fine tune that factor to adjust and optimize the motor model. Pay attention to correspondence of units.


Customer.h

```
/* ----------------------------------------------------------------------------------------------------------

                                         2.Motor Parameter
------------------------------------------------------------------------------------------------------ */
#define Pole_Pairs                    (1.0)                         ///< Pole pairs
#define LD                            (0.000030*1.0)                ///< (H) d-Axis inductance
#define LQ                            (0.000030*1.0)                ///< (H) q-Axis inductance
#define RS                            (0.010*1.0)                   ///< (Ω) Phase resistance
#define Ke                            (0.1345*1.0)                  ///< (V/kRPM) Back EMF constant
#define MOTOR_SPEED_BASE              (160000.0)                    ///< (RPM) Motor speed base

/* ----------------------------------------------------------------------------------------------------------
```

8.2 Base Current

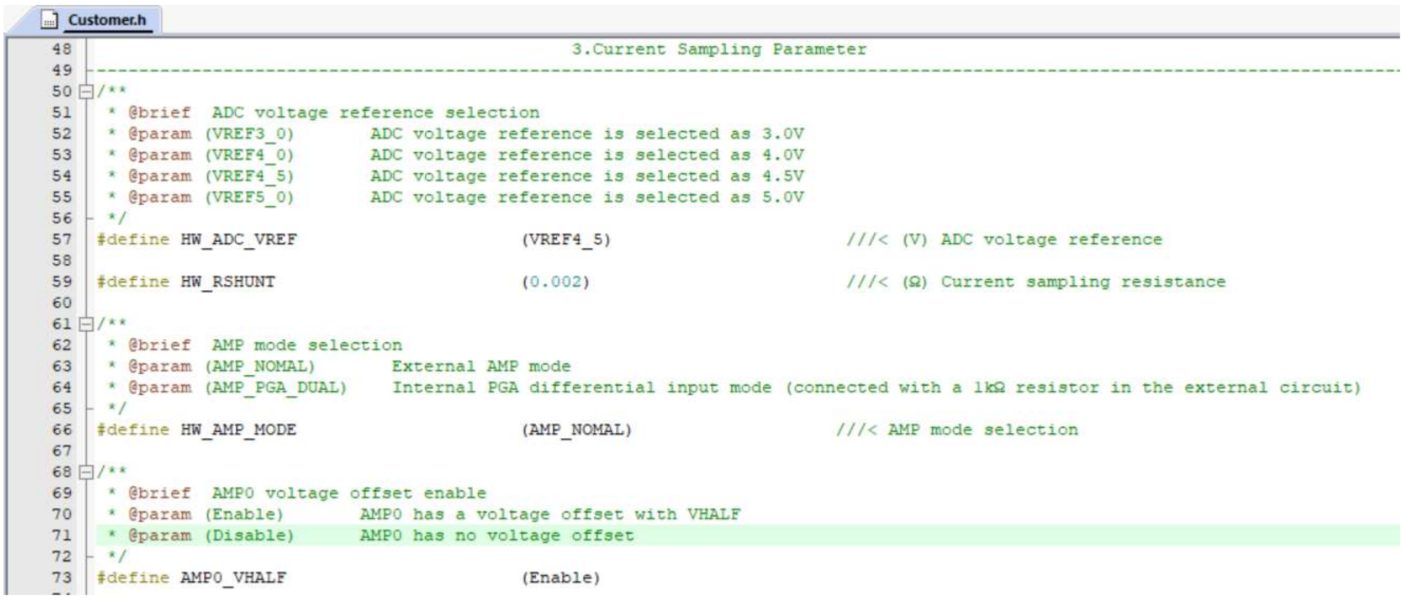
The base current, maximum sampling current and minimum sampling current can be calculated on the basis of the shunt resistance Rsample, the op-amp gain Amp, and the ADC reference voltage Vs of MCU. The motor current should be tuned so that it falls within the range between the minimum sampling current and the maximum sampling current. If not, Rsample and Amp must be reconfigured as needed. The power of shunt resistor should be selected by following the principle $\mathrm{P} = \mathrm{I}^2\mathrm{Rsample} < Pmax$.

Base current $\text{Ibase} = \frac{\text{Vs}}{\text{Rsamle}*\text{Amp}}$, maximum sampling current $\text{Ismax} = \frac{\text{Ibase}}{2}$, minimum sampling current $\text{Ismin} = -\frac{\text{Ibase}}{2}$.

Example: Suppose Rsample = 0.5Ω, Amp = 4, Vs = 5.0V, then $\text{Ibase} = 2.5\text{A}, \text{Ismax} = 1.25\text{A}, \text{Ismin} = -1.25\text{A}.$

8.2.1 Code Implementation

1. Detect whether VHALF bias voltage is applied to the current sampling circuitry. It is recommended to attach bias voltage to the hardware. When bias voltage is applied, you need to set AMP0_ VHALF as 1;
2. Write the above shunt resistance, gain and ADC reference voltage to the following software code.


Customer.h

```
                                               3.Current Sampling Parameter
-----------------------------------------------------------------------------------------------------------------------------
/**
  * @brief  ADC voltage reference selection
  * @param (VREF3_0)       ADC voltage reference is selected as 3.0V
  * @param (VREF4_0)       ADC voltage reference is selected as 4.0V
  * @param (VREF4_5)       ADC voltage reference is selected as 4.5V
  * @param (VREF5_0)       ADC voltage reference is selected as 5.0V
  */
#define HW_ADC_VREF                         (VREF4_5)                        ///< (V) ADC voltage reference

#define HW_RSHUNT                           (0.002)                          ///< (Ω) Current sampling resistance

/**
  * @brief  AMP mode selection
  * @param (AMP_NOMAL)       External AMP mode
  * @param (AMP_PGA_DUAL)    Internal PGA differential input mode (connected with a 1kΩ resistor in the external circuit)
  */
#define HW_AMP_MODE                         (AMP_NOMAL)                     ///< AMP mode selection

/**
  * @brief  AMP0 voltage offset enable
  * @param (Enable)        AMP0 has a voltage offset with VHALF
  * @param (Disable)       AMP0 has no voltage offset
  */
#define AMP0_VHALF                          (Enable)
```

8.3 DC Bus Voltage Divider Ratio

In the FOC - SVPWM module, the DC bus voltage must be sampled for calculation. In the HV/LV applications, there is a difference between the supply voltage and the ADC maximum sampling voltage of MCU, so the DC bus voltage should be processed through voltage scaling as needed. Configure the resistance values of resistors RV1, RV2, and RV3 in the circuit (RV2 is omitted in LV applications, that is, RV2=0), then the corresponding scaling and maximum shunt resistor value are calculated as follows:

$$\text{RV} = \frac{\text{RV1} + \text{RV2} + \text{RV3}}{\text{RV3}}$$

The maximum sampling voltage Vsmax = RV*Vs, and the voltage spike of practical applications should be lower than 0.8*Vsmax.

Example: RV1 = 330KΩ, RV2 = 300KΩ, RV3 = 6.8KΩ, Vs = 5.0V, the maximum sampling voltage

Vsmax=$\frac{330+300+\ .8}{6.8} * 5.0 = 468\mathrm{V}$

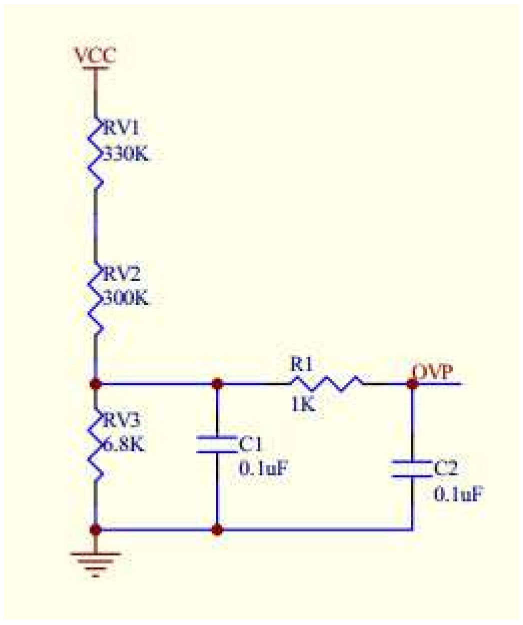
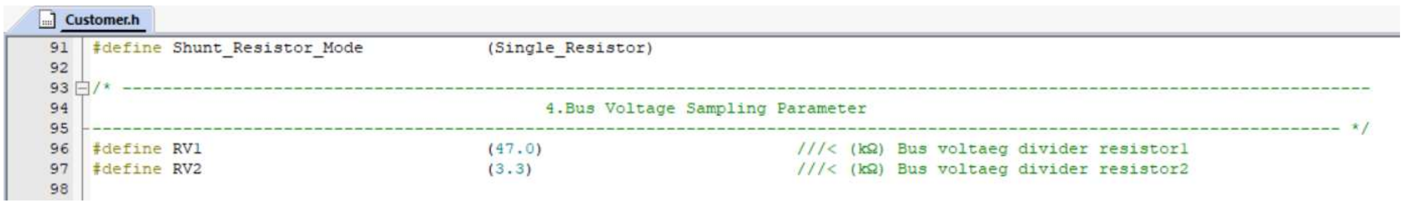


Figure 8-2 DC Bus Voltage Divider Circuit

8.3.1 Code Implementation

Write the above bus voltage divider parameters to the following software code. If only two shunt resistors are, fill in 0 for one of RV1 and RV2, VC1 is the voltage compensation coefficient.



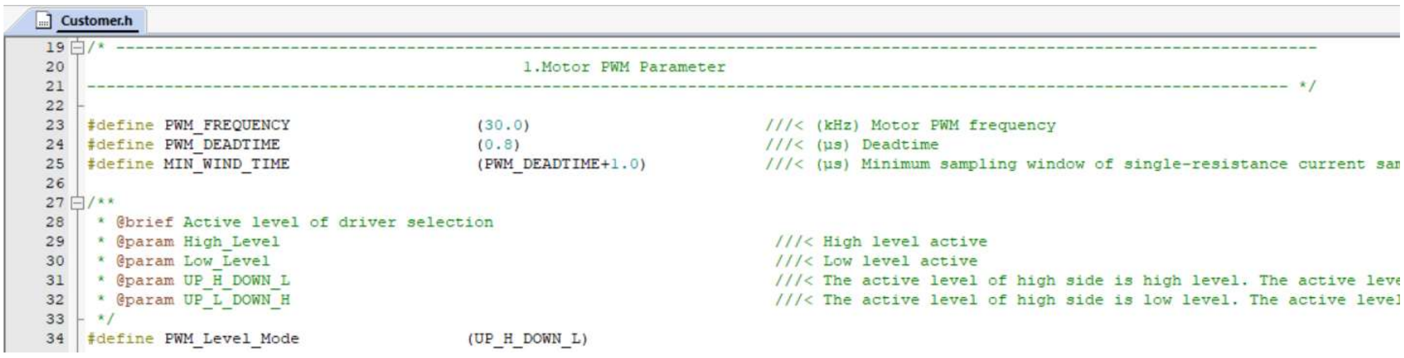
```
#define Shunt_Resistor_Mode           (Single_Resistor)

/* -----------------------------------------------------------------------------------------------
                                        4.Bus Voltage Sampling Parameter
------------------------------------------------------------------------------------------------- */
#define RV1                           (47.0)                       ///< (kΩ) Bus voltaeg divider resistor1
#define RV2                           (3.3)                        ///< (kΩ) Bus voltaeg divider resistor2

```

8.4 Carrier Frequency and Deadtime

It is recommended that the carrier frequency should not be less than 10 times of the maximum electrical frequency. The dead time should be determined by actual test of the VGS waveform of the upper and lower Bridges to ensure that there is no risk of micro-conduction.

When sampling with single resistance, pay attention to avoid internal parameter overflows. When sampling with single resistance, set the minimum sampling window value of FOC_TSMIN, FOC_TSMIN < carrier period /16(unit: us), for example: Carrier frequency is set to 30KHZ, MIN_WIND_TIME < 30/16 = 1.875μs

8.4.1 Code Implementation

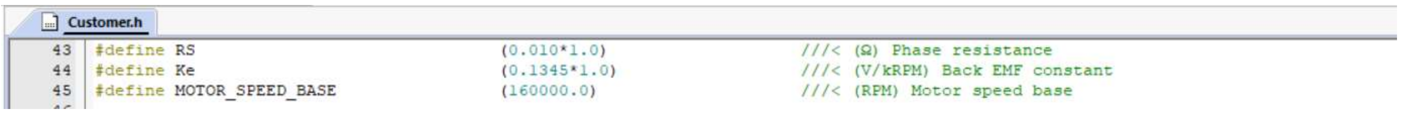


```
/* ---------------------------------------------------------------------------------------------------------------
                                          1.Motor PWM Parameter
---------------------------------------------------------------------------------------------------------------- */

#define PWM_FREQUENCY                 (30.0)                    ///< (kHz) Motor PWM frequency
#define PWM_DEADTIME                  (0.8)                     ///< (μs) Deadtime
#define MIN_WIND_TIME                 (PWM_DEADTIME+1.0)        ///< (μs) Minimum sampling window of single-resistance current sam

/**
 * @brief Active level of driver selection
 * @param High_Level                                             ///< High level active
 * @param Low_Level                                              ///< Low level active
 * @param UP_H_DOWN_L                                            ///< The active level of high side is high level. The active leve
 * @param UP_L_DOWN_H                                            ///< The active level of high side is low level. The active level
 */
#define PWM_Level_Mode               (UP_H_DOWN_L)
```

8.5 Base Speed

The MOTOR_SPEED_BASE is typically set to 1.5 ~ 2 times of the maximum speed.

8.5.1 Code Implementation



```
#define RS                        (0.010*1.0)           ///< (Ω) Phase resistance
#define Ke                        (0.1345*1.0)          ///< (V/kRPM) Back EMF constant
#define MOTOR_SPEED_BASE          (160000.0)            ///< (RPM) Motor speed base
```

8.6 Hardware Driver Test

1. Disable Startup Protection, Motor Lock Protection and Phase Loss Protection to avoid false triggering of protection functionality in test mode;
2. Set the hardware OCP threshold to prevent hardware short circuit that may burn out the power device;
3. Enable AlignTestMode = 1;
4. Do not connect the motor to U, V, W terminals;
   Test with oscilloscope to check whether the driver board U, V and W have tachometer output. If yes, indicate that the driving output of the board is normal and the normal waveform is referred to
5. Figure 8-3.

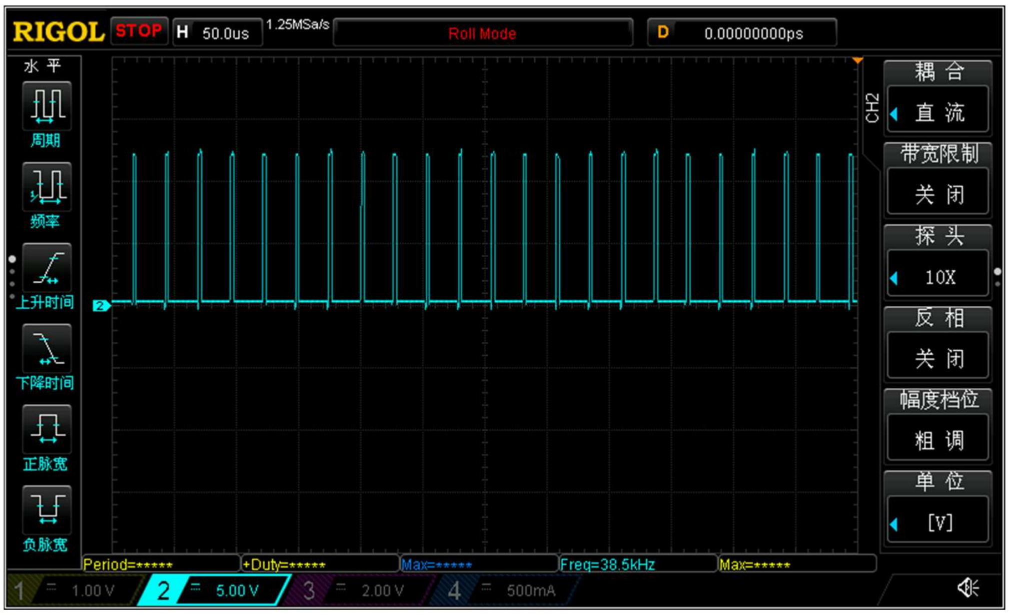
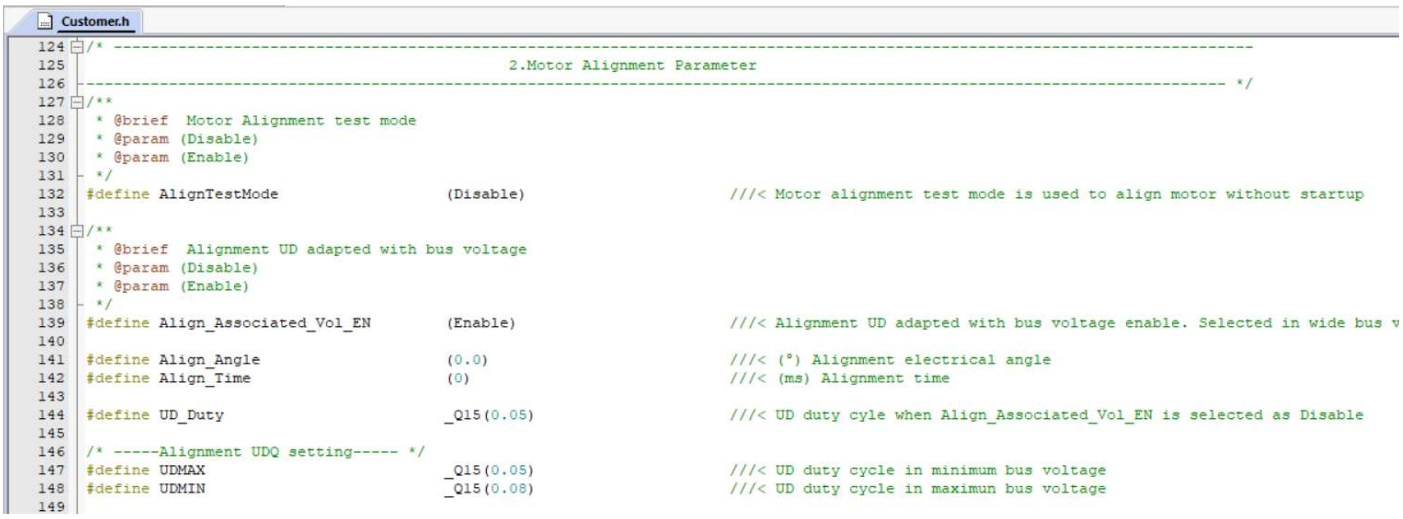


Figure 8-3 U-phase Output Waveform in Test Mode

8.6.1 Code Implementation



Customer.h

```
/* ----------------------------------------------------------------------------------------------------------------
                                          2.Motor Alignment Parameter
------------------------------------------------------------------------------------------------------------------ */
/**
 * @brief  Motor Alignment test mode
 * @param (Disable)
 * @param (Enable)
 */
#define AlignTestMode                   (Disable)                     ///< Motor alignment test mode is used to align motor without startup

/**
 * @brief  Alignment UD adapted with bus voltage
 * @param (Disable)
 * @param (Enable)
 */
#define Align_Associated_Vol_EN         (Enable)                      ///< Alignment UD adapted with bus voltage enable. Selected in wide bus v

#define Align_Angle                     (0.0)                         ///< (°) Alignment electrical angle
#define Align_Time                      (0)                           ///< (ms) Alignment time

#define UD_Duty                        _Q15(0.05)                     ///< UD duty cyle when Align_Associated_Vol_EN is selected as Disable

/* -----Alignment UDQ setting----- */
#define UDMAX                          _Q15(0.05)                     ///< UD duty cycle in minimum bus voltage
#define UDMIN                          _Q15(0.08)                     ///< UD duty cycle in maximun bus voltage
```

9 Protection Functionality

This Section mainly introduces the protection basics and parameter setting commonly used in the program. The basic protection functionalities provided in the program include hardware OVP, software OVP, MLP, startup protection, phase loss protection, OVLO/UVLO and bias voltage protection. Users may customize new protection functionalities as needed.

Note that protection values vary with project, motor and PCB, and the protection value of each protection function fit with the actual project. If you find that the protection functionality, especially motor lock protection or phase protection, cannot be triggered, or the protection is mistakenly triggered when motor operates normally, it suggests that the protection setpoint is incorrect and must be adjusted.

9.1 Hardware OCP

9.1.1 Theories

Hardware OCP is generally triggered by CMP3 interrupt and INT0 external interrupt, and the time from threshold triggered to protection finished is less than 2μs, allowing for hardware protection with a quick response.

The following is the port diagram of CMP3. The hardware OCP threshold can be set either through P23 input or through the output of the internal 9-bit DAC configuration. The OverHardcurrentValue cannot be greater than the maximum sampled current and the Peak current indicated in the power device’s user guide, generally set as 1.5 times or more of the peak phase current.

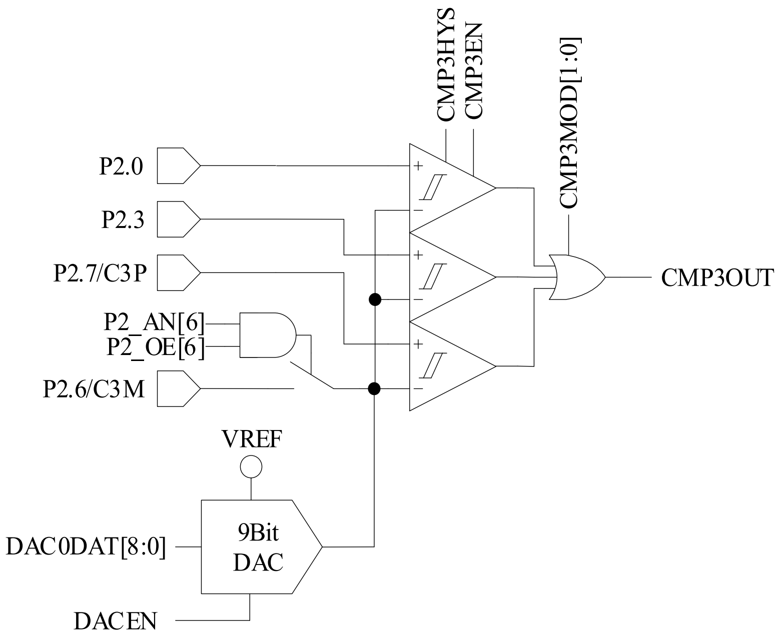
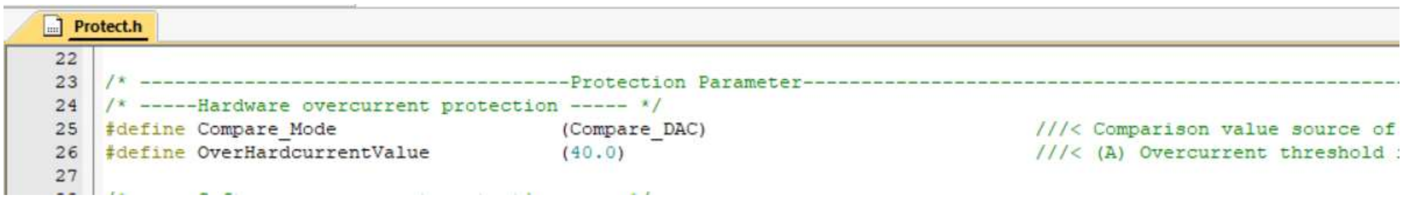


Figure 9-1 CMP3 Port Diagram

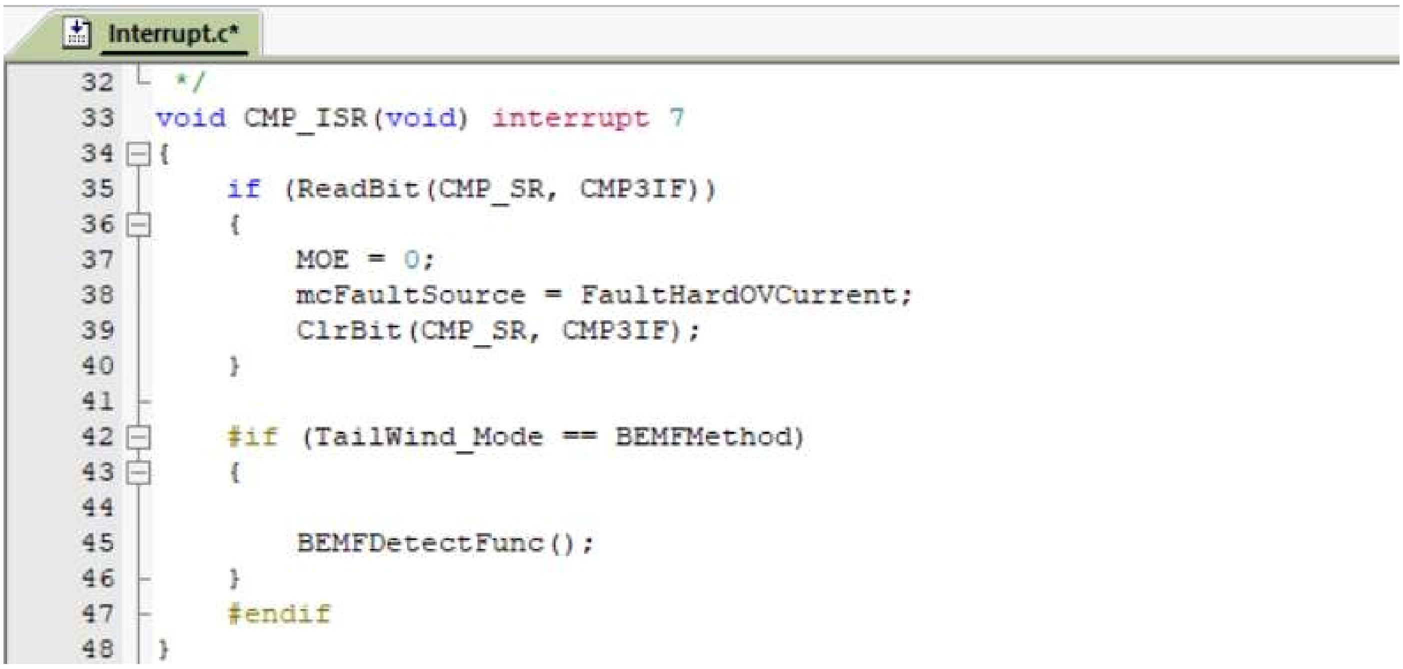
9.1.2 Code Implementation

- Configure the hardware OCP threshold source as internal DAC
- Configure the hardware OCP threshold



```
/* ---------------------------------------Protection Parameter----------------------------------------------------------
/* -----Hardware overcurrent protection ----- */
#define Compare_Mode                    (Compare_DAC)                           ///< Comparison value source of
#define OverHardcurrentValue            (40.0)                                  ///< (A) Overcurrent threshold :
```

When hardware OCP is detected, that is, the CMP_ISR(void) interrupt 7 InterruptFunction is invoked, where the driver output is disabled, i.e. MOE = 0, and the state machine shifts to the Fault state, i.e. mcState = mcFault. The system returns error source mcFaultSource = FaultHardOVCurrent.



Interrupt.c*

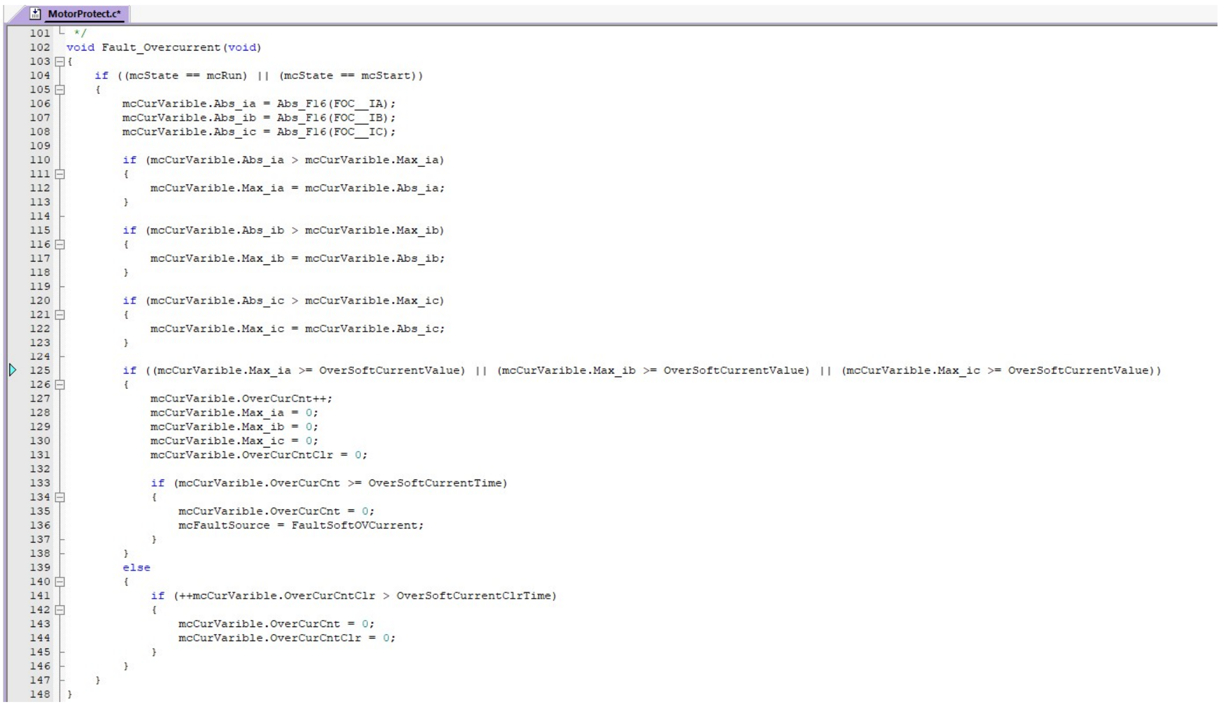
```c
 */
void CMP_ISR(void) interrupt 7
{
    if (ReadBit(CMP_SR, CMP3IF))
    {
        MOE = 0;
        mcFaultSource = FaultHardOVCurrent;
        ClrBit(CMP_SR, CMP3IF);
    }

    #if (TailWind_Mode == BEMFMethod)
    {

        BEMFDetectFunc();
    }
    #endif
}
```

9.2 Software OCP

9.2.1 Theories

Software OCP is mainly triggered after reading three-phase currents FOC__IA, FOC__IB and FOC__IC. If the phase currents read within OverSoftCurrentClrTime exceed the software OCP for OverSoftCurrentTime, the program will return an overcurrent error and the motor will shift to the mcFault state for lockout. Note that the software OCP response finishes within 1ms interrupt, and slow response time cannot replace the hardware OVP.

9.2.2 Code Implementation



MotorProtect.c*

```c
 */
void Fault_Overcurrent(void)
{
    if ((mcState == mcRun) || (mcState == mcStart))
    {
        mcCurVarible.Abs_ia = Abs_F16(FOC__IA);
        mcCurVarible.Abs_ib = Abs_F16(FOC__IB);
        mcCurVarible.Abs_ic = Abs_F16(FOC__IC);

        if (mcCurVarible.Abs_ia > mcCurVarible.Max_ia)
        {
            mcCurVarible.Max_ia = mcCurVarible.Abs_ia;
        }

        if (mcCurVarible.Abs_ib > mcCurVarible.Max_ib)
        {
            mcCurVarible.Max_ib = mcCurVarible.Abs_ib;
        }

        if (mcCurVarible.Abs_ic > mcCurVarible.Max_ic)
        {
            mcCurVarible.Max_ic = mcCurVarible.Abs_ic;
        }

        if ((mcCurVarible.Max_ia >= OverSoftCurrentValue) || (mcCurVarible.Max_ib >= OverSoftCurrentValue) || (mcCurVarible.Max_ic >= OverSoftCurrentValue))
        {
            mcCurVarible.OverCurCnt++;
            mcCurVarible.Max_ia = 0;
            mcCurVarible.Max_ib = 0;
            mcCurVarible.Max_ic = 0;
            mcCurVarible.OverCurCntClr = 0;

            if (mcCurVarible.OverCurCnt >= OverSoftCurrentTime)
            {
                mcCurVarible.OverCurCnt = 0;
                mcFaultSource = FaultSoftOVCurrent;
            }
        }
        else
        {
            if (++mcCurVarible.OverCurCntClr > OverSoftCurrentClrTime)
            {
                mcCurVarible.OverCurCnt = 0;
                mcCurVarible.OverCurCntClr = 0;
            }
        }
    }
}
```

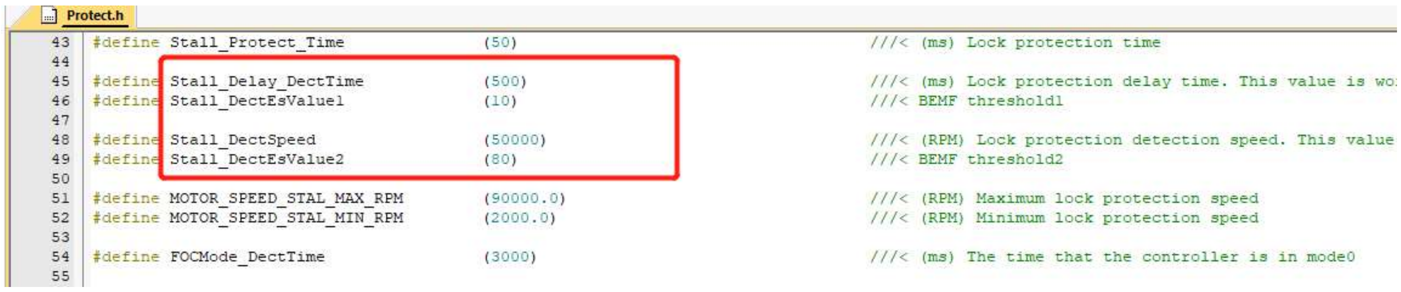
9.3 Motor Lock Protection (MLP)

9.3.1 Theories

MLP is mainly triggered to shut down the motor for protection or restart the motor when motor lock condition is detected, which is implemented in three ways:

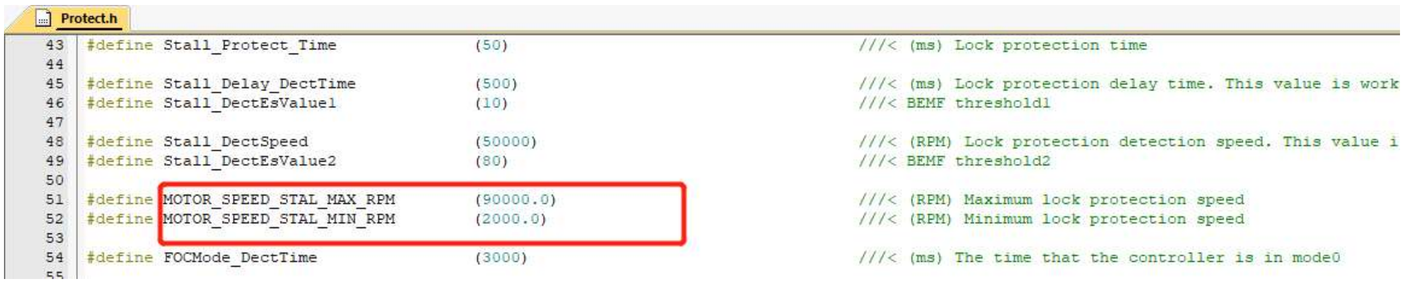
Method 1: By detecting the square of BEMF FOC_ESQU calculated by the estimator; under normal conditions, the higher the motor speed, the larger the FOC_ESQU will be. In the case of motor stall, the estimated speed will be very high when motor goes out-of-step but FOC_ESQU will be low.

Program implementation: Motor stall is established when FOC_ESQU value is smaller than the preset Stall_DectEsValue1 after Stall_Delay_DectTime; or when the estimated speed is higher than the preset Stall_DectSpeed value, but the FOC_ESQU value is smaller than the preset Stall_DectEsValue2.


Protect.h

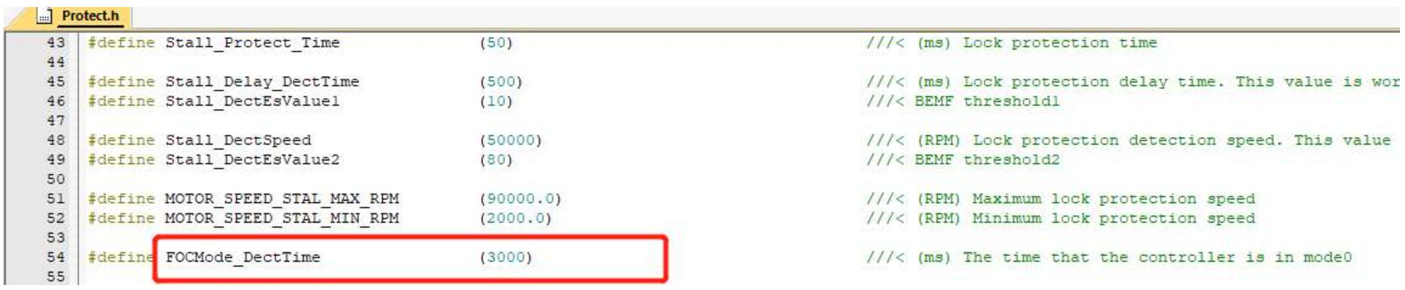
```
#define Stall_Protect_Time              (50)            ///< (ms) Lock protection time

#define Stall_Delay_DectTime            (500)           ///< (ms) Lock protection delay time. This value is wo:
#define Stall_DectEsValue1              (10)            ///< BEMF threshold1

#define Stall_DectSpeed                 (50000)         ///< (RPM) Lock protection detection speed. This value
#define Stall_DectEsValue2              (80)            ///< BEMF threshold2

#define MOTOR_SPEED_STAL_MAX_RPM        (90000.0)       ///< (RPM) Maximum lock protection speed
#define MOTOR_SPEED_STAL_MIN_RPM        (2000.0)        ///< (RPM) Minimum lock protection speed

#define FOCMode_DectTime                (3000)          ///< (ms) The time that the controller is in mode0
```

Method 2: By detecting the estimated speed; motor stall is established when the estimated speed is higher than the preset MOTOR_SPEED_STAL_MAX_RPM value or is lower than the preset MOTOR_SPEED_STAL_MIN_RPM value.


Protect.h

```
#define Stall_Protect_Time              (50)            ///< (ms) Lock protection time

#define Stall_Delay_DectTime            (500)           ///< (ms) Lock protection delay time. This value is work
#define Stall_DectEsValue1              (10)            ///< BEMF threshold1

#define Stall_DectSpeed                 (50000)         ///< (RPM) Lock protection detection speed. This value i
#define Stall_DectEsValue2              (80)            ///< BEMF threshold2

#define MOTOR_SPEED_STAL_MAX_RPM        (90000.0)       ///< (RPM) Maximum lock protection speed
#define MOTOR_SPEED_STAL_MIN_RPM        (2000.0)        ///< (RPM) Minimum lock protection speed

#define FOCMode_DectTime                (3000)          ///< (ms) The time that the controller is in mode0
```

Method 3: When the motor is started, the program will shift from Mode 0 to Mode 1 after judging that the estimated speed is greater than MOTOR_LOOP_RPM, where the motor starts with a fixed current before running in the normal loop. User can determine whether the motor lock condition is present from the Mode. If Mode 0 persists after the FOCMode_DectTime has elapsed, it suggests startup failure, that is, motor lock has occurred.



```
Protect.h
#define Stall_Protect_Time              (50)            ///< (ms) Lock protection time

#define Stall_Delay_DectTime            (500)           ///< (ms) Lock protection delay time. This value is wor
#define Stall_DectEsValue1              (10)            ///< BEMF threshold1

#define Stall_DectSpeed                 (50000)         ///< (RPM) Lock protection detection speed. This value
#define Stall_DectEsValue2              (80)            ///< BEMF threshold2

#define MOTOR_SPEED_STAL_MAX_RPM        (90000.0)       ///< (RPM) Maximum lock protection speed
#define MOTOR_SPEED_STAL_MIN_RPM        (2000.0)        ///< (RPM) Minimum lock protection speed

#define FOCMode_DectTime                (3000)          ///< (ms) The time that the controller is in mode0

```

9.4 Thermal Shutdown (TSD)

9.4.1 Theories

The figure below provides a common schematic representation of thermal shutdown, 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. Temperature detector (TD) is connected to one AD 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.

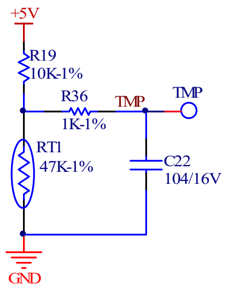
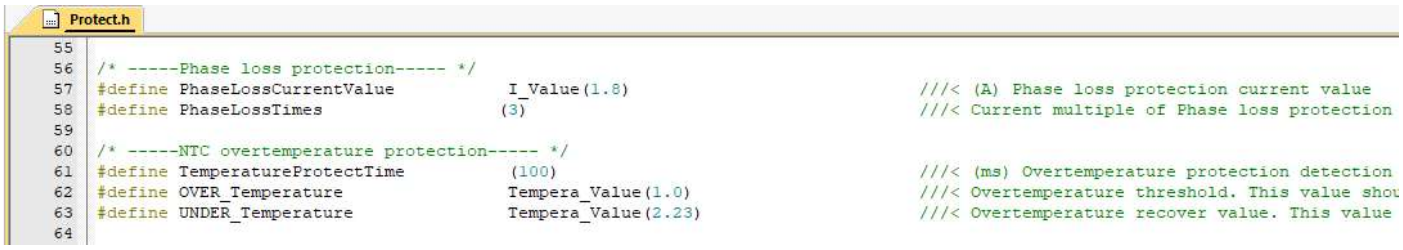


Figure 9-2 Temperature Detection Schematic

9.4.2 Code Implementation



```
Protect.h

/* -----Phase loss protection----- */
#define PhaseLossCurrentValue           I_Value(1.8)            ///< (A) Phase loss protection current value
#define PhaseLossTimes                  (3)                     ///< Current multiple of Phase loss protection

/* -----NTC overtemperature protection----- */
#define TemperatureProtectTime          (100)                   ///< (ms) Overtemperature protection detection
#define OVER_Temperature                Tempera_Value(1.0)      ///< Overtemperature threshold. This value shou
#define UNDER_Temperature               Tempera_Value(2.23)     ///< Overtemperature recover value. This value

```

Where, OVER_ Temperature is the protection setpoint, 1.0 is the resistance 1 Ω of NTC resistor at 80℃, UNDER_Temperature is the recovery value, 2.23 is the resistance 2.23 Ω of NTC resistor at 70℃ .

Note: If the pull-up resistor value is not 10K and the pull-up voltage is not 5V, the defied formula should be modified.

9.5 Phase Loss Protection

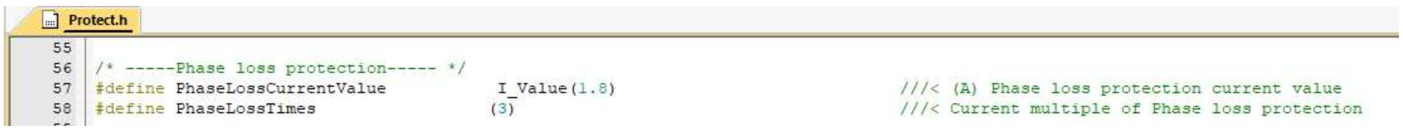
9.5.1 Theories

When one phase of a three-phase motor is lost, the three-phase current becomes asymmetrical. Hence, phase loss protection can be implemented by detecting the maximum value of three-phase current within a given period of time in the program and judging whether it is asymmetrical.

Program implementation: Phase loss is established if the maximum current of one phase measured is greater than PhaseLossTimes the maximum current of the other phase which is greather than PhaseLossCurrentValue.

Notes: In some applications, phase loss may not be detected in the above ways because the lost phase may have burrs, which may cause the maximum sensed current to be close to those of the other two phases. Solution: Phase loss can be determined by comparing the accumulated current value within a given period of time through integration.

9.5.2 Code Implementation



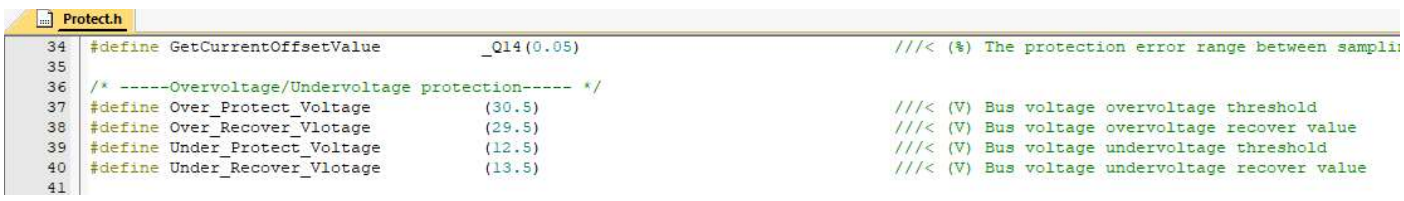
```c

/* -----Phase loss protection----- */
#define PhaseLossCurrentValue          I_Value(1.8)                  ///< (A) Phase loss protection current value
#define PhaseLossTimes                (3)                            ///< Current multiple of Phase loss protection
```

9.6 Overvoltage/Undervoltage Lockout (OVLO/UVLO)

9.6.1 Theories

The program senses the voltage applied on the AD2 port and triggers overvoltage lockout when the sensed voltage exceeds the setpoint; and the overvoltage fault is cleared when the sensed voltage is again below the overvoltage recovery value. When the sensed voltage is lower than the undervoltage lockout setpoint, the program will trigger undervoltage lockout; and the undervoltage fault is cleared when the sensed voltage is again below the undervoltage recovery value.

9.6.2 Code Implementation



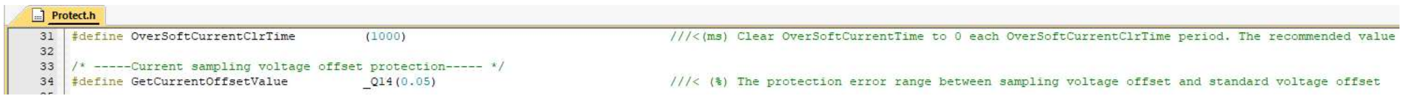
```c
#define GetCurrentOffsetValue          _Q14(0.05)                    ///< (%) The protection error range between sampli

/* -----Overvoltage/Undervoltage protection----- */
#define Over_Protect_Voltage           (30.5)                        ///< (V) Bus voltage overvoltage threshold
#define Over_Recover_Vlotage           (29.5)                        ///< (V) Bus voltage overvoltage recover value
#define Under_Protect_Voltage          (12.5)                        ///< (V) Bus voltage undervoltage threshold
#define Under_Recover_Vlotage          (13.5)                        ///< (V) Bus voltage undervoltage recover value

```

9.7 Bias Voltage Protection

9.7.1 Theories

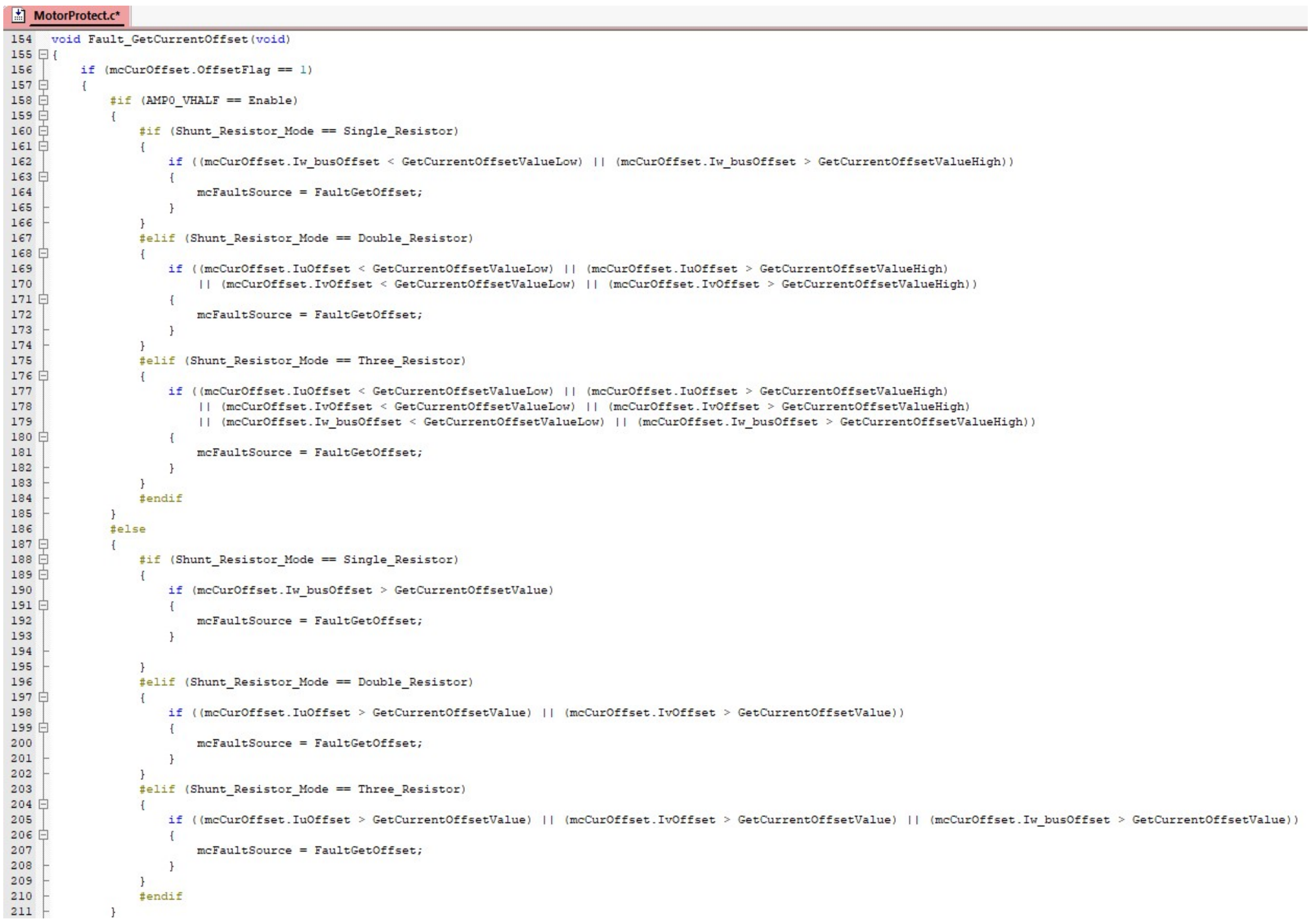
Bias voltage is measured before the motor starts up. When VHALF is attached, the theoretical value of bias voltage is 2048 which converts to 16383 after being moved left by 3 bits; when VHALF is not connected, the theoretical value of bias voltage is 0; when the error of measured bias voltage exceeds GetCurrentOffsetValue of its theoretical value, the bias voltage is considered abnormal. 0.05 represents 5%.

9.7.2 Code Implementation



Protect.h

```
#define OverSoftCurrentClrTime          (1000)                              ///<(ms) Clear OverSoftCurrentTime to 0 each OverSoftCurrentClrTime period. The recommended value

/* -----Current sampling voltage offset protection----- */
#define GetCurrentOffsetValue           _Q14(0.05)                          ///< (%) The protection error range between sampling voltage offset and standard voltage offset
```



MotorProtect.c*

```
void Fault_GetCurrentOffset(void)
{
    if (mcCurOffset.OffsetFlag == 1)
    {
        #if (AMP0_VHALF == Enable)
        {
            #if (Shunt_Resistor_Mode == Single_Resistor)
            {
                if ((mcCurOffset.Iw_busOffset < GetCurrentOffsetValueLow) || (mcCurOffset.Iw_busOffset > GetCurrentOffsetValueHigh))
                {
                    mcFaultSource = FaultGetOffset;
                }
            }
            #elif (Shunt_Resistor_Mode == Double_Resistor)
            {
                if ((mcCurOffset.IuOffset < GetCurrentOffsetValueLow) || (mcCurOffset.IuOffset > GetCurrentOffsetValueHigh)
                    || (mcCurOffset.IvOffset < GetCurrentOffsetValueLow) || (mcCurOffset.IvOffset > GetCurrentOffsetValueHigh))
                {
                    mcFaultSource = FaultGetOffset;
                }
            }
            #elif (Shunt_Resistor_Mode == Three_Resistor)
            {
                if ((mcCurOffset.IuOffset < GetCurrentOffsetValueLow) || (mcCurOffset.IuOffset > GetCurrentOffsetValueHigh)
                    || (mcCurOffset.IvOffset < GetCurrentOffsetValueLow) || (mcCurOffset.IvOffset > GetCurrentOffsetValueHigh)
                    || (mcCurOffset.Iw_busOffset < GetCurrentOffsetValueLow) || (mcCurOffset.Iw_busOffset > GetCurrentOffsetValueHigh))
                {
                    mcFaultSource = FaultGetOffset;
                }
            }
            #endif
        }
        #else
        {
            #if (Shunt_Resistor_Mode == Single_Resistor)
            {
                if (mcCurOffset.Iw_busOffset > GetCurrentOffsetValue)
                {
                    mcFaultSource = FaultGetOffset;
                }

            }
            #elif (Shunt_Resistor_Mode == Double_Resistor)
            {
                if ((mcCurOffset.IuOffset > GetCurrentOffsetValue) || (mcCurOffset.IvOffset > GetCurrentOffsetValue))
                {
                    mcFaultSource = FaultGetOffset;
                }
            }
            #elif (Shunt_Resistor_Mode == Three_Resistor)
            {
                if ((mcCurOffset.IuOffset > GetCurrentOffsetValue) || (mcCurOffset.IvOffset > GetCurrentOffsetValue) || (mcCurOffset.Iw_busOffset > GetCurrentOffsetValue))
                {
                    mcFaultSource = FaultGetOffset;
                }
            }
            #endif
        }
```

10 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	2022-06-16	You Yuchao	

11 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/