Freescale Semiconductor, Inc.
Application Note

Document Number: AN4911
Rev. 1, 02/2015

PMSM Sensorless Control on TWR-KV31F120M Using Kinetis SDK

by: Marek Zeman

1 Introduction

This application note represents an addendum to *DRM140 – Sensorless PMSM Field-Oriented Control*. It describes the MCU peripherals used in the application, the hardware setup and results of the measurement built under the Freescale SDK suite for Kinetis MCUs.

Contents

2 MCU peripherals

Table 1 summarizes the peripherals of the Kinetis KV31F120M MCU and their usage in the PMSM sensorless vector control application:

Table 1. Kinetis KV31F120M peripherals overview

Kinetis KV31F120M peripherals			Used in the application	Purpose
Group	**Module**	**Number of modules or channels**	–	–
Analog	ADC0	24 single-ended channels, four of them in differential pairs	Three channels	DC-bus voltage and motor phase currents sensing
–	ADC1	24 single-ended channels, four of them in differential pairs	Two channels	–
–	Comparators	Two modules, each has seven channels	–	–
–	DAC	One module	–	–
Communications	SPI	Two modules, six chip-select signals on SPI0 and four chip-select signals for SPI1	One module	MOSFET driver configuration
–	UART	Three modules	One module	FreeMASTER communication
–	LPUART	One module	–	–
–	I^{2}C	Two modules	–	–
Timers	FlexTimer	Eight channels	Six channels	Generation of six-channel PWM for motor control
–	–	Two channels	–	–
–	–	Two channels	–	–
–	–	Eight channels	–	–
–	PDB	Two channels for ADC triggering	Two channels	DC-bus voltage and phase current sampling initiation
–	–	Two channels for DAC triggering	–	–
–	PIT	Four channels	–	–
–	LPT	One module	–	–
Other	eDMA	16 channels	–	–

2.1 FlexTimer configuration for generating a six-channel PWM

The FlexTimer module (FTM) is a two-to-eight channel timer which supports input capture, output compare and the generation of PWM signals to control an electric motor and power management applications. The FTM time reference is a 16-bit counter that can be used as an unsigned or signed counter. Kinetis KV31F120M has four instances of FTM, where two FTMs have eight channels and the other two FTMs have two channels.

The procedure of the FlexTimer configuration for generating a center-aligned PWM with dead time insertion is described in the *AN3729* application note [1].

Since the reference application note supports an earlier version of the FlexTimer (1.0) with respect to the hardware used (TWR-MC-LV3PH), there are a few differences needed in the configuration, described as follows:

- Enable the system clock for the FlexTimer module by:

  ```
  SIM_HAL_EnableFtmClock(SIM_BASE, 0);
  ```
- Disable the write protection of some registers before they can be updated:

  ```
  FTM_HAL_SetWriteProtectionCmd(FTM0_BASE, false);
  ```
- Enable the internal FlexTimer counter to run in the debug mode:

  ```
  FTM_HAL_SetBdmMode(FTM0_BASE, 3);
  ```

  While the hardware debugging interface (JLink, Multilink, or other) is connected to the MCU, it is in the debug mode. It is not dependent on whether the running code contains breakpoints or not.
- The PWM signals generated by the FlexTimer are directly connected to the MOSFET driver. For safety reasons, the input signals for the top transistors on the MOSFET driver used on the TWR-MC-LV3PH board have inversed polarity. Therefore, it is also necessary to set the right polarity of the PWM signals:

  ```
  FTM_HAL_SetChnOutputPolarity(FTM0_BASE, 0, 1);
  FTM_HAL_SetChnOutputPolarity(FTM0_BASE, 2, 1);
  FTM_HAL_SetChnOutputPolarity(FTM0_BASE, 4, 1);
  ```
- The duty cycle is changed by adjusting the value of the FlexTimer value registers. These registers are double-buffered, meaning that their values are updated not only by writing the number but it is necessary to confirm the change by setting the Load OK (LDOK) bit. This ensures that all values are updated at the same instance:

  ```
  FTM_HAL_SetPwmLoadCmd(FTM0_BASE, true);
  ```

  It is necessary to write the LDOK bit every time the value registers are changed (not only at the stage of loading them with initial values, but with every update after the duty cycle value is computed in the vector control algorithm).
- As mentioned in Section 2.3, “ADC conversion timing, currents and voltage sampling,” hardware triggering of the AD converter is employed in the application. The Initialization Trigger signal from the FlexTimer is used as the primary triggering signal and is fed into the Programmable Delay Block that services the timing of the AD conversion initiation.

  ```
  FTM_HAL_SetInitTriggerCmd(FTM0_BASE, true);
  ```
- The output pins of the MCU have to be configured in order to bring the signals out of the chip. The assignment of signals to output pins is set in the Pin Control Register, while the available signals are listed in the Signal Multiplexing chapter of reference manual *KV31P100M120SF7RM* [2] and are package-dependent.

  ```
  PORT_HAL_SetMuxMode(PORTC_BASE, 1, kPortMuxAlt4);  // FTM0 CH0, PTC1
  PORT_HAL_SetMuxMode(PORTC_BASE, 2, kPortMuxAlt4);  // FTM0 CH1, PTC2
  PORT_HAL_SetMuxMode(PORTC_BASE, 5, kPortMuxAlt7);  // FTM0 CH2, PTC5
  PORT_HAL_SetMuxMode(PORTC_BASE, 4, kPortMuxAlt4);  // FTM0 CH3, PTC4
  PORT_HAL_SetMuxMode(PORTD_BASE, 4, kPortMuxAlt4);  // FTM0 CH4, PTD4
  PORT_HAL_SetMuxMode(PORTD_BASE, 5, kPortMuxAlt4);  // FTM0 CH5, PTD5
  ```

The port settings implemented in the application code reflect the hardware solution built on the Tower System board.

2.2 Configuring the ADC and PDB modules

The on-chip ADC module is used to sample feedback signals (motor phase currents and DC-bus voltage) that are necessary to successfully perform the vector control algorithm. The Programmable Delay Block (PDB) closely cooperates with the ADC and serves for hardware triggering of sampling.

It is required to perform a self-calibration procedure of the ADC module before it is used in the application in order to obtain specified accuracy. The calibration process also requires programmer’s intervention to generate the plus-side and minus-side gain calibration results and store them in the ADC plus-side gain and minus-side gain registers after the calibration function completes. The calibration has to be performed for both ADC modules. After the calibration, the ADC modules are configured for a 12-bit accuracy. The CPU frequency is set to 120 MHz, so by using the available pre-scaler values the input clock to the ADC module is set to 15 MHz. That setting yields a conversion time of 2.2 µs (33 ADC clock cycles). Finally, the hardware trigger has to be enabled in the Status and Control Register 2.

The PDB provides controllable delays from either an internal or external trigger, or a programmable interval tick to the hardware-trigger inputs of the ADCs, so that a precise timing between ADC conversions is achieved. The PDB module has an internal counter that overflows on a modulo value. Since the input trigger comes periodically from the FTM0, the value of the modulo register can be set to its reset value. The values in the channel delay registers are set to generate triggers to start sampling the DC-bus voltage and the motor phase AD conversions. The PDB module on the KV31F120M MCU allows for 15 different input trigger sources. They are listed in the “Chip configuration” chapter in the "PDB Configuration” section of the device reference manual *KV31P100M120SF7RM* [2]. For FTM0, the LDOK bit has to be set in order to acknowledge the changes in the delay registers.

2.3 ADC conversion timing, currents and voltage sampling

The FTM0 is configured to trigger an internal hardware signal when its counter is reset to the initialization value after overflow. This signal is supplied into the PDB which consequently triggers the AD conversion of the voltage and currents with a pre-defined delay. There are two ADC modules implemented on the Kinetis KV31F120M 120 MHz MCU. Each ADC module associates to one channel of the PDB module. Each ADC module has two result registers (two channels) and they correspond to two programmable pre-trigger delays of the PDB channels. It is possible to perform four AD conversions without requesting an interrupt (with respect that the DMA is not used for data transfer). In this application, only three conversions need to be triggered without CPU intervention (two motor phase currents and the DC-Bus voltage). Figure 1 shows the interconnection of modules and the ADC interrupt generation.

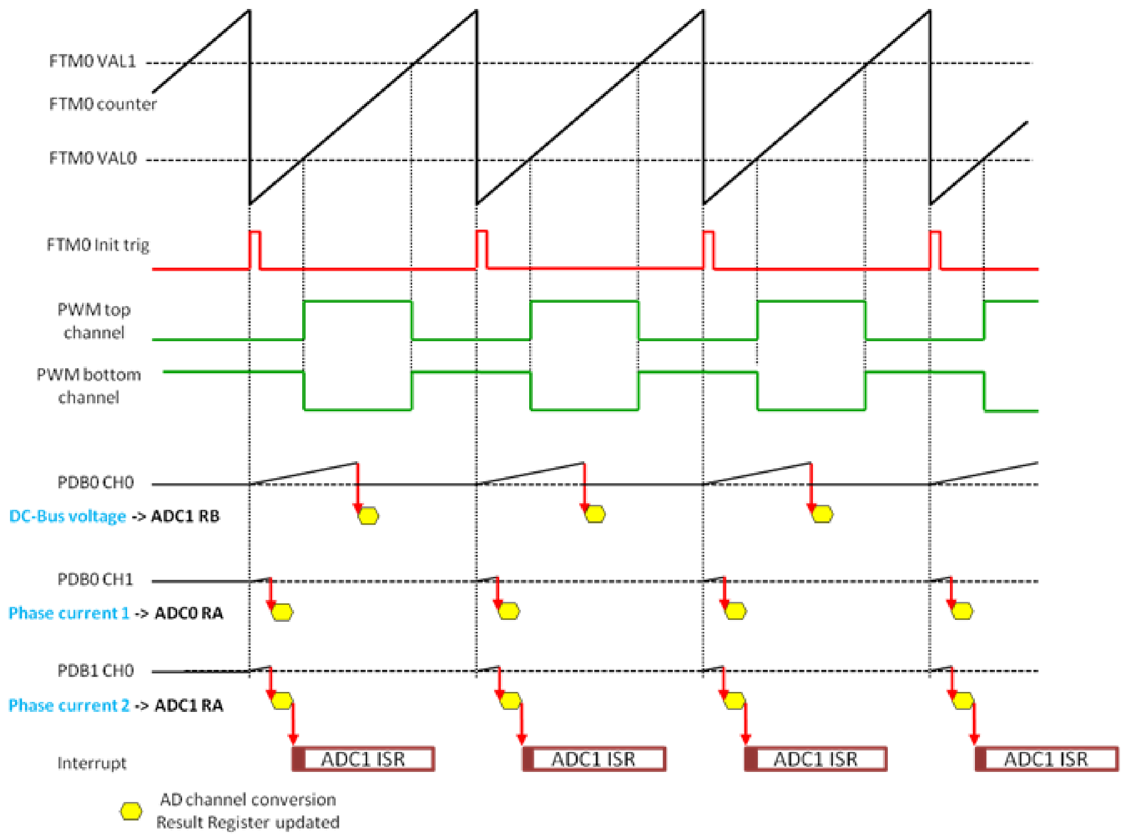

FTM0 VAL1
FTM0 counter
FTM0 VAL0
FTM0 Init trig
PWM top channel
PWM bottom channel
PDB0 CH0
DC-Bus voltage -> ADC1 RB
PDB0 CH1
Phase current 1 -> ADC0 RA
PDB1 CH0
Phase current 2 -> ADC1 RA
Interrupt
ADC1 ISR
AD channel conversion Result Register updated

Figure 1. ADC conversion timing

2.4 Current measurement

The assignment of the ADC channels to the measured analog signals is closely related to the ADC conversion trigger timing. For computation of the fast (current) control loop of the FOC, it is necessary to know the values of all three motor phase currents. Since there are only two ADC modules, it is possible to sample only two analog quantities at one instance. Assuming the motor represents a symmetrical three-phase system, the sum of all three instantaneous phase currents is zero.

$$0 = i_A + i_B + i_C \qquad \text{Eqn. 1}$$

Since the phase currents are measured in the instance when the bottom transistors are conducting, the time when the current can be measured is too short, especially in the case of high duty cycle ratios (current value is in the area of the maximum of the sine curve). The bottom transistor must be switched on at least for a critical pulse width to get a stabilized current shunt resistor voltage drop. The selection of the channels is based on the section where the space vector of the stator current is generated. This assignment is performed at the end of the ADC1 interrupt service routine. Therefore, it is enough to sample only two phase currents while the third can be easily calculated according to Equation 2:

$$\begin{gathered} \text{Sector 1, 6: } i_A = -i_B - i_C \\ \text{Sector 2, 3: } i_B = -i_A - i_C \\ \text{Sector 4, 5: } i_C = -i_B - i_A \end{gathered} \qquad \text{Eqn. 2}$$

Figure 2 explains the reason why the third current cannot be measured directly in some time instances. Case I shows the phase currents at 30° and Case II at 60°.

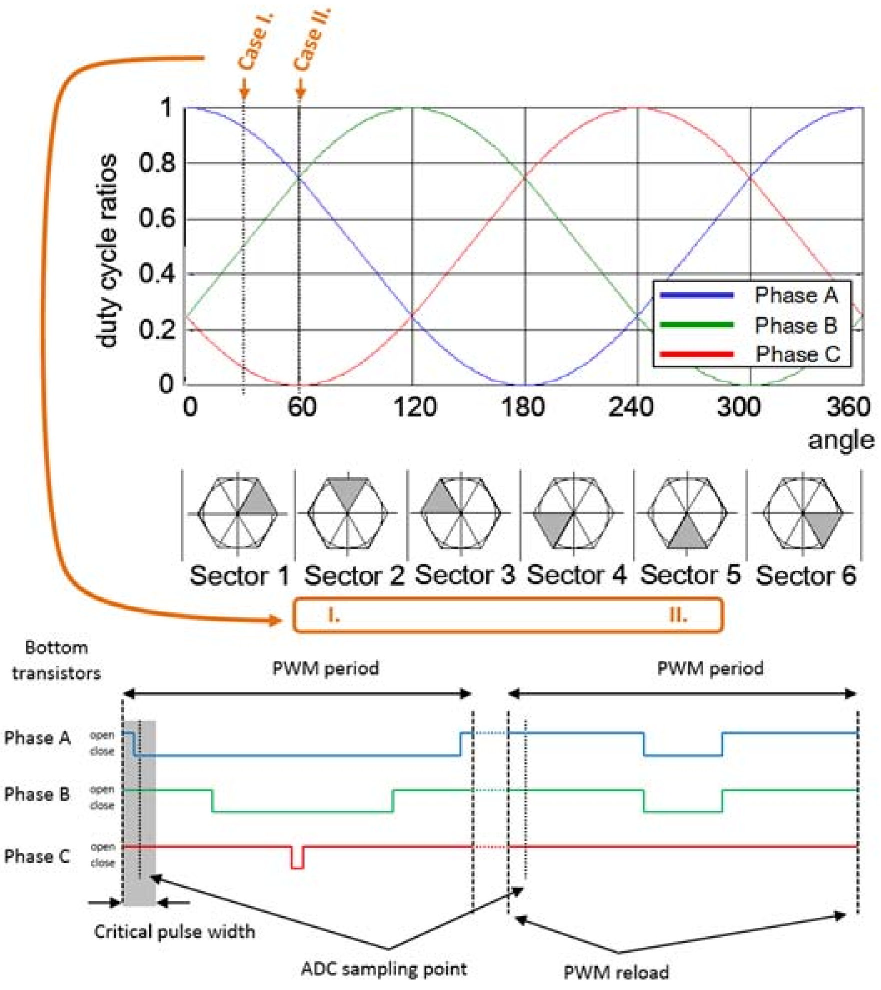


Figure 2. Current sensing

2.5 Over-current level

The over-current signal is connected via the TWR-Elevator IRQ_A (B62) pin to the GPIO_C9 pin of the KV31F120M device. This pin is internally connected to the GPIO signal that handles the fault.
See Figure 3.

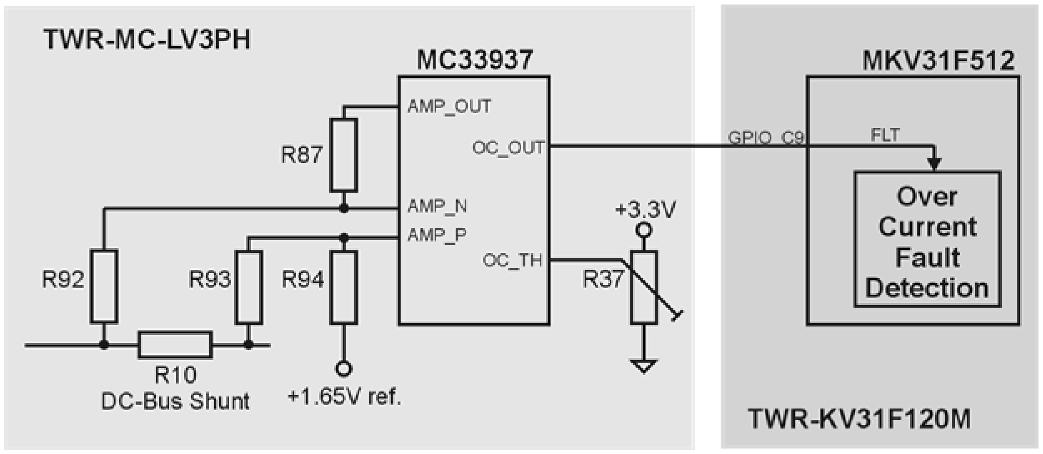


Figure 3. Over-current level

The over-current level can be set in the range of 0 ~ 8 A by the trimmer R37 on the TWR-MC-LV3PH board. The maximum current level can be set by turning the trimmer counterclockwise. The user can find the level by turning the trimmer counterclockwise while the motor is running until the red LED is lit. It is recommended to set the trimmer to a somewhat higher level so the motor can operate at maximum speed.

2.6 SPI configuration

The SPI interface is used in the application for communication between the intelligent MOSFET gate driver MC33937 and the KV31F120M MCU. The MC33937 gate driver is placed on the Tower System low-voltage power module and serves for driving the high-side and low-side MOSFET transistors of the three-phase inverter. In the application, the initialization of the MC33937 has to be performed, mainly in setting the dead time. While the motor is running, there is also a periodic checking of the status register of the driver in order to provide information on the latched faults. The MC33937 driver requires precise timing of the SPI signals. The default setting of the SPI module on the MCU is not possible to be used. The exact timing of the SPI signals is listed in the *Three Phase Field Effect Transistor Pre-driver Data Sheet* [3].

2.7 SCI (UART) configuration

The SCI is used in the application for communication between the master system and the embedded application. The master system is the PC where the FreeMASTER software is installed in order to control the application and visualize its state. On the Kinetis KV31F120M, there are three UART modules implemented. Due to the hardware solution being based on the Tower System modules, the UART0 is used. The communication speed is set to 19200 Bd.

3 Interrupts

The application requires minimum number of interrupts because of the MCU’s capability to hardware-trigger the AD conversion. There are three interrupts in the application. The ADC1 interrupt serves to execute fast-loop (current FOC) algorithms while the port interrupt handles the user buttons and PDB error interrupt clears the sequence error fault.

3.1 ADC1 interrupt

This interrupt request is triggered when the conversion of channel A of the ADC1 module is completed and has the highest priority. In the time of the interrupt generation, there are sampled values of physical quantities ready as follows:

- in the result register A of the ADC0 – motor phase current one
- in the result register A of the ADC1 module – motor phase current two
- in the result register B of the ADC1 – DC-bus voltage

The interrupt is always enabled only for one module at a time; otherwise there would be two interrupt requests generated at the same time since the triggers for the motor phase currents are generated in the same instance. The application state machine function is called in the beginning of the ADC1 ISR execution. If the application is in the RUN state then the execution of the fast (current) control loop of the PMSM vector control algorithm follows, including the position and speed estimation. The slow (speed) control loop is calculated based on the value of software counter that is decremented each time the fast control loop is passed. The interrupt flag is cleared by reading the result register of the ADC channel that triggered the interrupt. Therefore, the results of the AD conversion are read at the beginning of each particular state machine function, even when the values are not later used in the program execution.

The flowchart depicted in Figure 4 provides an overview of the program flow during the execution of the ADC interrupt service routine when the application is in the RUN state and SPIN sub-state.

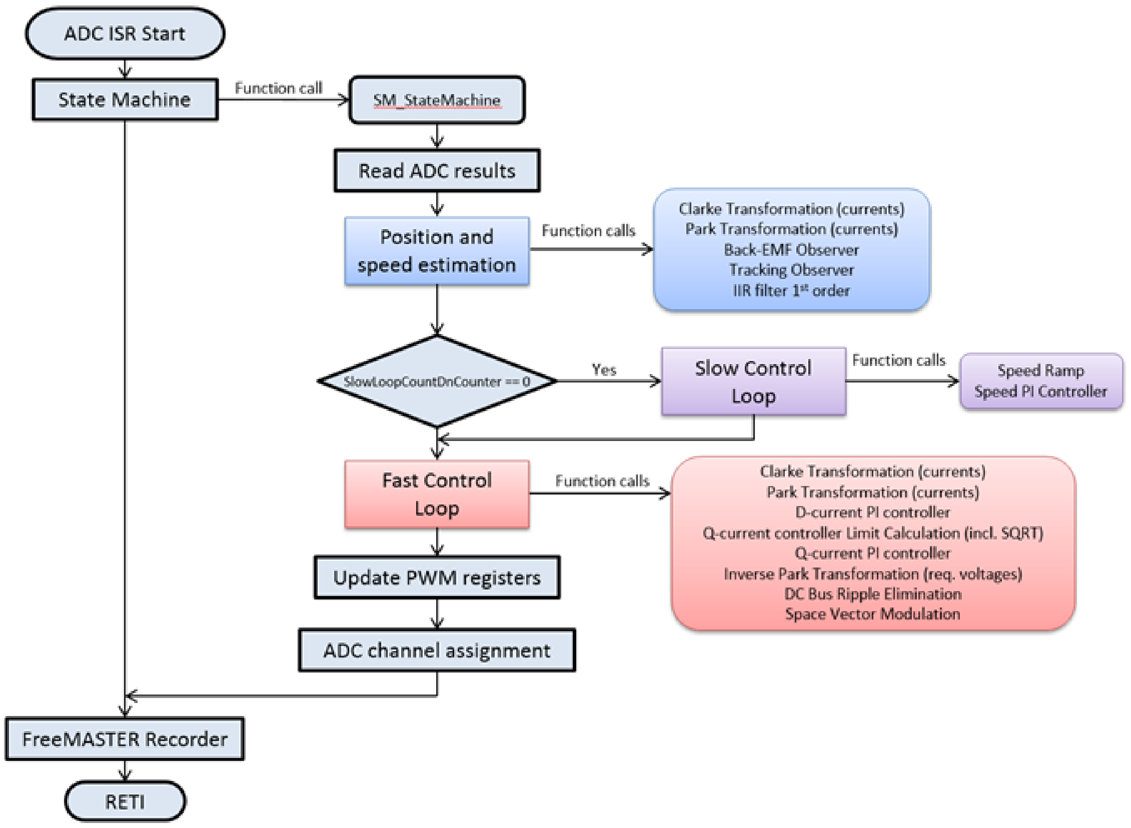


Figure 4. ADC ISR flowchart

3.2 Ports interrupt

Handling of the user's buttons on the KV31F120M Tower System board is performed in the ISR associated with the ports interrupt, generated whenever one of the buttons is pressed. At the beginning of the ISR, a simple logic is executed to evaluate which button was pressed, and the interrupt flag is cleared. Pressing the SW1 button causes the speed to increase in 10% steps. Pressing of the SW2 button causes the speed to decrease in 10% steps and also causes the transition back to STOP state.

For more information about the application control via the user buttons see *TWRKV31F120M* [4], where the use of the FreeMASTER control interface for enhanced control and diagnostics is described.

3.3 PDB error interrupt

The PDB error ISR serves for clearing the sequence error fault that is generated in situations when the PDB initiates the sampling of the AD converter, but the COCO flag in particular ADCx_SC1n register of the ADC module was not cleared (the values from result registers were not read due to multiple reasons described below). In such case, the PDB counter stops working and the interrupt is asserted. The PDB module is then re-initiated in the ISR. The PDB generates trigger signals with the same period as is the period of the ADC conversion complete interrupt, and this period is the same as the PWM period. If the user places an interrupt in the code, the execution will stop. However, the PDB will generate triggers for the next conversion even when the program execution stops. The COCO flags are not cleared and the PDB generates a sequence error. Another reason is that the execution of the ADC conversion complete interrupt (where also the fast control loop is calculated) would extend over one period of PWM. This might happen if the user put additional tasks into the ADC conversion complete interrupt. Besides the generation of the PDB sequence error, a more serious impact is on the quality of the control process, as one of the key assumptions is not met: the execution of control algorithms extends the sampling period. The real-time control application has to be designed in such a way that this situation never occurs.

4 Application operation

The application can be operated either via the user button on the TWR-KV31F120M MCU board (as mentioned in Section 3.2, "Ports interrupt") or using the FreeMASTER software, which enables visualization of the application variables. The FreeMASTER application consists of two parts: the PC application used for variables visualization, and the set of software drivers running in the embedded application. The data between the PC and the embedded application is transferred via the RS232 interface.

4.1 Installing FreeMASTER on a PC

The FreeMASTER application for PC can be downloaded from the Freescale webpage freescale.com/freemaster; from the Download tab, select "FreeMASTER 1.4 Application Installation." Since downloading of the FreeMASTER application requires registration, it is necessary to create an account before you can log-in. After logging into the system, the license agreement appears. User should read the license agreement and then accept the agreement by clicking the "I Accept" button. If using Internet Explorer, then a bar asking to authorize the file download appears at the top of the web page. Please click on the bar and select "Download File." A dialog box where the user can choose either "Run" or "Save" appears. In both cases, the installation archive will be stored on user PC.

After selecting “Save,” user has the option to select the preferred location for saving the installation archive; otherwise it will be saved to a temporary folder in user system. The library installation archive will now be downloaded to user computer.

To run the installation, please click the “Run” button. Follow the instructions on the screen to complete the installation process.

4.2 Establishing a connection between the PC and the embedded application

The FreeMASTER allows using multiple communication interfaces between the embedded application and the PC (UART (RS232), CAN, Ethernet, USB, BDM, and other). For this application, the RS232 was used because the software overhead for the data transfer represents the lowest additional load on the CPU. Nowadays, the notebooks are not equipped with the COM port, so for this purpose the TWR-KV31F120M module has a USB-to-RS232 interface in place (CDC Serial Port). By connecting the TWR-KV31F120M module to a notebook via the USB cable, a virtual serial port will be established in the Windows system.

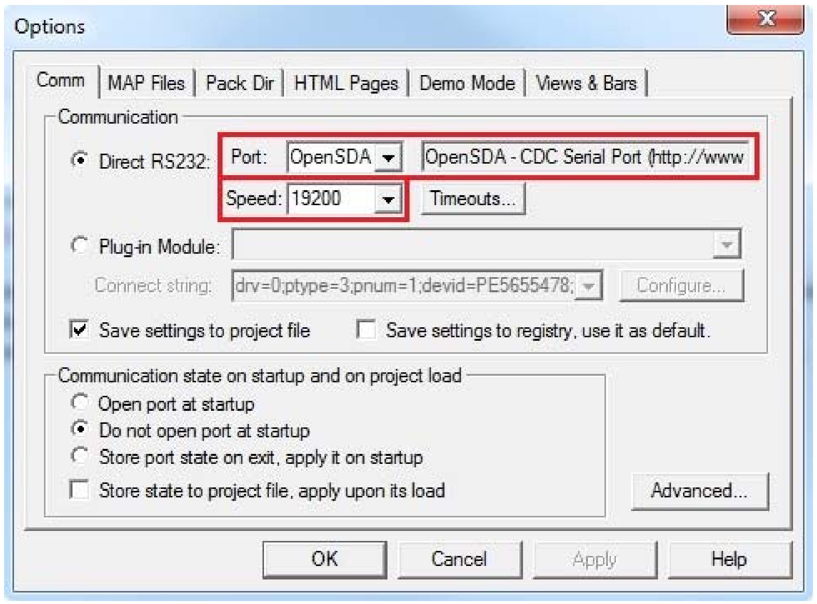
To run the FreeMASTER application, double click the *PMSM_FOC_KV31x.pmp* file located in the *\freemaster\PMSM_Sensorless_FOC_MID* folder. The FreeMASTER application starts and the environment will be automatically created, as defined in the **.pmp* file.

4.3 Setting up the communication

When the notebook is connected to the TWR-KV31F120M board via a USB cable, the operating system assigns the number of the COM port to the OpenSDA CDC serial port interface. This number is assigned randomly, therefore it is necessary to set the right communication port when the connection is established (re-plugging the USB cable might cause a different port number assignment).

To set the port number, select “Project \ Options …” from the menu.

It is necessary to assign a correct port number and/or value. Add the “OpenSDA” value to the “Port” list box and the FreeMASTER automatically assigns a correct communication channel, or select “COM” port from the dropdown listbox menu. The correct port number selection is confirmed by the text “OpenSDA – CDC Serial Port” next to the list box with the available serial port numbers. Communication speed of the interface is 19200 Bd.


Options

Comm | MAP Files | Pack Dir | HTML Pages | Demo Mode | Views & Bars

Communication

(●) Direct RS232: Port: OpenSDA | OpenSDA - CDC Serial Port (http://www

Speed: 19200 | Timeouts...

( ) Plug-in Module:

Connect string: drv=0;ptype=3;pnum=1;devid=PE5655478; | Configure...

[✓] Save settings to project file [ ] Save settings to registry, use it as default.

Communication state on startup and on project load

( ) Open port at startup

(●) Do not open port at startup

( ) Store port state on exit, apply it on startup

[ ] Store state to project file, apply upon its load

Advanced...

OK | Cancel | Apply | Help

Figure 5. FreeMASTER communication settings

4.4 Starting / stopping the communication and the application

When the communication settings are performed, the communication between the PC and the embedded application can be initiated. Click the “Stop” button in the FreeMASTER toolbar, as shown in Figure 6.

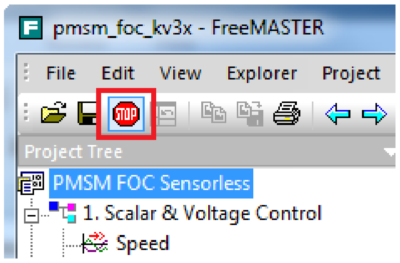
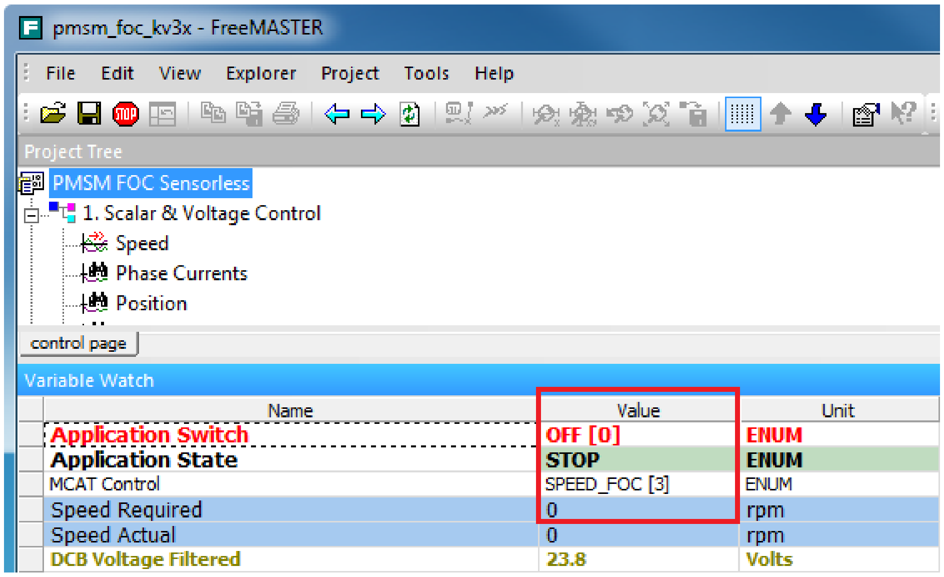


Figure 6. Initiating the communication with the embedded side

The next step is switching the application to the RUN state. Click on the drop-down list next to the “Application Switch” variable name in the FreeMASTER window in the “Variable Watch” grid and select ON, as shown in Figure 7.

Once the application is switched to the RUN state, the required speed can be changed to a non-zero value. The procedure is similar to the previous step; in the “Variable Watch” grid, enter a positive or negative value next to the “Speed required” variable name.


pmsm_foc_kv3x - FreeMASTER
File Edit View Explorer Project Tools Help
Project Tree
PMSM FOC Sensorless
1. Scalar & Voltage Control
Speed
Phase Currents
Position
control page
Variable Watch

Name	Value	Unit
Application Switch	OFF [0]	ENUM
Application State	STOP	ENUM
MCAT Control	SPEED_FOC [3]	ENUM
Speed Required	0	rpm
Speed Actual	0	rpm
DCB Voltage Filtered	23.8	Volts

Figure 7. Starting the application

4.5 MCAT tool control

To run the FreeMASTER application including the MCAT tool, double click on the *MCAT_PMSM_FOC_KV3x.pmp* file located in the *\freemaster\PMSM_Sensorless_FOC_MID* folder. The FreeMASTER application starts and the environment will be automatically created, as defined in the **.pmp* file.

The application enables tuning of the PMSM sensorless application for any motor. For this purpose, the field-oriented control can be divided into four modes as follows:

- Scalar
- Voltage FOC
- Current FOC
- Speed FOC

This enables tuning of the application in several steps, with each step containing a limited number of unknown parameters that can be adjusted. To switch between these modes, the MCAT tab “Control Struc” is used. For more information on tuning of the application for any motor, see *AN4912* [5].

The application code contains routines for a PM synchronous motor parameters identification. The algorithm simplifies and speeds up controller constant calculations and settings. The identification process is disabled by default in the application and can be enabled from the MCAT control tab “Motor Identif.” For more information on user motor identification, see *AN4986* [6].

5 SDK implementation

Kinetis SDK (KSDK) provides a comprehensive software support for Freescale Kinetis devices such as the KV31F120M MCU. The KSDK includes all necessary source and header files required by the application as startup code, clock configuration and peripherals configuration provided by Hardware Abstraction Layer (HAL) as “low” level configuration routines and “high” level peripherals drivers based on HAL routines. This suite provides easy portable basics of applications for the Kinetis family MCUs with no or little code modifications, depending on MCU and its peripherals and core equipments used. More information about the KSDK can be found in *KSDKDEMOUG* [7] and on freescale.com/ksdk.

Next steps explain how to run a demo application under KSDK with IAR workbench and how to pre-build the platform driver library in IAR workbench, which is necessary to run the example application.

5.1 Building the platform driver library in IAR

In the first step, it is required to generate the platform library file before building and debugging the demo application. This library contains all HAL and peripheral driver functions that are device-specific. Therefore, each device has its own library (*platform.a*). It should not be necessary to build the library after initially downloading the KSDK because the platform library is already pre-built. However, if this is necessary, follow the following directions to rebuild the platform library.

Open the workspace file by double clicking on *pmsm_sensorless_foc.eww* in IAR, located in *build\iar\kv31\pmsm_sensorless_foc\*. The IAR workspace launches and contains two projects: “ksdk_platform_lib” and “pmsm_sensorless_foc.” It is needed to build the platform library before the PMSM sensorless project can be used. As shown in Figure 8, the workspace contains two projects. Right-click on “ksdk_platform_lib” and a drop-down menu appears. Click on “Set as Active,” and then click on the “Make” button (highlighted by a red rectangle in Figure 8). When the build is complete, the library (*ksdk_platform_lib.a*) is generated in the output directory according to the build target, and can be located depending on the build target:

- Debug –*<ksdk_install_dir>/lib/ksdk_platform_lib<toolchain>/<device_name>/output/Debug*
- Release –*<ksdk_install_dir>/lib/ksdk_platform_lib/<toolchain>/<device_name>/output/Release*

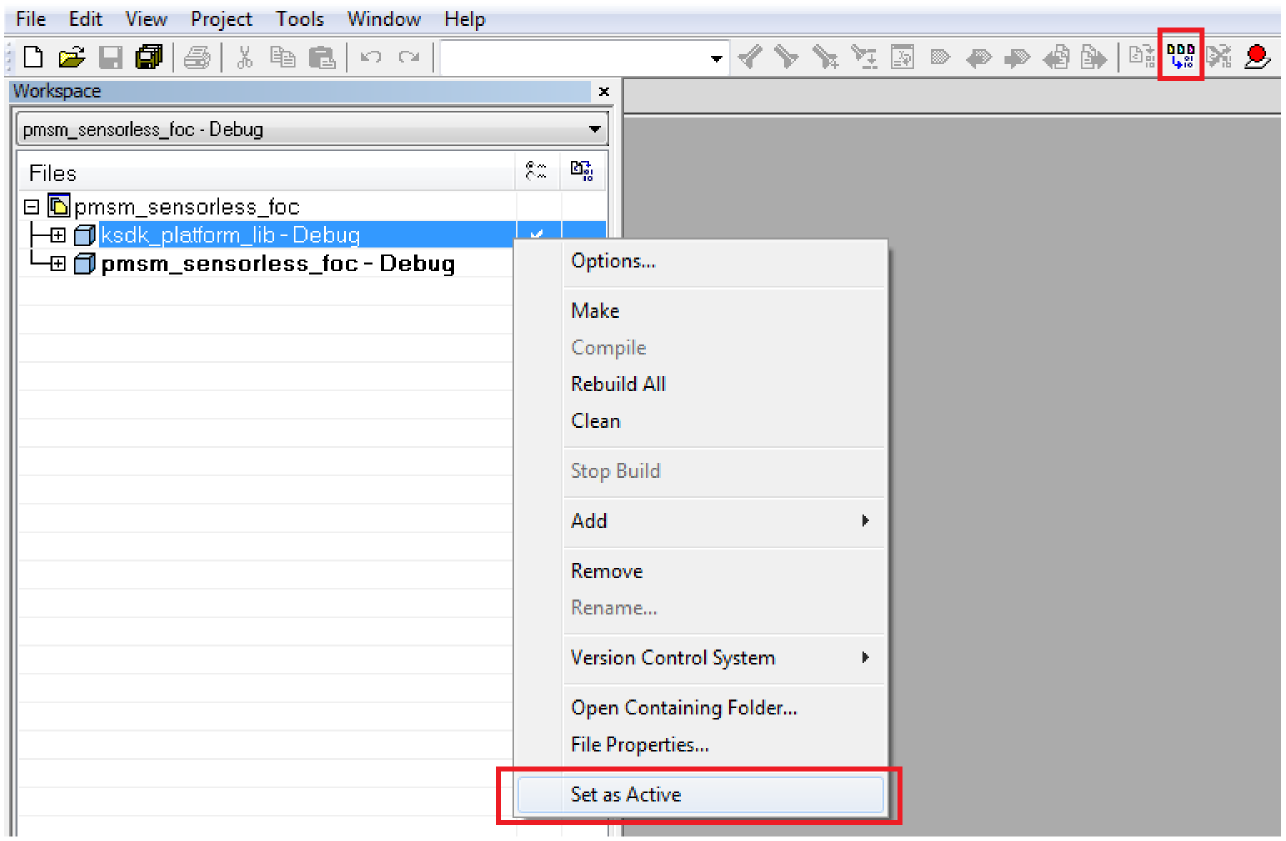


Figure 8. Platform driver library build

5.2 Building the PMSM project in IAR

After the platform library build is complete, the PMSM sensorless project can be generated. As mentioned in Section 5.1, "Building the platform driver library in IAR" and as shown in Figure 8, the workspace contains two projects. It is required to activate the "pmsm_sensorless_foc" project by right-clicking the project name, and then selecting "Set as Active." The PMSM project is activated and can be built by clicking the "Make" button. After the output file *pmsm_sensorless_foc.out* is generated in the output directory, it can be loaded into the KV31F120M MCU with the debugger or linked to the FreeMASTER communication tool.

6 Project file structure

The total number of source (**.c*) and header files (**.h*) in the project exceeds one hundred. Therefore, only the key project files will be described in detail, and the rest will be described in groups.

The main project folder is divided into three directories:

- *build\iar\kv31\PMSM_Sensorless_FOC* – contains configuration files for the IAR compiler as well as the compiler's output executable and object files. If the IAR Embedded Workbench for ARM® is installed on your computer, double clicking the workspace file *PMSM_SENSORLESS_FOC.eww* will launch the IAR IDE.

- *freemaster\PMSM_Sensorless_FOC* – contains the FreeMASTER configuration file (*PMSM_Sensorless_FOC.pmp*) and supporting files (control page in HTML format and the binary file with addresses of variables).
- *\src* – contains the project source and header files, and its contents will be described further in the text.

Files in the *src\projects\kv31\PMSM_Sensorless_FOC* folder are:

- *main.c* and *main.h* – contain basic application initialization (enabling interrupts), sub-routines accessing the MCU peripherals and interrupt service routines. The FreeMASTER communication is performed in the background infinite loop.
- *m1_state_machine.c* and *m1_state_machine.h* – contain the software routines that are executed when the application is in a particular state or state transition.
- *freemaster_cfg.h* – configuration file for the FreeMASTER interface.
- *pmsm_appconfig.h* – contains constants' definitions of the application control processes (parameters of the motor and regulators and the constants for other vector control related algorithms). When the application is tailored for another motor using the Motor Control Application Tuning Tool (MCAT), this file is generated by the tool at the end of the tuning process.
- *\peripherals* – contains important files for static configuration of the peripherals used in the application (FlexTimers, ADC, PDB, SPI, PIT).

Subfolders in the *\src* folder are:

- *\state_machine* – contains the application state machine structure definition which handles the switching between the application states and application state transitions.
- *\control_algorithms* – contains the structure definitions and subroutines dedicated for performing the the motor control algorithm (vector control algorithm, position and speed estimation algorithm, speed control loop).

Files in the *src\CM4_MMCLIB_r1.0\* folder are:

- *CM4_MMCLIB_IAR.a* – a software library containing motor control, general math and filter algorithms.

  Other files in the folder and subfolders are associated header files, each one for a particular function of the library.
- *ACLIB\ CM4_ACLIB_F16.a* – contains the advanced control algorithms for rotor position and speed estimation (back-EMF observer and tracking observer).

Other subdirectories in the *src* folders are:

- *\freemaster* – contains all source files of the FreeMASTER application, it is not necessary to access it or change anything inside. The interface to the programmer is handled via the *freemaster_cfg.h* file located in the *src\projects\kv31\PMSM_Sensorless_FOC* folder.
- *\sac* (sensor and actuator components) – contains routines for accessing peripherals used by the motor control algorithm to sense input feedback physical quantities (currents, voltage, speed, position) and to set the actuators based on calculated output variables (FlexTimer, MOSFET pre-driver).
- *\sdk\startup* – contains pre-compiled start-up library *ksdk_startup_lib.a* with clock setting possibilities and pre-defined header for clock setup *system_KV31F51212.h*.

7 Memory usage and CPU load

The following tables show the memory usage of the KV31F120M MCU, total available data and RAM memory, size of used memory, and memory used by individual algorithm blocks.

Table 2. Memory usage, values in bytes

Memory	Total available on the kinetis KV31F120M	Used by the application
Program Flash (application code)	512 KB	27 214 B
Data Flash (application constants)		1094 B
Data RAM (application variables)	96 KB	9 657 B (*)

(*) including 4096 B FreeMASTER recorder buffer

Table 3. Memory usage, values in bytes

Algorithm block	Program memory (code + constants) [Bytes]	Data memory [Bytes]
FOC (fast + slow loops)	4472	0
FreeMASTER	2950	4365
SAC (Sensor and Actuator Components)	2696	4
MID (Motor Identification)	3412	590
Application State Machine	4638	596

The CPU load is influenced mainly by the execution of the ADC1 ISR, in which the execution of the application state machine and calculation of the fast (current) and slow (speed) control loop of the PMSM vector control is performed.

The complete ADC1 ISR requires 5506 (state machine and fast / slow control loop) machine cycles in the worst case, the total consumed time per interrupt is 45 μs. The ADC1 interrupt is generated periodically with the same frequency as the PWM re-load event, when the values of the duty cycles are updated.

In this application, the ADC ISR is generated once per 100 μs, which corresponds to 10 kHz of the PWM frequency. At 120 MHz on the Kinetis KV31F120M MCU, it consumes 45,85 % of the total CPU performance.

8 Hardware setup

The Tower System – a modular development system is used as a hardware platform for the PMSM sensorless application on Kinetis KV31F120M MCU. It consists of the following modules:

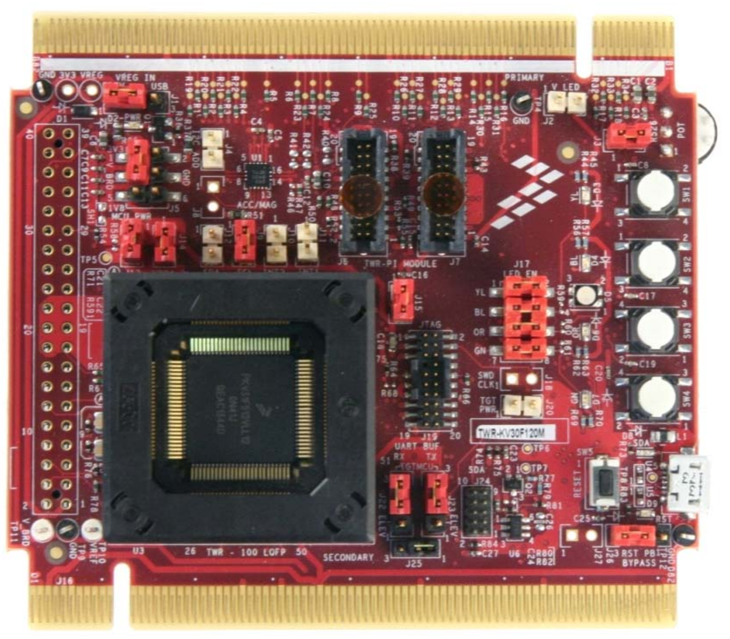
- Tower System elevator modules (TWR-ELEV)
- Kinetis KV31F120M Tower System module (TWR-KV30F120M)
- Three-phase low-voltage power module (TWR-MC-LV3PH) with included motor

All modules of the Tower System are available to order via the Freescale web page or from distributors, so the user can easily build the hardware platform for which the application is targeted.

8.1 Hardware setup and jumpers configuration

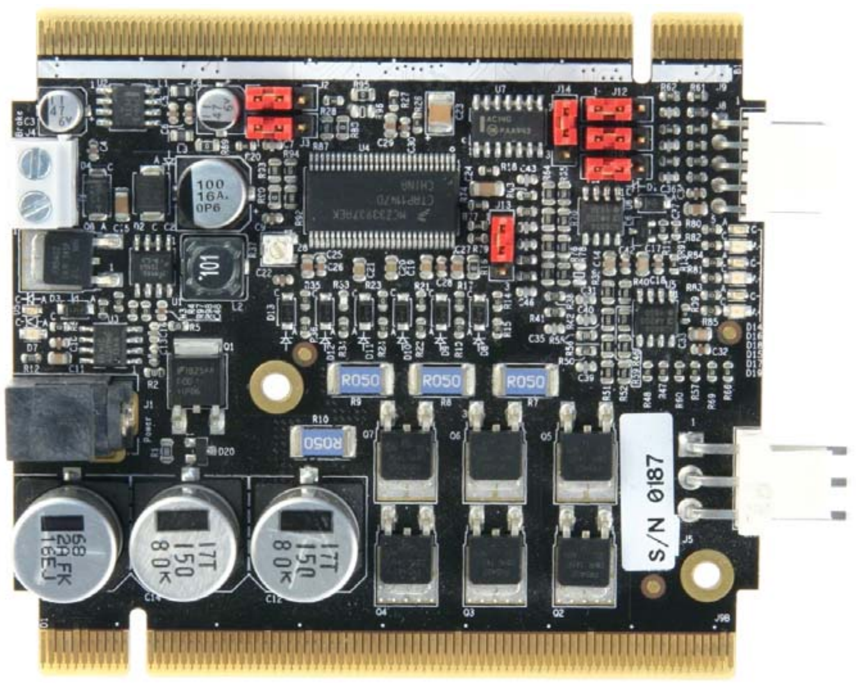
Building the system using the modules of the Tower System is not difficult. The peripheral modules and the MCU module are plugged into the elevator connectors, while the white stripe on the side of the module boards determines the orientation to the Functional elevator (the elevator with mini USB connector, power supplies and switch).

Table 4. Jumper settings of the TWR-KV31F120M board

	Jumper #	Settings	Jumper #	Settings
	J1	1-2	J15	1-2
	J2	open	J17	1-2
	J3	1-2	–	3-4
	J4	open	–	5-6
	J5	1-3	–	7-8
	J8	open	J18	open
	J9	1-2	J20	open
	J10	open	J22	2-3
	J11	open	J23	2-3
	J12	1-2	J25	1-2
	J13	1-2	J26	2-3
	J14	1-2	J27	open

The jumper settings for TWR-MC-LV3PH board are listed and highlighted in Table 5. See the user's manual *TWRMCLV3PHUG* [8] for more details (for example the hardware over-current threshold setting) of the TWR-MC-LV3PH stage.

Table 5. Jumper settings of the TWR-MC-LV3PH board

	Jumper #	Settings
	J2	1-2
	J3	1-2
	J10	1-2
	J11	1-2
	J12	1-2
	J13	1-2
	J14	1-2

9 Conclusion

The results of the execution time measurement show that the Kinetis KV31F120M MCU can be used to drive the PMSM sensorless vector control algorithm for high-dynamic applications. The CPU load at 10 kHz PWM frequency represents only 45%, so there is much space to either increase the frequency of the fast control loop or to perform additional user’s tasks.

10 References

The following resources are available on freescale.com:

1. *Using FlexTimer in ACIM/PMSM Motor Control Applications (Document AN3729)*
2. *KV31F Sub-Family Reference Manual (Document KV31P100M120SF7RM)*
3. *Three Phase Field Effect Transistor Pre-driver Data Sheet (Document number MC33937)*
4. *TWR-KV31F120M OOBE Demo User Guide (Document TWRKV31F120M)*
5. *Tuning 3- Phase PMSM Sensorless Control Application Using MCAT Tool (Document number AN4912)*
6. *Automated PMSM Parameters Identification (Document number AN4986)*
7. *Kinetis SDK Demo Applications User's Guide (Document KSDKDEMOUG)*
8. *TWR-MC-LV3PH User’s Manual (Document number TWRMCLV3PHUG)*
9. *Kinetis SDK KV31 User's Guide (Document number KSDKKV31FUG)*

11 Acronyms and abbreviated terms

Table 6 contains abbreviated terms used in this document.

Table 6. Acronyms and abbreviated terms

Term	Meaning
ADC	Analog-to-digital converter
Back-EMF	Back-electromotive force: a voltage generated by a rotating motor
FOC	Field-oriented control
PMSM	Permanent magnet synchronous motor
PWM	Pulse-width modulation
SVM	Space-vector modulation: the algorithm used for generation of signals for PWM output
SDK	Software development kit

12 Revision history

Table 7. Revision history

Revision number	Date	Substantial changes
0	04/2014	Initial release
1	02/2015	Updated sections 4 and 5, Kinetis SDK compatible.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4911
Rev. 1
02/2015