Freescale Semiconductor
Application Note

Document Number: AN4490
Rev. 0, 3/2012

How to Build an FOC Code Structure Based on the 56F8006 Using a Quick-Start Tool

by: Xu wei Zhou
Applications Engineer
Shanghai, China

Contents

1 Introduction

This application note shows how to build a general Field Oriented Control (FOC) algorithm code structure, based on the 56F8006 using a Graphic Configuration Tool (GCT) called the DSP56800E_Quick_Start, integrated in CodeWarrior. The configuration of the 56F8006 in this document is based on a 3-Phase BLDC/PMSM Low-Voltage Motor Control Drive Board together with the MC56F8006 Controller Daughter Board whose information can be found in the user guide titled *3-Phase BLDC/PMSM Low-Voltage Motor Control Drive* (document number LVMCDBLDCPMSMUG)and the *MC56F8006 Controller Daughter Board for BLDC/PMSM Motor Control Drive* (document number MC56F8006DBUM).

2 Initialize peripherals for motor control using GCT

Quick Start provides a graphic configuration interface for users to initialize peripherals in an intuitive way. This graphic interface helps configure peripheral functions, while users do not have to memorize all register definitions.

freescale semiconductor

2.1 About Quick Start

The 56F800E_Quick_Start tool is to the processor expert but dedicated to the DSC series. It provides more compact and fully debugged peripheral drivers, examples, and interfaces for programmers to create their own C application code while independent of core architecture. It provides a software infrastructure that allows development of efficient, ready to use high level software applications that are fully portable and reusable between different core architectures. Maximum portability is achieved for devices with compatible on-chip peripheral modules. For example, the initialization code of the PWM can be the same for the 56F8006 and 56F80xx.

NOTE

This quick start tool (the latest version is 2.5) is not integrated in the CodeWarrior development environment by default. You will have to download it from the Freescale web page.

After downloaded from the web, install it according to the manual to integrate it into CodeWarrior. The remainder of the work is focused on only the algorithm.

2.2 Initialization of peripherals for FOC

The 56F8006 is one of the cost-effective chips in the DSC series and has some unique peripheral characteristics compared to other DSC chips, such as the ADC module and PDB module. This document introduces configurations of motor control related peripherals using Quick Start's GCT. Configure the system at 32 MHz using the internal oscillator.The PWM runs at three times the system clock speed. It uses a PWM internal synchronization signal to trigger the PDB which triggers the ADC in the middle of a PWM period where a 000 NULL voltage vector occurs. The FOC algorithm is executed in the ADC interrupt service routine.

2.2.1 Setup a new project

After the quick start tool has been integrated into CodeWarrior, setup a new project based on the quick-start stationery. Click New from File→New, and select DSP56800E Quick Start r2.5 Stationery from the Project tab. Type a project name and all project files will be in a folder with the same name as the project. Figure 1 shows the New dialogue.

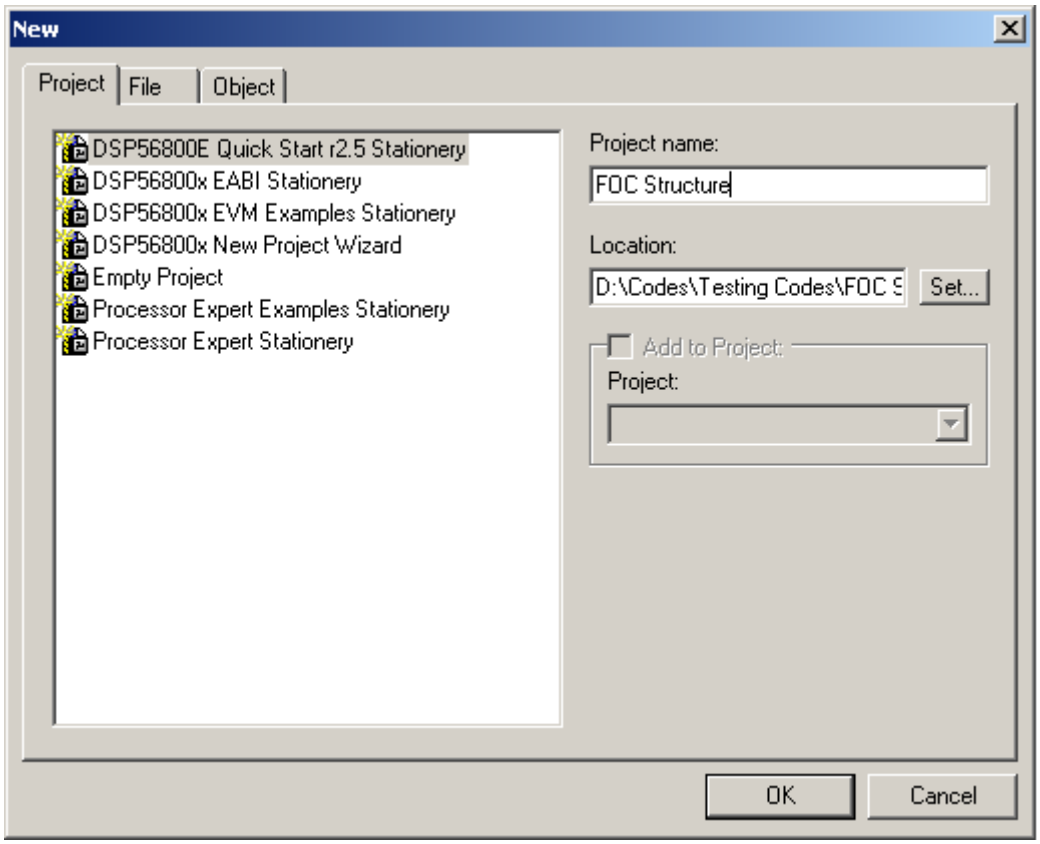

Figure 1. File->New dialogue

Click OK and a New Project dialogue window opens, see Figure 2. Choose the Standalone_C_application from the MC56F8006 this means all the files used will be in the project folder. You can move this folder to wherever you want without having any problems. If you choose the C_application, some of the files used in the project will be in folders where CodeWarrior is installed.

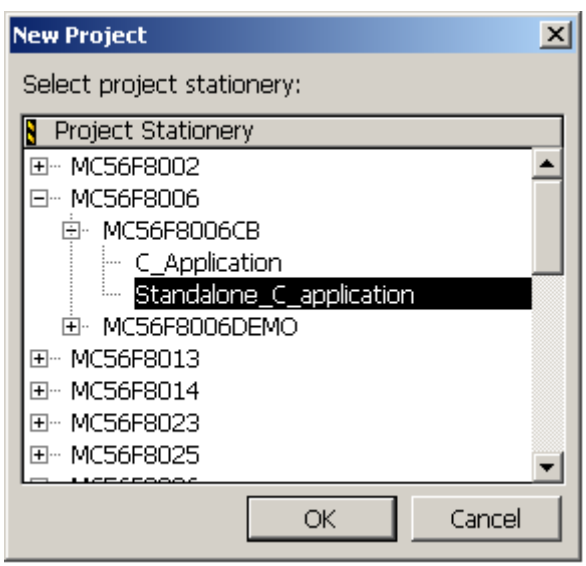

Figure 2. New Project dialogue

Click OK and then click Make, in the tools bar, to compile and link all the files so the global variables, macros, and functions turn into the global color which is azury by default. Double click on main.c on the left Files tab, notice all the peripheral head files have been included. There is a main function definition with a loop inside. All you have to do is add code to this main.c file with the interfaces the quick start provides.

2.2.2 Peripheral configuration using GCT

Open GCT, check View→Registers Summary and View→Warnings Summary to see the initialization value for module registers and warnings if any configuration was not done properly. This configuration via the graphic interface is to setup appropriate values for the registers used in the project.

2.2.2.1 On chip clock configuration

Check the OCCS-On-Chip Clock Synthesis on the left side of GCT, here the internal relaxation oscillator is used with the factory trim value. Figure 3 shows the configuration.

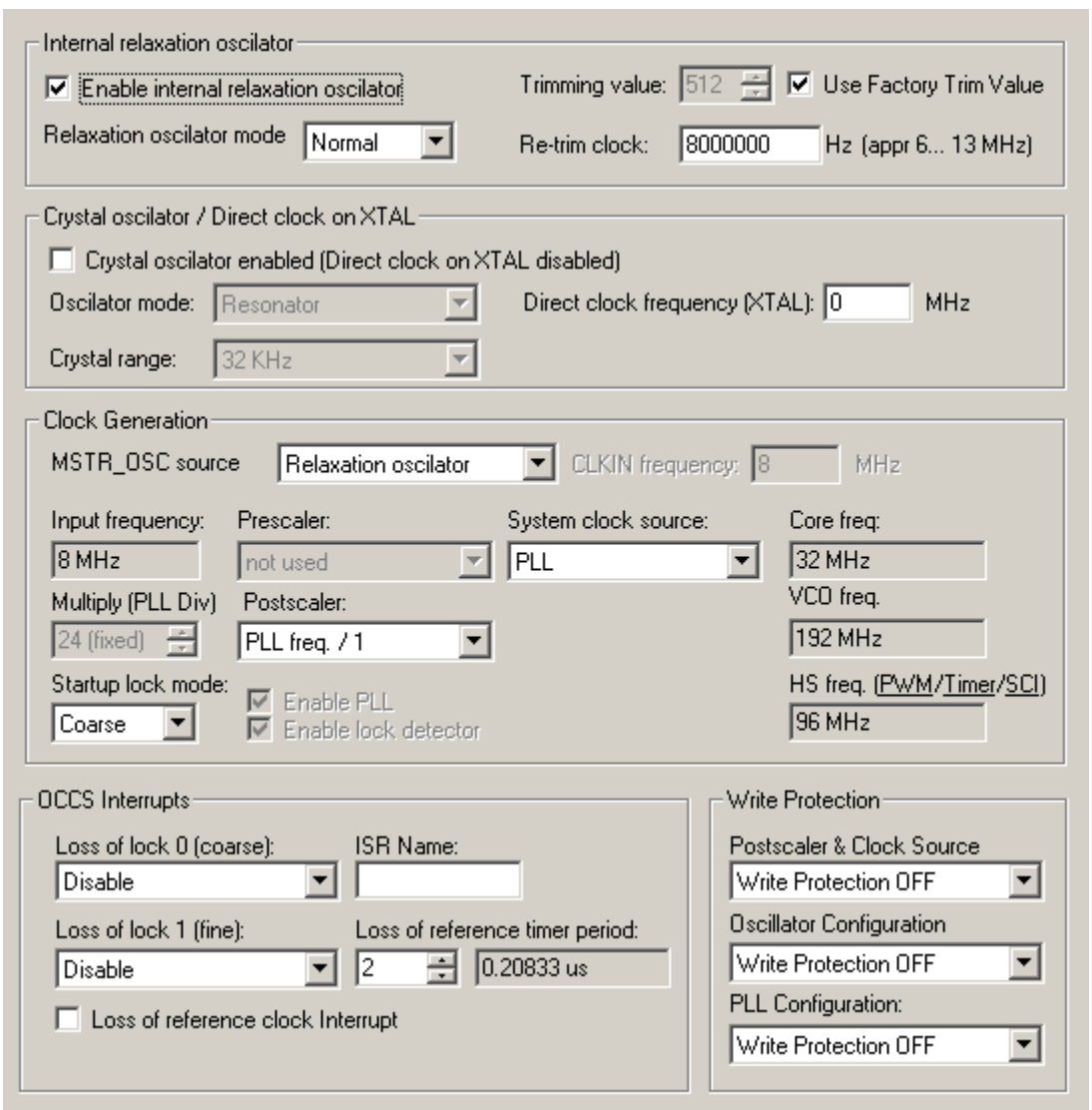

Figure 3. OCCS configuration

Remember to check the Use Factory Trim Value because the default trimming value 512 is never good enough.

2.2.2.2 COP configuration

The watchdog can be enabled depending on your needs; Here the watchdog is disabled. Check COP-Computer Operating Properly on the left side of GCT and configure this module as shown in Figure 4. Leave all checkboxes unchecked.

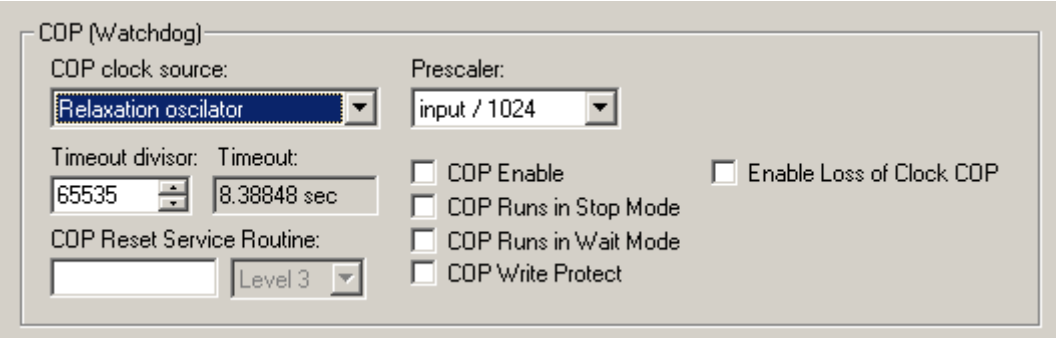

Figure 4. COP configuration

2.2.2.3 IO configuration

Here are some peripherals used:

- SCI for the interface with a PC using FreeMASTER
- SPI for configuring the MC33927 which is the inverter drive
- PWM/PDB/ADC0/ADC1 for FOC algorithm
- Qtimer0 for interface with incremental encoder

Table 1 shows the pin assignment for the 56F8006 daughter board schematic.

Table 1. LV Motor Control Kit pin assignments

Module Name	Secondary Function	GPIO Pin
PWM	PWM0~PWM2	GPIOA0~GPIOA2
	PWM3	GPIOB0
	PWM4~PWM5	GPIOA4~GPIOA5
	FAULT0	GPIOA6
SIM	RESET	GPIOA7
ADC	ANA0~ANA2	GPIOC0~GPIOC2
	ANB0~ANB2	GPIOC4~GPIOC6
QTimer0	TIN2	GPIOB2
	TIN3	**GPIOB3**
GPIO	**UP_SWITCH**	**GPIOB5 (input)**
	DOWN_SWITCH	**GPIOB1 (input)**
	TOGGLE_SWITCH	GPIOF0 (input)

Table continues on the next page...

Table 1. LV Motor Control Kit pin assignments (continued)

Module Name	Secondary Function	GPIO Pin
SPI	MISO	GPIOB4
	MOSI	**GPIOB3**
	SCLK	**GPIOB5**
	SS (Used as GPIO output)	**GPIOB1**
SCI	TXD	GPIOB7
	RXD	GPIOB6
JTAG	TDI,TDO,TCK,TMS	GPIOD0~GPIOD3

Some of the pins in the SPI module have a functional conflict with the GPIO and QTimer0 module as indicated in Table 1, but SPI is only for MC33927 initialization. Therefore configure the GPIOB1, GPIOB3, and GPIOB5 an SPI function, change their function to be QTimer0 and GPIO as shown in Table 1, after the MC33927 has been initialized.

Figure 5 shows the GPIO_B configuration, check GPIO_B-General Purpose I/O Port B on the left side of GCT.

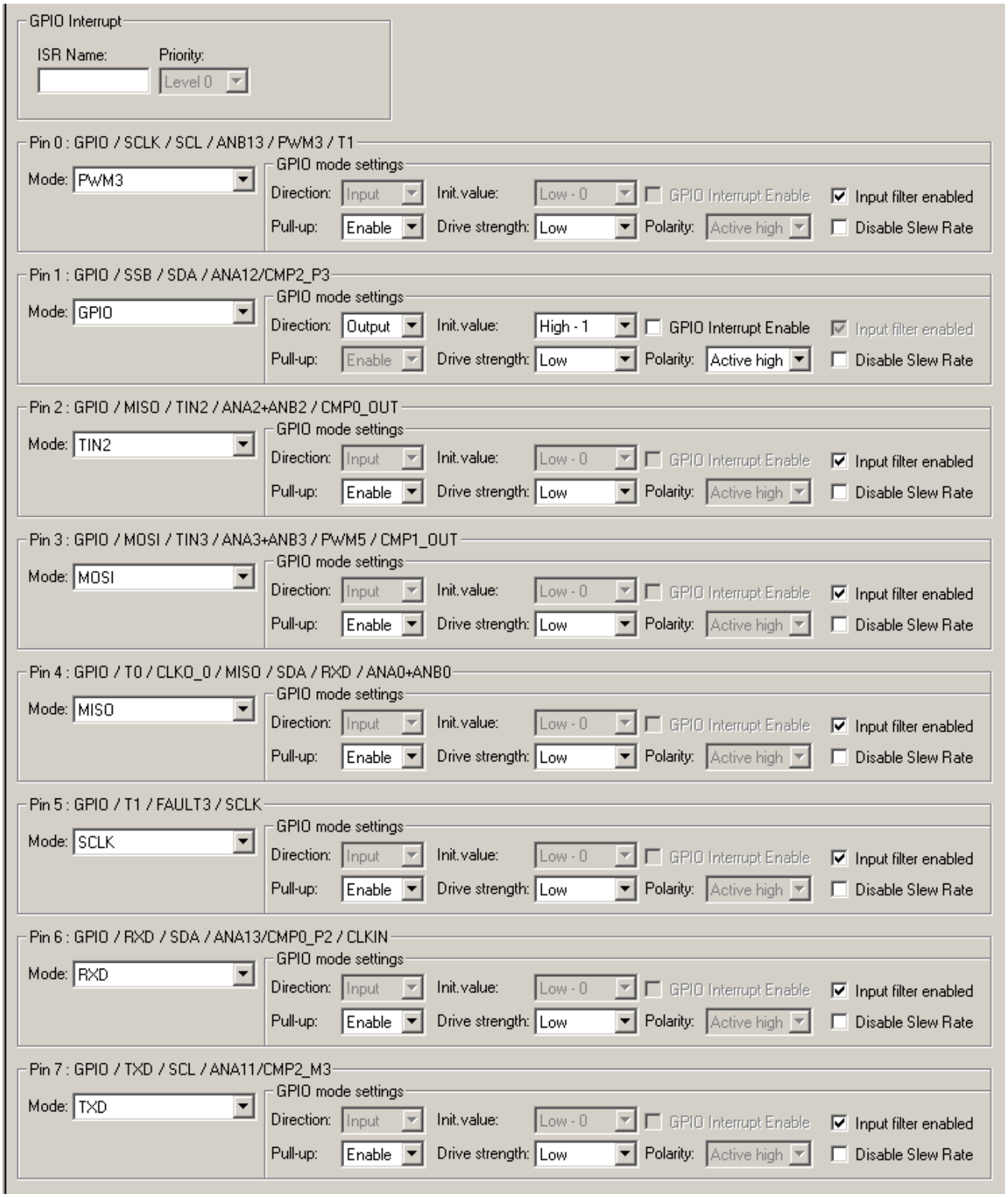

Figure 5. GPIO_B configuration

Notice that the GPIOB1 which is for the SPI SS function is configured as an output GPIO with the initial level being high.

2.2.2.4 System integration module configuration

Check SYS-System Support Control on the left side of the GCT. There are some sections in this configuration interface. First, enable all peripherals clock you will use. See Figure 6. If the corresponding peripheral clock is not enabled here and configured in the GCT, there will be warnings in the warning window.

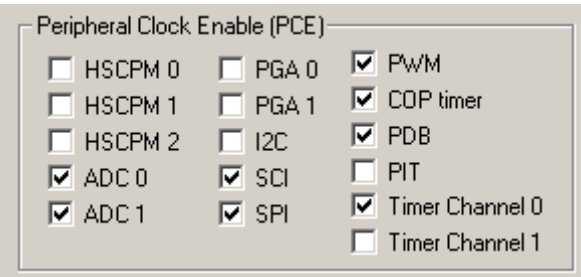

Figure 6. Peripheral clocks checkboxes

There are internal input source selections for the PWM, Comparator and QTimer modules, Figure 7. Default settings are used here.

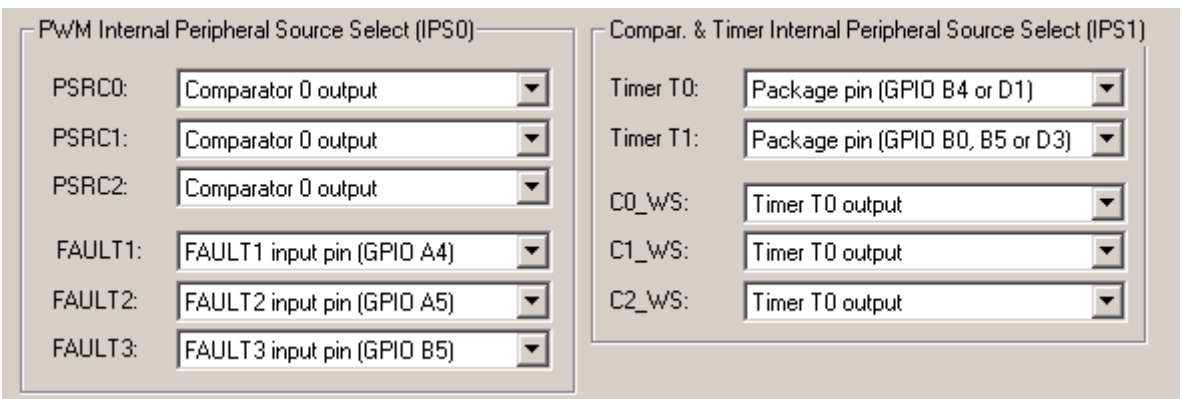

Figure 7. Selection of input signals for PWM/Comparator/QTimer

The PWM, QTimer, and SCI module can use a clock which is three times the system clock. Use 96 MHz as PWM clock to achieve the highest PWM resolution. See Figure 8.

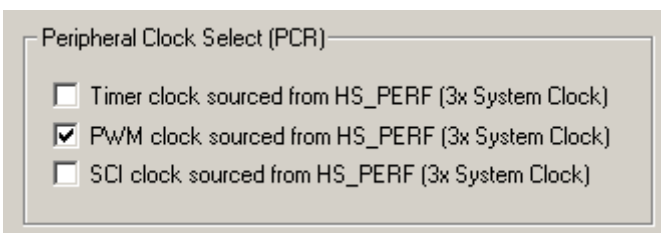

Figure 8. Choose 3x System Clock as PWM clock

Some pins have many functions integrated, for example GPIO_B6 can be RXD, SDA, CMP0_P2/ANA13, and CLKIN. Registers in the SIM module decide what function uses the GPIO_B6 after being configured as a peripheral pin in the GPIO module. In fact, you do not have to decide the function here if you have chosen the GPIO configuration interface function. Figure 5 shows the GPIO_B6 set as the RXD function. In the SIM configuration interface, GPIO_B6 is set as RXD automatically. See Figure 9.

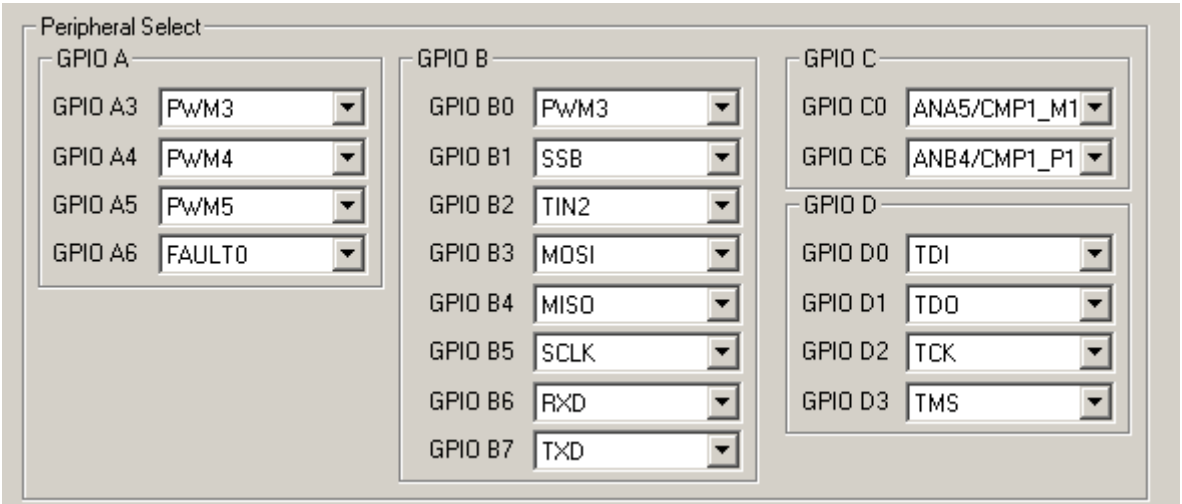

Figure 9. Selection of peripheral functions for GPIO pin

2.2.2.5 PWM configuration

PWM is an important module in motor control. Figure 10 shows the settings for the 3-Phase BLDC/PMSM Low-Voltage Motor Control Drive board. PWM frequency, deadtime, and the polarity setting is shown in Figure 10.

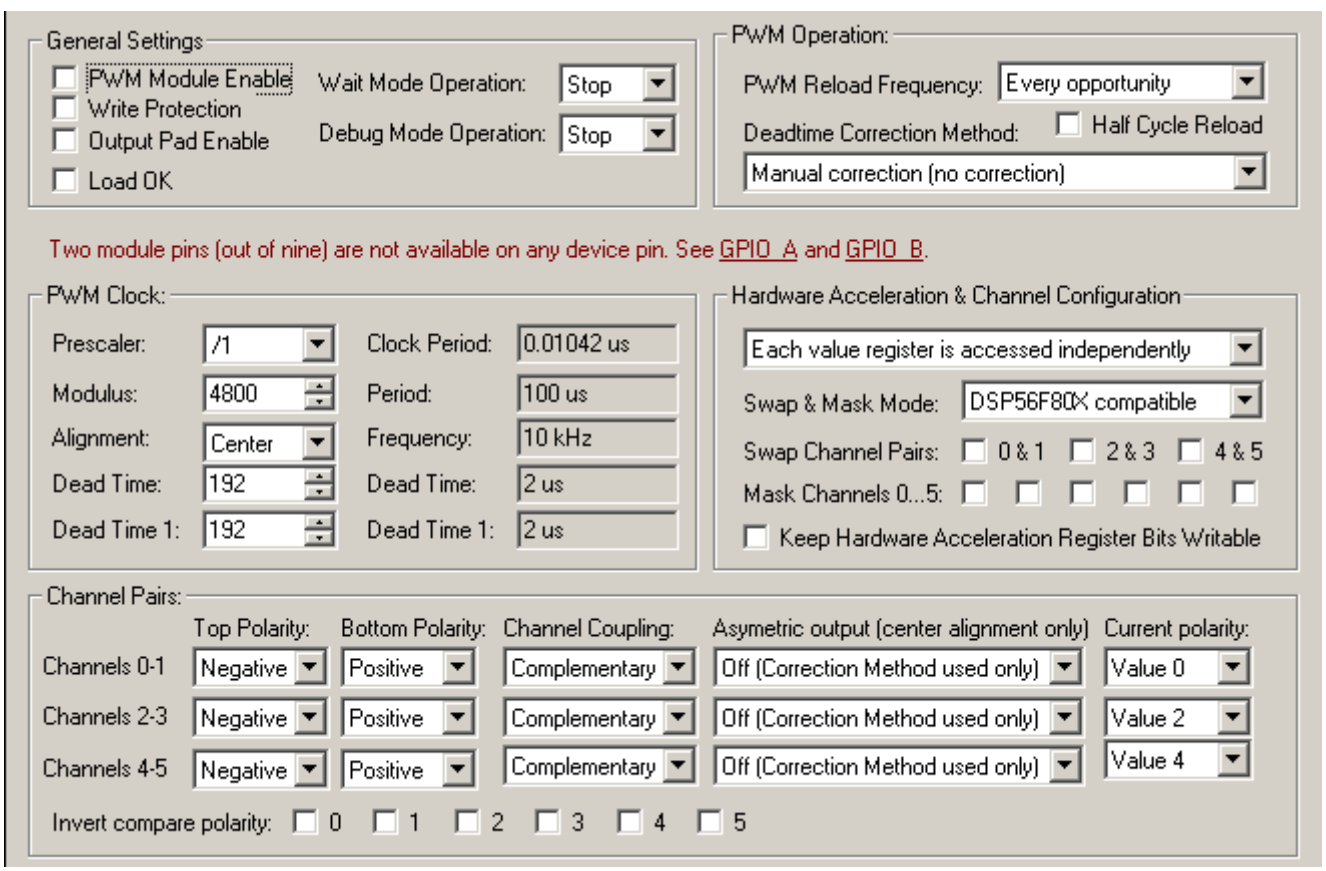

Figure 10. Frequency, deadtime, reload mode, and PWM polarity settings

Set the PWM frequency with center-aligned mode at 10 KHz. There is an important relationship between the SYNC signal and RELOAD signal that needs to be clarified first:

1. If Every opportunity is chosen for PWM Reload Frequency and Half Cycle Reload is not enabled, the RELOAD signal always appears at the start time of a PWM period. If the Half Cycle Reload is enabled, the RELOAD signal will appear at both the start time and the midpoint of a PWM period.
2. SYNC signal only comes with the RELOAD signal that occurs at the start time of a PWM period. There will not be any SYNC signals at midpoint of a PWM period.
3. The PWM Reload Frequency indicates this option is for RELOAD signals. But because the SYNC signal comes with the RELOAD signal that occurs at the start time of a PWM period, then the SYNC signal frequency can also be changed if the corresponding RELOAD signal frequency is changed.

The SYNC and RELOAD signals in configuration of Figure 10 is shown in Figure 11 . All the top legs are negative polarity which means low voltage turns on the top MOSFETs as indicated in Figure 10. The reason why the top MOSFETs are turned on by low voltage is due to the MC33927. Refer to the datasheet titled *Three-Phase Field Effect Transistor Pre-Drive* (document number MC33927) for details.

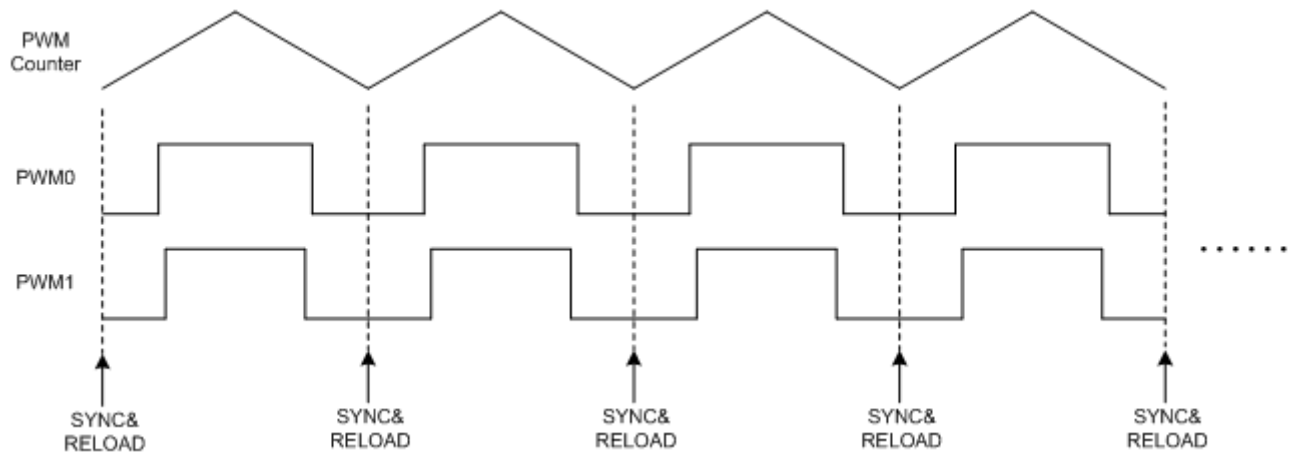

Figure 11. SYNC and RELOAD signals in configuration of figure 10

Some other examples of SYNC and RELOAD signals are in Figure 12 - Figure 15 where all six switches of the inverter can be turned on for a general purpose through a high voltage on the gates. Figure 12 shows Every 2 opportunities with half cycle reload not enabled. Figure 13 shows Every opportunity with half cycle reload enabled. Figure 14 shows every 3 opportunities with half cycle reload enabled.

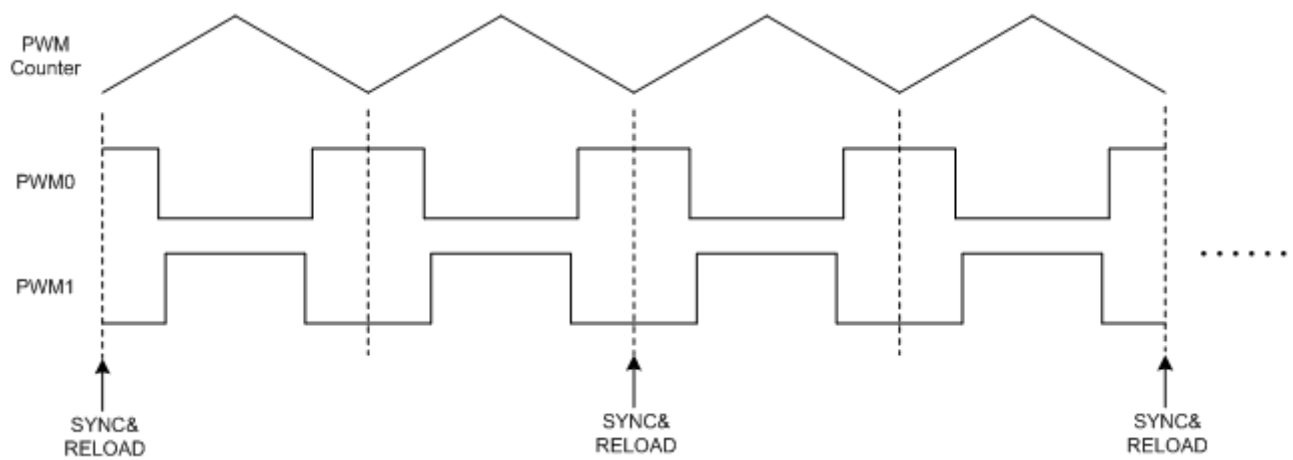

Figure 12. SYNC and RELOAD signals in every 2 opportunities with no half cycle reload

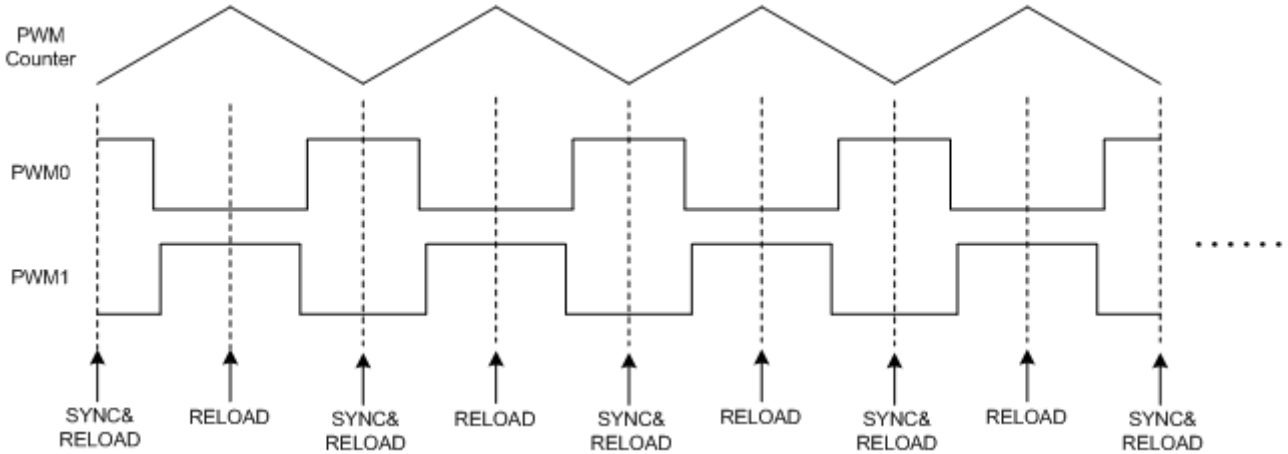

Figure 13. SYNC and RELOAD signals in every opportunity with half cycle reload

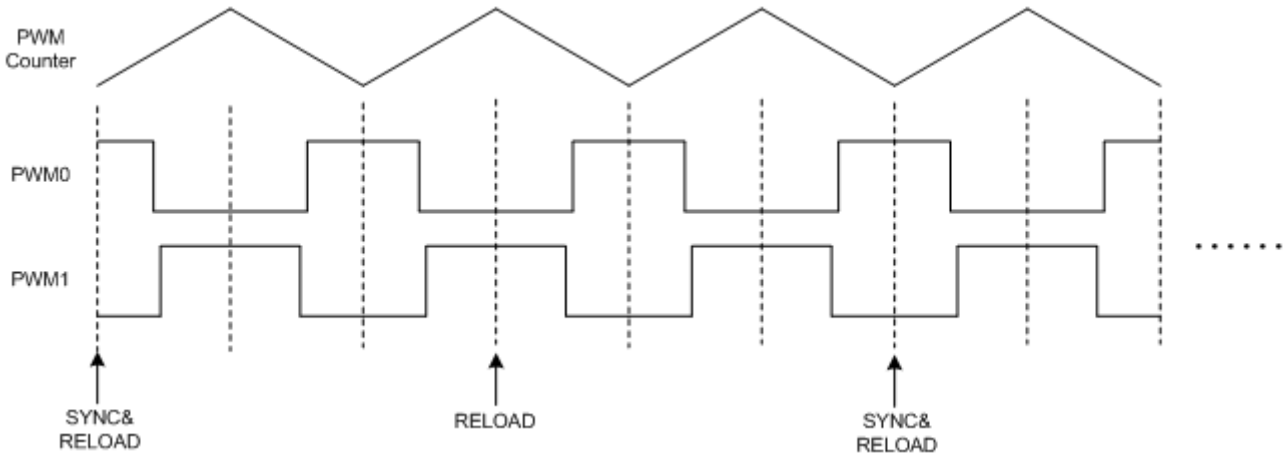

Figure 14. SYNC and RELOAD signals in every 3 opportunities with half cycle reload

The fault setting of the PWM module is fault0 as a protection signal with high level active,set all initial duties to 50%. Therefore there will be no line to line voltage even if the PWM outputs are enabled. The configuration is shown Figure 15.

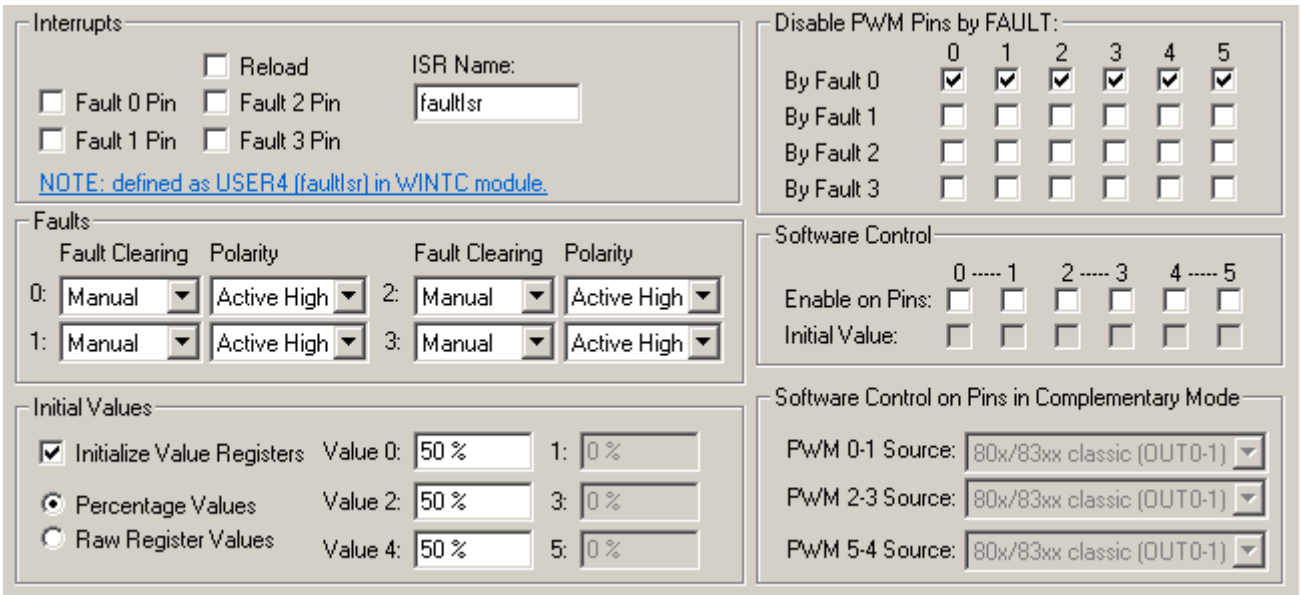

Figure 15. Fault signal configuration

A level 2 interrupt priority is used as the fault interrupt and is defined in the WINTC module. Most peripheral module interrupt priorities are the lowest which is level 0. To change it, take the fault interrupt and first enable it in the PWM module and fill the interrupt ISR Name to store the interrupt setting in the WINTC module, then back to WINTC module, fill in the ISR Name and corresponding interrupt source number in the interrupt levels you want. See in Figure 16.

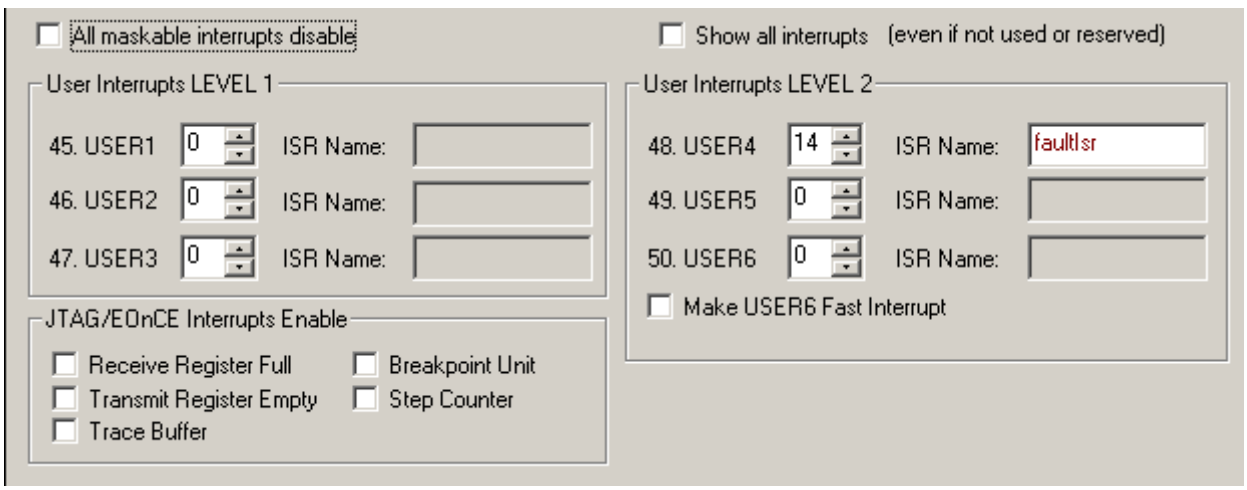

Figure 16. Change interrupt priority through WINTC

Set the filter time for the fault pin through this GCT interface. The filter time is set to 0.958 us for fault0 signal. See Figure 17.

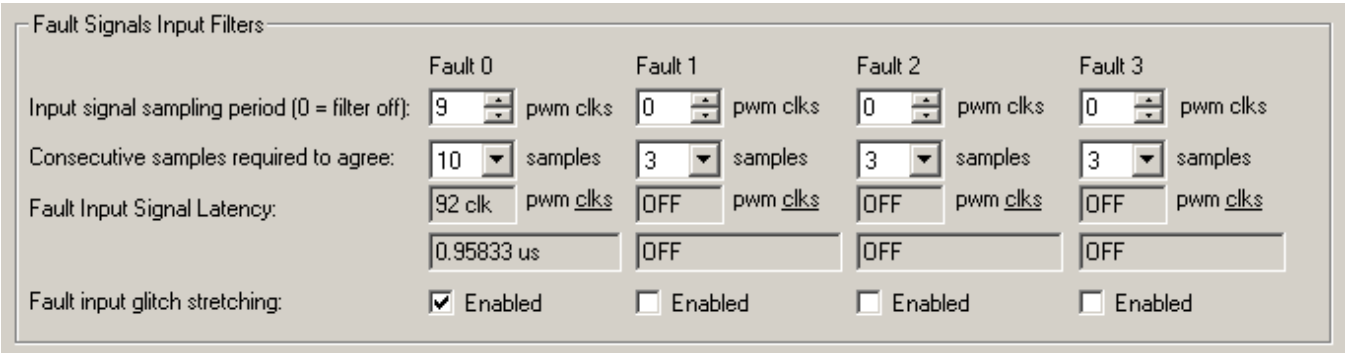

Figure 17. Fault input filter time setting

2.2.2.6 PDB configuration

This PDB module is used for triggering the ADC module to sample currents of motor windings and DC bus voltage. The PWM SYNC signal starts the PDB module counter. Phase currents should be sampled at the midpoint of the time interval where 000 NULL voltage vector lies. This is because the shunt resistor is connected in series with the lower legs. Figure 18 shows the PDB configuration.

Trigger A and Trigger B delay time is important. The SYNC signal is shown in Figure 11 . This is a detailed look of a complementary PWM signal where you can get the correct value for Trigger A and Trigger B. See Figure 19, because of the deadtime, the time where the PWM counter reaches the value of the period register (namely Modulus register) it is no longer at midpoint of the time interval where 000 NULL voltage vector lies. Set Trigger A delay register to pwm period+deadtime/2, this way when the PDB counter reaches the Trigger A delay value of the register value, two phase currents can be sampled and converted simultaneously at midpoint of the 000 NULL voltage vector. Use Trigger B to enable the ADC ping pong mode to have another conversion for the dc bus voltage after two currents have been converted. The value for Trigger B delay register should be pwm period+deadtime/2+ADC conversion time for one S/C.

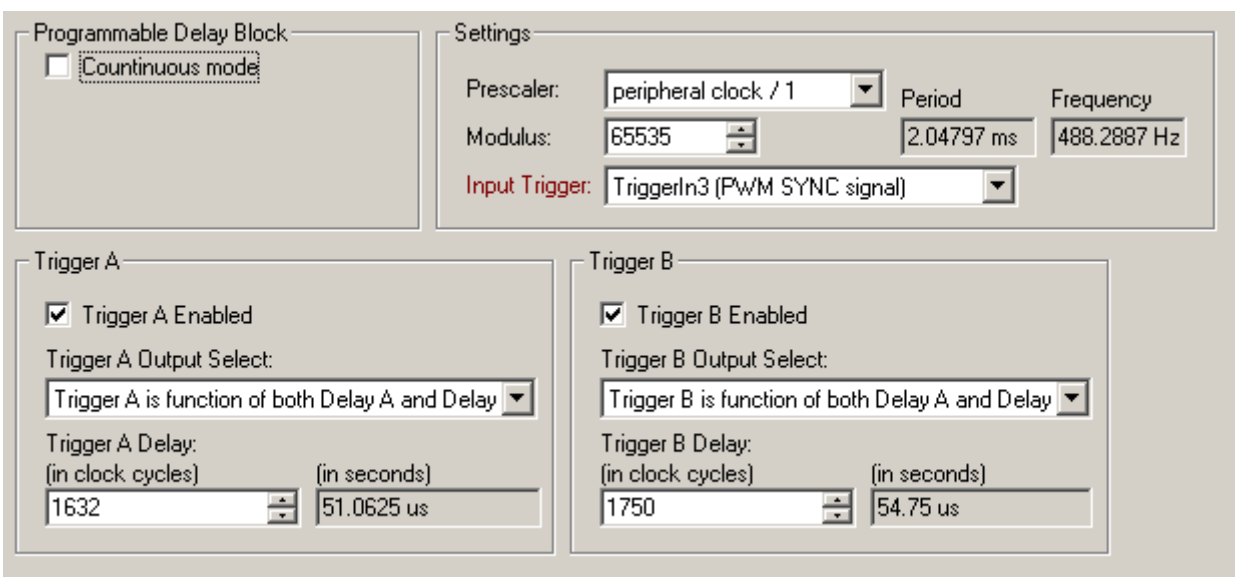

Figure 18. PDB configuration

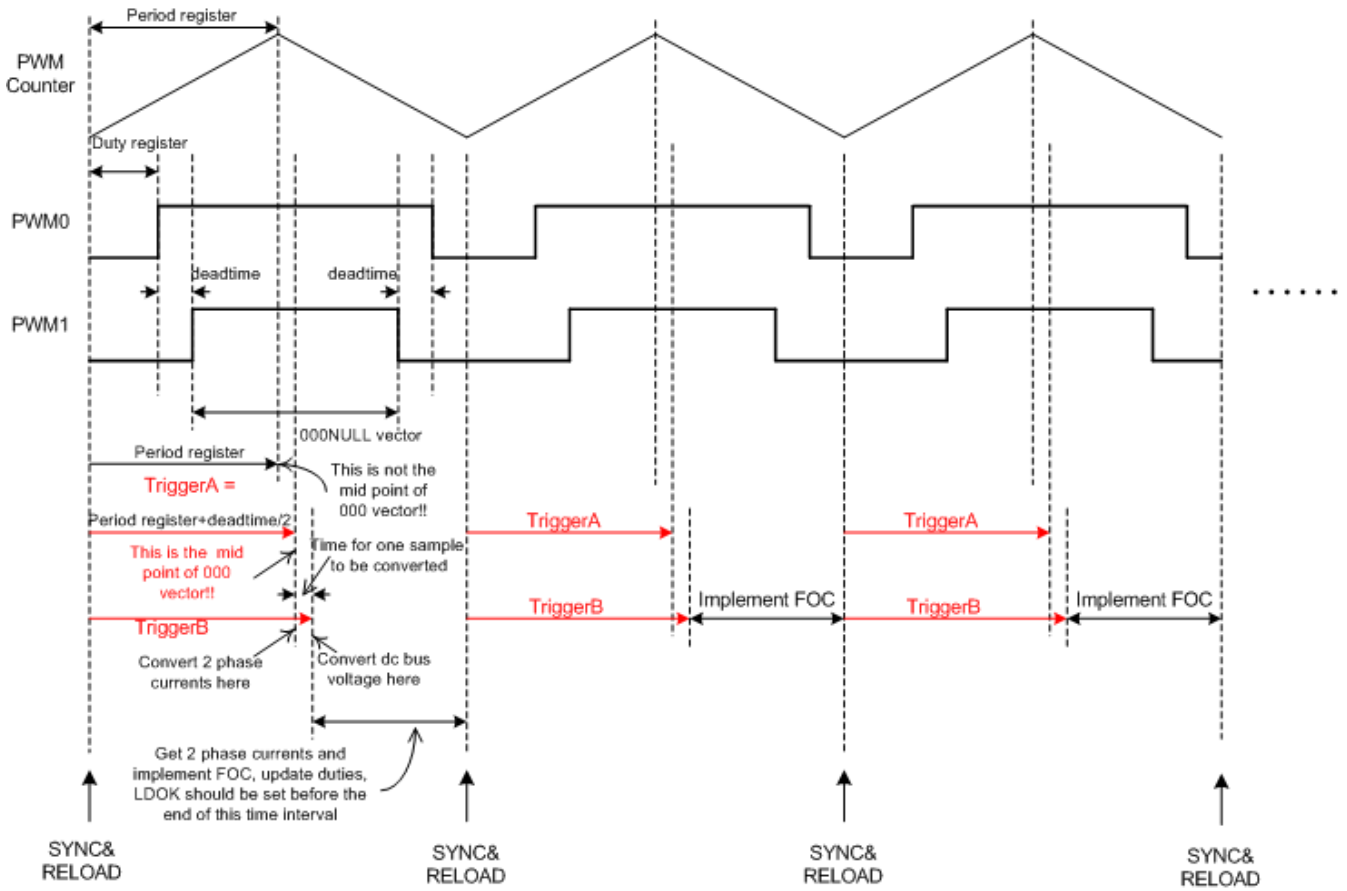

Figure 19. Setup value for Trigger A and Trigger B

The PWM clock is configured to 96MHz while the system clock is 32 MHz. Figure 19 shows 2 phase currents to be Sampled and Converted (S/C) at the point where delay for Trigger A ends, and the dc bus voltage has started to S/C at the point the where delay for Trigger B ends. Because the same ADC module is used to convert both 2 phase currents and dc bus voltage, you must assure the time between those two points is long enough to S/C 2 phase currents. In Figure 21 and Figure 22 it takes 3.03125 us to sample and convert 2 phase currents. The 118 system clock is 118/32=3.6875 us to assure that S/C for 2 phase currents is larger than 3.03 us.

2.2.2.7 ADC configuration

There are two independent ADC modules in the 56F8006 called ADC0 and ADC1. Each ADC module can convert up to 32 channels. There are a maximum of 14 channels for each ADC module with ANA0~ANA13 for ADC0 and ANB0~ANB13 for ADC1.

There are 2 control registers for each ADC module: ADC0_ADCSC1A and ADC0_ADCSC1B for ADC0, ADC1_ADCSC1A and ADC1_ADCSC1B for ADC1. For example ADC0, ADC0_ADCSC1A can tell ADC0 to sample and convert a channel specified in this register and the result is ADC0_ADCRA. ADC0_ADCSC1B can also tell ADC0 to sample and convert a channel specified in this register and the result is ADC0_ADCRB. ADC0 can be under control by either ADC0_ADCSC1A or ADC0_ADCSC1B.

Configure both ADC0 and ADC1 to be triggered by the hardware and disable the PGA module, ADC0 and ADC1 are completely controlled by the PDB trigger signals, see Figure 20.

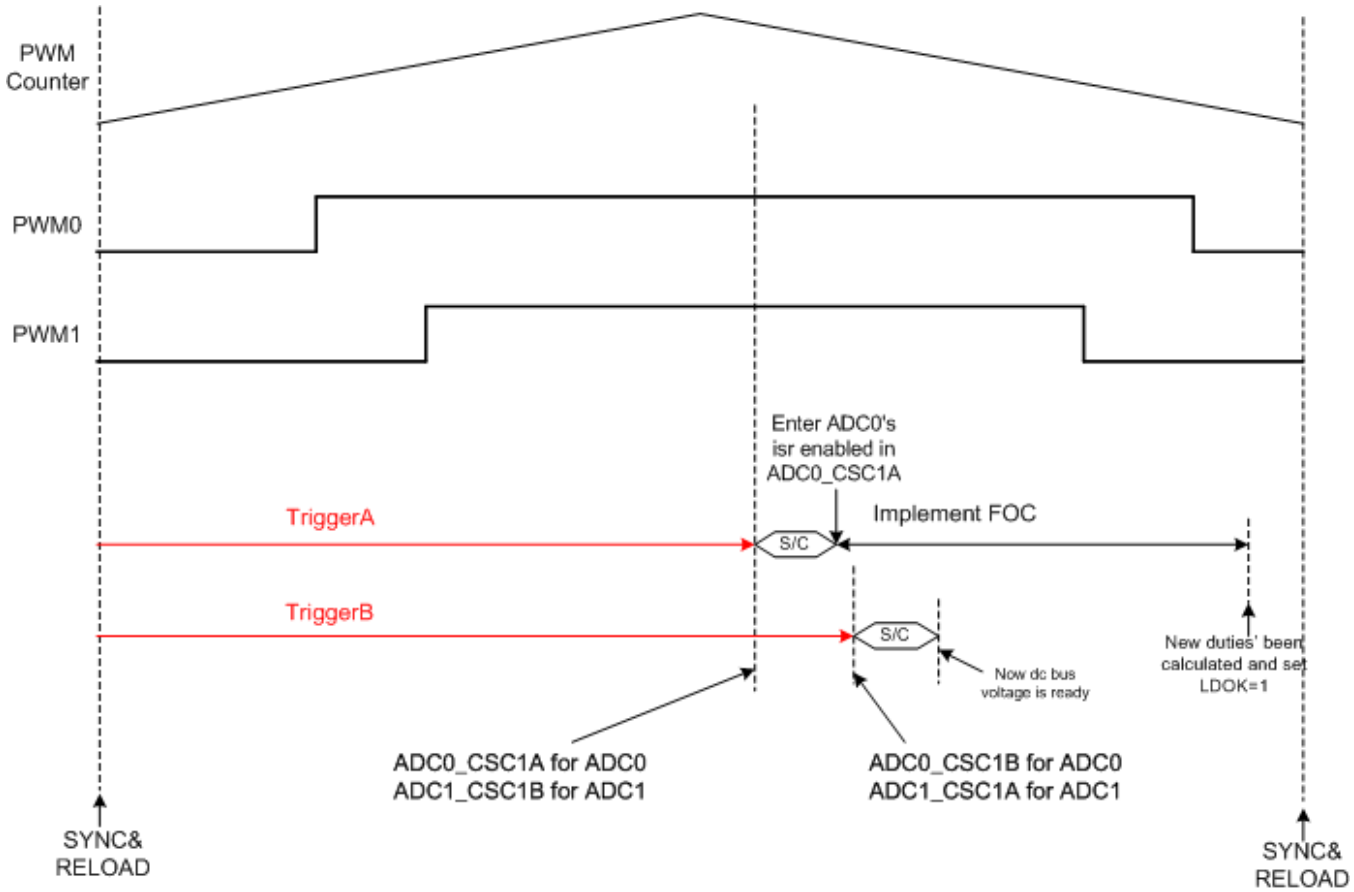

Figure 20. Ping-Pong mode for ADC0 and ADC1

When Trigger A signal occurs, ADC0 samples and convert (S/C) the channel defined in ADC0_CSC1A while ADC1 will sample and convert the channel defined in ADC1_CSC1B. Trigger A is used to convert two phase currents as mentioned in PDB configuration. After ADC0 and ADC1 finish their conversion, Trigger B signal occurs and ADC0 will sample and convert the channel defined in ADC0_CSC1B, while ADC1 samples and converts the channel defined in ADC1_CSC1A. Usually Trigger B is used to convert the dc bus voltage and another analog signal.

This mode supported by PDB and ADC0/1 is called Ping-Pong mode.The interrupt enable bit is usually set in ADC0_CSC1A to be 1, this way the ADC interrupt service routine (ISR) is entered the moment two phase currents are converted. Field Oriented Control algorithm is executed in this ADC ISR, new duties for three legs must be calculated before the next reload signal occurs, see Figure 20.

The configuration interface of ADC module are shown in Figure 21 and Figure 22 shows.

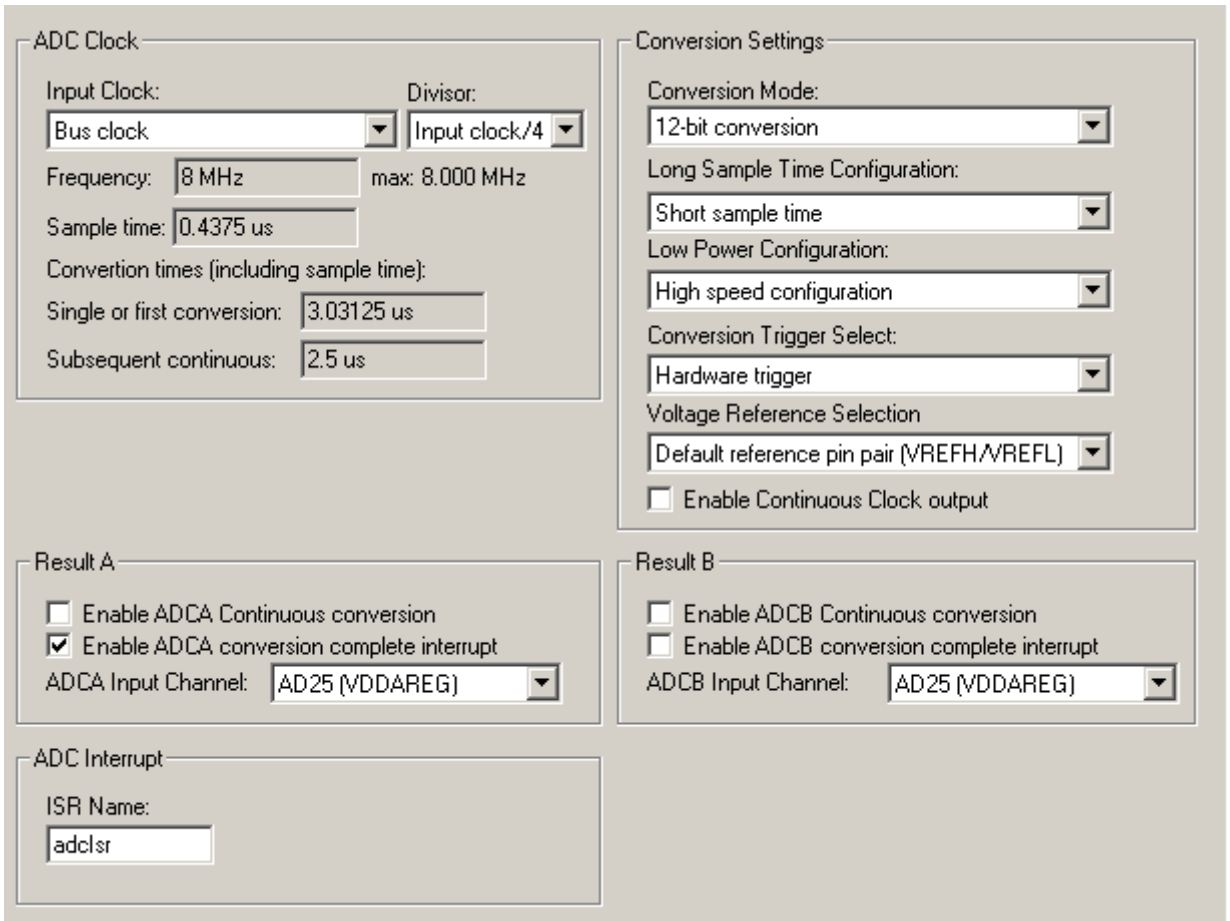

Figure 21. ADC0 configuration

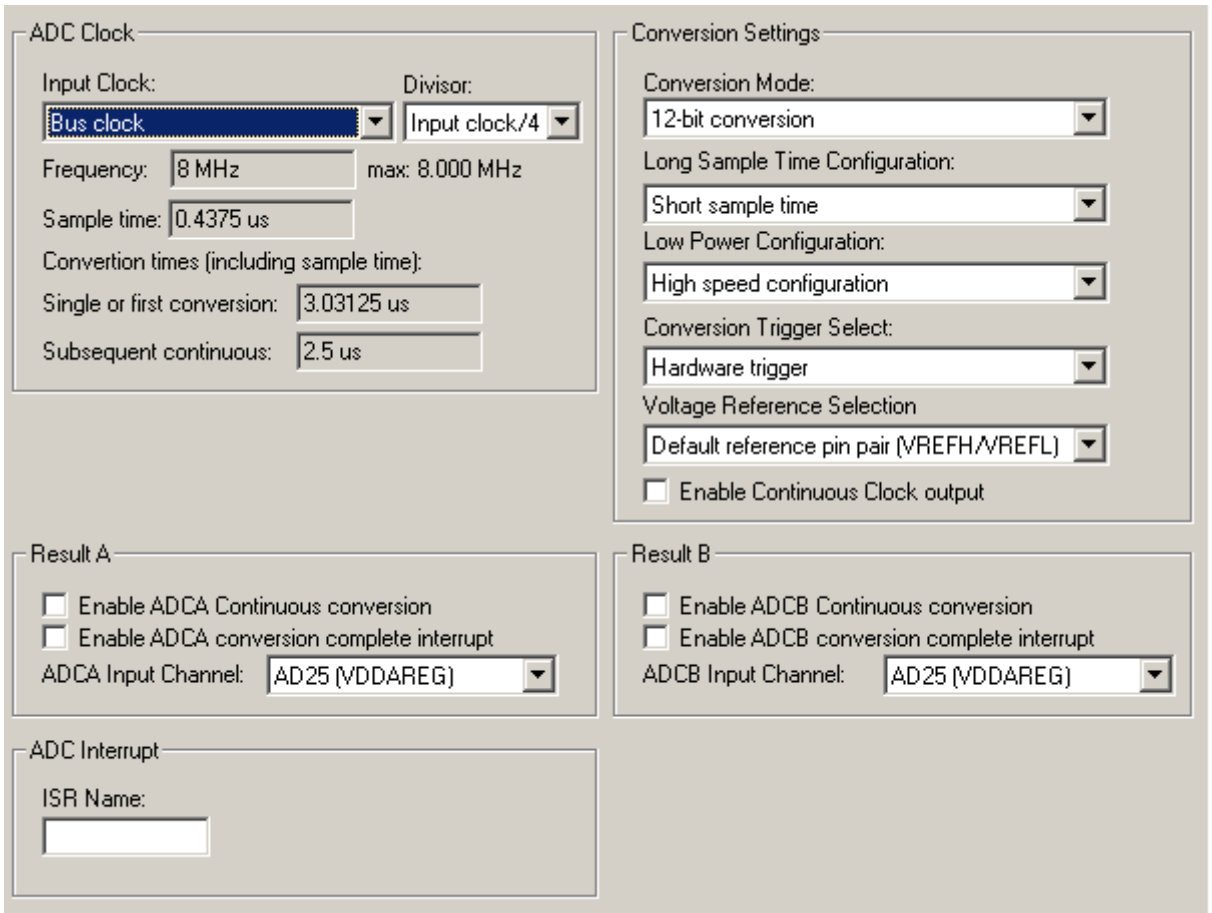

Figure 22. ADC1 configuration

3 Build FOC code structure using Quick-Start API

Quick Start provides lots of API (Application Program Interface) functions for every module in the DSC series. These functions have the same function name and unified parameter style which makes DSC easy to use and makes code easy to understand, you do not have to memorize the register bit definitions, use this API to configure all the modules.

The unified form of these API functions — ioctl (peripheral_module_identifier, command, command_specific_parameter);

You can find all the reference and detailed information about these APIs in the DSP56800E Quick Start Users Manual after installing Quick Start.

The code together with this application note provides a general structure and some key functions for the FOC algorithm based on the 3-Phase BLDC/PMSM Low-Voltage Motor Control Drive board and the MC56F8006 Controller Daughter Board.

3.1 Code structure

The entire code structure is simple, the motor control related algorithm is executed in ADC ISR. There is a state machine that makes it easy to add some customized functions. See Figure 23 and Figure 24 for the flowchart structure.

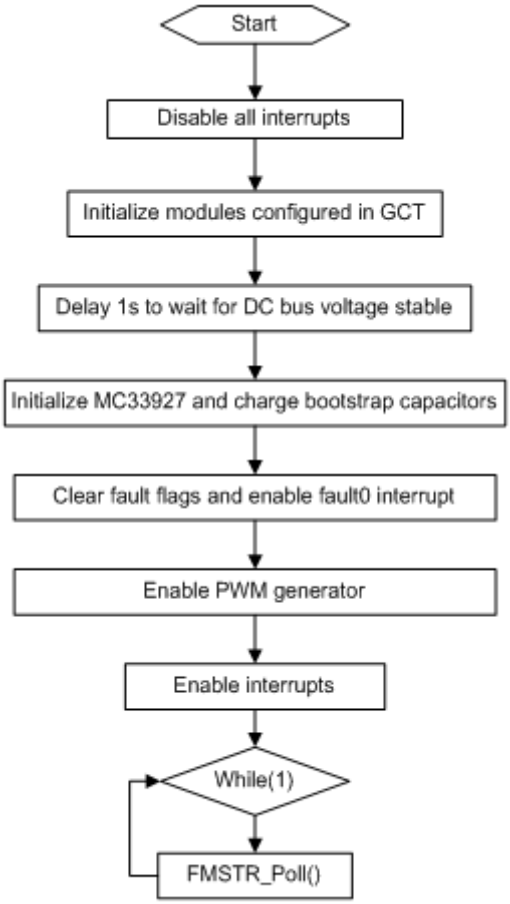

Figure 23. Main structure of the code

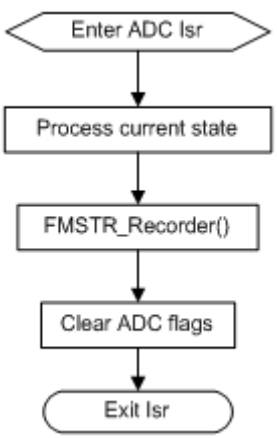

Figure 24. ADC ISR

It is important to enter the ADC interrupt the moment two phase currents are converted to make the most of the time executing algorithm. See Figure 20.

3.2 Key codes explanation

Some important uses for Quick Start API functions are pointed out in the following paragraph which includes module initialization, clearing fault flags, charging bootstrap capacitors, updating duties, and so on.

3.2.1 Initialize modules as configured in GCT

In Peripheral configuration using GCT, you can configure the modules needed, but it does not generate all the initialization codes automatically, invoke API to execute initialization as below:

```
    /* initialize SYS module */
    ioctl(SYS, SYS_INIT, NULL);
    /* configure all GPIO modules */
    ioctl(GPIO, GPIO_INIT_ALL, NULL);
    /* initialize PWM module */
    ioctl(PWM,PWM_INIT,NULL);
    /* initialize SPI module */
    ioctl(SPI,SPI_INIT,NULL);
    /* initialize PDB module */
    ioctl(PDB,PDB_INIT,NULL);
    /* initialize ADC module */
    ioctl(ADC0,ADC_INIT,NULL);
    ioctl(ADC1,ADC_INIT,NULL);
```

3.2.2 Clear fault flags

Fault flags can be set when the board is powered up. It is best to clear these flags before enabling fault interrupt:

```
/* enable Fault0's interrupt  */
ioctl(PWM,PWM_CLEAR_FAULT_FLAG,PWM_FAULT_0|PWM_FAULT_1|PWM_FAULT_2|PWM_FAULT_3); // Clear
fault flags
ioctl(PWM,PWM_FAULT_INT_ENABLE,PWM_FAULT_0);  // enable fault0 interrupt
```

3.2.3 Charge bootstrap capacitors

Most of the conventional non-isolated inverters for motor control use only one power supply that share the same ground with the Dc bus voltage to drive all six switches (either IGBT or MOSFET). It has the issue of how to drive the three upper switches. The bootstrap circuits solve this problem. This circuit is inside of the driving chip MC33927, the bootstrap capacitor of one switch in the upper leg is charged whenever the corresponding lower leg is on. But, because all switches are turned off when boards are powered up, it is essential to charge the three bootstrap capacitors before using the inverter.

```
/* Begin-Charge bootstrap capacitors  */
    // A0,A2,A4 are three upper legs controlling pwm io
    // A1,B0,A5 are three lower legs controlling pwm io
    ioctl(GPIO_A,GPIO_SET_PIN,BIT_0|BIT_2|BIT_4); // A0,A2,A4 = 1 to disable
                                                                  upper legs
    ioctl(GPIO_A,GPIO_CLEAR_PIN,BIT_1|BIT_5);
    ioctl(GPIO_B,GPIO_CLEAR_PIN,BIT_0);           // A1,A5,B0 = 0 to disable
                                                                  lower legs
    ioctl(GPIO_A,GPIO_SETAS_OUTPUT,BIT_0|BIT_1|BIT_2|BIT_4|BIT_5);
    ioctl(GPIO_B,GPIO_SETAS_OUTPUT,BIT_0); // set all PWM outputs as outputs
    ioctl(GPIO_A,GPIO_SETAS_GPIO,BIT_0|BIT_1|BIT_2|BIT_4|BIT_5);
    ioctl(GPIO_B,GPIO_SETAS_GPIO,BIT_0);   // set all PWM outputs as IO
    // turn on 3 bottom Mos
    ioctl(GPIO_A, GPIO_SET_PIN, BIT_1|BIT_5);
    ioctl(GPIO_B, GPIO_SET_PIN, BIT_0);
    Cpu_Delay(200);
    // turn off 3 bottom Mos
    ioctl(GPIO_A, GPIO_CLEAR_PIN, BIT_1|BIT_5);
```

```
    ioctl(GPIO_B, GPIO_CLEAR_PIN, BIT_0);
     Cpu_Delay(200);
    // turn on 3 top Mos
    ioctl(GPIO_A, GPIO_CLEAR_PIN, BIT_0|BIT_2|BIT_4);
    Cpu_Delay(200);
    // turn off 3 top Mos
    ioctl(GPIO_A, GPIO_SET_PIN, BIT_0|BIT_2|BIT_4);
    // change 6 pwm IO back to pwm functions
    ioctl(GPIO_A, GPIO_SETAS_PERIPHERAL, BIT_0|BIT_1|BIT_2|BIT_4|BIT_5);
    ioctl(GPIO_B, GPIO_SETAS_PERIPHERAL, BIT_0);  //set all 6 pwm IO as
                                          peripheral controlled by PWM module
    /* End-Charge bootstrap capacitors  */
```

Configure the 6 PWM outputs to be IO and turn-on the three lower MOSFETs to charge the bootstrap capacitors. It is intuitive and convenient using API to configure IO.

3.2.4 Update duties

Quick Start provides many APIs and one which is for updating duties in a concise way.

```
static MCLIB_3_COOR_SYST_T       mcDutyABC;      // duty
       mcDutyABC.f16A = FRAC16(0.5); // set all duty to be 50%,enable pwm output
       mcDutyABC.f16B = FRAC16(0.5);
       mcDutyABC.f16C = FRAC16(0.5);
       ioctl(PWM,PWM_UPDATE_VALUE_REGS_COMPL,&mcDutyABC);
```

You should be concerned with the duty percentage, it will calculate three duties according to the percentage and PWM period, update three duties, and set LDOK to 1.

3.2.5 State machine

All motor control related behaviors are executed using the state machine, see Figure 25.

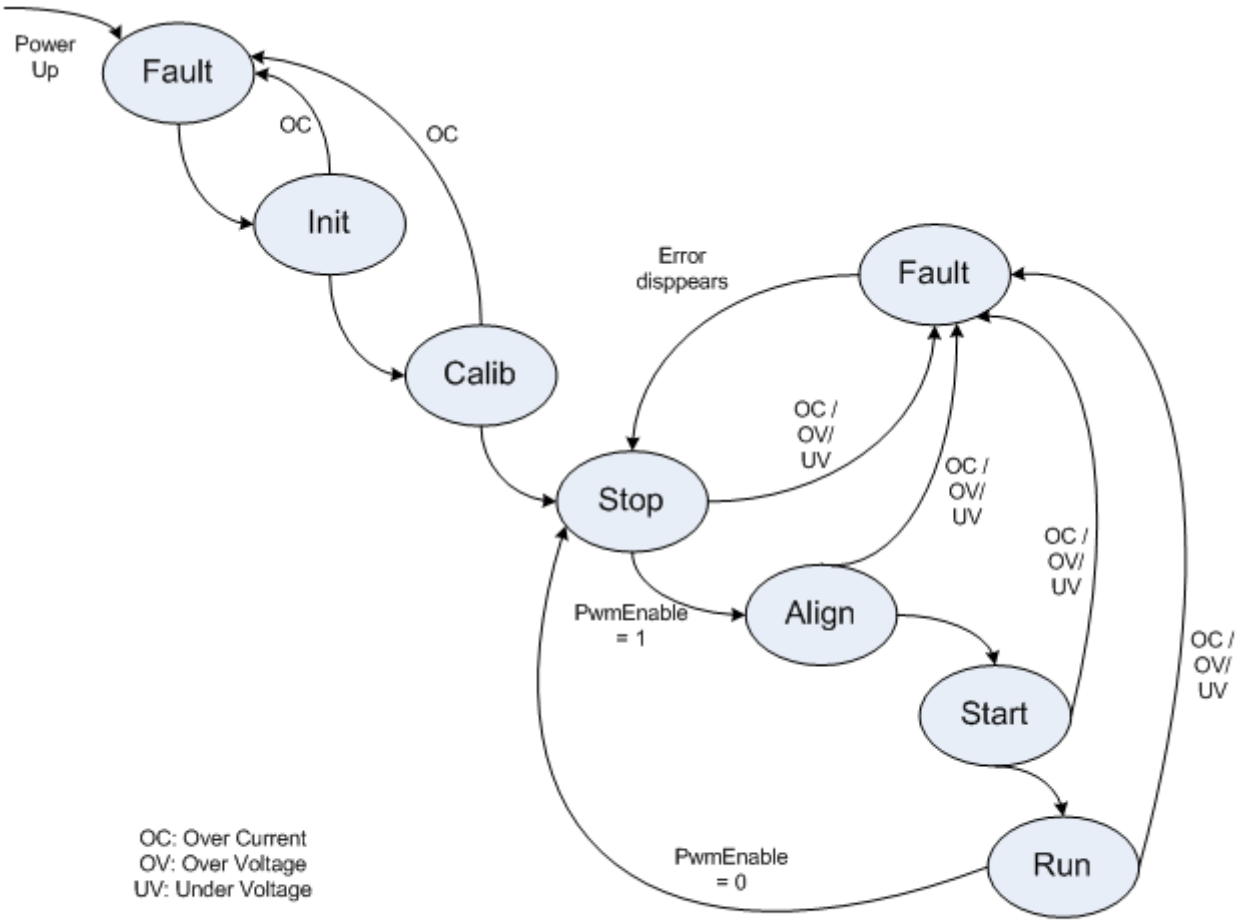

Figure 25. State machine

Because the PWM is enabled all the time, the ADC is triggered every PWM period at midpoint of the 000 NULL voltage vector through the PDB. Use ADC ISR to update this state machine.

```
/* pointer to function */
typedef void (*PFCN_VOID_VOID)(void);
static  void      TaskFault(void);
static  void      TaskInit(void);
static  void      TaskAlignment(void);
static  void      TaskRun(void);
static  void      TaskCalibration(void);
static  void      TaskStart(void);
static  void      TaskStop(void);
static  PFCN_VOID_VOID  DispatchTaskFast[] = { TaskFault, TaskInit,
TaskAlignment,                          TaskRun, TaskCalibration,TaskStart, TaskStop };
```

The ADC ISR routine is as follows:

```
#pragma interrupt saveall
void adcIsr(void)
{
    /* Begin - task dispatcher for fast inner control loop */
    DispatchTaskFast[eTASK]();
    /* End - task dispatcher for fast inner control loop */
    /* FreeMASTER recorder engine */
```

```
    FMSTR_Recorder();
    ioctl(ADC0,ADC_READ_SAMPLE_B,NULL); // dummy read of ADC results to clear flags
    ioctl(ADC0,ADC_READ_SAMPLE_A,NULL);
    ioctl(ADC1,ADC_READ_SAMPLE_B,NULL);
    ioctl(ADC1,ADC_READ_SAMPLE_A,NULL);
}
#pragma interrupt off
```

When the board is powered up, it enters a Fault state to check all fault signals and then back to the Init state to initialize all variables used in the algorithm. Next, calibrate ADC channels for currents. The reason for this calibration is that the current of –4 A~4 A corresponds to 0~3.3 V in the ADC channel input. There is an offset when current is zero. A 50% duty PWM is generated to get these offsets because only NULL voltage vectors are applied on the motor. There is no current in the three phases.

3.2.6 ADC channel mapping

A total of three phase current values are needed in the FOC algorithm, but the sum of these three current values are always zero at any point, so it is enough to get two phase current values using the ADC module and the other one can be calculated. There are some extreme circumstances where currents can not be sampled properly if sampling the constant two currents like phase A and phase B.

Figure 26 shows a voltage vector whose position is near basic vector 100 (in the ABC sequence). This vector lies in sector 1 and is composed of 100 and 110 vectors.The 100 vector lasts a long time while 110 will not. The corresponding PWM waveform is shown in Figure 27 . The lasting time of 000 NULL vector is short and because there could be spikes superposed on the current waveform of phase A, current spike could be sampled in phase A instead of the real current value.

To avoid this situation, sample current B and C when the voltage vector lies in sector 1 and 6; sample current A and C when voltage vector lies in sector 2 and 3; sample current A and B when voltage vector lies in sector 4 and 5.

In the case based on the 3-Phase BLDC/PMSM Low-Voltage Motor Control Drive board together with the MC56F8006 Controller Daughter Board, current of phase A is routed to both ADC0 and ADC1 channels, current of phase C is also routed to both ADC0 and ADC1 channels, and current of phase B is routed to a channel of ADC1, this setting makes it possible to sample different two phase currents according to the sector where the voltage vector is in. It is only essential to route one of the three phase currents to both ADC0 and ADC1 channels, one of the rest two currents to the ADC0 channel and the other one to the ADC channel. This means at least four ADC channels are in needed

AdcMapping (uw16Sector) changes channels according to the sector.

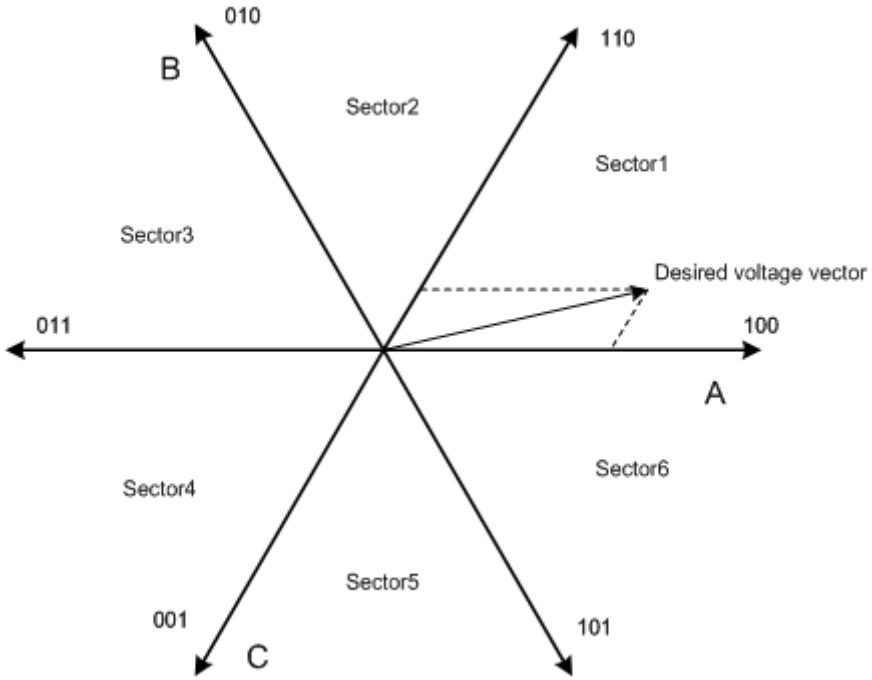

Figure 26. Voltage vector near basic vector

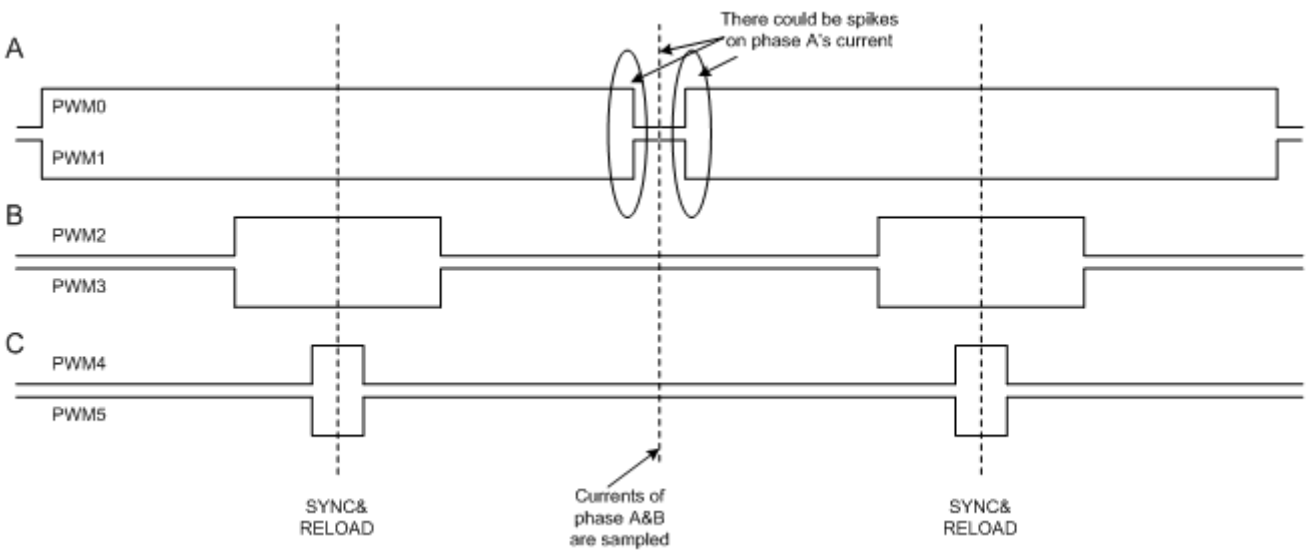

Figure 27. PWM waveform corresponding to Figure 26

```
#define ADC0_I_A ADC_AD7
#define ADC0_I_C ADC_AD9
#define ADC0_V_DC ADC_AD5
#define ADC1_I_B ADC_AD6
#define ADC1_I_C ADC_AD8
#define ADC1_I_A ADC_AD4

void AdcMapping(UWord16 uw16Sector)

{

     switch(uw16Sector)
```

```
{
            case 1:
            case 6:
           default:
                // measure current C,B
                ioctl(ADC0,ADC_SET_INPUT_CHANNEL_A,ADC0_I_C);
                ioctl(ADC1,ADC_SET_INPUT_CHANNEL_B,ADC1_I_B);
                ioctl(ADC0,ADC_SET_INPUT_CHANNEL_B,ADC0_V_DC);
                ioctl(ADC1,ADC_SET_INPUT_CHANNEL_A,ADC1_I_A);
                break;
            case 2:
            case 3:
                // measure current A,C
                ioctl(ADC0,ADC_SET_INPUT_CHANNEL_A,ADC0_I_A);
                ioctl(ADC1,ADC_SET_INPUT_CHANNEL_B,ADC1_I_C);
                ioctl(ADC0,ADC_SET_INPUT_CHANNEL_B,ADC0_V_DC);
                ioctl(ADC1,ADC_SET_INPUT_CHANNEL_A,ADC1_I_B);
                break;
            case 4:
            case 5:

                // measure current A,B
                ioctl(ADC0,ADC_SET_INPUT_CHANNEL_A,ADC0_I_A);
                ioctl(ADC1,ADC_SET_INPUT_CHANNEL_B,ADC1_I_B);
                ioctl(ADC0,ADC_SET_INPUT_CHANNEL_B,ADC0_V_DC);
                ioctl(ADC1,ADC_SET_INPUT_CHANNEL_A,ADC1_I_C);
                break;

        }
}
```

4 Conclusion

A tool called Quick Start is introduced for building FOC code structure based on the 56F8006. Detailed settings and codes are given in this application note. This code structure has proven to be suitable for PMSM field oriented control.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4490
Rev. 0, 3/2012