

Refrigerator Compressor BLDC Sensorless Control Based on MC56F82xxx

Contents

1. Introduction

For the purpose of increasing the efficiency, Brushless DC motor (BLDC) are used more and more often in the reciprocating compressors of refrigerators. Considering there's nowhere to install sensors in the compressor, the motor has to be controlled in a sensorless way. This application note provides a reliable sensorless Field Oriented Control (FOC) method targeting the BLDC control inside a refrigerator compressor. The control algorithm is realized on MC56F82xxx.

2. Anatomy of refrigerator control system

There're usually two parts of control existing in a refrigerator, one is for compressor control and the other is for the system control. See *Figure 1*, The system control part mainly reads the temperatures of the chambers, the environment, and so on to decide the speed of the compressor, the states of the fans and the defrost heater in the fridge based on a control strategy. It also drives a control panel with display and key inputs on it. The system control part outputs a PWM signal with its frequency indicating the command speed, and the compressor control part drives the motor per this command. Typically, a frequency range of 40 Hz ~ 150 Hz corresponds to 1200 RPM ~ 4500 RPM.

1.	Introduction.....	1
2.	Anatomy of refrigerator control system	1
3.	Background of refrigeration system	3
4.	Some features in refrigerator compressor control	3
4.1.	Startup under high pressure difference.....	3
4.2.	Efficiency.....	4
4.3.	Protections	4
5.	Sensorless FOC for BLDC in a compressor.....	5
5.1.	Alignment	5
5.2.	Startup with predicted position	5
5.3.	Spin with estimated position (flux observer)– speed open-loop	8
5.4.	Spin with estimated position (flux observer) – speed closed-loop.....	11
6.	Software implementation	12
6.1.	Main state machine	12
6.2.	Sub state machine	13
6.3.	Faults handling.....	14
7.	User guide of the software	15
7.1.	Project structure	15
7.2.	How to adapt to a new compressor	16
7.3.	Startup performance tuning.....	17
7.4.	Protection settings.....	18
7.5.	Other settings	22
7.6.	Use FreeMASTER to control the compressor.....	27
8.	Conclusion	28
9.	Revision history	30

The system control and compressor control parts are usually implemented on separate MCUs in current markets. It is common that there's a system control board and a compressor control board together with a HMI module inside a refrigerator. These two parts can be also implemented on one board so as to eliminate a set of AC-DC power circuits. This application note focuses on the compressor control part with a controller of MC56F82xxx, which has been implemented on NXP High-Voltage Development Platform (HVP-MC3PH) and HVP-56F82748 daughter card. For more information about HVP platform, please refer to www.nxp.com/hvp.

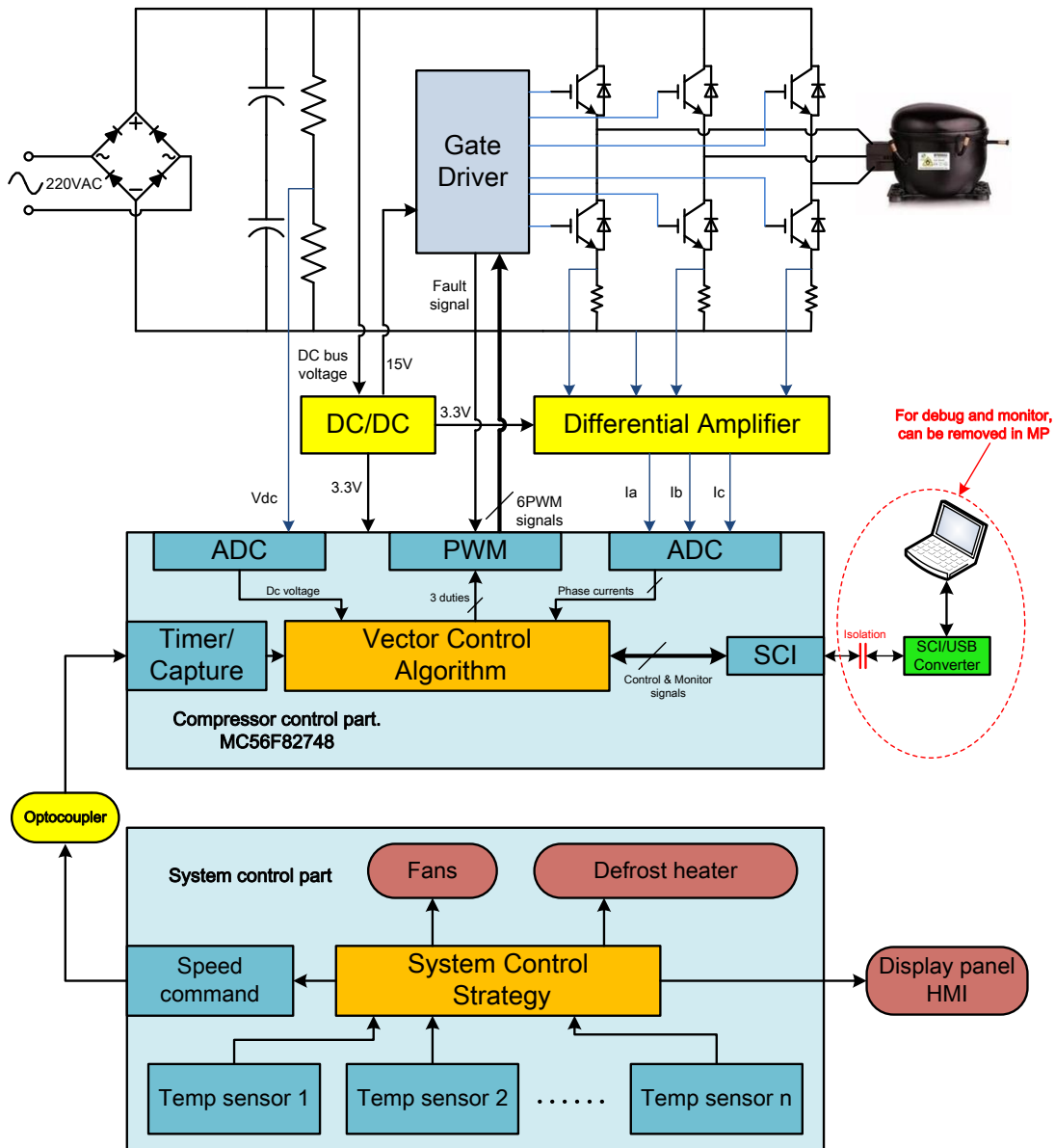


Figure 1. Refrigerator system anatomy

3. Background of refrigeration system

A typical refrigeration system is composed of a compressor, a condenser, a metering device and an evaporator. The metering device is often a capillary tube in refrigerator. When the system starts to work, the compressor compresses the low pressure vapor refrigerant from its inlet and generates high pressure high temperature vapor at its outlet. This high pressure high temperature vapor refrigerant flows into the condenser. Because the ambient air is cooler than the condenser, the heat is transferred to the cooler air and the vapor refrigerant becomes a high pressure liquid status. Then this high pressure liquid refrigerant leaves the condenser and flows into the metering device which is a capillary tube. The refrigerant becomes a low pressure and cooler liquid when it reaches to the evaporator. The cooler refrigerant in the evaporator tubes absorbs the heat in the air where the evaporator is placed, and it changes to a low pressure cold vapor when it reaches to the inlet of the compressor. The low pressure vapor refrigerant is sucked into the compressor and the cycle starts over. The high-side pressure (measured at the outlet of the compressor) increases significantly as the refrigerating cycle goes on, and the low-side pressure (measured at the inlet of the compressor) also decreases a little bit.

4. Some features in refrigerator compressor control

There are some major features in refrigerator compressor control:

- The loading is not constant but changes periodically every mechanical revolution, namely, there's a maximum loading torque and a minimum loading torque in each mechanical revolution due to the reciprocating motion of the piston.
- The residual pressure difference between the inlet and outlet of the compressor can be very large, which makes startup difficult.
- Efficiency is very important to refrigerator. Since the compressor is stopped most of the time, the influence of the control board consumption becomes important.
- There are all kinds of protections.

4.1. Startup under high pressure difference

The motor inside the compressor drives a crankshaft, which in turn drives a piston moving in a reciprocating motion. The vapor is compressed in this motion. Since the high-side pressure is much higher than the low-side pressure, there's a significant load torque change in one mechanical revolution. When motor is running at high speed, this periodic load torque change is not a big problem because the load change in a very short period of time won't lead to much speed variation. When the compressor stops after working for a while, the pressure difference between high-side and low-side still exists, and it'll come to zero over time. When there's a large residual pressure difference, the loading can be either large or small at the very moment of startup because the exact rotor and piston position are not known, hence we don't know whether the piston is to move against the pressure or the other way around at this very moment of startup, which makes it rather difficult to start the motor in traditional open-loop start up fashion due to the absence of position sensors. In practical use, when a compressor is stopped, it won't be started immediately even if there's a valid speed command, unless a couple of minutes (usually 5~10 min) have passed by. But even so, when the ambient temperature of a refrigerator is high, the residual pressure difference could be still large, which makes start up really challenging. The startup method

mentioned here uses a fast converging flux observer at open-loop startup, which greatly shortens the startup time. This method has been tested and proved to be reliable under production. Typically, the startup can be reliable when the pressure difference is around 0.6 MPa.

4.2. Efficiency

The popular working pattern of the refrigerator compressor is still on/off mode even though the control method of the compressor is FOC. The motor only runs at several specified speeds, e.g. 1200 RPM, 2700 RPM, 3400 RPM and 4300 RPM. These speeds are decided based on the efficiency of the compressor, so different compressors may have different optimal running speeds. The system control strategy of the refrigerator affects the temperature stability and the system efficiency. For instance, when the chamber temperature is higher than desired, the compressor should be turned on, but which speed should be used? It really makes a difference on the efficiency when different control strategies are applied. There may be around 50% of the time that the compressor isn't working at all. There are several key factors that affect the system efficiency.

- The cooling efficiency of the compressor itself
- The motor running efficiency
- The control strategy of the whole system
- Since the compressor stops almost half the time, the power consumption of the control boards becomes crucial

From the perspective of the motor controller, the estimated position accuracy affects the motor running efficiency. The observers mentioned in this note are suitable for compressor control. In order to reduce the power consumption of the control boards, it is recommended that switching mode power supplies should be used instead of LDO, also enable STOP mode of MC56F82xxx when the speed command is zero. Switching loss can be reduced by using a lower PWM frequency and SVPWM without 111 NULL vector. This implementation uses a PWM of 8 KHz, but 5~6 KHz can also do the work due to the large inductance of the windings.

4.3. Protections

The protections on compressor control part are various. Most systems include protections of hardware triggered over current, DC bus under voltage, DC bus over voltage, startup fail (stall), open phase detection. Other systems may require additional software triggered over current or over power protection.

5. Sensorless FOC for BLDC in a compressor

The startup procedure is well designed as four stages, during which startup fail is under detection. A quasi-synchronous reference frame d-q is used before estimated rotor position is used. The four stages are:

- **Alignment**
- **Startup**: speed open-loop startup with predicted position
- **Spin**: speed open-loop spin with estimated position
- **Spin**: speed closed-loop with estimated position.

5.1. Alignment

Alignment is to align the rotor to a known position. In this case, a current vector of 1.5 A is placed at the q-axis, and the position of d-axis is located at -90° . So the rotor is actually expected to be pulled on A-axis or α -axis. See *Figure 2*. The alignment lasts two seconds, and the current rises from 0 to 1.5 A at a ramp of 1.5 A/s.

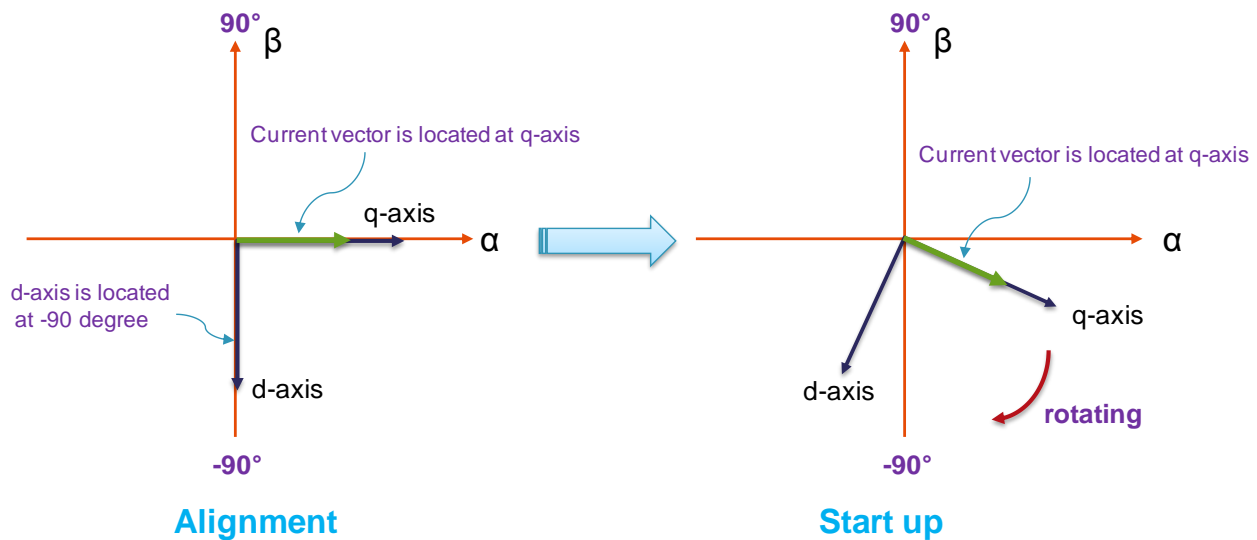


Figure 2. Current vector placement at alignment

5.2. Startup with predicted position

After alignment, the current vector starts to rotate. The rotating speed increases from 0 to a certain value with a ramp of -200 RPM/s, and the predicted position is an integration of this given predicted speed. The current vector is still placed at the q-axis, and the d-axis rotates inversely from -90° to 90° . This stage ends as soon as the d-axis reaches 90° . *Figure 3* shows the rotation of the current vector in this stage.

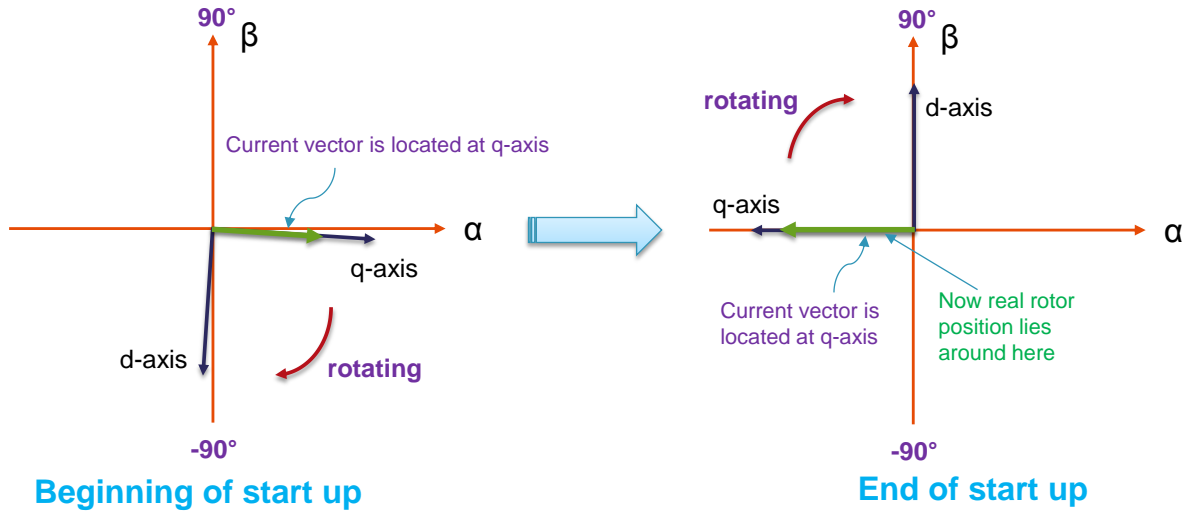


Figure 3. Startup with predicted position

Figure 4 shows the real values of the variables in this stage. There are four scopes in figure 4:

- The first one on the very top is the predicted speed illustrated in red line
- The second one with green and blue lines are predicted position and estimated position
- The third one with an orange line is the estimated speed
- The last one with a purple line is a state variable, a value 3 indicates this open-loop startup stage, which is from time point T1 to T2, as enclosed in a shadowed rectangular

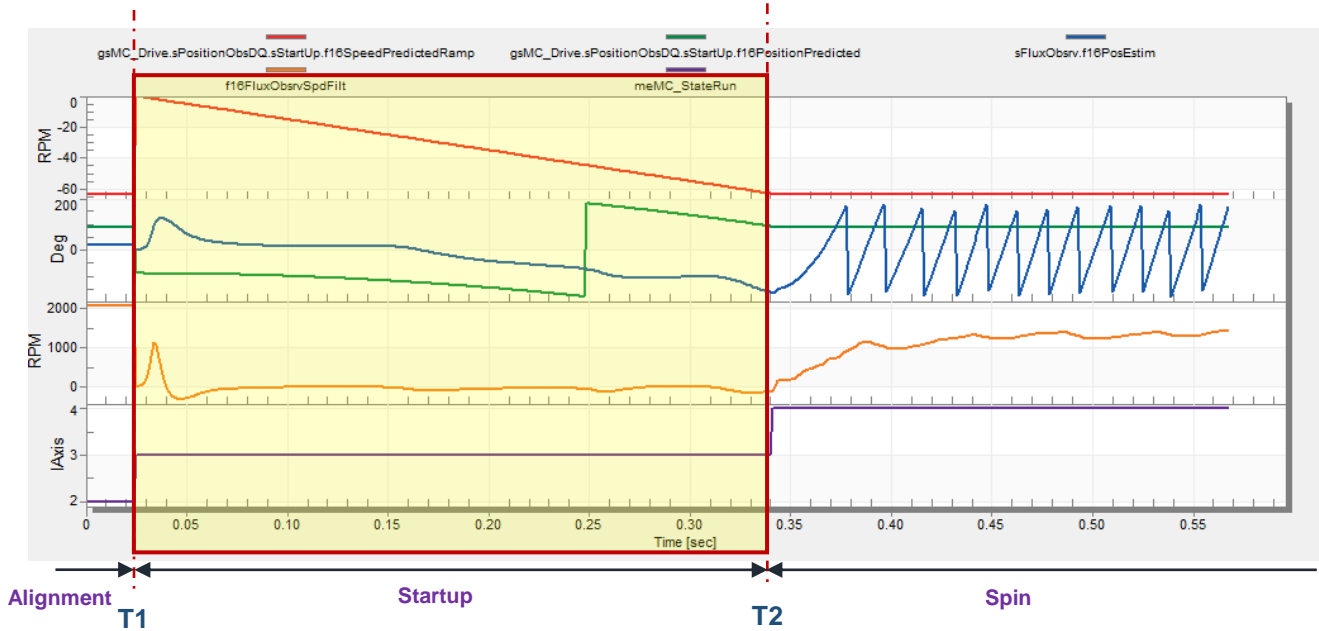


Figure 4. Predicted and estimated position and speed during open-loop startup

Here is what happens in this open-loop startup stage:

- 1) Alignment stage ends at the time point of T1, and the rotor is expected to be already aligned with α -axis where the current vector is located.
- 2) Open-loop startup stage starts from time point T1 and ends at T2. During this stage, the current vector is also located at q-axis. Since the predicted speed increases slowly from 0 RPM to around -60 RPM as indicated by the red line, the d-axis rotates very slowly from -90° to 90° as indicated by the green line. Since current vector is located at q-axis, the current vector also rotates from 0° to -180° . The rotor naturally follows the current vector, since it rotates very slowly.
- 3) The current vector only rotates 180 electrical degrees, so the duration of this stage is short and the rotor is supposed to be pulled near -180° in the end, which is time point T2. A flux observer is enabled in this stage, which is expected to converge and get the rotor position at the end of this stage (time point T2). As shown in Figure 4, the estimated rotor position (blue line) is around -150° in the end (time point T2).

After time point T2, the state machine enters spin state, where estimated rotor position is used for FOC as shown in [Figure 10](#) and [Figure 11](#).

This open-loop startup lasts about 300 ms (from T1 to T2) which is indicated by the last purple line. During this stage, the rotating speed should be as slow as possible so that the rotor can still follow the current vector under large pressure difference. And the duration of this open-loop startup should be as short as possible because we don't want to experience too much loading torque fluctuations due to the reciprocation movement, and that's why the current vector only moves 180° in this case. It's also important that the observer is able to converge and get the correct position in the end of this stage.

[Figure 5](#) shows the block diagram of startup. The estimated position is used at the end of this stage.

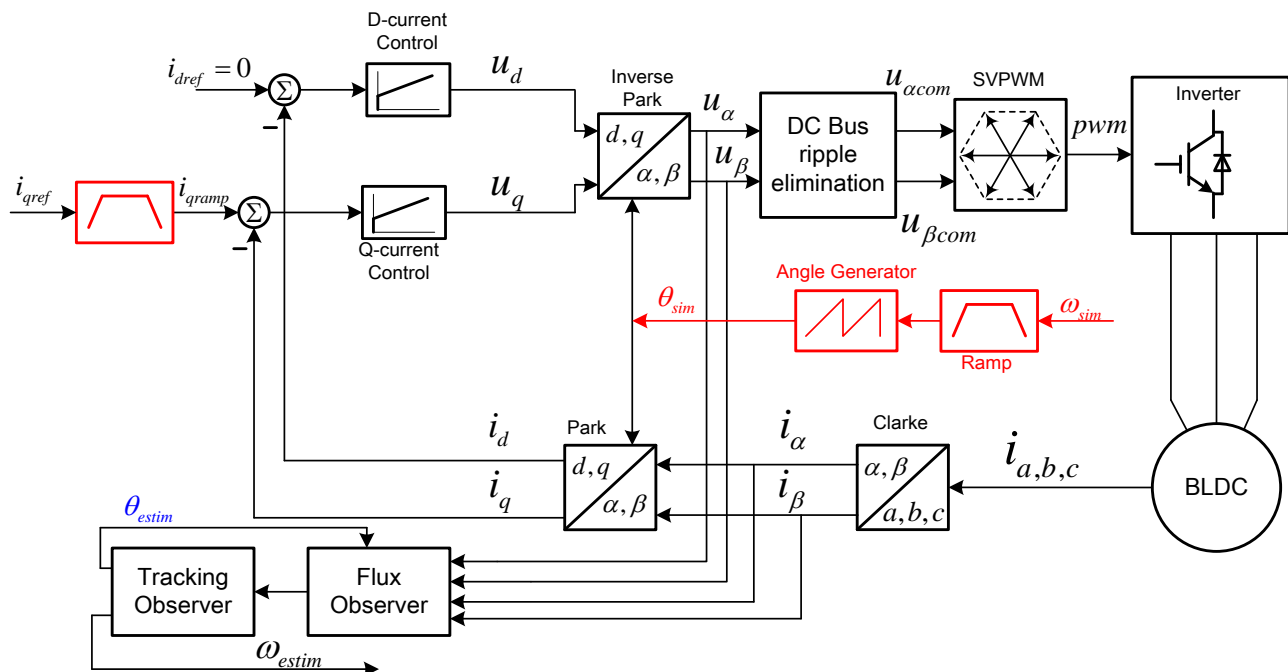


Figure 5. Block diagram of open-loop startup

5.3. Spin with estimated position (flux observer)– speed open-loop

Figure 6 shows the real values of some key variables in this stage, where speed is not controlled and estimated rotor position is used for FOC. It is startup stage before time point T2. Torque current reference is maintained as a constant between T2 and T4, which means electrical torque is a constant and speed is accelerated naturally with no regulation. Once the estimated speed is over 1000 RPM (after time point T4), speed regulator is enabled.

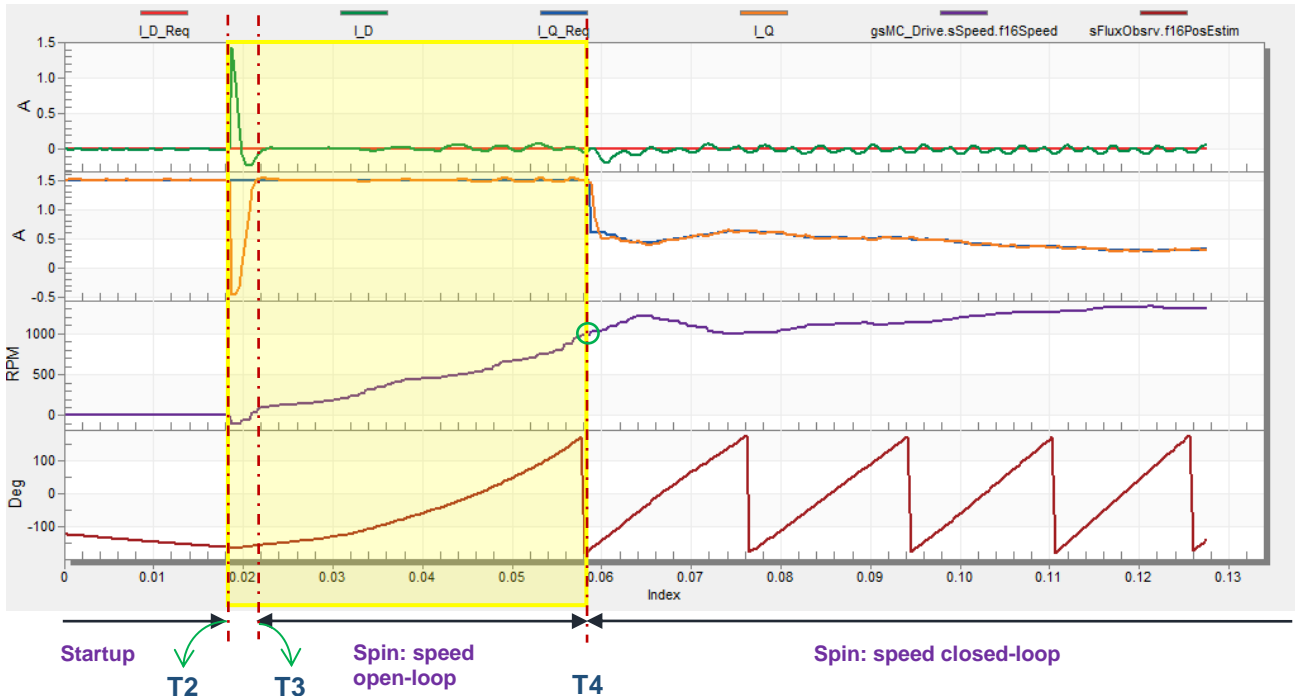


Figure 6. Id and Iq feedback from startup to spin state

The shadowed part in figure 6 reflects this stage. The meanings of the variables in figure 6 are:

- The first scope (the top one) contains reference and feedback d-axis currents. The green one is the feedback Id and the pinkish red one is the reference Id.
- The second scope contains reference and feedback q-axis currents. The blue one is the reference Iq and the orange one is the feedback Iq.
- The third scope (counting from top) with a purple line is the estimated speed from the flux observer.
- The last scope is the estimated rotor position from flux observer.

Here is what happens in this speed open-loop spin stage:

At the end of startup (time point T2 in figure 6), the predicted position of d-axis is at 90° , but the estimated position is around -180° as shown in Figure 3. Since the estimated position is used in this stage, the position of d-axis is changed abruptly from 90° to a position around -180° , as illustrated in Figure 7. This position change of dq frame leads to the fact: the feedback of Id jumps to a positive value and the feedback of Iq jumps to a value near zero at the beginning of spin state as shown in Figure 7. This is also shown in Figure 6 at the time point of T2, where I_D and I_Q in the upper first and second scopes are feedback dq currents.

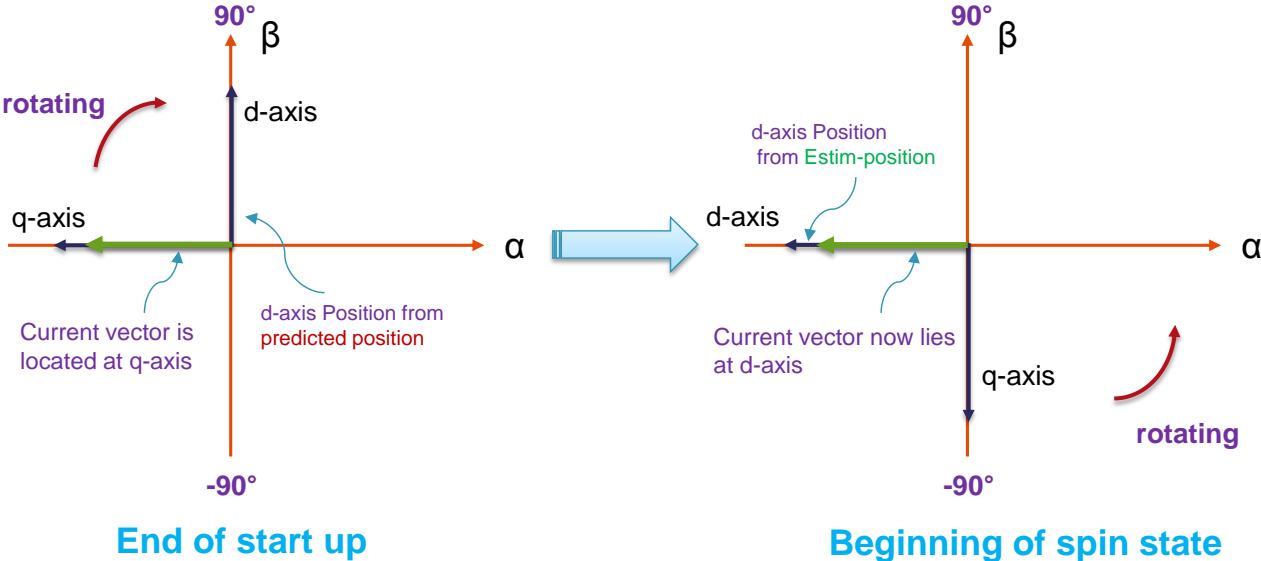


Figure 7. Position jump between startup and spin

The reference currents of dq frame maintain the same, which means Id reference is still zero, and the Iq reference is still 1.5 A. Since current loop is much faster than speed loop, the current vector will jump ahead for 90° very quickly, which leaves an angle of 90° between the current vector and the rotor, so a maximum electrical torque is generated. This is shown in time frame T2~T3 in Figure 6, where I_D_Req and I_Q_Req are dq current references, and the motor is quickly accelerated from time point T3 on. Figure 8 shows how the current vector jumps to accelerate the motor in vector diagram. The duration of T2~T3 is about 4ms in Figure 6, which means current controller dynamic response is much faster compared with speed response.

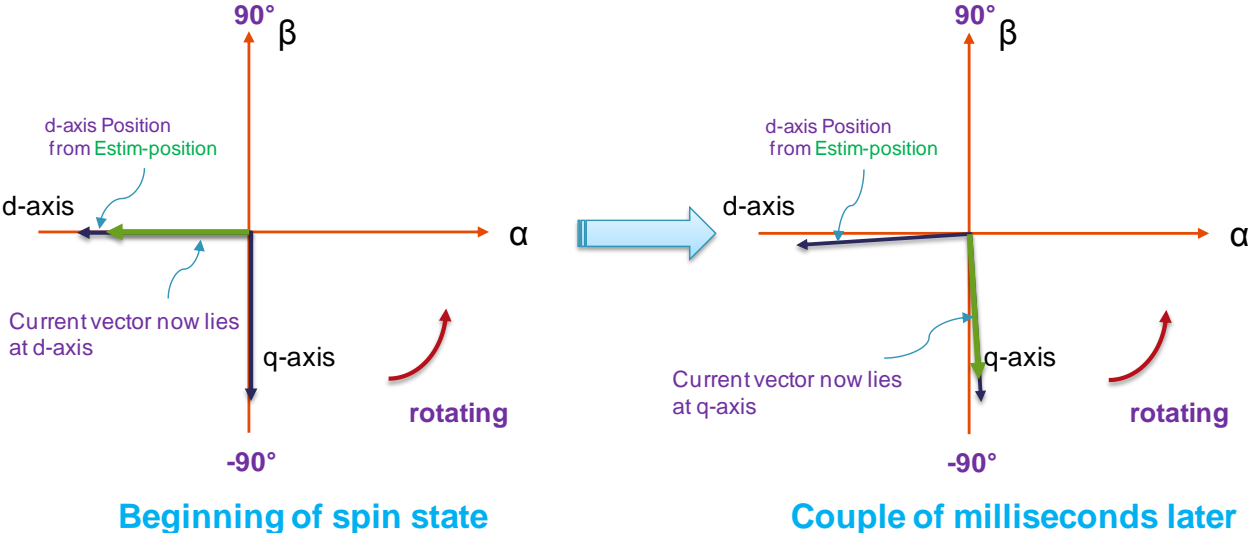


Figure 8. Current vector jumps 90° at the beginning of spin state

Current vector is placed at q-axis which is 90° ahead of rotor flux from time point T3 on. Motor is accelerated under a constant electrical torque (this torque should be designed as large as possible so as to cope with large loading, but also should make compromise with speed over-shoot and copper loss). Once the estimated speed reaches 1000 RPM, speed regulator is enabled, which is the time point of T4 in Figure 6.

Another state observer (based on DQ rotation frame) is enabled from the beginning (T2) of this stage, as shown in Figure 9. The time stamps and the shadowed part in figure 9 share the same meaning of which in figure 6. The startup is deemed as failure if the estimated speed of flux observer doesn't reach 1000 RPM within 0.35 s, the motor will start up again with a current of 2.5 A.

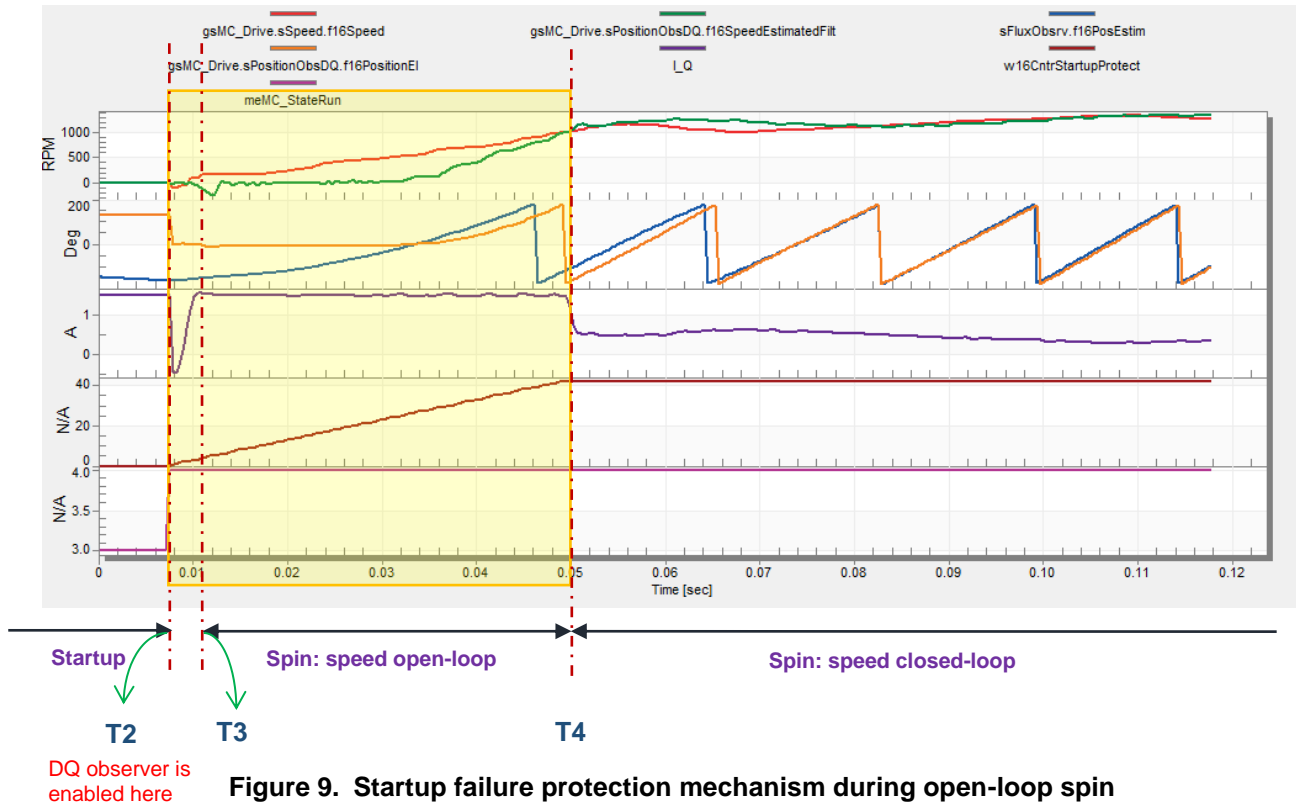


Figure 9. Startup failure protection mechanism during open-loop spin

In figure 9, the meanings of the variables are:

- The first scope (the top one) illustrates the estimated speeds during open-loop spin stage: the red one is out of the flux observer, and the green one is from the state observer.
- In the second scope, the blue one is the position generated by flux observer, while the orange one is the position generated by the state observer.
- In the third scope, a counter counts the time when the estimated speed of the flux observer is below 1000 RPM.

Once the counter exceeds 350 (which means 0.35 s because the counter increases in slow loop of 1 KHz), the startup is deemed fail. In this case, it takes about 40ms to accelerate to 1000 RPM. The block

diagram of this stage is shown in *Figure 10*

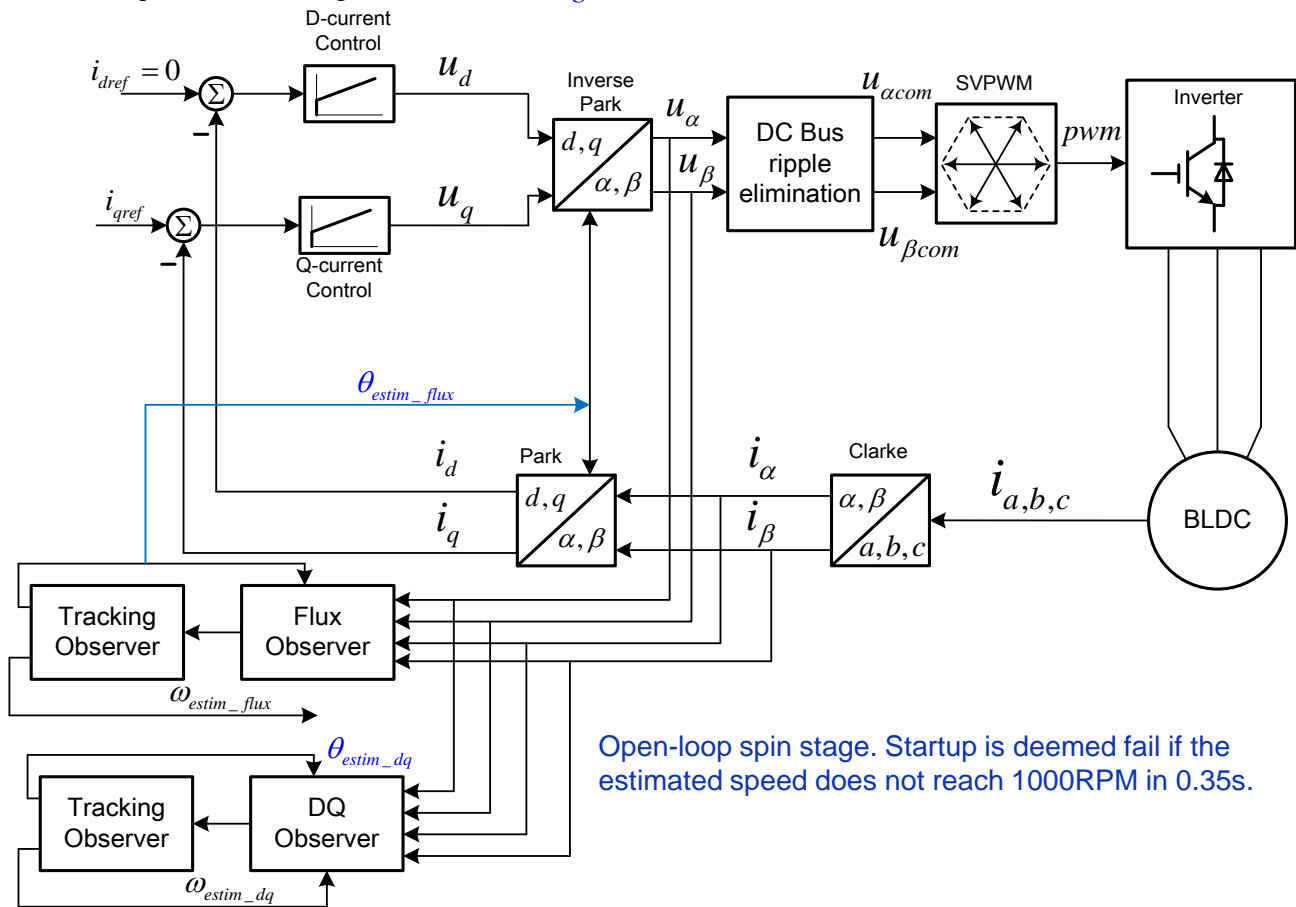


Figure 10. Block diagram of open-loop spin stage

5.4. Spin with estimated position (flux observer) – speed closed-loop

Once the estimated speed of flux observer exceeds 1000 RPM, the speed regulator is enabled, and the estimated speeds of the two observers are compared for the coming two seconds. If the speed difference is larger than 250 RPM for over 1.5 s during this two seconds, the startup is also deemed fail. The estimated speed of flux observer is used as feedback. The block diagram of speed closed-loop using estimated position and speed of flux observer is shown in *Figure 11*. The extra state observer is used as a safety mechanism for the startup. It can be used for other purposes or eliminated if desired. There's a lubrication stage after speed closed-loop is enabled, which means the compressor runs at a pre-defined speed profile for several minutes regardless of the speed command. Different compressors have different lubrication requirements.

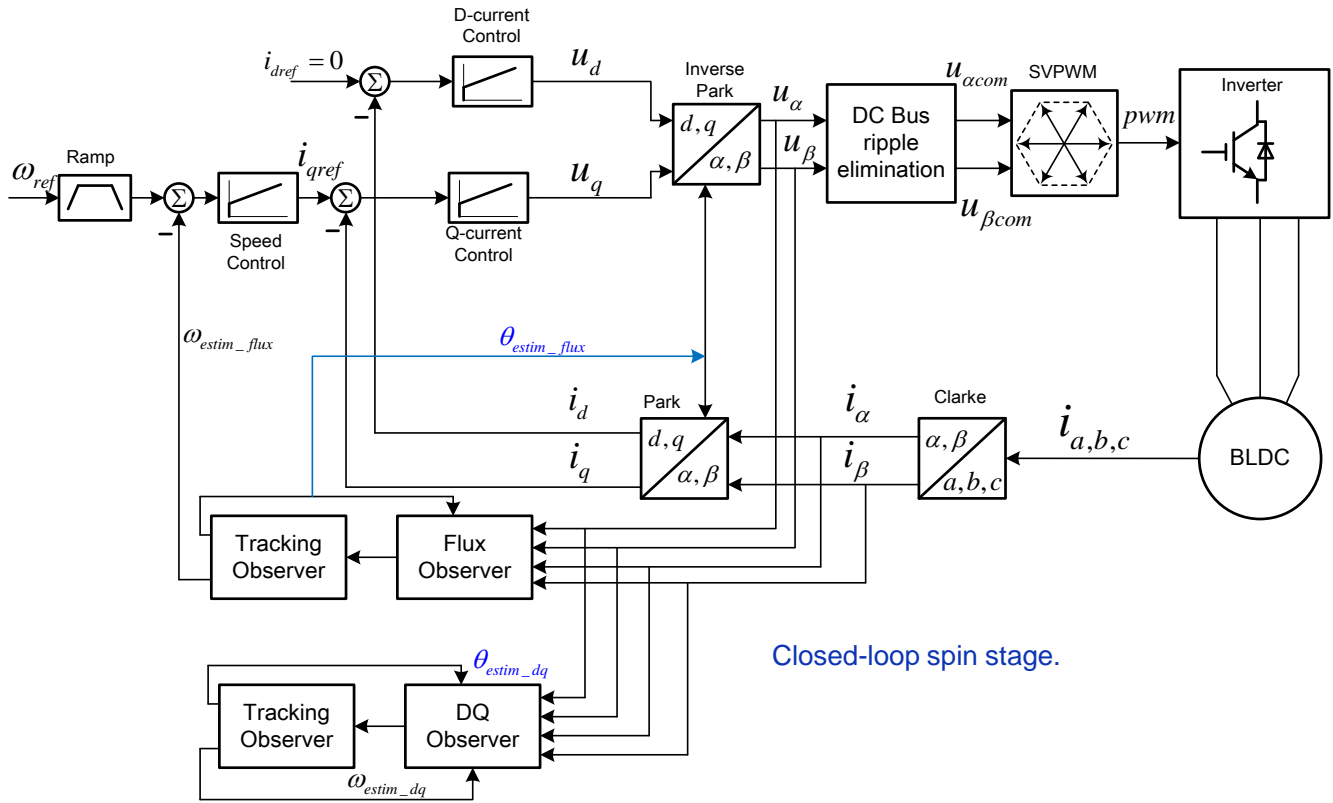


Figure 11. Speed closed-loop in spin stage using the outputs of flux observer

6. Software implementation

MC56F82xxx is a member of DSC family. The peripherals and the CPU core combining MCU and DSP features makes it powerful in motor control and power conversion applications. The algorithm is realized in ADC ISR, which is triggered by PWM module in a frequency of 8 KHz. A main state machine is invoked in the ADC ISR and the three duties are updated every PWM period. The main state machine includes four states: INIT, STOP, RUN and FAULT. There's a sub state machine in the RUN state to manage the detailed behaviors of the motor.

6.1. Main state machine

The main state machine diagram is shown in [Figure 12](#). The default first state is INIT, where all the variables are initialized. Then it jump to STOP state naturally. A switch variable called mbMC_SwitchAppOnOff decides if RUN state shall be entered or not. The default value of mbMC_SwitchAppOnOff is 1, so the program actually goes to RUN state shortly after the system is powered on. The system goes to FAULT state if any fault occurs, and goes back to INIT when all faults disappear.

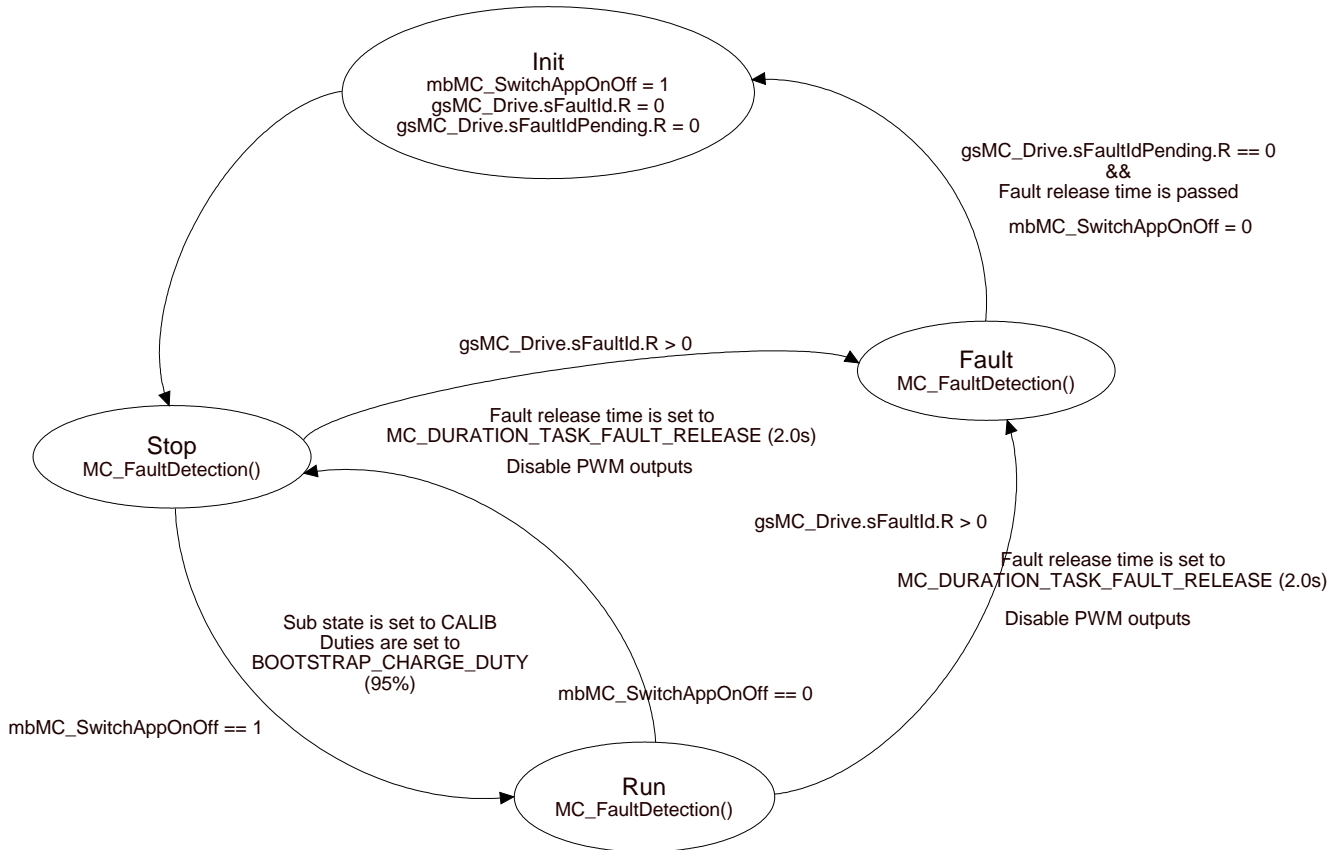


Figure 12. Main state machine

6.2. Sub state machine

A sub state machine resides in RUN state of the main state machine to manage the motor running behaviors. [Figure 13](#) shows this sub state machine. There are six states here:

Calib: Phase current offsets are measured and stored.

Ready: This is the state where motor is stopped waiting for speed command. The motor won't start if the value of variable uw32CounterPressureRelax is not 0, so as to avoid starting up under high pressure difference.

Align: The rotor is aligned to A-axis as described in [Section 5.1, "Alignment"](#)

Startup: Open-loop startup as described in [Section 5.2, "Startup with predicted position"](#)

Spin: Motor runs with estimated position as described in [Section 5.3, "Spin with estimated position \(flux observer\)– speed open-loop"](#) & [Section 5.4, "Spin with estimated position \(flux observer\) – speed closed-loop"](#)

Freewheel: A transition state where all transistors are off, motor is freewheeling.

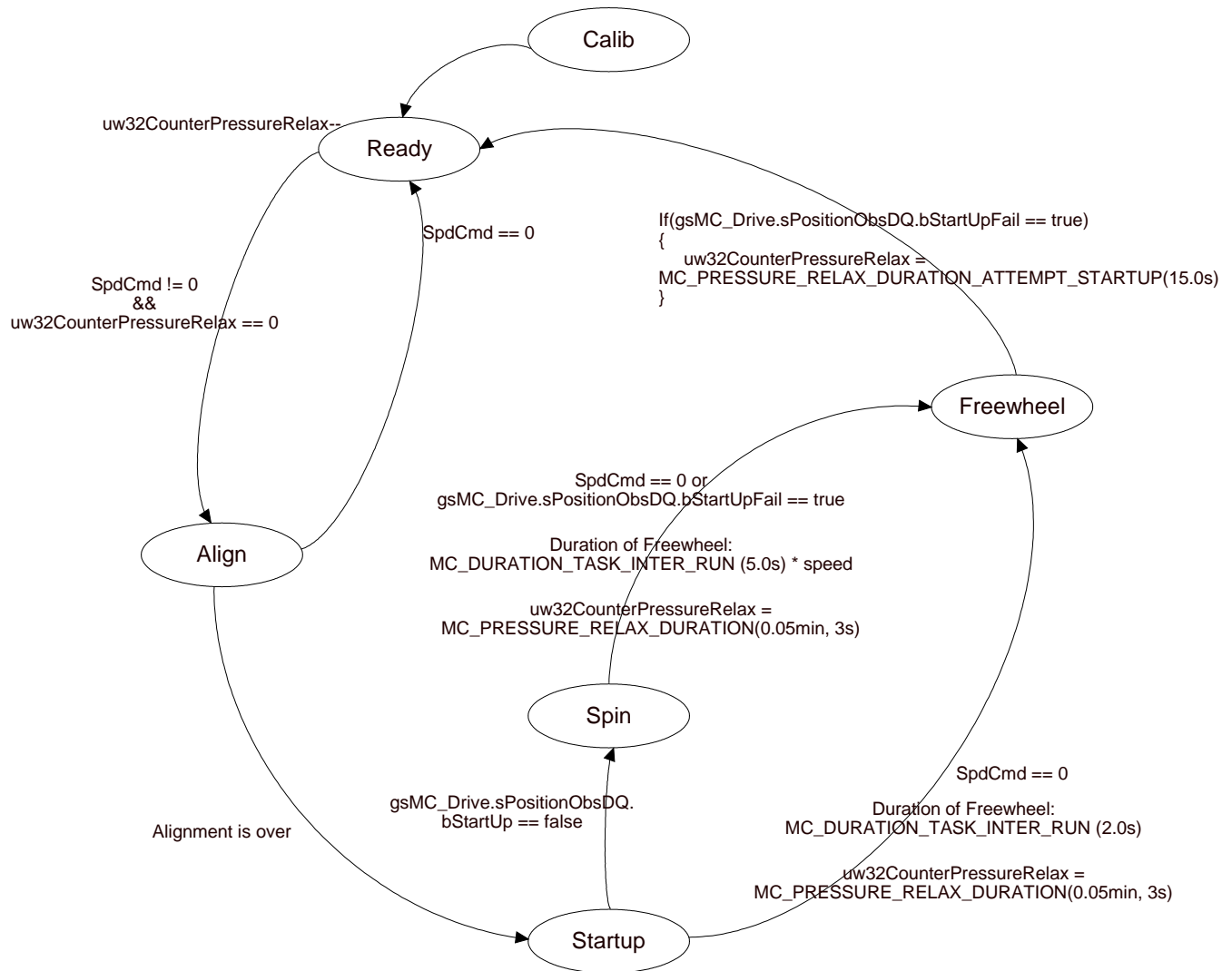


Figure 13. Sub state machine

Figure 13 also shows the duration of Freewheeling state and the waiting time for pressure releasing represented by variable `uw32CounterPressureRelax`. Normally, the time interval between two startups is determined by speed command from system control part as shown in Figure 1. This application adds a minimum time interval of 3 s between two consecutive startups if nothing goes wrong. The time interval is set to 15 s if startup fails and 6min if any fault occurs.

6.3. Faults handling

There are several faults need to be detected and dealt with. A LED is used to indicate the type of fault when anyone of them occurs. The LED will blink for 6 min and after which the motor can be started again if there exists no fault.

Overcurrent protected by hardware: This protection is provided by eFlexPWM module inside DSC chip. An active fault signal can turn off the six transistors without CPU overhead.

Overcurrent protected by software: The software monitors the length of the current vector, the PWM is turned off if the amplitude of the phase current is over 3 A.

DC bus Overvoltage protection: The threshold is 390 V. The PWM is turned off once the detected DC bus voltage is above this threshold.

DC bus Undervoltage protection: The threshold is 180 V. The PWM is turned off if the detected DC bus voltage is under this threshold for a continuous duration time of 125 ms.

Overload protection: When lubrication stage is over and the speed command is smaller than 1800 RPM, the PWM is turned off if the estimated speed is under 600 RPM for an accumulative time of 5 ms.

Open-phase protection: When one or more phases of the motor are disconnected from the controlling board, the PWM is turned off. The motor won't start up anymore if there've been open-phase faults occurring for totally 5 times, unless the system is powered off and then powered on again.

Stall protection: The rotor is deemed to be stalled if there've been continuous startup failures of 3 times.

7. User guide of the software

The software is realized in CodeWarrior10.6 with ProcessorExpert based on HVP-MC3PH and HVP-56F82748 daughter card. All the necessary functions have been integrated into the code, all users have to do is modify some macros in PMSM_SpeedVectorCtrl.h or some ProcessorExpert settings if different compressor is used.

Steps to set up the demo:

1. Get the HVP-MC3PH and HVP daughter card to set up the hardware platform
2. Install the latest CodeWarrior
3. Download sample project from www.nxp.com
4. Configure the header file based on your compressor following the instructions in this chapter

7.1. Project structure

The project view in CodeWarrior and some explanations about the folders and files are shown in figure 14. When system is powered up, the first instruction is fetched from the hardware reset vector (Vectors.c), which is a jump instruction jumping to “_EntryPoint” routine (CPU_Init.c). System clock is initialized in _EntryPoint routine, and then it jumps to “init_56800_” routine (56F83x_init.asm) to initialize stack, memory areas etc. At the end of “init_56800_” routine, the program jump to main() function in ProcessorExpert.c, where the compressor control begins.

For different compressors, depending on the hardware and PWM configurations, what users have to do with this software is:

- Modify some macros in PMSM_SpeedVectorCtrl.h to fit the new motor parameters and system characteristics if HVP platform is used and PWM is still 8 KHz.
- Modify some macros in PMSM_SpeedVectorCtrl.h to fit the new motor parameters and system characteristics as well as PE configurations of eFlexPWM module if HVP platform is used but PWM is no longer 8 KHz.

- Modify some macros in PMSM_SpeedVectorCtrl.h and PE configurations as well as current sampling related functions in MC_statemachine.c if other platform is used and I/O assignments are different from HVP platform.

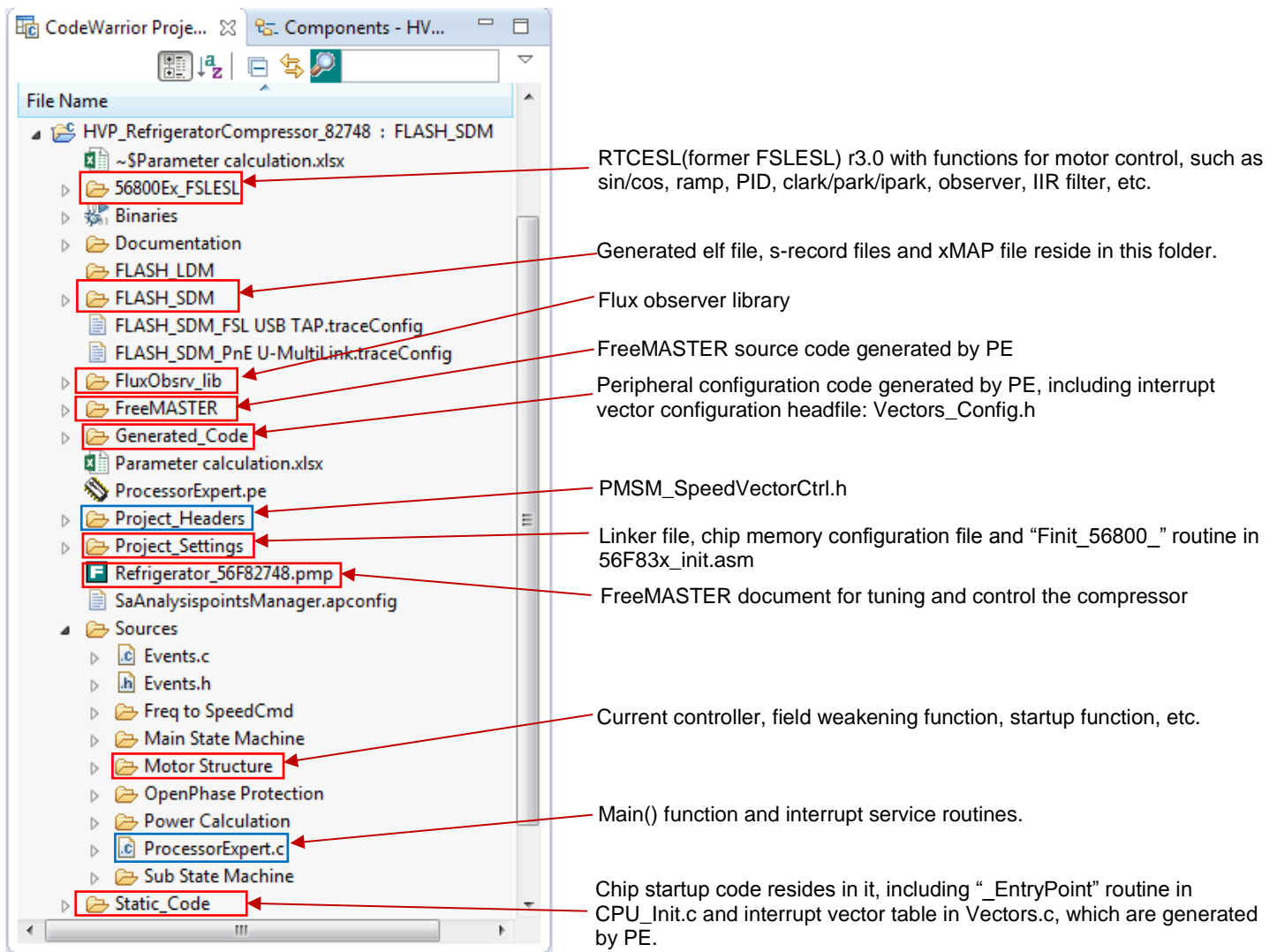


Figure 14. Project structure view

7.2. How to adapt to a new compressor

All the motor and control related constants are in "PMSM_SpeedVectCtrl.h" head file. There are two parts in this file: Manually input macros and Auto-generated macros. Auto-generated macros are copied from sheet2 of an excel file "Parameter calculation.xlsx" which can be found in the project. These macros are mainly motor parameters and observers related. Manually input macros are mainly protection and performance related.

If the motor parameters are changed, users must change them accordingly in the excel file. All the cells in red background in sheet1 are meant to be input by users manually, while the other cells in the "Value" column are generated accordingly. *Figure 15* shows the motor parameters in the excel file.

Rs	6	ohm	Phase resistance
Ld	0.076	H	Phase inductance of d-axis
Lq	0.118	H	Phase inductance of q-axis
Pn	3	Pairs	Pole pairs
Rotor_Flux	0.14066	Wb	Rotor flux

Figure 15. Motor parameters in excel file

Five motor parameters are essential to this application: phase (line-to-neutral) resistance, phase inductances of d and q axes, pole-pairs and rotor flux. The accuracy of rotor flux affects the startup performance. If users wish to use this HVP platform to test their own compressors, it's possible that only these five parameters need to be changed in this excel file and users don't have to change other parameters, such as the bandwidth of current and observer controllers. All parameters that need to be entered by users are noted in the excel file. After the update of the excel file, copy sheet2 to head file "PMSM_SpeedVectCtrl.h" and replace the macros listed below "Auto-generated Macros".

7.3. Startup performance tuning

Alignment and open-loop startup parameters affect the performance of startup. These parameters are defined as macros in the "Manually input Macros" part. Here are the alignment related macros.

```
#define MC_DURATION_TASK_ALIGN    2.0    /* [s]; Duration of alignment */
/* Bootstrap capacitor charging duty and duration */
#define BOOTSTRAP_CHARGE_DUTY    0.95    /* This duty is used to charge bootstrap capacitors
before alignment */
#define MC_BOOTSTRAP_CHARGE      0.1     /* [s];Duration of charging bootstrap, must be less
than alignment time */
#define MC_ALIGN_CURRENT         1.5     /* [A]; Alignment current */
#define MC_ALIGN_SPEED           0.0     /* [RPM]; Alignment speed in RPM */
#define VOLTAGE_ALIGN_MAX        170.0  /* [Volt]; Output limit of current controllers during
alignment */
#define MC_ALIGN_CURRENT_RAMP    1.5     /* [A/S] */
```

There's a bootstrap capacitor charging stage at the beginning of alignment, the lasting time is MC_BOOTSTRAP_CHARGE, and the duty for three phases is BOOTSTRAP_CHARGE_DUTY. The charging duty (applied on the top transistors) can't be very small or else there could be large charging current that leads to false over current. The total alignment duration is defined by MC_DURATION_TASK_ALIGN, during which the current rises from 0 A to MC_ALIGN_CURRENT with an acceleration of MC_ALIGN_CURRENT_RAMP. There must be a ramp for the alignment current, or there could be abrupt vibration when the alignment starts under the circumstances that the pressure difference is high. For the configuration above, the current rises from 0 to 1.5 A in 1 s and lasts another 1 s with 1.5 A. Set MC_ALIGN_SPEED to zero for this application.

VOLTAGE_ALIGN_MAX is the maximum voltage that can be allowed to output in d-q frame. In general, users may need to modify alignment duration, alignment current value and ramp to tune the performance.

Open-loop startup related macros are listed as below.

```
#define MC_START_UP_SPEED_MAX          300.0 /* [RPM]; Desired final speed in open-loop
start up */

#define MC_START_UP_SPEED_RAMP         200.0 /* [RPM/s]; Start-up speed ramp */

#define MC_START_UP_CURRENT_RAMP       12.0  /* [A/s]; Start-up current ramp */

#define MC_START_UP_CURRENT_PULL_OUT   0.30  /* [A]; Speed starts to increase when current
reaches this value */

#define MC_START_UP_CURRENT_MAX_ATTEMPT 2.5   /* [A]; Start-up current max value, when last
time start up is fail; */

#define MC_START_UP_CURRENT_MAX        1.5   /* [A]; Start-up current max value, when last
time start up is successful; */

/* Using estimated position while speed open loop running parameters */

#define SPEED_MAX_POSITION_CLOOP        1000.0 /* [RPM]; Transition speed in this phase.
Speed closed loop is enabled when it reaches this speed */

#define MAX_DURATION_POSITION_CLOOP     0.35  /* [s]; Start up is deemed as failure if real
speed doesn't reach SPEED_MAX_POSITION_CLOOP in this period of time */

#define SPEED_RAMP_INIT                 100.0  /* [RPM]; Speed ramp output when first
switched to speed closed loop */
```

In open-loop, when Iq reference is larger than MC_START_UP_CURRENT_PULL_OUT, the predicted speed starts from 0RPM towards a target speed of the negative value of MC_START_UP_SPEED_MAX with a ramp of MC_START_UP_SPEED_RAMP. Meanwhile, Iq reference increases towards MC_START_UP_CURRENT_MAX with a ramp of MC_START_UP_CURRENT_RAMP. Since Iq reference is already a large value during alignment, motor will start up immediately the moment startup state is entered. If startup fails, the next startup will use a Iq reference value of MC_START_UP_CURRENT_MAX_ATTEMPT.

When motor has rotated for 180° electrical degrees, it'll enter spin state where estimated position is used and speed is not under control as described in section 5.3. The startup is deemed fail if the estimated speed doesn't reach SPEED_MAX_POSITION_CLOOP in a time period of MAX_DURATION_POSITION_CLOOP. Once the estimated speed is over SPEED_MAX_POSITION_CLOOP in time, speed closed-loop is enabled as described in [Section 5.4, "Spin with estimated position \(flux observer\) – speed closed-loop"](#) and the speed ramp reference is set to SPEED_RAMP_INIT to reduce Iq reference so that there's not much speed overshoot. For different compressors, the value of SPEED_RAMP_INIT and SPEED_MAX_POSITION_CLOOP may need to be adjusted for reliable startup. Increasing the value of SPEED_RAMP_INIT may enhance the capability of startup with high pressure difference, but there could be large speed overshoot.

7.4. Protection settings

Protection settings are also defined as macros in the “Manually input Macros” part. The controller doesn't respond to any speed command for couple of minutes if any fault occurs.

```

#define MC_PRESSURE_RELAX_DURATION           0.05 /* [min]; default waiting time
before next start up */

#define MC_PRESSURE_RELAX_DURATION_RUNTOFAULT 6.0 /* [min]; waiting time before
next start up when fault occurs */

#define MC_PRESSURE_RELAX_DURATION_ATTEMPT_STARTUP 15.0 /* [s]; waiting time before next
startup when last startup fails (Not fault); */

#define MC_START_UP_ATTEMPTS                 3 /* Number of start-up attempts
before it goes to startup failure(STALL) fault */

```

When any fault occurs and then disappears, the controller won't respond to any valid speed command for `MC_PRESSURE_RELAX_DURATION_RUNTOFAULT` minutes in order to decrease the pressure difference before next startup. If the fault doesn't disappear, such as under-voltage, the state machine will be stuck in fault state, and the compressor won't start up unless all faults are gone and a period of `MC_PRESSURE_RELAX_DURATION_RUNTOFAULT` has passed by.

For normal operation, the time interval between startups is actually determined by the system control part, the compressor control part just simply responds to the speed command. Even though, a minimum time interval of `MC_PRESSURE_RELAX_DURATION` is added in the compressor control part to prevent starting up under high pressure difference. In this case, it's 0.05 min namely, 3 s. So from the time the compressor is stopped, the controller won't respond to any valid speed command for 3s.

7.4.1. Stall fault

When startup fails, it will try to start up again, and it takes continuous startup failures of `MC_START_UP_ATTEMPTS` times to trigger stall fault. The time interval between two attempting startups is `MC_PRESSURE_RELAX_DURATION_ATTEMPT_STARTUP`, during which the controller doesn't respond to any valid speed command.

7.4.2. DC bus over-voltage and under-voltage protection

```

/* Over voltage protection threshold */
#define MC_OVERVOLT_LIMIT                   390.0 /* [Volt] */

/* Under voltage protection threshold */
#define MC_UNDERVOLT_LIMIT                  180.0 /* [Volt] */
#define UNDER_VOLT_DELAY_PROTECT_TIME      0.125 /* [s]; */

```

PWM is disabled once the detected DC bus voltage is larger than `MC_OVERVOLT_LIMIT`. PWM is also disabled when the detected DC bus voltage is smaller than `MC_UNDERVOLT_LIMIT` for a continuous time of `UNDER_VOLT_DELAY_PROTECT_TIME`.

7.4.3. Over current protection by software

```

/* Software over current threshold */
#define SOFTWARE_OC_TH_SHRK          3.0    /* [A]; PWM is disabled when current length
reaches this value */
#define I_LENGTH_MA32_NUMBER        4

```

When the state comes to spin, the actually current vector length is calculated and filtered through a moving average filter with filtered points of $2^I_LENGTH_MA32_NUMBER$. In this case, the filtered points are 16. A fault is triggered if the filtered current vector length is larger than `SOFTWARE_OC_TH_SHRK`.

7.4.4. Over load protection

```

/* Over load protection */
#define OVER_LOAD_DELAY_PROTECT_TIME 0.005 /* [s]; */
#define OVER_LOAD_PROTECT_BOUNDARY_SPEED 1800.0 /* [RPM]; */
#define OVER_LOAD_PROTECT_MIN_SPEED 600.0 /* [RPM]; */

```

After the first stage of lubrication, and when the speed command is smaller than `OVER_LOAD_PROTECT_BOUNDARY_SPEED`, if the estimated speed is smaller than `OVER_LOAD_PROTECT_MIN_SPEED` for an accumulative time of `OVER_LOAD_DELAY_PROTECT_TIME`, over-load fault will be triggered.

7.4.5. Open-phase protection

```

/* Open phase protection; */

#define OP_CURRENT_THRESHOLD          0.1          /* [A]; Motor is assumed to be open
phase if phase current is less than this value for a period of time */

#define OP_DETECT_PERIOD              0.4          /* [s]; A time period to detect open
phase */

#define OP_PROTECT_DURATION_TH       0.3          /* [s]; A time period threshold */

#define OP_STARTUP_ATTEMPT_CNTR_MAX  5           /* Stop motor permanently after
continuous start-up attempts */

#define OP_CURRENT_VECT_SPD_FILT_N    4           /* [n/a]; Current vector speed MA
filter number. */

#define OP_ALIGN_DET_THRESHOLD       0.3          /* [A]; Open-phase detection begins
when alignment current request is larger than this value */

#define OP_SPEED_DIFF_ENTER_SPIN     4000.0       /* [RPM]; Threshold of speed difference
between current vector and ramp request when just entering SPIN */

#define OP_SPEED_DIFF_IN_SPIN        700.0       /* [RPM]; Threshold of speed difference
between current vector and ramp request after a period of time */

#define OP_DURATION_ENTER_SPIN       2.0         /* [S]; during the time from just
entering spin to this value, speed error threshold is larger */

#define OP_CURR_ERROR_THRESHOLD       0.5         /* [A]; Id & Iq error threshold. */

#define OP_SPEED_AND_CURR_DIFF_CNT_MAX 5         /* [n/a]; Open Phase flag is set when
speed error between ramp and current vector, Id error, Iq error meet the Open Phase criteria
for this number of times */

```

Open-phase is detected when the below two conditions are met:

- (a). Sampled current of one or more phases is very small in a period of time.
- (b). Rotation speed of the current vector deviates a lot from the reference speed of the motor, or the errors of Id or Iq are too large.

A counter is used to count a time period of `OP_DETECT_PERIOD` in the fast loop periodically, the current values of three phases are monitored within each period. If the current value of any phase is below `OP_CURRENT_THRESHOLD`, and lasts accumulatively over a time duration of `OP_PROTECT_DURATION_TH` in one period, a flag is set. In this case, detection period is 0.4 s, take phase A as an example, if the absolute value of I_a is less than 0.1 A for over an accumulative time of 0.3 s within this 0.4s period, a flag is set and condition (a) is met. The same rule applies to other phases.

During the stage of alignment and startup, since the reference current is always the same pattern which does not change in accordance with the loading, open-phase can be affirmed if any phase current is too small (namely, condition (a) is met). The open-phase detection is enabled only when reference current is larger than `OP_ALIGN_DET_THRESHOLD` during alignment.

In the state of spin, the moment speed controller is enabled, since the estimated speed will be generally larger than the reference speed, the reference q-axis current drops rapidly. It might go below `OP_CURRENT_THRESHOLD` if the loading is light (e.g. the ambient temperature is very low). So it's not enough to use condition (a) alone as the criteria of open-phase, condition (b) must be also met. When

Refrigerator Compressor BLDC Sensorless Control Based on MC56F82xxx, Application Note, Rev. 0, 12/2016

motor runs normally during spin state, the speed of current vector is almost the same with motor real speed, but when open-phase occurs, the current vector doesn't rotate continuously under the influence of current controllers, so the estimated speed of current vector deviates from the speed reference, and the current errors of d-axis and q-axis may increase. A moving average filter is used for the estimated speed of current vector, the filtered points are $2^{OP_CURRENT_VECT_SPD_FILT_N}$, so in this case, it is 16 points. Since a vector tracking observer is used to estimate the speed of current vector, and this observer is only enabled in spin state, it requires some time to converge the moment spin state is entered. During a time period of `OP_DURATION_ENTER_SPIN` after switching to spin state, condition (b) is met if the speed difference between current vector and speed reference is larger than `OP_SPEED_DIFF_ENTER_SPIN`. Then `OP_SPEED_DIFF_IN_SPIN` is used for this comparison. Condition (b) is also met if d-axis or q-axis current error is larger than `OP_CURR_ERROR_THRESHOLD`.

Finally, for the reliable detection during spin state, open-phase is affirmed only when condition (a) is met and condition (b) is met for `OP_SPEED_AND_CURR_DIFF_CNT_MAX` times.

If the system detects open-phase fault accumulatively for over `OP_STARTUP_ATTEMPT_CNTR_MAX` times, the state machine will be stuck in fault state and it requires reboot to start over.

7.4.6. LED indications for various faults:

There's a `USER_LED` on HVP platform (D20) for indicating the fault that just occurs. This LED will blink in a certain pattern in the time of `MC_PRESSURE_RELAX_DURATION_RUNTOFAULT` when compressor doesn't respond to any speed command. Take under-voltage fault for instance, the settings are as follow:

```
#define BLINK_TIMES_UV          5      /* [n/a]; Led blinking times */
#define BLINK_DURATION_UV      0.5    /* [S]; Led on/off duration during
blinking */
#define PATTERN_SPACE_UV      2.0    /* [S]; Time Duration between patterns */
```

It means a blink is composed of an ON and an OFF state, and each state lasts for 0.5 s. When under-voltage occurs, the LED will blink for 5 times then stay off for 2 s, and then blink for 5 times, stay off for 2 s., on and on, until `MC_PRESSURE_RELAX_DURATION_RUNTOFAULT` is over. Other faults have different blink pattern.

7.5. Other settings

7.5.1. Speed command and stop mode:

In real application, speed command typically comes from a frequency modulated square waveform with a duty of 50%. 40 Hz~150 Hz corresponds to 1200 RPM~4500 RPM. QTimer1 in 56F82xxx is configured to capture and recognize this frequency. Since on HVP platform, there's no such interface, speed command comes from FreeMASTER interface, but the frequency-to-speed code is maintained.

```

/* Speed command source control */
#define CMD_FROM_CONTROL_BOARD      0      /* 1:Enable  0:Disable */
/* Stop mode enable control */
#define STOP_MODE                    0      /* 1:Enable stop mode when speed cmd is zero
0: Disable; */
/* Power down mode */
#define DURATION_DELAY_ENTER_STOP    3.0    /* [s]; Duration before entering stop when
speed command is zero */

```

Clear `CMD_FROM_CONTROL_BOARD` to fetch speed command from FreeMASTER, or the speed command comes from the frequency on pin GPIOC6. Clear this macro when using HVP platform.

Setting `STOP_MODE` enables chip stop mode when speed command is zero. 56F82xxx will enter stop mode after a time of `DURATION_DELAY_ENTER_STOP` when speed command becomes zero. Clear this macro when using HVP platform or else FreeMASTER will lose connection when speed command is zero.

7.5.2. DC bus voltage affection during system power up:

```

/* DC bus voltage threshold at power on */
#define MC_POWERONVOLT_TH           250.0 /* [Volt]; State machine isn't invoked unless DC bus
voltage is above this value during system power on */

```

When system is powered up, the state machine is not enabled and FreeMASTER doesn't work until DC bus voltage is larger than `MC_POWERONVOLT_TH`. Carefully set this value, because refrigerator requires working under certain low voltage.

7.5.3. Bandwidth of rotor position tracking observer:

```

/* Change bandwidth of TO dynamically */
#define DYN_TO_PARA                  0

```

When `DYN_TO_PARA` is 1, the bandwidth of the tracking observer for rotor position changes dynamically from 30 Hz to 100 Hz as speed command goes from 2500 RPM to 3500 RPM. For most cases, this bandwidth doesn't have to change. A value 0 for this macro is fine. If dynamic bandwidth is required, beware that the constants used in the calculation is hard coded and it's targeted for motor with 3 pole-pairs, just take the code as an example:

```

#if DYN_TO_PARA == 1

if((gsMC_Drive.sSpeed.f16SpeedRamp >= FRAC16(2500.0/N_MAX)) &&
(gsMC_Drive.sSpeed.f16SpeedRamp <= FRAC16(3500.0/N_MAX))) // 30~100Hz
{
    gsMC_Drive.sObserverDQ.sTo.f16PropGain =
sub(mult(FRAC16(0.595),gsMC_Drive.sSpeed.f16SpeedRamp),FRAC16(0.2465))<<2;
    gsMC_Drive.sObserverDQ.sTo.f16IntegGain =
sub(mult(FRAC16(0.571775),gsMC_Drive.sSpeed.f16SpeedRamp),FRAC16(0.27457645))<<2;
}
#endif

```

7.5.4. Field weakening parameters:

Field weakening is realized with NXP patented algorithm, which generates Id reference automatically through a PI controller based on q-axis voltage and current request and limits.

```

/* Field-weakening parameters */

#define MC_SPEED_FW_ON                2000.0 /* [RPM]; Speed when the field weakening
is allowed */

// Field-weakening PI controller

#define MC_PI_FW_P_GAIN                0.05
#define MC_PI_FW_I_GAIN                0.03
#define MC_PI_FW_P_GAIN_SHIFT         0
#define MC_PI_FW_I_GAIN_SHIFT         0
#define MC_PI_FW_OUTPUT_LIMIT         1.0 /* [A]; Maximal d-axis current */

```

Field weakening function is enabled when estimated speed is above MC_SPEED_FW_ON, which means Id reference is set to constant zero when estimated speed is below MC_SPEED_FW_ON and Id reference is controlled by a PI controller when estimated speed is over MC_SPEED_FW_ON. The upper limit of this controller is zero, and the lower limit is the negative value of MC_PI_FW_OUTPUT_LIMIT. The proportional gain is $MC_PI_FW_P_GAIN \cdot (2^{MC_PI_FW_P_GAIN_SHIFT})$, and the integral gain is $MC_PI_FW_I_GAIN \cdot (2^{MC_PI_FW_I_GAIN_SHIFT})$. The PI gain values listed above can be reused for most cases.

7.5.5. Speed controller parameters:

Speed is controlled through a PI controller, which is realized in slow loop.

```

/* Speed controller parameters;Parallel type */
#define MC_PI_SPEED_P_GAIN          0.3    /* Speed controller proportional gain */
#define MC_PI_SPEED_I_GAIN          0.17   /* Speed controller integral gain */
#define MC_PI_SPEED_P_GAIN_SHIFT    1      /* Speed controller proportional gain shift */
#define MC_PI_SPEED_I_GAIN_SHIFT    -6     /* Speed controller integral gain shift */
#define MC_PI_SPEED_OUTPUT_LIMIT    1.5    /* [A]; Speed controller output limit in Amps,
used in initialisation only */
#define MC_I_MAX                     1.5    /* [A]; Max current used for flux weakening */
#define MC_SPEED_LOOP_LIMIT_AFTER_STARTUP 3.6 /* [A]; The ASR output limit after
startup(actually, it's after observers' merge) */

```

Proportional gain is $MC_PI_SPEED_P_GAIN * (2^{MC_PI_SPEED_P_GAIN_SHIFT})$, and integral gain is $MC_PI_SPEED_I_GAIN * (2^{MC_PI_SPEED_I_GAIN_SHIFT})$. The output limit is $MC_PI_SPEED_OUTPUT_LIMIT$ in initialization. When speed controller is enabled in spin state, the output limit is set to MC_I_MAX when flux observer is used. In this case, the rotor position and speed are switched to state observer shortly after speed controller is enabled, then the output limit of which is set to $MC_SPEED_LOOP_LIMIT_AFTER_STARTUP$.

7.5.6. Speed ramp and lubrication parameters:

Speed acceleration and lubrication settings in spin state is as follow.

```

#define SPEED_UP_RAMP_ASR          300.0 /* [RPM/s]; Speed ramp for speed up */
#define SPEED_DOWN_RAMP_ASR        300.0 /* [RPM/s]; Speed ramp for speed down */
#define SPEED_UP_RAMP_OIL_PUMP_ASR 3700.0 /* [RPM/s]; Speed ramp for speed up during oil
pumping */
#define SPEED_DOWN_RAMP_OIL_PUMP_ASR 3700.0 /* [RPM/s]; Speed ramp for speed down during
oil pumping */
#define SPEED_UP_RAMP_STOP_ASR      1000.0 /* [RPM/s]; Speed ramp for stopping the motor
*/
#define SPEED_DOWN_RAMP_STOP_ASR    1000.0 /* [RPM/s]; Speed ramp for stopping the motor
*/
#define SPEED_STOP_THRHL           2100.0 /* [RPM/s]; Stop motor when speed ramp is
below this value when speed command is zero */
#define STOP_DELAY_TIME            3.0 /* [s]; Delay this time to stop motor when
speed reaches to SPEED_STOP_THRHL */
/* Piston lubrication during start up */
#define LUBRICATION_SPEED_COMMAND_1 1500.0 /* [RPM]; Speed for lubrication of piston
during start up. */
#define LUBRICATION_SPEED_COMMAND_DURATION_1 12.0 /* [s]; Time duration of running at
lubrication speed. */
#define LUBRICATION_SPEED_COMMAND_2 2760.0 /* [RPM]; Speed for lubrication of
piston during start up. */
#define LUBRICATION_SPEED_COMMAND_DURATION_2 120.0 /* [s]; Time duration of running at
lubrication speed. */

```

When speed controller is just enabled, speed command is set to LUBRICATION_SPEED_COMMAND_1 for a time of LUBRICATION_SPEED_COMMAND_DURATION_1 s., and the speed ramp is set to SPEED_UP_RAMP_OIL_PUMP_ASR and SPEED_DOWN_RAMP_OIL_PUMP_ASR, which is the first stage of lubrication. In this case, the compressor runs at 1500RPM for 12s after startup to pump oil up for the purpose of lubrication. Then in the next time period of LUBRICATION_SPEED_COMMAND_DURATION_2, depending on the speed command, the compressor runs at LUBRICATION_SPEED_COMMAND_2 if the command is larger than LUBRICATION_SPEED_COMMAND_2 or runs at speed command if the command is smaller than LUBRICATION_SPEED_COMMAND_2, during which speed ramp is changed to SPEED_UP_RAMP_ASR and SPEED_DOWN_RAMP_ASR. The compressor runs at the speed command when lubrication stages are over.

When speed command is zero, and the motor runs at a speed less than SPEED_STOP_THRHL, the compressor is stopped immediately. But when speed command is zero and the motor runs at a speed larger than SPEED_STOP_THRHL, the speed will be decelerated to SPEED_STOP_THRHL at a ramp of SPEED_DOWN_RAMP_STOP_ASR, then the speed maintains at SPEED_STOP_THRHL for STOP_DELAY_TIME seconds before the motor is stopped.

7.6. Use FreeMASTER to control the compressor

HVP-MC3PH can be connected to PC through a virtual com port connection. The compressor can be controlled by FreeMASTER through this connection providing CMD_FROM_CONTROL_BOARD and STOP_MODE are all set to 0. Please refer to www.nxp.com/freemaster for more information on FreeMASTER.

Open Refrigerator_56F82748.pmp as shown in [Figure 14](#). Choose a baud rate of 19200 as shown in [Figure 16](#). Click scope “FOC Variables” under “User Interface” sub-block in Project Tree window. Now the Variable Watch window shows the variables for control and status.

In [Figure 18](#), gsMC_Ctrl.eState indicates the main state and meMC_StateRun indicates the sub state. faultRecordFlag.B indicates the fault that just occurs. Udc_Filt is the filtered DC bus voltage. gsMC_Drive.sSpeed.f16SpeedCmd is the speed command, users can input value between 1200 RPM~4500 RPM, and the compressor starts up. gsMC_Drive.sSpeed.f16Speed is the estimated speed. uw32CounterPressureRelax is a counter that indicates whether the compressor is ready for a valid speed command, a value of zero suggests the compressor is under the influence of speed command. gsMC_Drive.sPositionObsDQ.sStartUp.uw16AttemptCntr is a counter that records how many times the startup have failed continuously, three times will trigger stall fault as mentioned before.

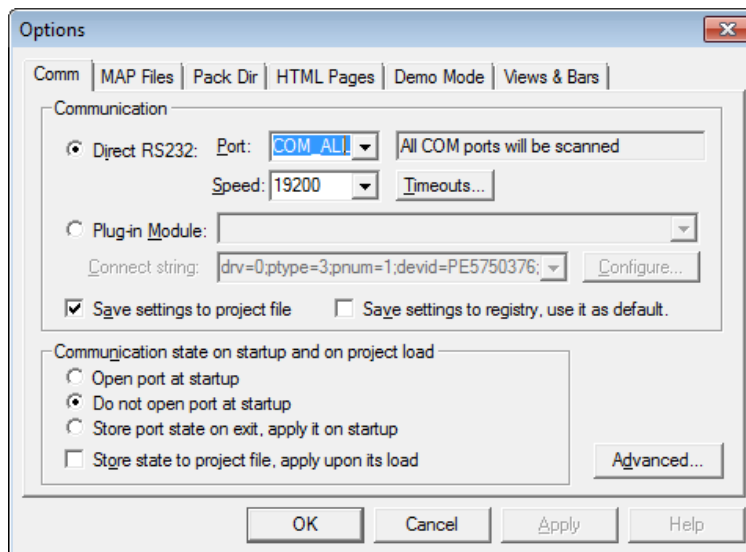


Figure 16. FreeMASTER connection setup

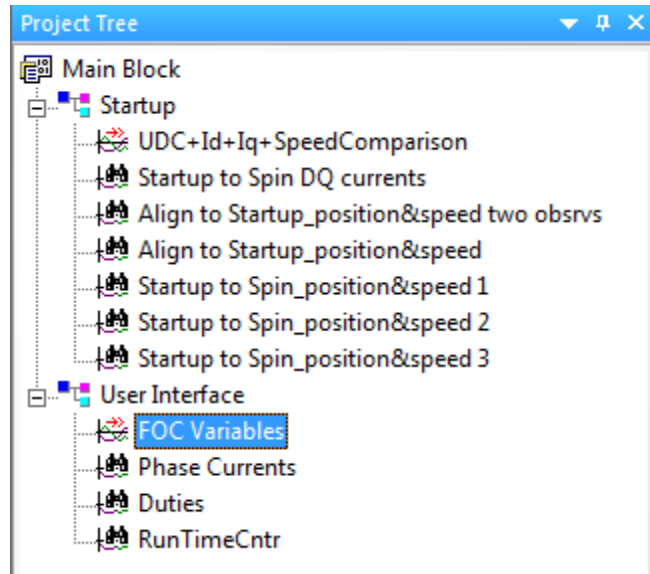


Figure 17. Project tree of FreeMASTER

Variable Watch				
Name	Value	Unit	Period	
gsMC_Ctrl.eState	Run	ENUM	1000	
meMC_StateRun	Spin	ENUM	1000	
faultRecordFlag.B	Normal	ENUM	1000	
Udc_Filt	319.253	Volt	1000	
gsMC_Drive.sSpeed.f16SpeedCmd	1999.97	RPM	1000	
gsMC_Drive.sSpeed.f16Speed	2008.67	RPM	1000	
uw32CounterPressureRelax	0	DEC	1000	
gsMC_Drive.sPositionObsDQ.sStartUp.uw16AttemptCntr	0	DEC	1000	

Figure 18. Variable watch window of FreeMASTER

8. Conclusion

The compressor control method mentioned above provides reliable startup capability and comprehensive protections. The startup has been tested on several refrigeration systems with different reciprocating compressors. *Figure 19* shows the startup current waveform of one phase, the motor has three pole-pairs.

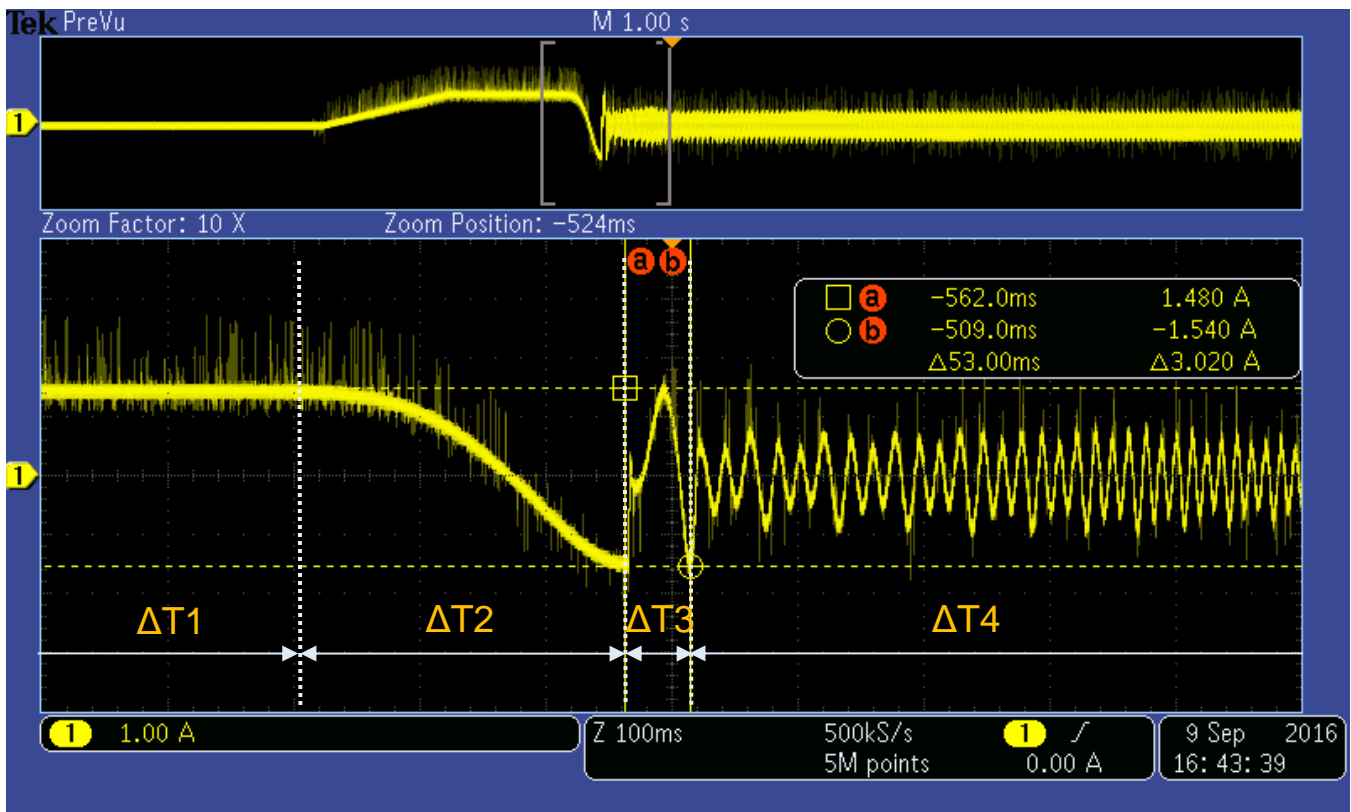


Figure 19. Startup current waveform of phase A

During time interval of $\Delta T1$, the motor runs at alignment. Then it goes to open-loop startup during $\Delta T2$ as described in [section 5.2](#). During $\Delta T3$, the motor runs at spin stage with speed open-loop as described in [section 5.3](#). And finally reaches to speed closed-loop in duration $\Delta T4$ as described in [section 5.4](#). Startup under low pressure and high pressure are shown in [Figure 20](#) and [Figure 21](#).



Figure 20. Startup under low pressure difference

Revision history

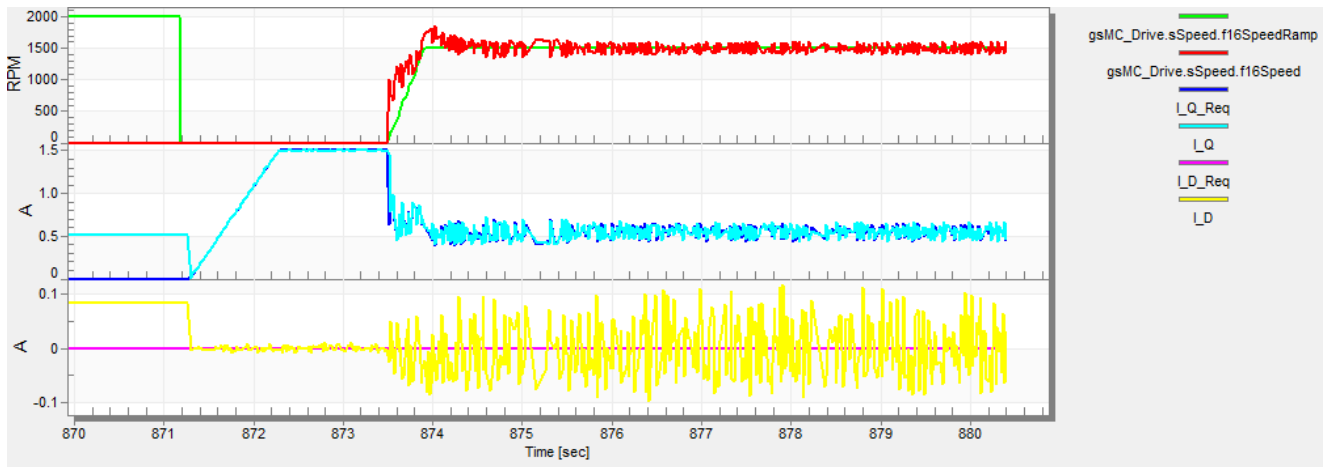


Figure 21. Startup under high pressure difference

9. Revision history

Revision number	Date	Substantive changes
0	12/2016	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address:
nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, and the Freescale logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners.

ARM, the ARM logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 NXP B.V.

Document Number: AN5387
Rev. 0
12/2016

