NXP Semiconductors
User Guide

Document Number: S12ZVMEFPSWUG
Rev. 0, 04/2020

S12ZVM-EFP RDB Software User Manual

by: NXP Semiconductors

1. Introduction

S12ZVM-EFP is designed for sensorless PMSM automotive fuel pump application. This user guide mainly focus on the software usage, including the software environment setup, software architecture introduction, motor control implementation and how to use the demo. The user guide also covers system test performance.

2. Before you start

Before you start, you need download and install the following tools first. You also should have basic knowledge about the tools, like how to use these tools setup a new project, how to configure and so on.

2.1. Codewarrior IDE introduction

CodeWarrior® for Microcontrollers v11.0 integrates the development tools for the ColdFire®, ColdFire+, DSC, Kinetis®, MPC5xxx, RS08, S08 and S12Z architectures into a single product based on the Eclipse open development platform. The modular installer provided with the tools, allows you to select and install only the architecture support you need for your application development.

Contents

NOTE

It is recommend to download the latest version from official site.(The latest version available currently is **CodeWarrior for MCU 11.1**)

2.2. FreeMASTER introduction

FreeMASTER is a user-friendly real-time debug monitor and data visualization tool, which enables runtime configuration and tuning of embedded software applications. Both automotive OEMs and industry-leading appliance manufacturers widely adopted FreeMASTER as it is suitable for use in a broad range of applications.

FreeMASTER supports non-intrusive monitoring of variables on a running system and can display multiple variables on oscilloscope, like displays as standard widgets (gauges, sliders, and more) or as data in text form, offering simple-to-use data recorders. FreeMASTER can link with custom HTML, MATLAB®, or migrate it to other scriptable frameworks (using Excel) to add MCU hardware into control loops. Connection to the target system from a host running FreeMASTER may be made directly over a broad range of communication peripherals or debug channels. FreeMASTER 3.0 embeds graphs, tabular grids, and web views directly in the desktop application. FreeMASTER connections are made via a network connection using JSON RPC calls, with client implementations available for Python, C/C++/C#, and other languages.

FreeMASTER 3.0 offers a new component, FreeMASTER Lite. It is a lightweight service leveraging the JSON RPC protocol that can run on Windows or Linux host PC and enables the implementation of custom UI applications on a web browser application (running on a local or remote host computer or mobile device).

Features:

- Real Time Data Monitoring
- Control Panels
- Host Communication Options
- Demonstration Platforms Integration

In order to debug the motor conveniently, a customized motor debugging GUI tool based on FreeMASTER, named **MCAT** (**M**otor **C**ontrol **A**pplication **T**uning Tool) is available, you can download it from the link.

2.3. Abbreviation

Abbreviation	Description
RD	**R**eference **D**esign
HW	Hardware

SW	Software
SDK	**S**oftware **D**evelopment **K**it
BSP	**B**oard **S**upport **P**ackage
LLD	**L**ow-**L**evel **D**river
API	**A**pplication **I**nterface
POR	**P**ower-**O**n **R**eset
BLDC	**B**rushless **D**irect **C**urrent
PMSM	**P**ermanent-**M**agnet **S**ynchronous **M**otor
FOC	**F**ield-**O**riented **C**ontrol
IPD	**I**nitial **P**osition **D**etection
RD	**R**eference **D**esign
HW	Hardware
SW	Software
SDK	**S**oftware **D**evelopment **K**it
BSP	**B**oard **S**upport **P**ackage
LLD	**L**ow-**L**evel **D**river
API	**A**pplication **I**nterface
POR	**P**ower-**O**n **R**eset
BLDC	**B**rushless **D**irect **C**urrent
PMSM	**P**ermanent-**M**agnet **S**ynchronous **M**otor
FOC	**F**ield-**O**riented **C**ontrol
IPD	**I**nitial **P**osition **D**etection

3. System Features

Most of automotive Electrical Fuel Pumps (EFP) are based on DC motor, but BLDC based EFPs have become more and more popular, especially in high-end car brands for both diesel pump and gasoline pump. NXP has developed the S12ZVM-EFP RD to meet this growing market.

The software package has the following features:

- Out-of-box motor control and tuning via FreeMASTER MCAT.
- Support PMSM sensorless FOC control, both dual shunt and single shunt.
- Dedicated optimized for fuel pump application to achieve robust and fast start up. It meets the strict start up condition < 150 ms from standstill to the rated speed.
- Smooth cross-over I-F start up and Initial position detection algorithm make sure the startup successful.
- Support multiple diagnose and protection covering UV, OV, OT, OC, Short, Stall Detection, etc.
- Not only support fuel pump, but also for other automotive PMSM applications.

4. Software Architecture Overview

The S12ZVM-EFP RD software package is developed on NXP **MTRCKTSPNZVM128 Software V1.3** and **AMMCLIB 1.1.15**.

The software architecture is shown in *AMMCLIB components structure*. The bottom layer is the hardware and the second layer is the peripheral low level driver and middleware, which includes AMMCLIB, FreeMASTER, LIN and PWM. The top layer is “User APP” and “Motor Control” module.

The S12ZVM has no SDK drivers for peripherals. There are two solutions for S12ZVM peripherals drivers, one is using PE (Processor Expert), another is using the peripheral example code. The S12ZVM-EFP uses peripheral example code as low level drivers as it is more efficient.

CPMU module is used to configure the system clock. ADC module is used to sample motor phase currents, bus voltage and silicon temperature. PMF is used to generate six channel PWM signals and

PTU is used to trigger the ADC in specific PMF point. These low level modules are the basic modules for PMSM control. GDU is the gate driver unit and it can amplify the PMF signals to turn the MOSFETs on or off, it can also achieve the charge pump management, fault mechanism configuration and etc.

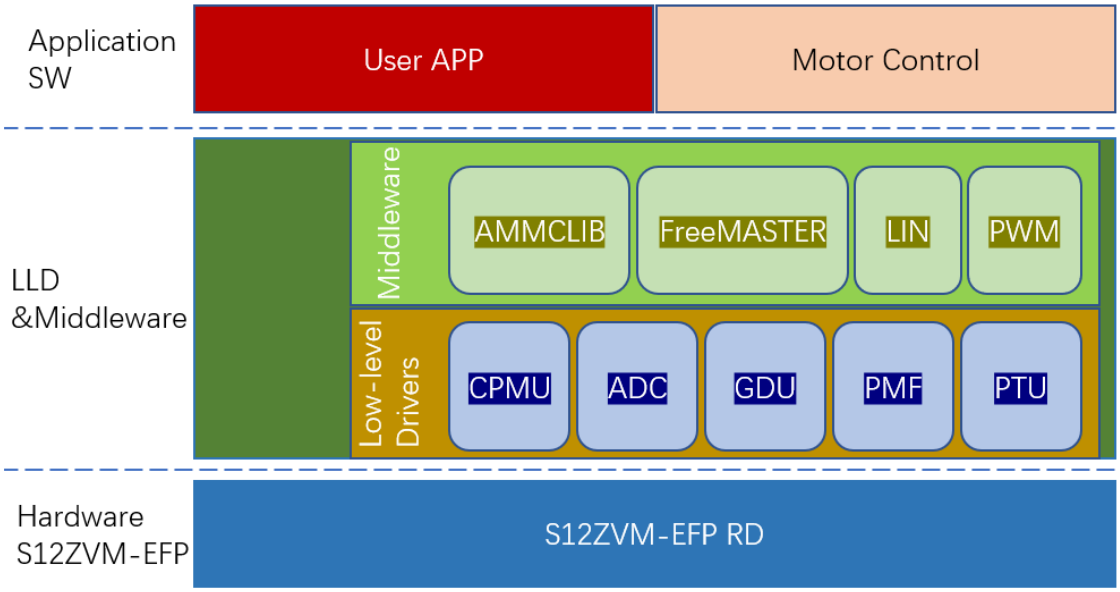

Figure 1. **The RD software architecture**

4.1. S12ZVM peripheral driver introduction

4.1.1. CPMU

For proper operation the CPMU needs to have a stable power supply. The power supply is stable when the GDUF_GLVLSF is cleared. The application uses Internal Reference which provides a 1 MHz internal clock (CPMUOSC_OSCE = 1). Out of the 1 MHz internal clock bus and the core clock is derived by the following settings:

```
CPMUREFDIV_REFDIV = CPMU_REFDIV;       // CPMU_REFDIV=0
CPMUREFDIV_REFFRQ = CPMU_REFFRQ;       // CPMU_REFFRQ=0
CPMUSYNR_SYNDIV = CPMU_SYNDIV;         // CPMU_SYNDIV=49
CPMUSYNR_VCOFRQ = CPMU_VCOFRQ;         // CPMU_VCOFRQ=3
CPMUPOSTDIV_POSTDIV = CPMU_POSTDIV;    // CPMU_POSTDIV=0
```

Set the bus and core clock to 50 MHz and 100 MHz respectively. The SW needs to wait until the PLL lock (CPMUIFLG_LOCK set to 1).

The CPMU module setting provide also possibility to enable the High Temperature Sensor which is routed to ADC internal channel.

4.1.2. PMF

The Pulse Width Modulator with Fault Protection (PMF) module is configured to generate a center-aligned (PMFCFG0_EDGEx = 0) PWM with a frequency of 20 kHz (PMFMODA = 2500). In order to protect the MOSFET devices in the same leg of the inverter, deadtime is set to approximately 0.25 us

(PMFDTMA = 25). PWM generator A runs as the master and generates the Reload Signal as a synchronization signal for the other submodules (PMFCFG2_REV[0:1] = 1). For dual shunt, the reload signal is generated at every fourth PWM opportunity (PMFFQCA = 3). Pair A, Pair B and Pair C PWMs are synchronized to PWM generator A (PMFCFG0_MTG = 0). A PWM pulse width PMFVAL registers are double buffered and are swapped when GLDOK is set and the PWM reload signal occurs. The GLDOK is an external signal generated by the PTU module. The GLDOK is enabled at the PWM module (PMFENCA_GLDOKA = 1).

4.1.3. PTU

Programmable Trigger Unit (PTU) is intended to completely avoid CPU involvement in the time acquisitions of state variables during the control cycle. The PTU module consists of two trigger generators (TG). For each TG, a separate enable bit is available, so that both TGs can be enabled independently. Trigger generator zero is connected to ADC0, and trigger generator one is connected to ADC1. The trigger generation of the PTU module is synchronized to the incoming reload event. This reload event resets and restarts the internal time base counter and makes sure that the first trigger value from the actual trigger list is loaded. Furthermore, the corresponding ADC is informed that a new control cycle has started. If the counter value matches the current trigger value, then a trigger event is generated. In this way, the reload event is delayed by the number of bus clock cycles defined by the current trigger value. All acquisition time values are stored inside the global memory map, basically, inside the system memory as a three dimensional array of integers (PTUTriggerEventList). The exact location of the acquisition time values (PTUTriggerEventList) in the system memory is given by the linker command file and linked to the PTU module during the initialization phase.

```
PTUPTR = ptuTriggerEventList;
```

Each trigger generator uses only one list to load the trigger values from the memory. The pointers for the primary (TG0L0IDX/ TG1L0IDX) and alternate (TG0L1IDX/ TG1L1IDX) lists are equal.

```
TG0L1IDX = (unsigned char)(((long)&ptuTriggerEventList[0][0][0] -
(long)ptuTriggerEventList) >> 1); // same as TG0L0IDX
TG1L0IDX = (unsigned char)(((long)&ptuTriggerEventList[1][0][0] -
(long)ptuTriggerEventList) >> 1);
TG1L1IDX = (unsigned char)(((long)&ptuTriggerEventList[1][0][0] -
(long)ptuTriggerEventList) >> 1); // same as TG1L0IDX
```

The trigger generator is using only one physical list of trigger events, even if the trigger generator logic is switching between both pointers. The PTU module generates the LDOK signal used to inform other modules that the double buffered registers were updated by software.

4.1.4. GDU

The Gate Drive Unit (GDU) is a Field Effect Transistor (FET) pre-driver designed for three-phase motor control applications. The following GDU features are used in PMSM FOC sensorless control.

- Charge Pump: The charge pump is used to maintain the high-side driver gate source voltage VGS when PWM is running at a 100% duty cycle. The clock for the charge pump is set to be fbus/64 (GDUCLK2_GCPCD = 4) .
- Desaturation Error: The GDU integrates three desaturation comparators for the low-side FET pre-drivers and three desaturation comparators for the high-side FET pre-drivers. The

desaturation level is set to be 1.35 V (GDUDSLVL = 0x77) for both low-side and high-side FET. A blanking time during the FET transients needs to be employed. The blanking time is set to be approximately 8 us (GDUCTR = 0x13).

- Current Sense Amplifiers: Internal current sense amplifier 0 and 1 (GDUE_GCSE0 = 1 and GDUE_GCSE1 = 1) is used to measure motor phase currents in phase A and phase B. The output of the current sense amplifier 0 is routed internally to ADC0 channel 0. The output of the current sense amplifier 1 is routed internally to ADC1 channel 1.

4.1.5. ADC

The MC9S12ZVML128 uses two independent Analogue-to-Digital Converters (ADC). Both ADCs are n-channel multiplexed input successive approximation analogue-to-digital converters. The List Based Architecture (LBA) provides a flexible conversion sequence definition, as well as flexible oversampling. Both ADC conversion command lists are stored inside the global memory map, basically, inside the system memory as two dimensional arrays of bytes (ADC0CommandList, ADC1CommandList). The exact location of the ADC conversion commands in the system memory is given by the linker command file and linked to the respective ADC module during the initialization phase. The same strategy is used for the ADC Results. The Conversion results are stored in an array of shorts (ADC0ResultList, ADC1ResultList) located in system memory.

```
ADC0CBP = ADC0CommandList;  // ADC0 Command Base Pointer
ADC0RBP = ADC0ResultList;   // ADC0 Result Base Pointer

ADC1CBP = ADC1CommandList;  // ADC1 Command Base Pointer
ADC1RBP = ADC1ResultList;   // ADC1 Result Base Pointer
```

The ADC conversion clocks are set to be 8.33 MHz (ADC0TIM = 2; ADC1TIM = 2). The results are stored in memory as 12-bit (ADC0FMT_SRES = 4; ADC1FMT_SRES = 4) left-justified data (ADC0FMT_DJM = 0; ADC1FMT_DJM = 0).

Conversion flow of both ADCs is controlled by internal signals (generated by the PTU) and by the DataBus (ADC0CTL_0_ACC_CFG = 3; ADC1CTL_0_ACC_CFG = 3). The results are stored in system memory even if commutation occurs when conversion is ongoing (ADC0CTL_0_STR_SEQA = 1; ADC1CTL_0_STR_SEQA = 1).

The ADC1 schedules the end of list interrupt (ADC1CONIE_1_EOL_IE = 1) to perform application logic and calculate the PMSM FOC Sensorless algorithm.

The PMSM sensorless FOC algorithm uses ADC0 to measure the motor phase A current and DC-Bus voltage. The ADC1 is used to measure the motor phase B current and temperature.

4.2. Application data flow overview

The application software is interrupt driven running in real time. There is one periodic interrupt service routine associated with the ADC end of sequence interrupt, executing all motor control tasks. These include both fast current and slow speed loop control. All tasks are performed in an order described by the application state machine shown in *Figure 2. State Machine of S12ZVM-EFP* .

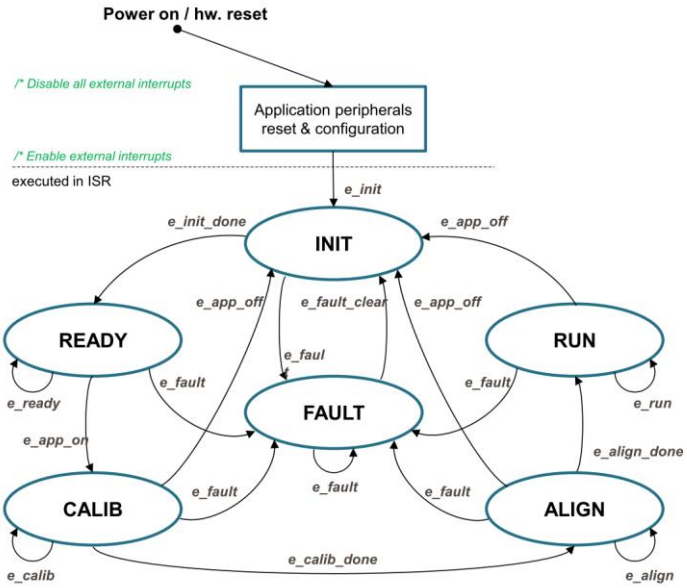

Figure 2. **State Machine of S12ZVM-EFP**

To achieve precise and deterministic sampling of analog quantities and to execute all necessary motor control calculations, the state machine functions are called within a periodic interrupt service routine. Hence in order to actually call state machine functions the periphery causing this periodic interrupt must be properly configured and interrupt enabled. As described in section MCS12ZVM Device initialization, all peripherals are initially configured and all interrupts are enabled after a RESET of the device. As soon as interrupts are enabled and all peripheries are correctly configured, the state machine functions are called from the ADC end of sequence interrupt service routine. The background loop handles non-critical timing tasks, such as the FreeMASTER communication polling.

For more details, refer to AN5135 (dual shunt FOC) and AN5327 (single shunt FOC).

The main application flowcharts and the key interrupt flowcharts are shown in *Figure 3. Application Flowcharts of Main* and *Figure 4. Key Interrupt Flowcharts*.

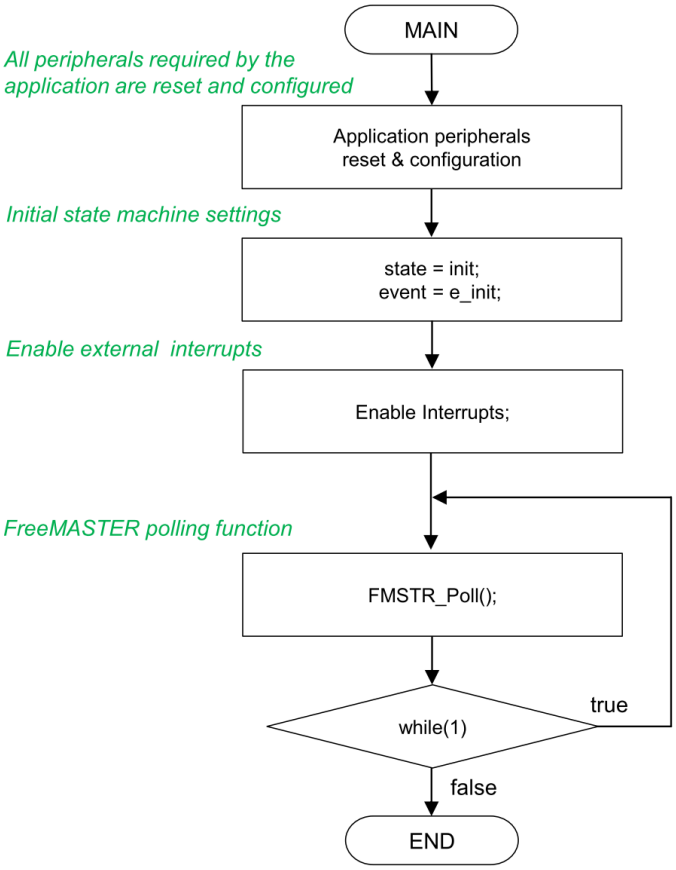

Figure 3. **Application flowcharts of main**

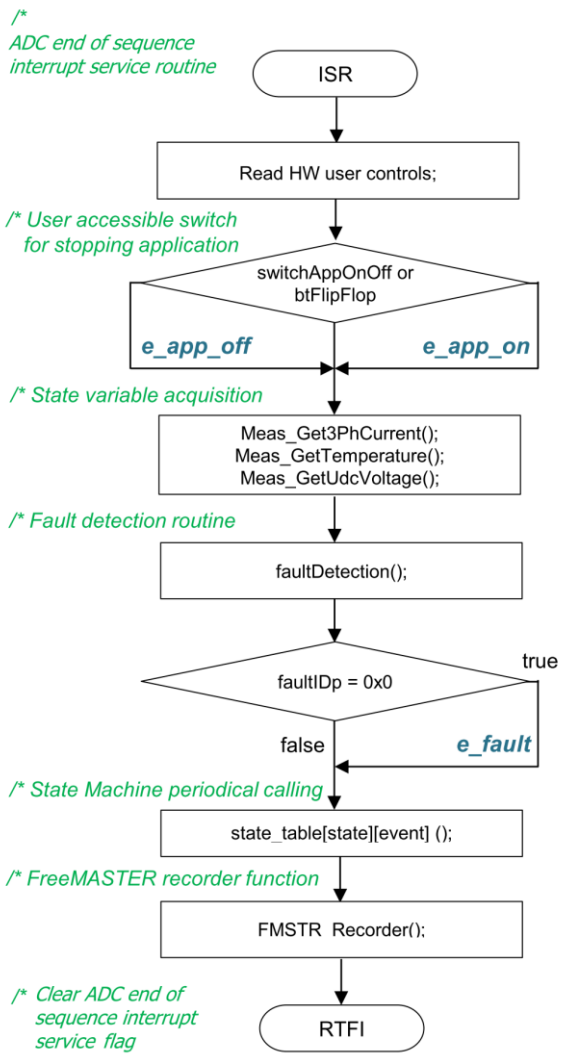

Figure 4. **Key interrupt flowcharts**

4.3. NXP AMMCLIB introduction

The AMMCLIB (Automotive Math and Motor Control Library) Set for NXP S12ZVMx devices consists of several sub-libraries, functionally connected as depicted in *AMMCLIB components structure*.

The Automotive Math and Motor Control Library Set for NXP S12ZVMx devices sub libraries are as follows:

- **Mathematical Function Library (MLIB):** comprising basic mathematical operations such as addition, multiplication, etc.
- **General Function Library (GFLIB):** comprising basic trigonometric and general math functions such as sine, cosine, tan, hysteresis, limit, etc.
- **General Digital Filters Library (GDFLIB):** comprising digital IIR and FIR filters designed to

be used in a motor control application.

- **General Motor Control Library (GMCLIB):** comprising standard algorithms used for motor control such as Clarke/Park transformations, Space Vector Modulation, etc.
- **Advanced Motor Control Function Library (AMCLIB):** comprising advanced algorithms used for motor control purposes.

The Automotive Math and Motor Control Library Set for NXP S12ZVMx devices is developed to support these major implementations:

- Fixed-point 32-bit fractional
- Fixed-point 16-bit fractional

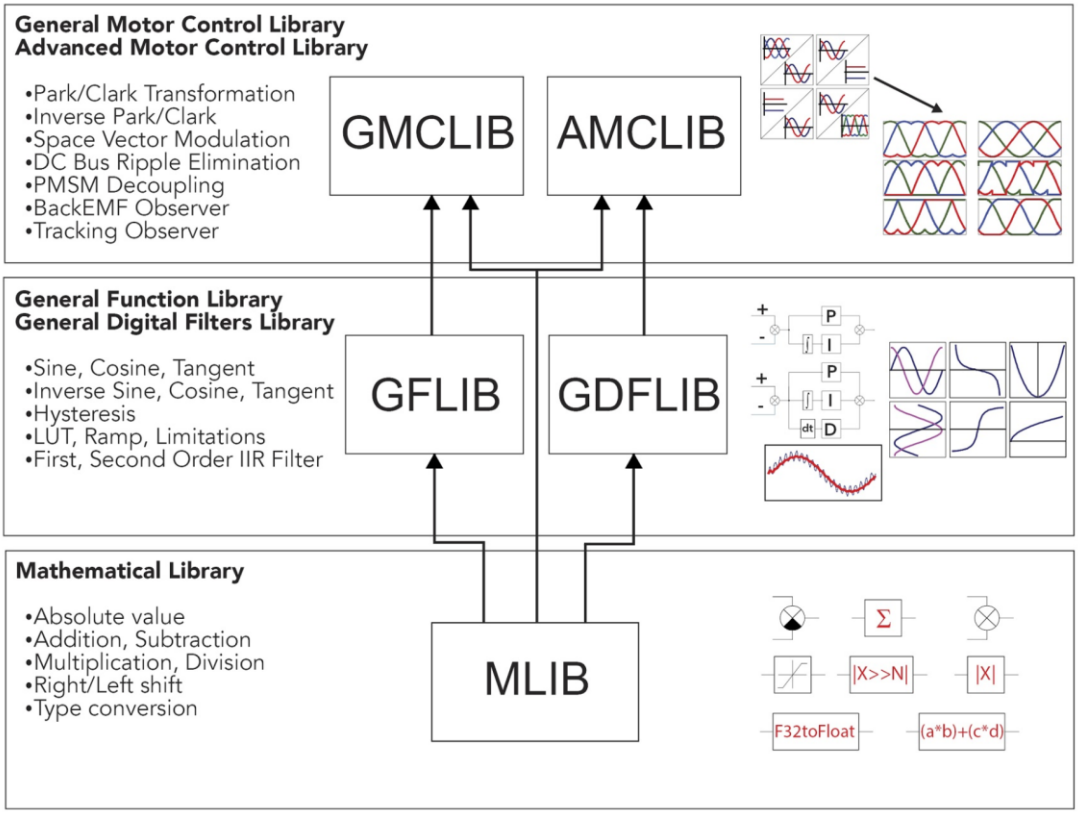

Figure 5. **AMMCLIB components structure**

More details about S12ZVM AMMCLIB refer to user guide **S12ZVMMCLUG Rev.20**.

4.3.1. Using AMMCLIB in CodeWarrior IDE

Actually, the AMMCLIB version 1.1.15 of S12ZVM is added in the S12ZVM-EFP RD example project. So no need to set the AMMCLIB path. But if you want to experience the latest version of AMMCLIB or create some new functions in the newer version of AMMLIB, you can add the path to your project and rebuild it.

First step is to set the include file path. Both of “Assembly Source File” and “GNU C”, just as shown in *Figure 6*. Second step is to set the library file “MC9S12ZVM_AMMCLIB.UC.a” path, as shown in *Figure 7*. If you just want to use version 1.1.15, you do not need do anything due to its default setting.

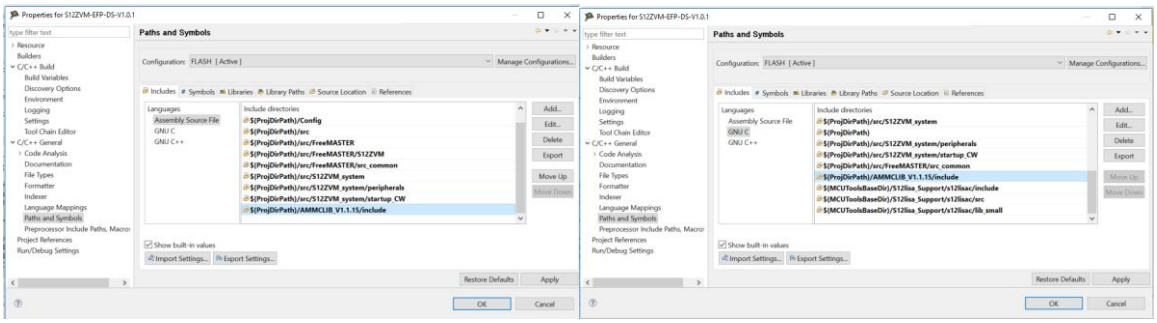

Figure 6. **Set AMMCLIB path in project**

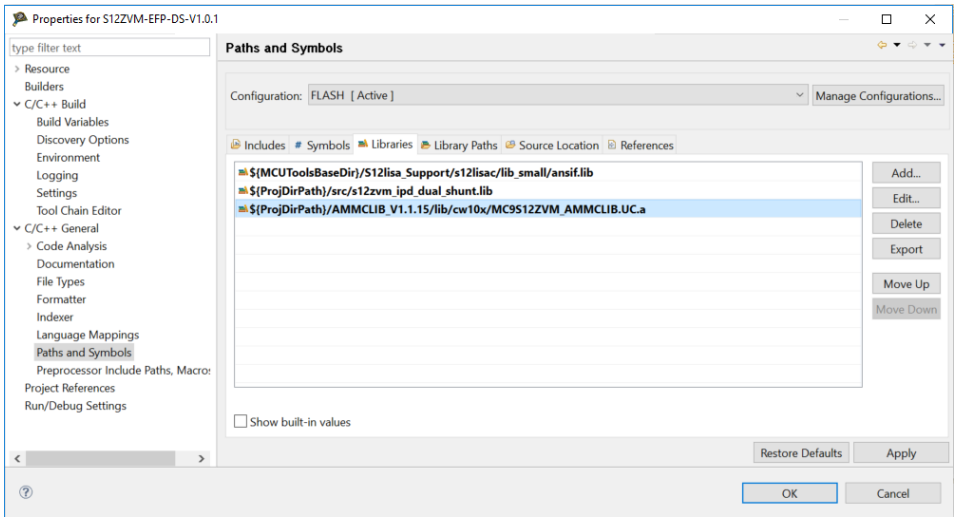

Figure 7. **Set AMMCLIB lib file path in project**

4.4. Project files tree and function briefing

The software project files tree is shown in the following figure.

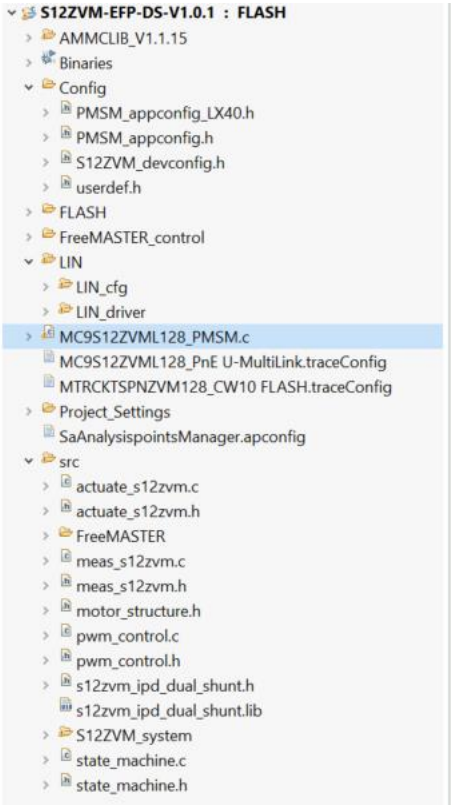

Figure 8. **Project files tree**

AMMCLIB_V1.1.15 is included in the project to avoid the compiling error if the path is not set correctly. In the “Config” folder, motor parameters header file configured by MCAT, device config and user macro define file are included.

LIN folder contains the NXP MagniV LIN stack. Low level driver (LLD) module is located in “S12ZVM_system” folder.

5. Motor Control Implementation

5.1. Fundamental principle of PMSM FOC

High-performance motor control is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/ deceleration. To achieve such control, Field Oriented Control is used for PM synchronous motors.

The FOC concept is based on an efficient torque control requirement, which is essential for achieving a high control dynamic. Analogous to standard DC machines, AC machines develop maximal torque

when the armature current vector is perpendicular to the flux linkage vector. Thus, if only the fundamental harmonic of stator magnetomotive force is considered, the torque T_e developed by an AC machine, in vector notation, is given by the following equation:

$$T_e = \frac{3}{2} \cdot pp \cdot \vec{\psi_s} \times \vec{i_s}$$

- **pp:** The number of motor pole-pairs
- $\vec{i_s}$**:** Stator current vector
- $\vec{\psi_s}$**:** Represents vector of the stator flux
- **3/2:** Indicates a non-power invariant transformation form

In instances of DC machines, the requirement to have the rotor flux vector perpendicular to the stator current vector is satisfied by the mechanical commutator. There is no such mechanical commutator in AC Permanent Magnet Synchronous Machines (PMSM), the functionality of the commutator has to be substituted electrically by enhanced current control. This reveal that stator current vector should be oriented in such a way that component necessary for magnetizing of the machine (flux component) is isolated from the torque producing component.

This can be accomplished by decomposing the current vector into two components projected in the reference frame, often called the dq frame that rotates synchronously with the rotor. It has become a standard to position the dq reference frame such that the d-axis is aligned with the position of the rotor flux vector, so that the current in the d-axis will alter the amplitude of the rotor flux linkage vector. The reference frame position must be updated so that the d-axis should be always aligned with the rotor flux axis.

The rotor flux axis is locked to the rotor position, when using PMSM machines. A mechanical position transducer or position observer can be utilized to measure the rotor position and the position of the rotor flux axis. When the reference frame phase is set such that the d-axis is aligned with the rotor flux axis, the current in the q-axis represents solely the torque producing current component.

What further resulted from setting the reference frame speed to be synchronous with the rotor flux axis speed is that both d and q axis current components are DC values. This implies utilization of simple current controllers to control the demanded torque and magnetizing flux of the machine, thus simplifying the control structure design.

To perform vector control, it is necessary to perform the following four steps:

1. Measure the motor quantities (DC link voltage and currents, rotor position/speed).
2. Transform measured currents into the two-phase orthogonal system (α, β) using a Clarke transformation. After that transform the currents in α, β coordinates into the d, q reference frame using a Park transformation.
3. The stator current torque (isq) and flux (isd) producing components are separately controlled in d, q rotating frame.
4. The output of the control is stator voltage space vector and it is transformed by an inverse Park transformation back from the d, q reference frame into the two-phase orthogonal system fixed with the stator. The output three-phase voltage is generated using a space vector modulation.

Clarke/Park transformations discussed above are part of the Automotive Math and Motor Control Library set (see section References).

The following two figures show the basic structure of the vector control algorithm for the PM synchronous motor.

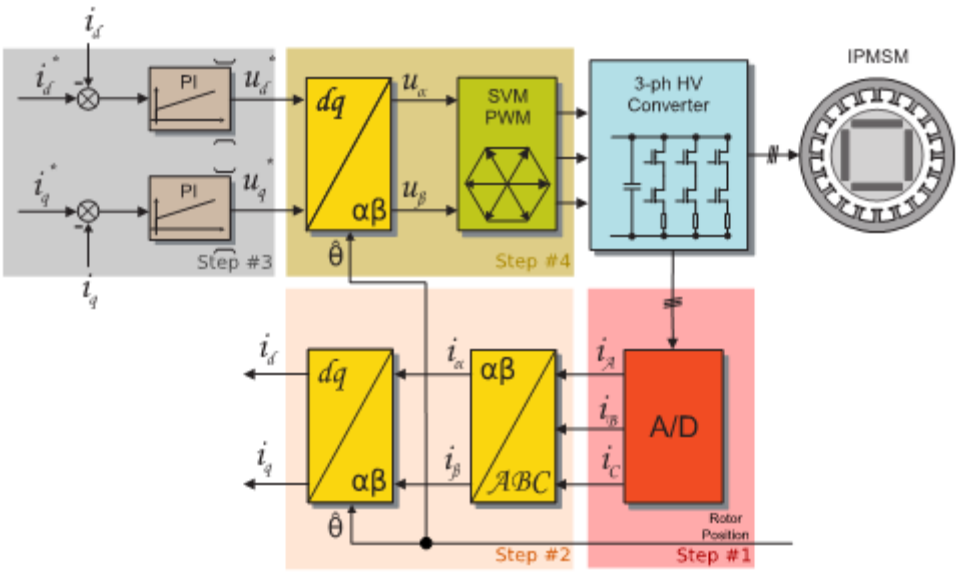

Figure 9. **Field oriented control structure**

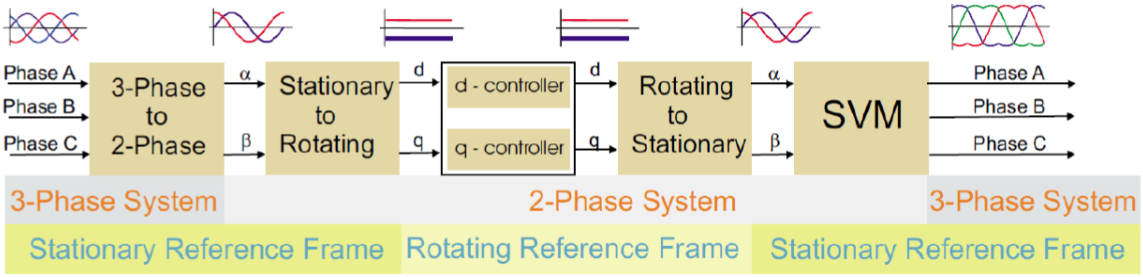

Figure 10. **Field oriented control transformations**

5.2. Sensorless control

To be able to decompose currents into torque and flux producing components (isd, isq), position of the motor-magnetizing flux has to be known. This requires knowledge of accurate rotor position as being strictly fixed with magnetic flux. This demo system deals with the sensorless FOC control where the position and velocity are obtained by either a position/velocity estimator or incremental Encoder sensor. The estimate method is using back-EMF observer, but back-EMF observer as well as incremental Encoder sensor provide only relative position. It is necessary to force alignment or initial position detection (IPD) algorithm for sensorless control system.

Alignment algorithm is the first stage of control system, the alignment algorithm applies DC voltage to d-axis resulting full DC voltage applied to phase A and negative half of the DC voltage applied to phase B and C for a certain period. This will cause the rotor to move to "align" position, where stator and rotor

fluxes are aligned. The rotor position in which the rotor stabilizes after applying DC voltage is set as zero position. Motor is ready to produce full startup torque once the rotor is properly aligned.

In the second stage, the field-oriented control is in open-loop mode (Application in sensorless mode must start with open loop), in order to move the motor up to a speed value where the observer provides sufficiently accurate speed and position estimations. As soon as the observer provides appropriate estimates, the rotor speed and position calculation are based on the estimation of a BEMF in the stationary reference frame using a Luenberger type of observer.

When the PMSM reaches a minimum operating speed, a minimum measurable level of BEMF is generated by the rotor's permanent magnets. The BEMF observer then transitions into the closed-loop mode. The feedback loops are then controlled by the estimated angle and estimated speed signals from the BEMF observer.

BEMF observer is as a part of the NXP's Automotive Math and Motor Control library. Following figure shows the BEMF structure.

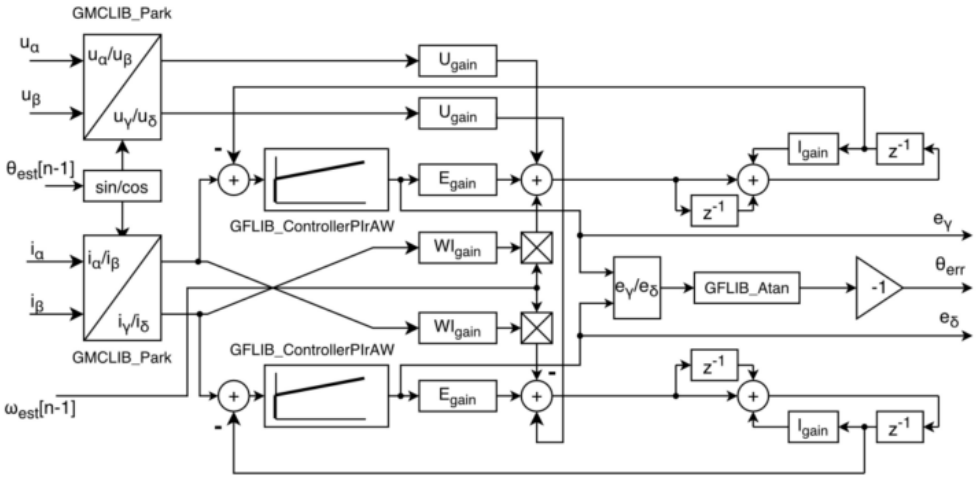

Figure 11. **BEMF observer structure**

5.3. Initial position detection implementation

If the salient polarity of a PMSM is obvious, the inductance and resistance of the PMSM will vary with the rotor position. In particular, if the reluctance of the winding changes in sinusoidal law, the inductance will also change in sinusoidal. As shown in *Figure 12*, the change period of inductance is twice than that of rotor. For example, when the voltage direction applied to the motor is V -> W, W -> U, U -> V, the inductance of the floating phase will be changed with the rotor position, so the voltage on that will also be changed. For example, when the phase voltage of U is equal to half of the bus voltage, the phase voltage of V is less than half of the bus voltage, and the W phase is greater than half of the bus voltage, it can be considered that the rotor is in the position of - 180 ° or 0 °, so that the position of the rotor is located within the range of 180 °.

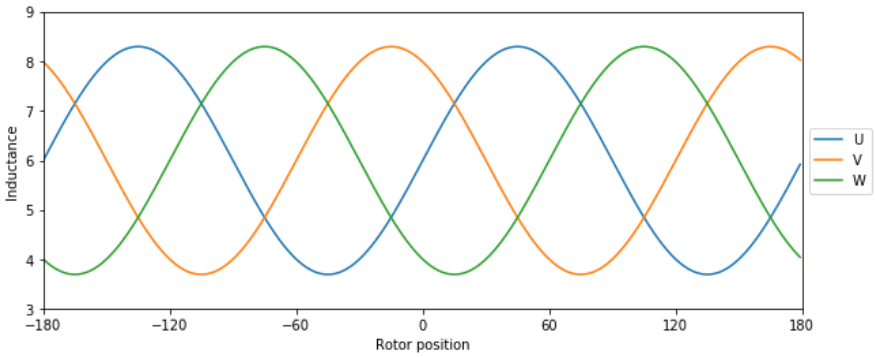

Figure 12. **Change of inductance according to rotor position**

In order to get the accurate rotor position, it is necessary to determine the direction of rotor N pole, which means to determine whether the rotor orientation is - 180 ° or 0 ° position. Generally speaking, for an inductance, the smaller the magnetic resistance is, the larger the inductance is, and they are inversely proportional. If it is an air-core inductor, then its magnetoresistance is certain, and the inductance is also a certain value. Therefore, an appropriate voltage vector can be applied to the motor to distinguish the polarity of the rotor by using the magnetic saturation effect. As shown in *Figure 13*, at the moment, the magnetic field generated by the rotor permanent magnet strengthens the magnetic field generated by the stator coil. Before reaching the magnetic saturation, the inductance of stator winding change rate is negative, and the magnetic resistance increases. If the rotor permanent magnet is rotated 180 °, the inductance change rate of the stator winding will be positive.

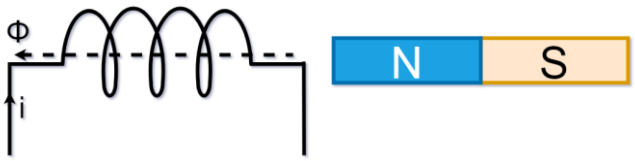

Figure 13. **Relation between magnet and inductor**

For the PMSM motor, the relationship between the rotor position and the three-phase winding is shown in *Figure 14*. The U-phase stator winding is closer to the d-axis of the rotor and the V-phase winding is closer to q-axis. Therefore, the magnetization effect of U and V phases is different, and the magnetization effect of U phase is higher than that of V phase. When a voltage vector is applied to the motor (U-phase connected to VDC, V-phase connected to GND), the inductance of phase U is less than that of phase V at the beginning, but the gap between them is gradually narrowing, and finally tends to be equal. That is to say, the direction of the N-pole of the rotor can be distinguished by the change law of the phase W voltage.

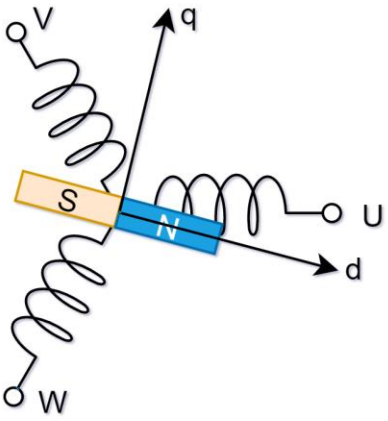

Figure 14. **Relation between the rotor and each phase**

5.4. Smooth Cross-Over I-F start-uP implementation

5.4.1. Principle

To improve alignment between current controllers and the reference frame, d-axis start-up with cross-over transition to q-axis is proposed. The start-up algorithm works with the current limit as required by the application. This limit is then passed to a cross-over block, which commands the Id and Iq currents with respect to the actual speed and open-loop to sensorless transition. Two set points are used: the first one is used to initiate transition of the current vector from the d-axis to the q-axis and the second set point is used as a hard switch to the sensorless closed-loop mode, as shown in the *Figure 15*.

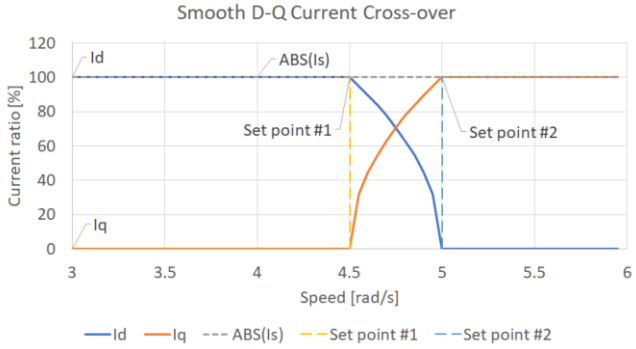

Figure 15. **Current Smooth Cross-over Transition Profile**

During the first phase (until the reference frame speed or frequency passes the first set point), the entire current limit Is is directed to the Idreq and the Iqreq is set to zero. Once

the expected rotor speed (or the stator frequency) passes the first set point, the current vector Is starts to move from d-axis towards q-axis by increasing the angle between the virtual frame and the current vector. This way, the virtual reference frame speed is de facto increased together with acceleration, thus more torque is put on the rotor and the torque angle is extended. Simulation result of this method is shown on *Figure 16*. The first set point speed is 4.5 rad/s, the second set point is 5 rad/s (the target speed). Currents in d and q axes are calculated as follows.

referenced to the virtual stator frame, which is originally aligned with the d-axis. Following equations 1 – 3 describe the cross-over calculation. The *weight* is changed between the set point #1 (ω_{set1}) and set point #2 (ω_{set2}) linearly, while limited to fit the interval of<0;1>. The required currents I_{dreq} and I_{qreq} are calculated to keep the I_s vector size constant.

$$weight = \frac{\omega_{start} - \omega_{set1}}{\omega_{set2} - \omega_{set1}}$$
$$I_d = |I_s|\sqrt{1 - weight}$$
$$I_q = |I_s|\sqrt{weight}$$

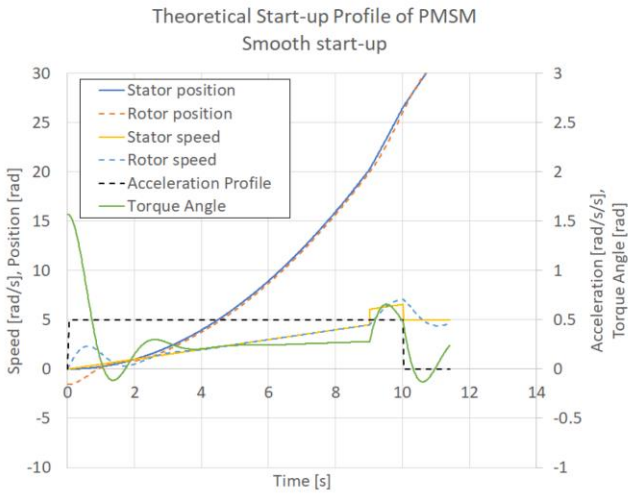

Figure 16. **Current smooth cross-over transition simulation**

5.4.2. Implementation

The implementation of this algorithm is not difficult. Firstly, the I_{dreq} and I_{qreq} are derived from speed loop calculation. So the algorithm is implemented in *focSlowLoop()* function.

In *focSlowLoop()*, the SpeedLoop is running all the time, but the output is not linked to I_{dreq} and I_{qreq} directly, it is linked to the temple variable *temIDQReq*.

The project uses "*switch, case*" to set the I_{dreq} and I_{qreq} value accordingly. Just as shown below.

```
switch(pos_mode)
{
  case force:
  current = drvFOC.pospeOpenLoop.iQUpperLimit;
  weight = 0;
  drvFOC.currentLoop.pIDQReq->f16Arg1 = current;
  drvFOC.currentLoop.pIDQReq->f16Arg2 = 0;
  drvFOC.speedLoop.pPIpAWQ.f16UpperLimit= drvFOC.pospeOpenLoop.iQUpperLimit;
  drvFOC.speedLoop.pPIpAWQ.f16LowerLimit=
MLIB_Neg_F16(drvFOC.speedLoop.pPIpAWQ.f16UpperLimit);
  break;

  case tracking:
```

```
  current = drvFOC.pospeOpenLoop.iQUpperLimit;
  weight=MLIB_SubSat_F16(MLIB_Abs_F16(drvFOC.pospeOpenLoop.wRotEl),drvFOC.pospeSensorless.wRo
tElMatch_1);
  weight=MLIB_DivSat_F16(weight,
MLIB_SubSat_F16(drvFOC.pospeSensorless.wRotElMatch_2,drvFOC.pospeSensorless.wRotElMatch_1));
  drvFOC.currentLoop.pIDQReq->f16Arg1=
MLIB_Mul_F16(GFLIB_Sqrt_F16(MLIB_SubSat_F16(FRAC16(1.0),weight)),current);
  drvFOC.currentLoop.pIDQReq->f16Arg2= MLIB_Mul_F16(GFLIB_Sqrt_F16(weight),current);

    break;
  case sensorless1:
    current = tempIDQReq.f16Arg2;
    weight = FRAC16(1.0);
    drvFOC.currentLoop.pIDQReq->f16Arg1 = 0;
    drvFOC.currentLoop.pIDQReq->f16Arg2 = current;
    drvFOC.speedLoop.pPIpAWQ.f16UpperLimit = drvFOC.pospeSensorless.iQUpperLimit;
    drvFOC.speedLoop.pPIpAWQ.f16LowerLimit = drvFOC.pospeSensorless.iQLowerLimit;
    break;
}
```

5.5. Stall detection implementation

In PMSM sensor-less application, motor stalls when the load become very large or rotor is stuck by something or the load changes dramatically. Usually it will trigger overcurrent protection, but sometime the motor phase current is not very big when motor is in stall condition. Meanwhile, sensor-less algorithm may still work, it can generate speed and angel regularly. This "fake running" should be detected to avoid the harm to system. So, the stall detection method should adopt to achieve the task.

5.5.1. Stall detection principle

There are several methods which can do stall detection, NXP uses BEMF consistency checking method to detect the stall case.

The BEMF of PMSM are linear with motor speed. BEMF of observer output should be consistent with motor KE multiply motor speed and plus the offset. The following equation shows the same.

$$E_q = K_e \cdot \omega + Ke_{offset}$$

- E_q: BEMF output of observer;
- K_e: BEMF coefficiency of the motor;
- ω: Motor speed;
- Ke_{offset} : BEMF offset;

So principle of the method is to check the consistency of two BEMFs. If observer output Eq is not linear with motor speed, it means observer is not working correctly and indicate the motor is in stall.

The following diagram can show the principle also.

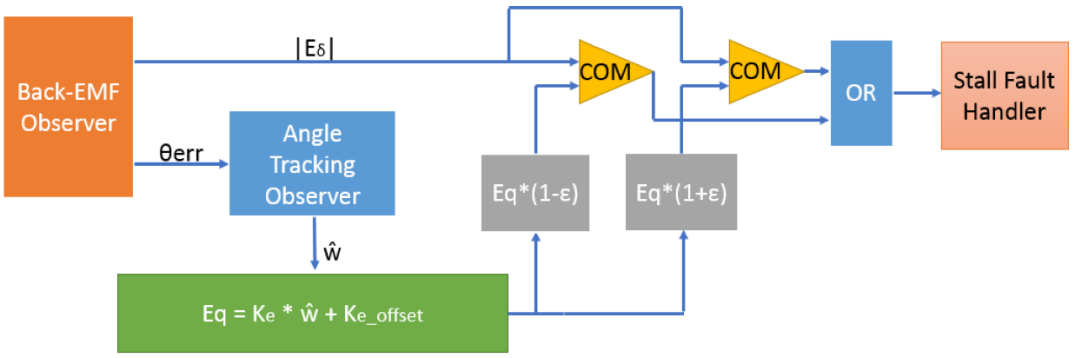

Figure 17. **Stall detect principle**

5.5.2. Implementation

The method is based on module design, so it is easy to implement in your own project.

- Copy the **stallDetection**.c and .h file in your project;
- Define the variable of structure, for example, stallDetection_T stallDetectionPsrams;
- Initial the stall detection function, **stallDetectionInit** (&stallDetectionPsrams);
- Add the stall detection function in **StateRun** function.
- if (TRUE == **stallDetection**(&stallDetectionPsrams))
  {
  permFaults.motor.B.StallError = 1;
  }

Stall detection parameters: The parameters should be correctly configuredy to make sure the stall detection function can work correctly.

There are some macros to configure the function. The following screenshot below shows all the macros.

```
#define  STALLDETECTION_BLANKCNT       20000
#define  STALLDETECTION_CHKCNT         30
#define  STALLDETECTION_CHKERRCNT
 STALLDETECTION_CHKCNT-5
#define  STALLDETECTION_COEFF          FRAC16(0.25)
#define  STALLDETECTION_COEFFKE        FRAC16(0.2655)
#define  STALLDETECTION_COEFFKEOFT     FRAC16(0.01)
#define  BEMFOBSFILTER_NSAMPLES        2
#define  ROTELFILTER_NSAMPLES          2
```

"**STALLDETECTION_BLANKCNT** " is used to set a blank time slot, in this slot, system will not do BEMF checking. The method is based on BEMF checking, so it is not applicable in startup stage.

"**STALLDETECTION_CHKCNT** " and "**STALLDETECTION_CHKERRCNT** " is used the check time and the error time. The allowable error check time is check time minus five. In the default setting,

check time is 30, if the error time is greater than 25, it will trigger the stall fault. You can change the checking time and the error checking time according to your application.

"**STALLDETECTION_COEFF**" is the threshold of allowable different range between observer Eq and calculated BEMF. In default setting, if the calculated BEMF is in range of 0.75*Eq and 1.25*Eq, it indicates the motor is not in stall status, but if the calculated BEMF is out of the range, meanwhile, the error checking time bigger than the setting, it will trigger the stall fault.

"**STALLDETECTION_COEFFKE**" and "**STALLDETECTION_COEFFKEOFT** " is the slope and the offset for the calculation equation. These parameters are very important and need manual offline calculation. Different type of motors may have different parameters.

Take 45ZWN24-90-B for example, if Eq = 1.725 V @1000rpm and 3.2 V@2000rpm.

Then there are two equations FRAC16(1000/4500) *a + b = FRAC16(1.725/25)， that is 0.2222*a + b =0.069 and FRAC16(2000/4500) *a + b = FRAC16(3.2/25)， that is 0.4444*a + b =0.098. After the calculation, a = FRAC16(0.2655) and b = FRAC16(0.01).

"BEMFOBSFILTER_NSAMPLES" is BEMF observer output MA filter smoothing factor. The range is from 0 to 15. Smaller, filter less. so 0 means no filter influence.

"ROTELFILTER_NSAMPLES" is the speed ω MA Filters. smoothing factor. Smaller, fitler less. So 0 means no filter influence.

5.5.3. Stall detection summary and enable

The BEMF consistency checking method is based on NXP patent **US20170126153A1**. The developer can check the original patent for more information.

In the demo project, this function disabled in default, if you want to enable it, just change the macro: **STALL_DETECTION** value from **STD_OFF** to **STD_ON** in “userdef.h”.

5.6. Current sampling method

5.6.1. Overview

There are three current sampling methods which uses shunts in inverter legs as current sensors, tri-shunt, dual-shunt and single-shunt. S12ZVM-EFP supports dual-shunt and single-shunt.

There are two demo projects, one is for dual-shunt and another one is for single-shunt.

5.6.2. Dual-shunt current sampling

Dual-shunt current sampling is the most popular method due to the best performance vs cost. S12ZVM microcontrollers have 2 OPAMP and 2 ADC modules and it's perfect to get two current sampling at one shot. The topology is shown as following.

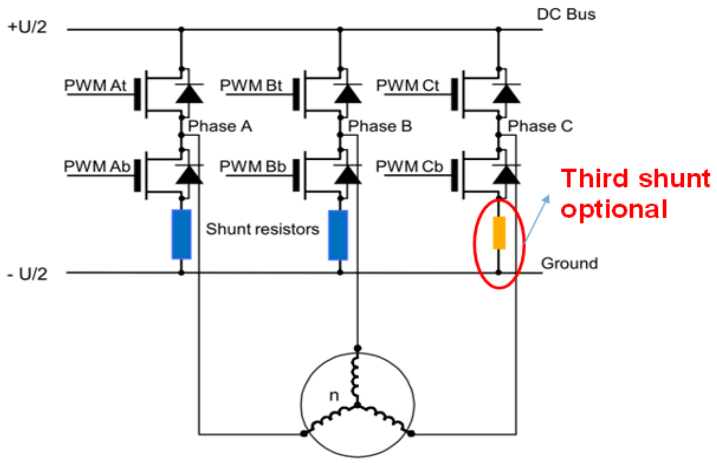

Figure 18. **Shunt resistors topology**

When all the bottom MOSFETs are ON, the motor phase current continues to flow due to the inductance effect. In this period, the voltage of shunts indicates the motor phase current which flow through the shunts.

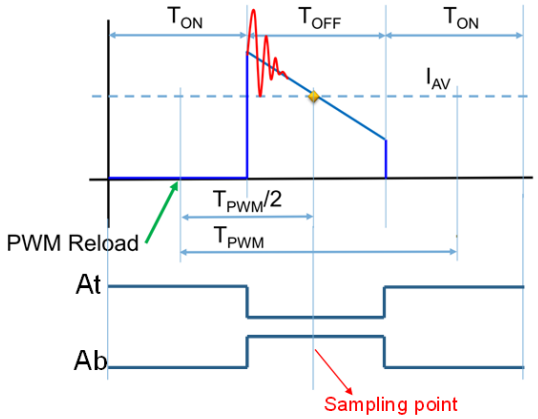

Figure 19. **Sampling theory diagram**

For dual-shunt sampling, the sampling point is no need change, the method is much easy to achieve. The phase currents obtained in the same time, so reconstruct phase current THD is low.

But if SVPWM waveform is shown as following phase II, there is a concept "minimal pulse width" which rely on hardware design. If the available duty is too short, the current sampling for Phase A would be bad quality. To avoid this case, the duty cycle limitation should be applied. 0.9 is the default value, for higher quality hardware, it can be set higher, for instance, 0.95 or more.

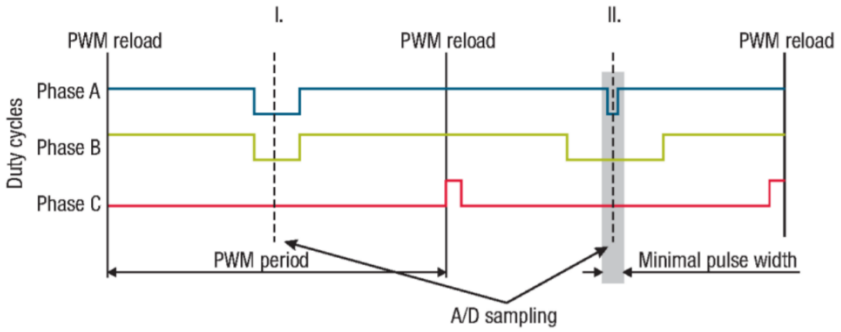

Figure 20. **PWM diagram when sampling**

5.6.3. Single-shunt current sampling

Single-shunt current sampling is low cost solution, but it is enough for most of the low dynamic application. It uses phase current reconstruction to capture the phase currents in different time slot.

Phase current reconstruction: The phase current sampling technique is a critical issue for detection of phase current differences and for acquiring full three phase information of stator current by its reconstruction. Phase current flowing through a shunt resistor produces a voltage drop which needs to be appropriately sampled by the AD converter when the DC bus voltage is connected to the motor, thus in six of eight (non-zero) voltage vectors (see the following figure).

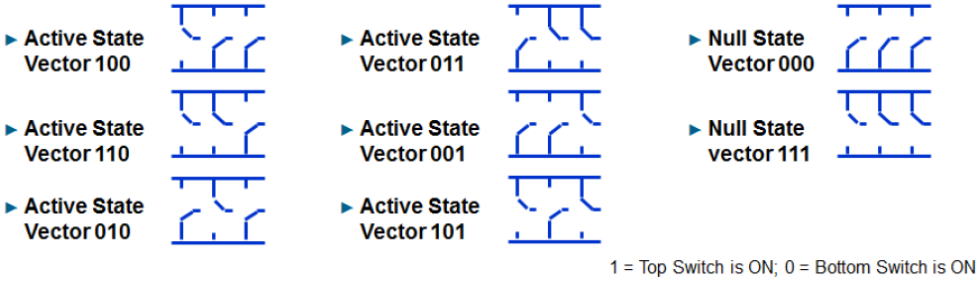

Figure 21. **Voltage vector states in terms of transistor states**

Figure 22 shows an example of a current measurement during vector 101, in which i_{SB} sample can be taken. Considering a symmetrical 3-phase system, Kirchhoff law can be used at any time, thus

$$i_{SA} + i_{SB} + i_{SC} = 0$$

Based on the above equation, at least two currents in a single PWM period are needed to have all the three currents available for the vector control. Thanks to the modulation of the voltage vector, two different combinations of non-zero voltage vectors are available during a single PWM period. Thus, two currents can be sensed as shown in Figure 23. The third current is then calculated based on Kirchhoff law.

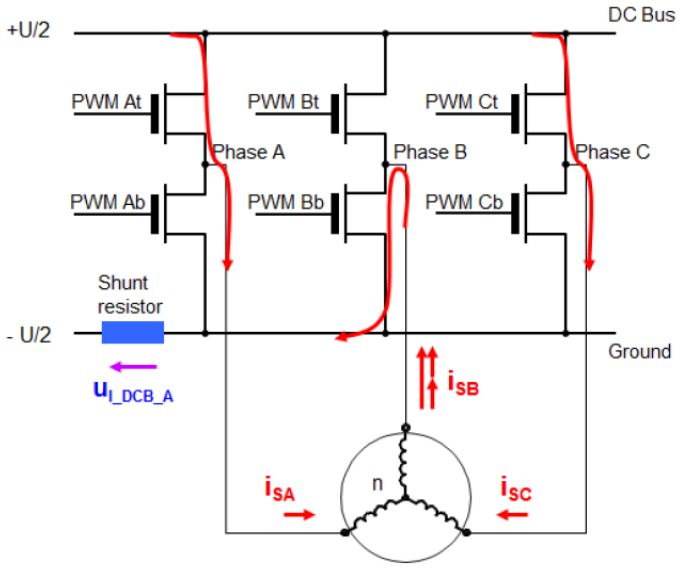

Figure 22. **Single-shunt three-phase current reconstruction**

Single-shunt 3-phase current reconstruction is available only if two voltage vectors are active for a sufficient time period to capture the current. As two PWM edges come close to each other, phase current signal pulse on the DC-link becomes too short to be captured or “disappears” at all, as shown in Figure 24. This makes that portion of 3-phase current information invisible for sensing circuit and can eventually disturb the phase current feedback. If all three phases come close enough, no phase current information can be recovered from the DC-link current sensor.

There are two main techniques to make the 3-phase current reconstruction available at any time. The first one, so called “phase shifting PWM” shifts one of the overlapping phases from another to make the DC-link current visible for a sufficient period of time. This method is described e.g. in Design reference manual DRM102 available at www.nxp.com.

Another option is to split one of the overlapped signals in two parts, thus insert a zero pulse in the middle of the pulse (see Figure 25, blue signal on the left, blue and brown signals on the right), so called “double switching PWM”. Instead of shifting one signal from another, one of the overlapped signals is split in two symmetrical signals and these two parts are shifted apart (Figure 25 on the left, the blue signal) so the total length of the signal is the same (in comparison to phase A).

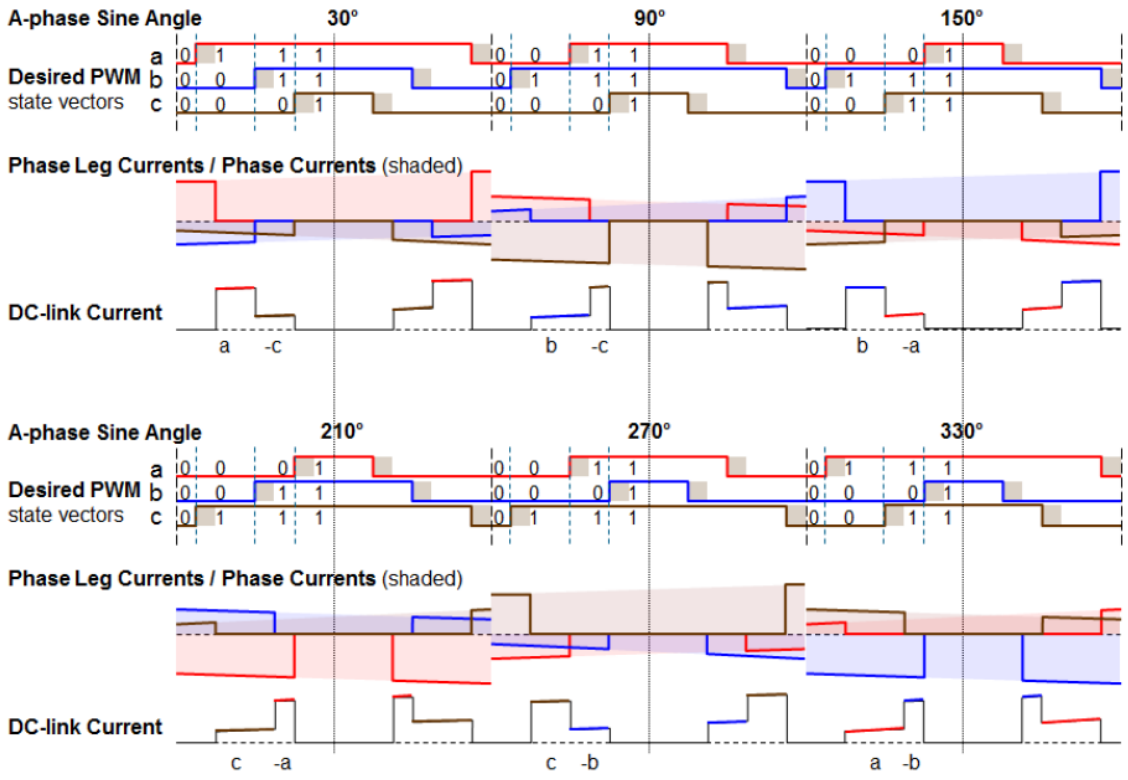

Figure 23. **DC-link current and phase currents connection**

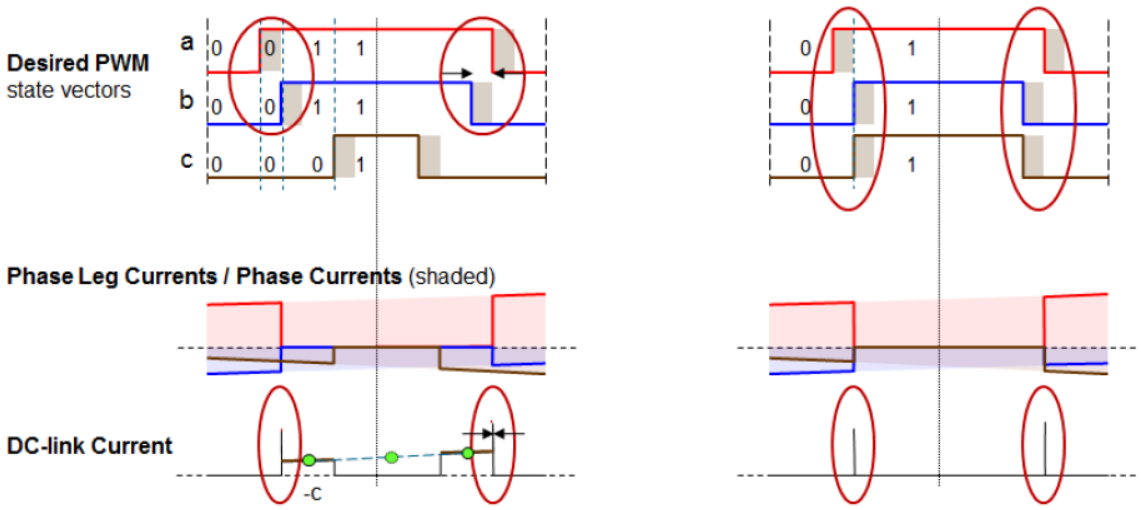

Figure 24. **Single-shunt current sensing limits during two phase edges overlapping**

However, there is a different number of switching operations in phase B. Considering a different number of dead-times inserted, the output voltage of the double-switched phase is lower. Another impact of such double switching is a different voltage vector being injected into the motor. These disturbances may cause a harmonic distortion to the flux and a sound noise.

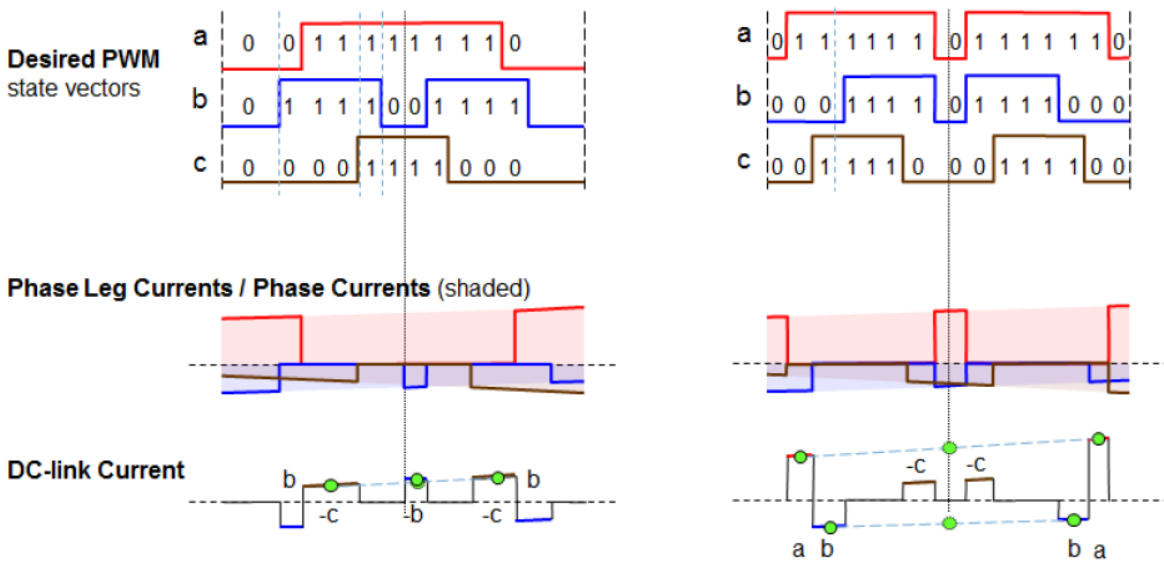

Figure 25. **Double switching PWM and 3-phase sensing opportunities**

To lower the noise and losses during double switching, all three signals are split in two parts, whilst one of the signals has the two parts shifted apart by a longer time span (see the above figure on the right, the brown signal is shifted from the blue signal). The unnecessary voltage vector (110) is switched for two short periods of time including a zero-voltage vector and the negative impact of double switching is reduced. The main disadvantage of such concept is the duty cycle is limited to approximately 93%. thanks to double switching, two samples for each current are available, an average value can be calculated.

Output voltage actuation: Generated phase signals based on duty cycles (phase A, phase B, phase C) of two PWM periods are shown in the following figure. These phase voltage waveforms correspond to a center-aligned double switched PWM with a space vector modulation input.

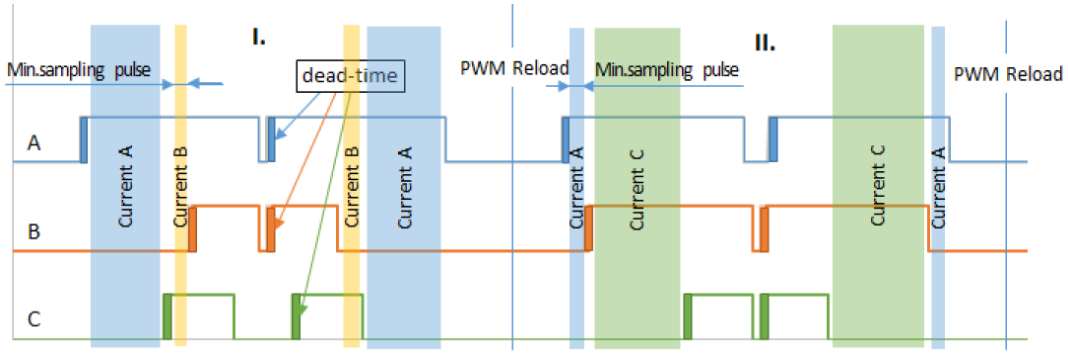

Figure 26. **Double-switching PWM and phase currents visibility**

The above figure shows two periods of PWM. In period I, phase B and phase C are of the same duty cycle, but shifted to enable three phase current reconstruction from a single shunt current signal. The width of the space between two pulses of phase C impacts the minimal sampling pulse width for current B sensing. Period II. introduces similar situation with phase A and B.

To benefit from the double switching feature of the PMF module, timings of the PWM signals edges are calculated. The algorithm (SetDutyCycle (SWLIBS_3Syst_F16 *f16pwm, tU16 sector) function in “actuate_s12zvm.c” module) calculates the edges based on 3-phase SVM generated reference voltage (f16pwm) and current sector within the voltage hexagon (shown in the following figure, SVM duty cycles are black dashed lines, double switched signals are blue, orange and green).

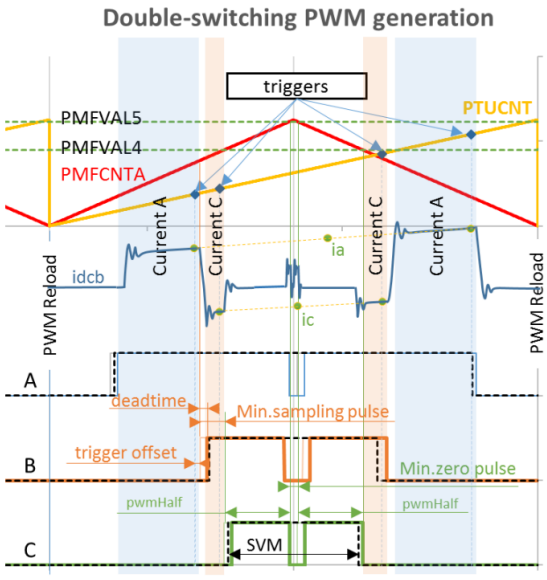

Figure 27. **Double switching edges calculation**

For more details of NXP patented double switch single-shunt solution, refer to AN5327.

5.6.4. Summary

There is no strict rule for selection of dual-shunt or single-shunt current sampling. But usually, if the motor inductance is very low or the application needs high dynamic performance, it strongly recommend dual-shunt method to avoid the big current THD.

6. Using the Demo Project

In the SW package, a demo project is provided for user to test the board with a real PMSM fuel pump application. In the following chapter, how to use the demo project will be introduced. Off course, the demo project also can be changed to be compatible with other automotive motor control applications, such as cooling fan, HVAC blower, oil pump and etc.

6.1. Motor parameters measurement

If do not have the specified motor type and want to use your own motor to replace, most time, it will not work normally due to parameters difference. So you need to change the configuration before running your own motor. You must know these parameters, you can get these parameters from datasheet or you can measure it by yourself.

To measure motor parameters, you need to prepare some common equipment like Multimeter, oscilloscope, LCR meter, power supply.

Refer to AN4680 for more insight on how to measure parameters. You need to prepare a project or other method that can keep motor running stably. Sometime, this may have limitations. So this section gives you another simple method to measure some basic parameters.

6.1.1. Measure L_d and L_q

Equipment needed: LCR meter

Steps:

1. Set the LCR meter to “L” measure mode, frequency to 10kHz.
2. Connect phase A and phase B to the measure probe.
3. Manually turn the motor rotor to run very slowly, make sure the induction value shown in LCR meter not changing so much.
4. Read the maximum and minimum value of the LCR meter.
5. Ld = Lmin/2, and Lq = Lmax/2.
6. If you cannot access the motor rotor axis, then you can use the “alignment” operation to align the motor rotor to the electrical angle 0, after the rotor stable, turn off the power supply and do the first test by using LCR meter. To do the second test set the “alignment” operation of the rotor to the electrical angle 90 degree.
7. The first test value is 2*Ld and the second test value is 2*Lq.
8. Please do the above steps again for phase B and phase C, phase A and phase C, then average the result to get the final value of Ld and Lq.
9. Notice, sometimes the Ld and Lq are very similar, it means the PMSM is surface mounted.

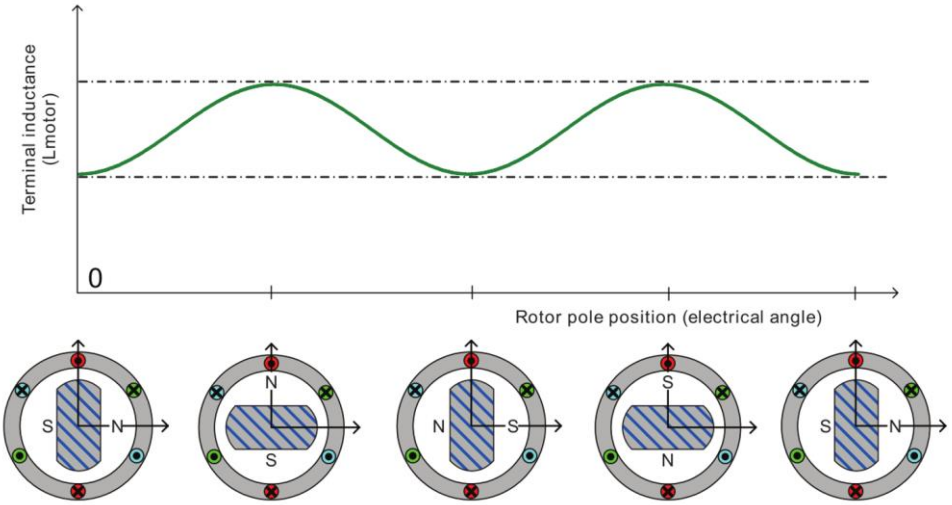

Figure 28. **MSM inductance variation with rotor position**

6.1.2. Measure pole pairs

Equipment needed: oscilloscope

Steps:

Disconnect power supply.

Connect any two phases port with oscilloscope probes.

First make a mark on the motor, then turn the motor by one hand cycle, observe and record the number of waveforms displayed on the oscilloscope, that is, the polar logarithm.

6.2. FreeMASTER configuration

After finishing the measurement of motor's parameter, you need to use FreeMASTERto regenerate and configure profile with the new parameter.

Go into FreeMASTER_control folder which is under project's root path. Double click **S12ZVM-EFP_Sensorless.pmp** file, it will open FreeMASTER as following shown in the following figure.

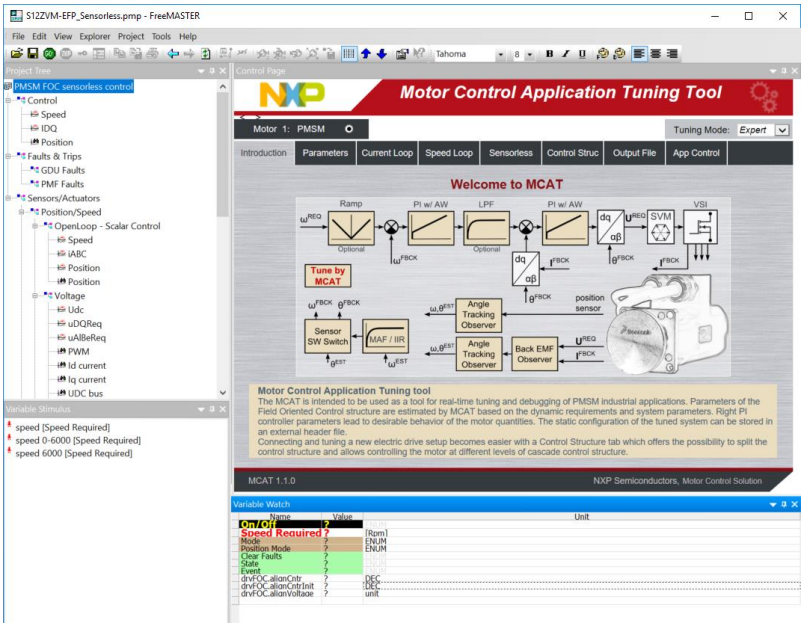

Figure 29. **FreeMASTER main window**

Switch to "**Parameters**" sheet as following picture and replace the parameters which in red rectangular box using your new values shown in the following figure.

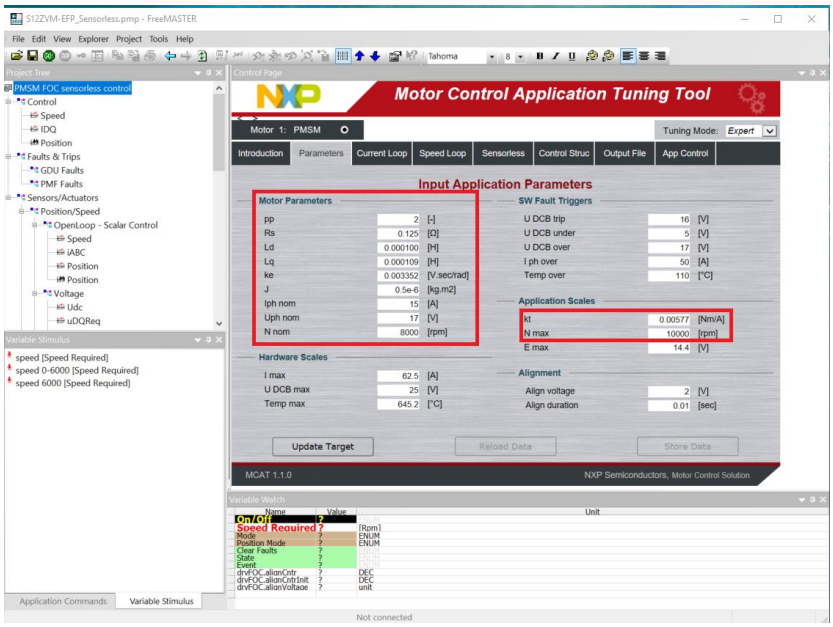

Figure 30. **Motor parameter configure window in FreeMASTER**

Switch to "**Current Loop**" sheet as shown in the following figure. Set the current loop parameters, controller limits and DC-bus voltage IIR filter settings.

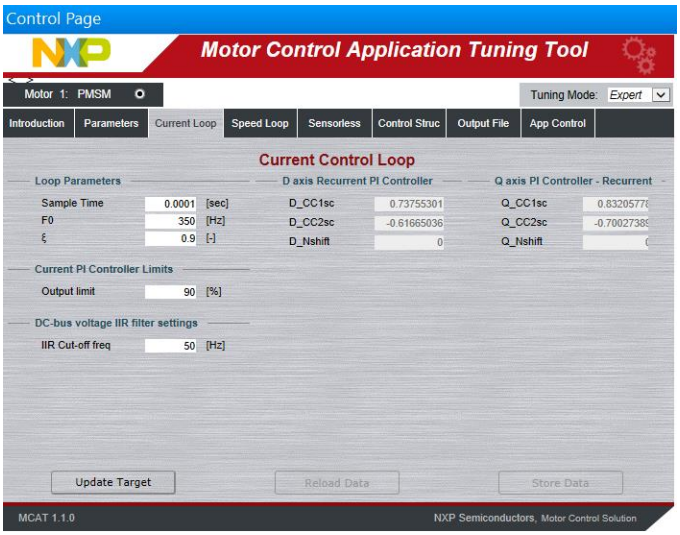

Figure 31. **Current Loop Configuration in FreeMASTER**

Switch to the "**Speed Loop**" sheet as shown in the following figure. Set the speed loop parameters, it include not only loop parameters, but also the speed ramp, speed PI controller limits and actual speed filter parameters.

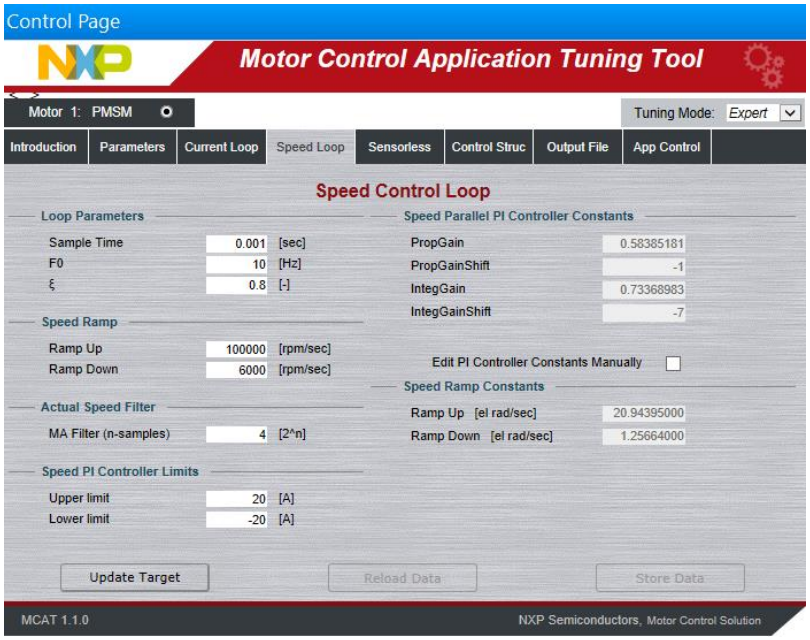

Figure 32. **Speed Loop Configuration in FreeMASTER**

To achieve the fast start up, the speed ramp value needs to be set as high as possible. In S12ZVM-EFP example project, it is set to 100000 rpm/s.

Switch to "**Sensorless**" sheet. It is used to configure the BEMF observer parameters, tracking observer parameters and the open loop start-up parameters which as shown in the following figure. To speed up the start-up process, the start-up ramp is set to 10000 which is bigger than usual application.

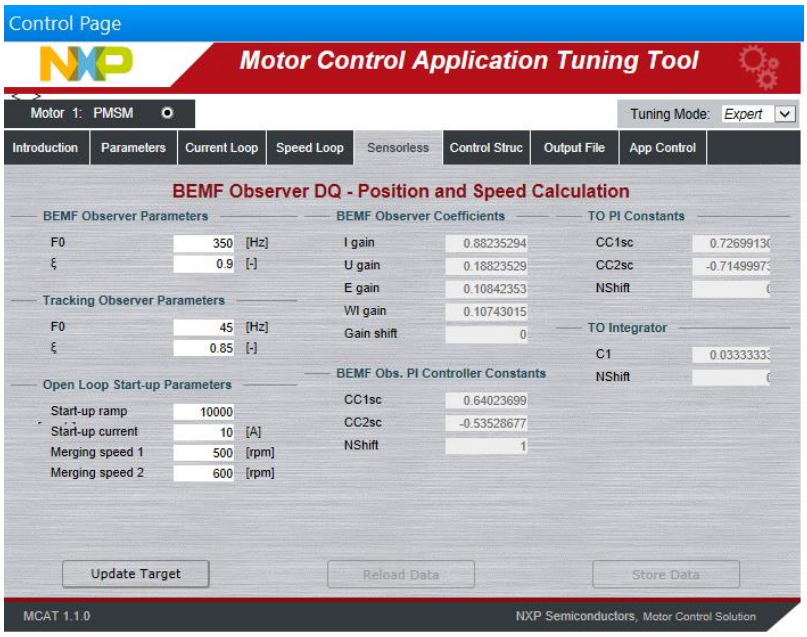

Figure 33. **Sensorless configuration in FreeMASTER**

Once all the parameters setting are complete, go to "**Output File**" sheet, click "**Generate Configuration File**" button. It will generate a new configuration file "PMSM_appconfig.h". Rebuild the firmware and the FreeMASTER configuration is successful completed.

6.3. Demo project work mode introduction

There are three methods to control the fuel pump running, FRM_CONTROL_MODE, LIN_CONTROL_MODE and PWM_CONTROL_MODE. It can be selected in "userdef.h". FRM_CONTROL_MODE means the fuel pump is controlled by FreeMASTER, it can be used in tuning stage. LIN_CONTROL_MODE means the fuel pump is controlled by LIN bus and the S12ZVM-EFP is used as slave. For PWM_CONTROL_MODE, it is a traditional control mode, it uses PWM duty cycle to control the fuel pump to start and stop and also controls the speed.

6.3.1. FreeMASTER control mode

FreeMASTER control mode is the default setting. In this mode, the motor is controlled by FreeMASTER MCAT GUI. It can control the ON/OFF, the running speed. It is very convenient to tune the control parameters, including the start-up parameters, current loop, speed loop and etc. It can update the parameters dynamic and very efficiency to tune the motor running performance.

If the performance reaches the desired goal, then other controls may be used. LIN or PWM which is decided by the system requirement.

6.3.2. LIN Control Mode

For LIN control mode, the example project of S12ZVM-EFP is running as slave. The S12ZVL32 in another hardware is work as master. The master send the "reqspeed" command to slave, the salve checks

the “reqspeed”, if the “reqspeed” is bigger than 500rpm, the motor will start to run and run to the target speed. If the “reqspeed” equal to 0, the motor will stop and enter idle state.

S12ZVM-EFP will feedback the motor real speed and the error information if triggered. More information can be found in LDF_LIN.ldf file.

6.3.3. PWM control mode

PWM control mode is the very traditional control method which using PWM duty cycle to control the target speed. Usually PWM control mode uses a linear to control the motor speed according to the PWM duty cycle. The following figure shows the speed vs duty cycle in PWM Control Mode.

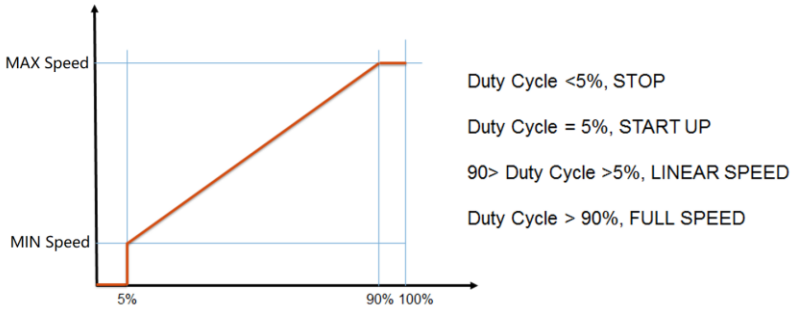

Figure 34. **Speed vs duty cycle in PWM control mode**

6.4. Build and Debug the demo project

6.4.1. Import the project with CodeWarrior

From menu File → Import…, open the **Import** window.

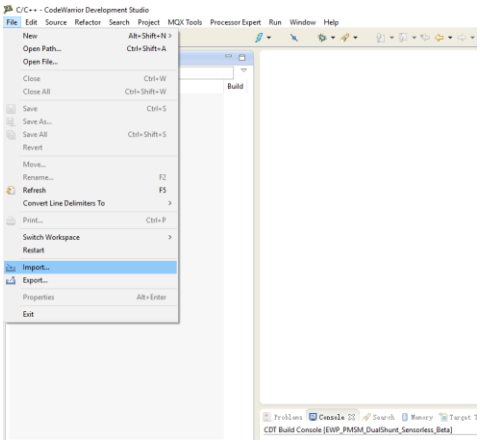

Figure 35. **Menu of File to open Import window**

Select General → Existing Projects into Workspace → Next.

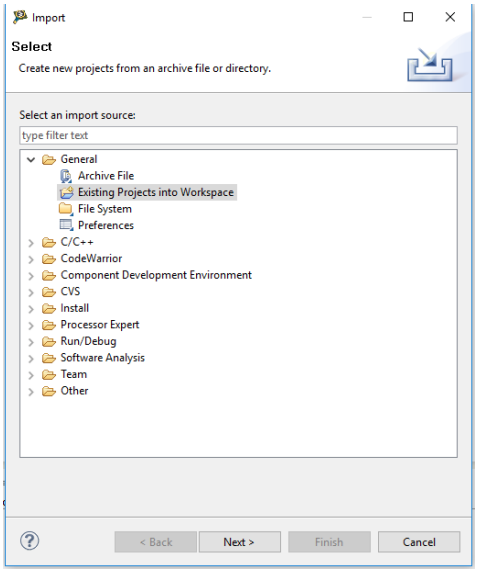

Figure 36. **Select "Existing Projects into Workspace"**

Copy the directory of the project and then push the "Enter" Key. The project name is shown in "Projects" and then Click "Finish" to complete the import.

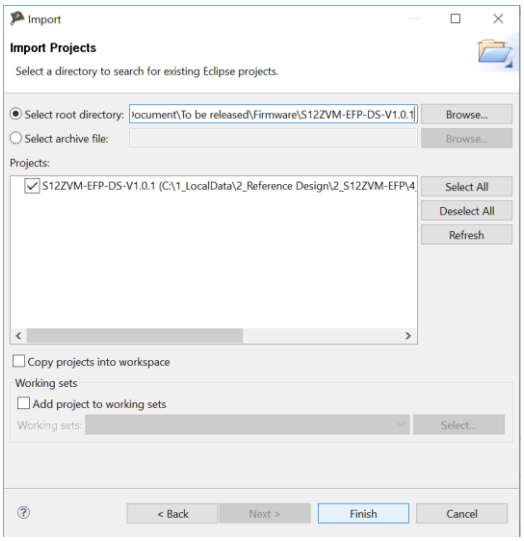

Figure 37. **Select the Target Project**

6.4.2. Rebuild and debug the project with CodeWarrior

After importing the existing example project, the first thing is to "clean" the project, the "clean" operation triggers "rebuild" event.

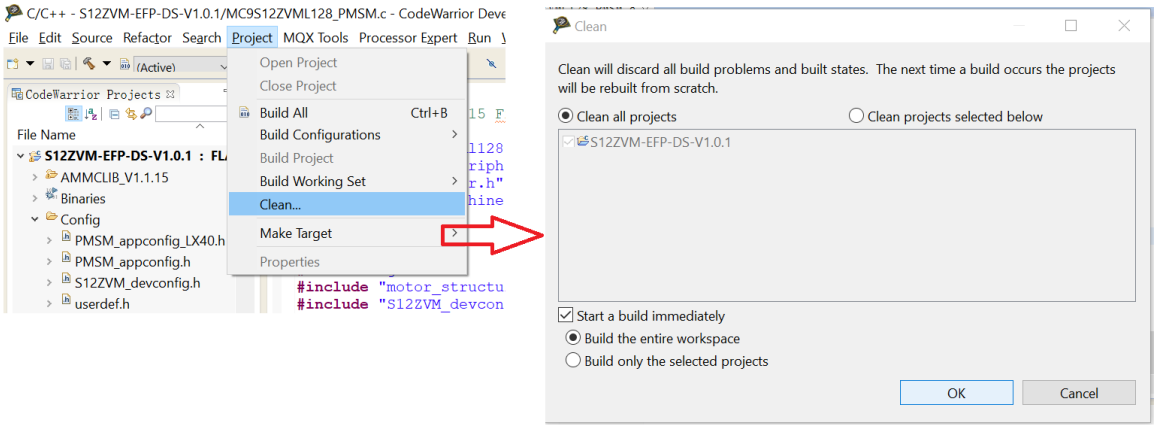

Figure 38. **Clean and build project**

If project is built successfully, following message is displayed on the Console window, the message is shown in the following figure.

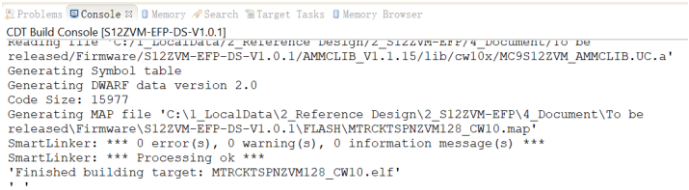

Figure 39. **The compile result in console window**

After the successful compilation, the next step is enter the debug step. Click the debug configurations, as shown in the following figure.

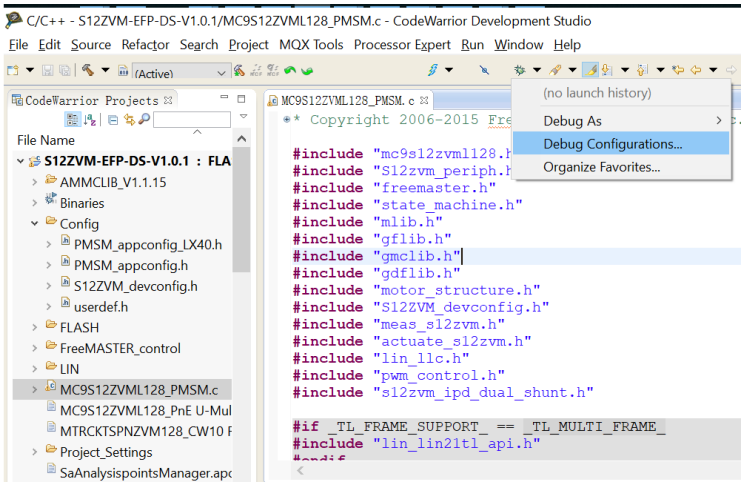

Figure 40. **Access the Debug Configurations**

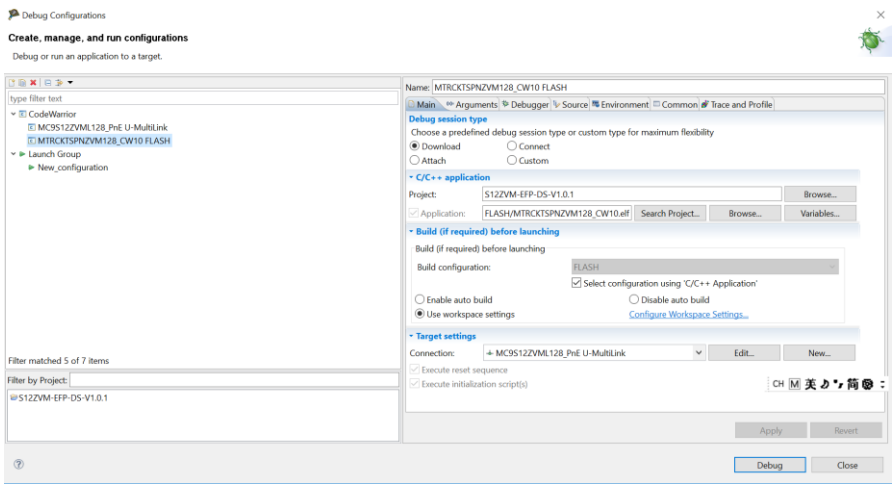

Figure 41. **Debug Configurations Details**

Double click the MTRCKTPNZVM128_CW10_FLASH and then enter the debug process and it stops during the “main” function if the hardware has no issue.

7. Fuel Pump System Test Performance

This chapter gives out the performance of S12ZVM-EFP RD SW from the following two aspects, fuel pump system test performance and used MCU resource. The fuel pump system test is based on a simplified fuel pump system which shown in the following figure. The fuel pump model is 3Q0919050. The main system test is as follows:

- Fast start up and ON/OFF test
- Fluctuation of load
- Fluctuation of power supply
- Velocity adjust by PWM change
- Temperature rise test

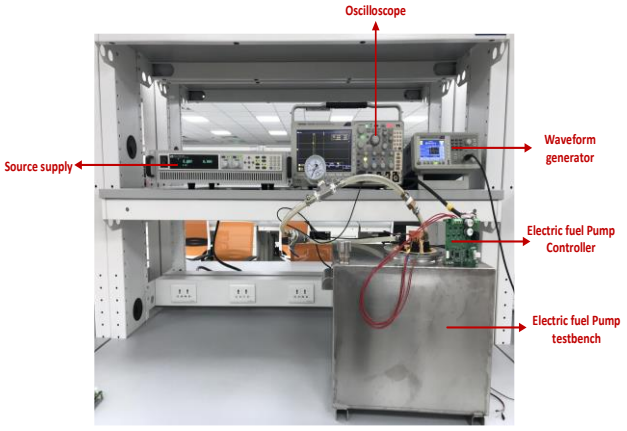

Figure 42. **Simplified Fuel Pump test platform**

7.1. Fast start up and ON/OFF test

Frequently turn the fuel pump ON and OFF. Try 100 counts and record the successful counts. One of the start-up current waveform is shown as shown in the following figure. The result is 100% percent successful start-up.

IPD and the crossing smooth I/F start up method make the start up fast and robust. As per the following figure, the start is up to 8000 rpm and the time is 107.6 ms and it is less than 150 ms.

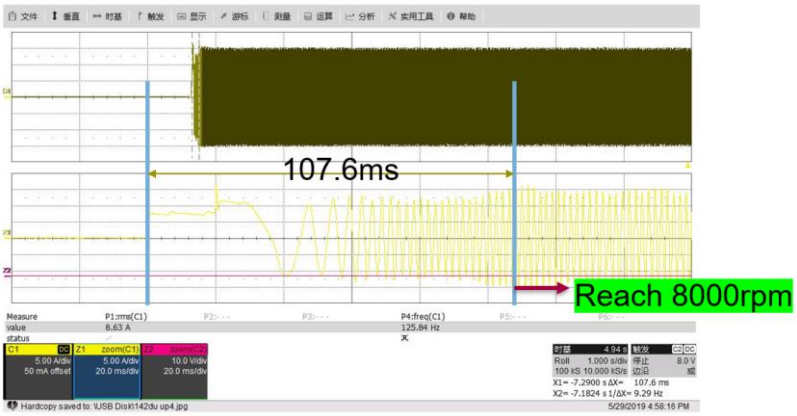

Figure 43. **A successful start-up phase current scope**

7.1.1. Fluctuation of load

S12ZVM-EFP work at 13 V, 8000rpm, then change the load of the pump. Check the speed variation for the load changing. From the following figure, even from light load to heavy load, or from heavy load to light load, the speed are still stable and does not change.

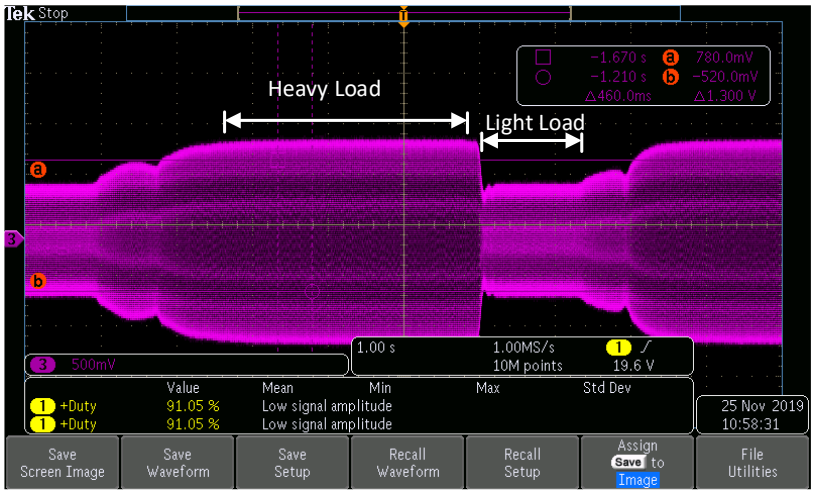

Figure 44. **Phase current scope for load changing test**

7.1.2. Fluctuation of power supply

Test the S12ZVM-EFP work state in 8 V, 13 V and 16 V, change the supply voltage dynamic and check the speed variation.

The test result is good and the speed does not change much in different voltage supply. But please note in 8 V, the speed may not achieve the rated speed.

7.1.3. Velocity adjust by PWM change

Test the fuel pump to check if the following speed command and the dynamic performance by using PWM input mode for speed command sending. The ramp down time is bigger due to the speed ramp setting for speed down is not as high as ramp up. The following figure shows the speed adjustment according to the PWM capture.

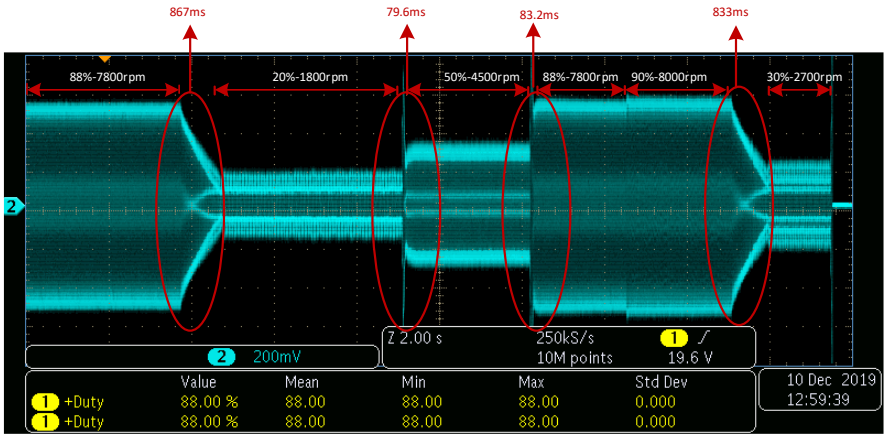

Figure 45. **Phase current scope for PWM vs speed**

Figure 46. **PWM Input Capture Shown in FreeMASTER**

The PWM capture value in FreeMASTER is shown in the above figure.

7.1.4. Temperature rise test

Running the S12ZVM-EFP board at rated power 250 W at 13 V. The temperature of MOSFET rises from 18° to 52°. At 5 min the temperature is stable and almost at 52°. The temperature change is 34° for long time. Please check the following figure for more information. The temperature rise result for S12ZVM-EFP is 34°

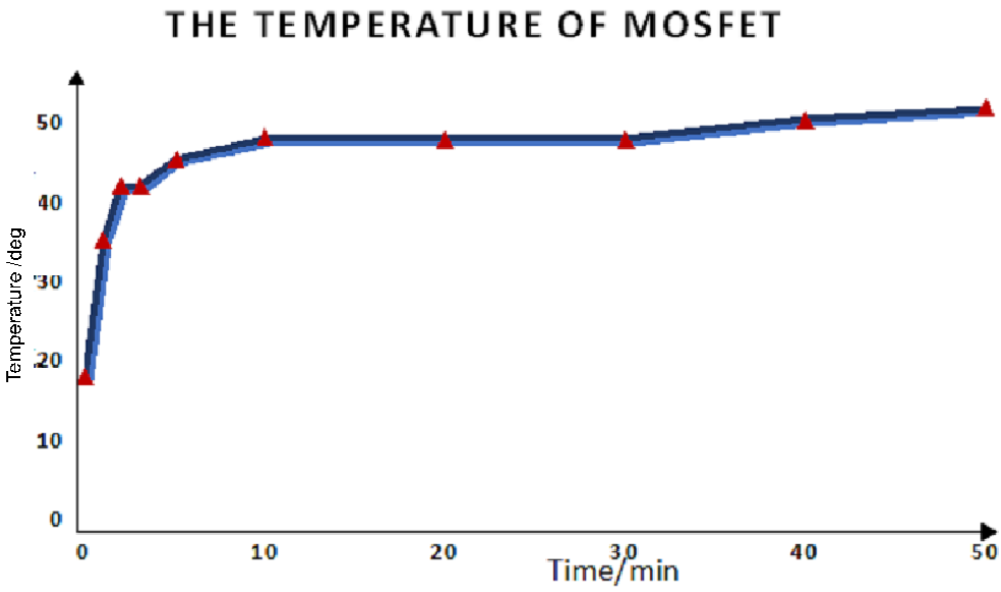

Figure 47. **S12ZVM-EFP temperature rising test result**

7.2. Used MCU resource

Set the optimization level “2” and “Speed” first as shown in the following figure.

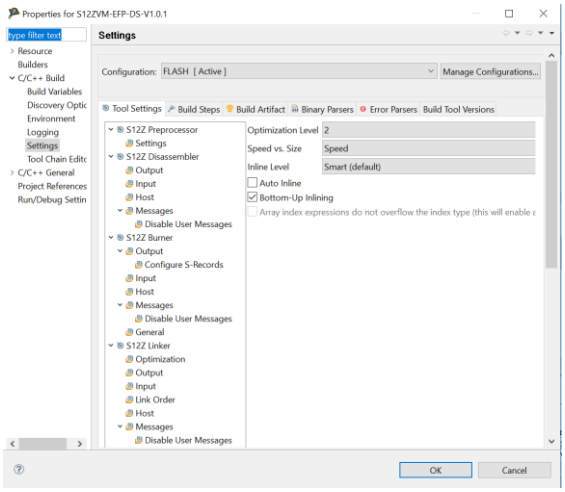

Figure 48. **Compiler optimization setting**

The memory usage result is shown as following by setting the control mode to "FRM_CONTROL_MODE"

```
Summary of section sizes per section type:
READ_ONLY (R):         4056 (dec:    16470)
READ_WRITE (R/W):       E91 (dec:     3729)
NO_INIT (N/I):          1E3 (dec:      483)
```

The S12ZVM-EFP is based on S12ZVML128. So for memory usage, it has lots of available memory which can be used.

8. Summary

The S12ZVM-EFP SW package provides an out-of-box runtime software based on NXP S12ZVM PMSM example project V1.3 and AMMCLIB v1.1.15.

The SW is dedicated optimized for fuel pump application, especially for fast and robust start up. The crossing smooth I/F start up and Initial Position Detection method are the key know-how for this project.

SW has three work mode, including FreeMASTER control, LIN control and PWM control. It can support dual shunt and single shunt, but this user guide is mainly for dual shunt.

Meanwhile, the SW is not only suitable for BLDC/PMSM fuel pump application, but also is a good reference for automotive small node motor control application, such as oil pump, water pump, cooling fan and HVAC blower.

9. Appendix A. Reference

1. S12ZVM MCU Family – Reference Manual(REV2.13), www.nxp.com, 2019/4/28.
2. Automotive Math and Motor Control Library Set for NXP S12ZVM devices, Rev.17, 2018/12.
3. PWM Input Control for S12Z, https://community.nxp.com/docs/DOC-342785, 2019/3/18.
4. AN5135, 3-phase Sensorless PMSM Motor Control Kit with MagniV MC9S12ZVM, Rev.1, www.nxp.com, 2016/5.
5. AN5327.Three-phase Sensorless Single-Shunt Current-Sensing PMSM Motor Control Application with MagniV MC9S12ZVM, Rev.0, www.nxp.com, 2016/8.
6. Improved Simple I-F Open-Loop Start-up of PMSM Drives Without Speed or Position Sensor, Matej Pacha, Simon Zossak, SLED 2019

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.

Document Number: S12ZVMEFPSWUG
Rev. 0
04/2020