Freescale Semiconductor, Inc.
Application Note

Document Number: AN4935
Rev. 1, 09/2014

PMSM Sensorless FOC for a Fan Using the Kinetis KV10

1 Introduction

This application note represents an addendum to the Sensorless PMSM Field-Oriented Control Design Reference Manual (DRM148). The original application is now enhanced to include an anti-wind function, which handles the specific start-up condition when the fan propeller is forced into rotation by the wind. The presented solution recognizes neither the speed nor the direction of the forced rotation. The moving rotor is gently stopped, then the initial rotor position is detected and the rotor starts to spin without alignment.

This application note describes the algorithms to brake the moving rotor and detect the initial rotor position, the software modifications, the MCU peripherals used in the application, the hardware set-up, and the results of the measurement.

Contents

2 Anti-wind function

The anti-wind function is a required feature for fan applications. Most ceiling fans with large plates are easily forced into rotation by air movement caused by a draft or the operation of other fans in the room. By using this application, a rotating PMS motor generates back electromotive force (back-EMF). To prevent current surges that can yield a system failure, the rotor is gently stopped before the voltage vector is applied.

The way to brake the rotor is simple. At the beginning of the process, the bottom transistors of all three phases are opened at 10%. If the rotor is moving, a small current excited by the generated back-EMF starts to flow. This current can be measured by shunt resistors and detected by the MCU. When the current starts flowing, the motor starts to generate braking torque which lowers the speed of the rotation. The energy of the moving rotor is dissipated as heat, mainly in the stator winding. If the current is higher than the pre-defined threshold (10% of nominal value in this application) then the MCU waits until the current decreases. If the current decreases to below the threshold, the PWM duty cycle will be increased until the current again reaches the threshold. In this way, the motor is gently braked to a standstill, when the bottom transistors are gradually opened to 100%. The flowchart in Figure 1 represents the braking process.

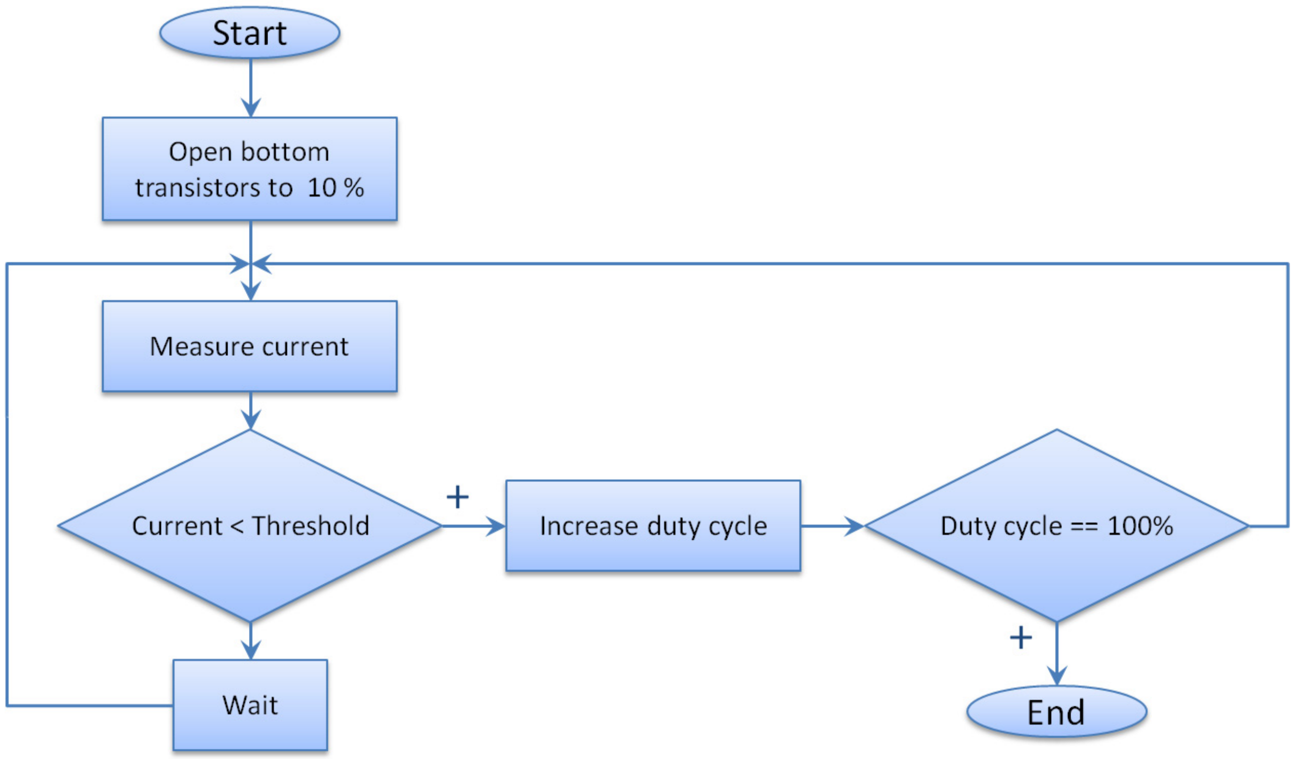


Figure 1. Braking Process

The robustness of the system can be further enhanced. A timer can be implemented to limit the braking process, for example in the case when there is still considerable current detected even at 100% duty cycle. The system can then issue a fault flag and the start-up of the motor will be disabled.

2.1 Initial rotor position detection

To successfully perform the vector control algorithm, the rotor position must be known in each instance of the control loop calculation. However, when the motor is at a standstill it is impossible to receive the positional information from the back-EMF. To obtain the position at a motor standstill, very often the so-called *rotor alignment* is used. During alignment, the voltage vector is generated along the d-axis of the d,q system, and the rotor is moved to the position when the rotor and stator magnetic field are aligned. In the fan application, the alignment method is not ideal because of the large amount of inertia of the ceiling fan plates.

After the rotor is stopped, the initial position detection algorithm is executed. The detection method is based on the saturation effect of the stator iron core due to the permanent magnet. Stator inductance varies with the rotor position as a result of the saturation caused by the rotor magnet. The effect can be observed by the rate of change of the current in the stator winding when a constant voltage is applied to the windings. By measuring the rate of change of the current in the stator winding due to the change of inductance, it is possible to estimate the relative position between a rotor magnet and a stator winding. The method is fully described in *A New Starting Method of BLDC Motors Without Position Sensor*, see Section 12, “References.” Only the fundamental principle and the software implementation will be presented in this document.

The estimation of the rotor position is based on the nonlinear magnetization characteristics of the stator core. Because the stator core is near a magnetic pole of the permanent magnet of the rotor it is strongly magnetized. Therefore, if a stator winding is close to the magnetic pole of the rotor, the rate of change of the current in the stator winding flowing in the magnetizing direction is large compared with that in the opposite direction, because of the magnetic saturation of the stator core.

In this application, six basic voltage vectors with the same length and the same amount of time, which are used in the Space Vector PWM module, are applied to the windings one by one. In this way, the six corresponding current pulses are generated and the rotor position of a 30 electrical degree resolution can be estimated according to the differences between the maximal values of these current pulses. An appropriate current vector can be generated according to this estimated initial rotor position, to achieve the maximum start-up torque. Figure 3 represents a simplified process of the initial position detection.

The article, *A New Starting Method of BLDC Motors Without Position Sensor* (see Section 12, “References”*)*, proposes evaluating the initial rotor position from the measured DC-bus current. Because this current is not measured in this application, the evaluation of the initial position is performed from the phase currents (in each sector, another phase current is considered), as Figure 2 represents.

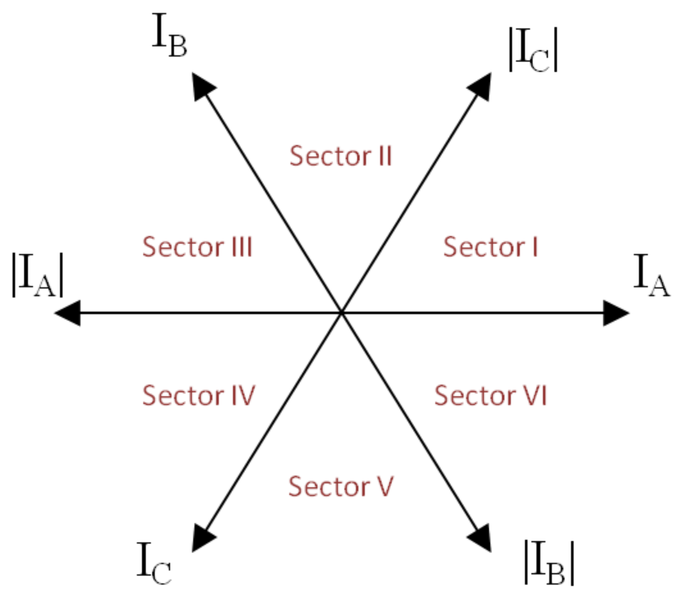


Figure 2. Initial position detection—the phase currents used for evaluation in particular sector

To adapt the anti-wind application to any motor, only knowing the electrical time constant of the stator coil is required. Because of CPU performance constraints—that is, the execution time and the 0.1 ms period of the fast control loop, the minimal electrical time constant of the motor must be 0.1 ms. If the application is not properly tuned to a particular motor, the differences in measured currents are so small that the detection of initial rotor position is not reliable. In such a case, the rotor position detection process is terminated and the fault flag is set. The application then continues with voltage alignment.

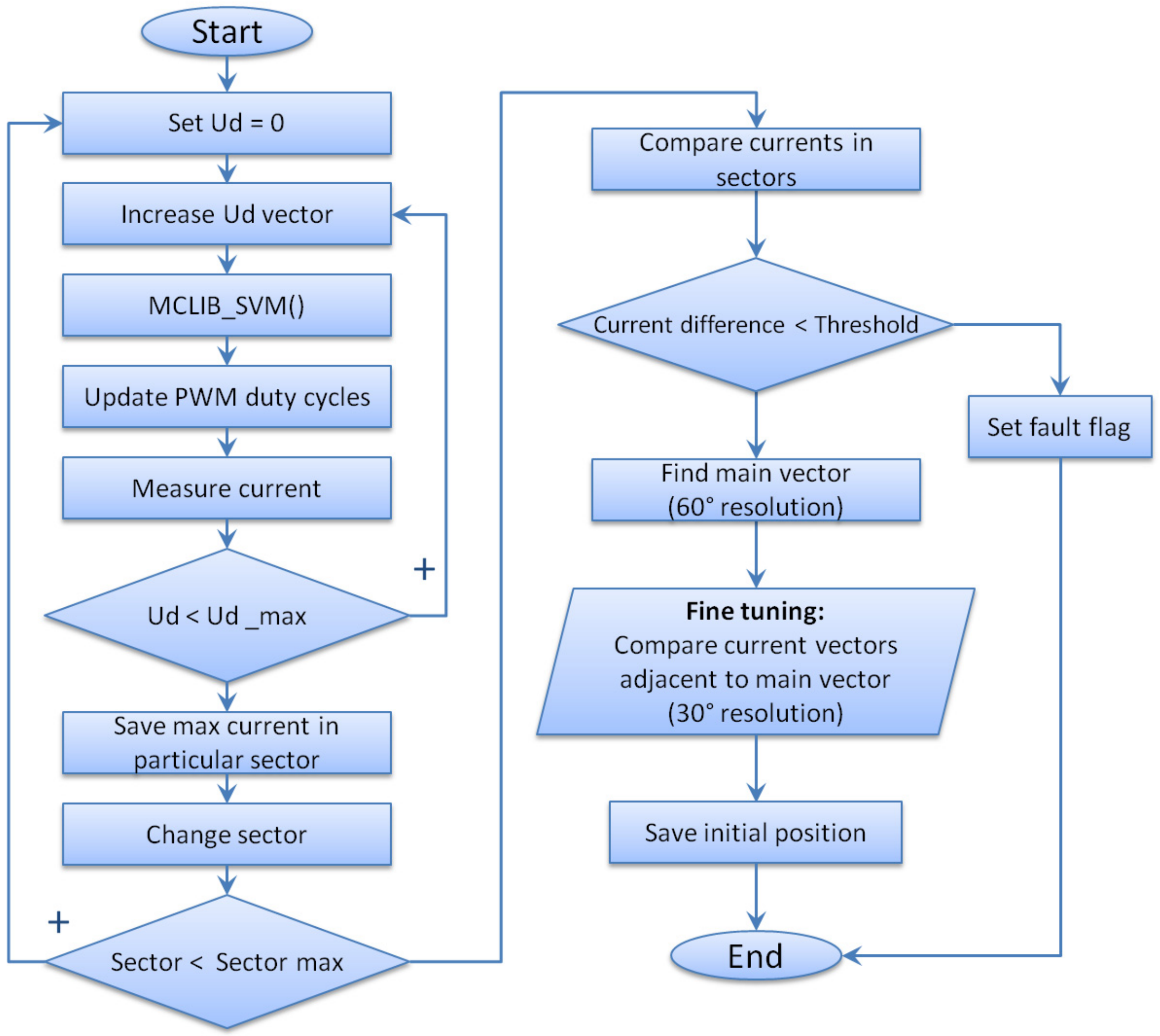

Start
Set Ud = 0
Increase Ud vector
MCLIB_SVM()
Update PWM duty cycles
Measure current
Ud < Ud _max
+
Save max current in particular sector
Change sector
Sector < Sector max
+
Compare currents in sectors
Current difference < Threshold
Set fault flag
Find main vector (60° resolution)
Fine tuning:
Compare current vectors adjacent to main vector (30° resolution)
Save initial position
End

Figure 3. Initial position detection process

3 Software description

3.1 Application state machine

The application state machine differs in a few details from the original state machine described in DRM148. The Run state is enhanced to include two additional sub-states, *Brake* and *PosDetection,* which enable the anti-wind feature and the initial rotor position detection without alignment. The *Calib* state is executed after the Brake substate, because the calibration of the offsets of the ADC channels used for current measurement is performed at the rotor stand-still. When the motor is rotating at the application start-up, it is not possible to measure the current offset. Therefore, the initial current offset values were entered during the application development. The impact on the performance is not significant. The current offset values are almost half of the maximum raw ADC value—that is, for a 12-bit resolution, $2^{11} = 2048$. The *Ready* sub-state was also modified. In the original application, during the *Ready* sub-state all the PWM outputs were enabled and set to 50%. When the motor is running, this causes a current surge with potential system failure. Figure 3 represents the Run state with sub-states and transitioning conditions.

The Run sub-states are called when the state machine is in the Run state. The Run sub-state functions are as follows:

- **Ready**:
  - The bottom transistors are open to 10% while the top transistors remain off.
  - The PWM output remains disabled.
  - The phase currents and DC-bus voltage are measured.
  - The ADC channels are set up.
  - Certain variables are initialized.
- **Brake:**
  - The PWM output is enabled, if the motor is rotating, it is braked.
  - The phase currents are measured.
  - The duty cycle of bottom transistors is increased up to 100%.
- **Calib:**
  - The duty cycle is set to 50% and top and bottom transistors are enabled.
  - The current channels offset is measured and filtered.
- **PosDetect**:
  - The currents are measured.
  - The position detection algorithm is executed.
  - If the initial rotor position detection fails, the application goes into the Align sub-state, otherwise it goes into the Startup sub-state.
- **Align:**
  - The currents are measured.
  - The ADC channels are set up.
  - The rotor alignment algorithm is called.
  - The PWM is updated.
  - The DC-bus voltage is measured.
  - After the alignment time expiration, the system is switched to Startup.
- **Startup**:
  - The currents are measured.
  - The ADC channels are set up.

- The Back-EMF observer algorithm is called to estimate the speed and position.
- The estimated speed is filtered.
- The FOC algorithm is called.
- The PWM is updated.
- The DC-bus voltage is measured and filtered.
- The open-loop start-up algorithm is called.

- **Spin:**
  - The currents are measured.
  - The ADC channels are set up.
  - The Back-EMF observer algorithm is called to estimate the speed and position.
  - The estimated speed is filtered.
  - The FOC algorithm is called.
  - The PWM is updated.
  - The motor spins.
  - The DC-bus voltage is measured.
  - The speed ramp and the speed PI controller algorithms are called.
  - The speed command is evaluated.
- **Freewheel:**
  - The PWM output is disabled, and the modulo is set to 50%.
  - The current is measured.
  - The ADC channels are set up.
  - The DC-bus voltage is measured.
  - The system waits in this sub-state for a certain period of time which is determined by the rotor inertia—that is, it waits until the rotor stops.
  - The system evaluates the conditions and proceeds into either the Align or Ready sub-state.

The Run sub-states also have functions that are called between sub-state transitions. The sub-state transition functions are as follows:

- **Ready > Brake**—non-zero speed command; entering the Brake state.
  - Top transistors are enabled.
- **Brake > Calib**
  - Top and bottom transistors are enabled.
- **Calib > PosDetect**
  - Variables used for position detection algorithm are initialized
- **PosDetect > Align**—Position detection failed; entering the Align state.
  - Certain variables are initialized (voltage, speed, position).
  - The alignment time is set up.
- **PosDetect > Startup**—Position detection successful.

— Clearing variables.

— Passing the detected rotor position to the start-up structure.

— Entering the Startup state.

- **Align > Ready**—zero speed command.

— Entering the Ready state.

- **Align > Startup**—alignment done.

— The filters and control variables are initialized.

— Entering the Startup state.

- **Startup > Spin**—entering the Spin state.
- **Startup > Freewheel**—no action occurs. Can be used to handle the start-up fail condition for a more robust application.
- **Spin > Freewheel**—zero speed command.

— PWM output is disabled.

— Entering the Freewheel state.

- **Freewheel > Ready**—freewheel time expired.

— The PWM output is enabled.

— Entering the Ready state.

- **Freewheel** > **Brake**—non-zero speed command.

— The PWM output is enabled.

— Certain variables are initialized (voltage, speed, position).

— Entering the Brake state.

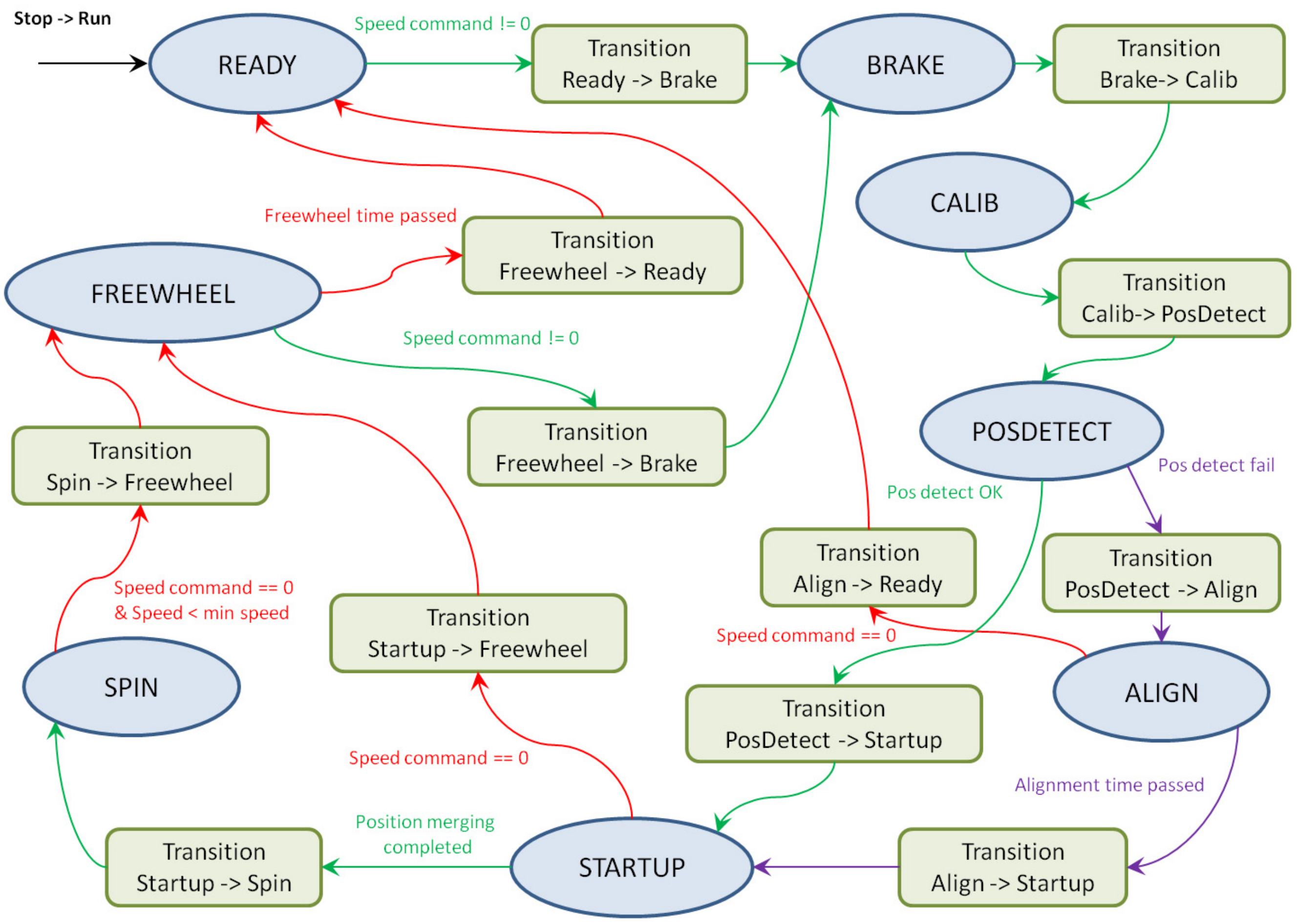

Stop -> Run
READY
Speed command != 0
Transition Ready -> Brake
BRAKE
Transition Brake-> Calib
CALIB
Transition Calib-> PosDetect
Freewheel time passed
Transition Freewheel -> Ready
FREEWHEEL
Speed command != 0
Transition Freewheel -> Brake
Transition Spin -> Freewheel
POSDETECT
Pos detect fail
Pos detect OK
Transition Align -> Ready
Transition PosDetect -> Align
Speed command == 0 & Speed < min speed
Transition Startup -> Freewheel
Speed command == 0
SPIN
ALIGN
Transition PosDetect -> Startup
Speed command == 0
Alignment time passed
Position merging completed
Transition Startup -> Spin
STARTUP
Transition Align -> Startup

Figure 4. Application flow in Run state

Run sub-state machine is declared. Thus, the Run sub-state identification variable has the following definitions:

```
typedef enum {
        CALIB       = 0,
        READY       = 1,
        BRAKE       = 2,
        POSDETECT   = 3,
        ALIGN       = 4,
        STARTUP     = 5,
        SPIN        = 6,
        FREEWHEEL   = 7,} M1_RUN_SUBSTATE_T;    /* Run sub-states */
```

As mentioned previously, the application does not pass the CALIB sub-state. Instead, after transition from the STOP state, the application goes directly into READY state. The CALIB sub-state is kept for compatibility with the original Sensorless PMSM FOC application.

For the Run sub-states, the following set of user functions are defined:

```
        static void M1_StateRunCalib(void);
        static void M1_StateRunReady(void);
        static void M1_StateRunBrake(void);
        static void M1_StateRunPosDetect(void);
        static void M1_StateRunAlign(void);
        static void M1_StateRunStartup(void);
        static void M1_StateRunSpin(void);
        static void M1_StateRunFreewheel(void);
```

The Run sub-states' user transient function prototypes:

```
        static void M1_TransRunReadyBrake(void);
        static void M1_TransRunReadyCalib(void);
        static void M1_TransRunCalibAlign(void);
        static void M1_TransRunBrakeCalib(void);
        static void M1_TransRunAlignStartup(void);
        static void M1_TransRunAlignReady(void);
        static void M1_TransRunCalibPosDetect(void);
        static void M1_TransRunPosDetectStartup(void);
        static void M1_TransRunPosDetectAlign(void);
        static void M1_TransRunStartupSpin(void);
        static void M1_TransRunStartupFreewheel(void);
        static void M1_TransRunSpinFreewheel(void);
        static void M1_TransRunFreewheelBrake(void);
        static void M1_TransRunFreewheelReady(void);
```

The Run sub-states functions table initialization:

```
/* Sub-state machine functions field */
        static const PFCN_VOID_VOID mM1_STATE_RUN_TABLE[8] =
        {M1_StateRunCalib,
         M1_StateRunReady,
         M1_StateRunBrake,
         M1_StateRunPosDetect,
         M1_StateRunAlign,
         M1_StateRunStartup,
         M1_StateRunSpin,
         M1_StateRunFreewheel};
```

3.2 Software implementation of braking and initial rotor position detection

3.2.1 Motor braking

The braking algorithm of the motor is performed in the application state machine, in the BRAKE sub-state of the RUN state. The current of Phase A is measured during the period of the fast control loop. To find the amplitude of the phase current, the maximum value of the measured Phase A current is tracked and is evaluated once per 1000 transitions of the fast loop. Thus, it is guaranteed that even at low speeds the amplitude of the phase current per one period will be successfully detected.

The variables used for storing the data used by the braking algorithm are integrated into the structure with the following definition:

```
typedef struct
      {
        UWord16  uw16BrakeLoopDuration;  /* duration of max current detection window */
        UWord16  uw16BrakeLoopCounter;   /* counter measuring the detection window */
        Frac16   f16MaxIA;               /* detected value of max. phase A current */
      } MCS_PMSM_BRAKE_T;
```

The description of the structure members is the following:

- Duration of the maximum current detection window, in a number of the fast control loop ticks
- Counter storing the actual number of ticks
- Captured value of the maximum Phase A current in the actual detection window

3.2.2 Detection of Initial rotor position

For better code readability, the detection of the initial rotor position algorithm is divided into two functions:

The prototype of the function which is called from the state machine and when the currents in all sectors are measured is:

```
      void MCS_ MCS_PMSMPositionDetectionInit(MCS_PMSM_FOC_T *psFocPMSM, MCS_PMSM_POS_DETECT_T
      *psPosDetect);
```

The prototype of the function used for evaluation of the initial position from the measured current is:

```
      void MCS_PMSMGetInitialPosition(MCS_PMSM_POS_DETECT_A1_T *psPosDetect)
```

The first function uses the FOC structure which is described in the DRM148. An additional structure referred to by the input/output structure pointer is defined as follows:

```
 typedef struct
    {
    GFLIB_RAMP_T  sVoltageVectorRampParams; /* Ud Voltage ramp parameters */
    Frac16        f16UdMax;                 /* Maximum applied Ud voltage */
    Frac16        f16UdMin;                 /* Minimum applied Ud voltage */
    Frac16       f16CurrentInSector[6];    /* Array of measured currents in sectors */
    Frac16       f16InitPos;               /* Detected initial rotor position */
    Frac16       f16MinCurrentDelta;       /* Minimum difference between currents */
    UWord16      uw16PulseCnt;             /* Counter used for measurement of the turn-on
                                              and turn-off of the PWM outputs*/
    UWord16      uw16PulseOnDuration;      /* Turn-on time of the PWM outputs */
    UWord16      uw16PulseOffDuration;     /* Turn-off time of the PWM outputs */
    Word8        w8MainVector;             /* Detected main vector of the rotor position */
    Word8        w8AuxVector;              /* Auxiliary vector used for fine tuning */
    bool         bPulseOffFlag;            /* Flag used for enabling/disabling of the PMW
                                              outputs */
    bool         bCurrentMeasurementCompleted;/* The current measurement completed */
    bool         bErrFlag;                 /* The position detection failed */
    }   MCS_PMSM_POS_DETECT_T;
```

The description of the structure members is as follows:

- Ud voltage ramp structure—used to control the applied Ud voltage
- Maximum Ud voltage (selected based on consideration of the maximum motor phase current)
- Minimum Ud voltage—the minimum value of the Ud voltage vector
- Array of the maximum measured currents in each SVM sector
- Detected initial position
- Minimum difference in measured currents, when the algorithm returns reliable values of the initial position
- Counter variable used for measuring the turn-on and turn-off times of the PWM outputs in ticks of the fast control loop
- Turn-on time of the PWM outputs while the Ud voltage vector is applied
- Turn-off time of the PWM outputs
- Detected main vector of the rotor position
- Auxiliary variable used for fine tuning of the initial rotor position
- Flag set when the current measurement is completed, so the position detection algorithm can be executed
- Error flag, set when the difference of the measured currents is too low, so the position cannot be reliably detected

3.2.3 Modifying the position detection algorithm to any motor

The algorithm of the initial rotor position detection can be tuned to any motor. The only limitation is that the electric time constant of the motor stator coils $\tau = L_S / R_S$ must be greater or equal to the period of the fast (current) control loop. The voltage applied on the stator winding must be high enough that the current flowing through the stator coil will cause the saturation phenomena to take effect. On the other hand, the applied current during the initial rotor position detection process must not be so high that torque is produced and the motor moves.

To modify the position detection algorithm to any motor, the following macro definitions must be changed:

```
#define CONTROL_FREQ     10000    /* in Hz, frequency of the fast control loop */
#define MAX_U_LENGTH      2       /* in Volts - should be In * R * 0.5 (such a voltage
                                     that the current will be 50 % of nominal value )*/
#define MIN_U_LENGTH      0.1     /* in Volts – should be set to 10-30% of MAX_U_LENGTH */
#define MIN_TO_MAX_TIME   1       /* in ms - should be equal to Tau */
#define MIN_DELTA_CUR    20       /* in mA - should be up to 5% of nominal current */
```

4 MCU peripherals

Table 1summarizes the peripherals on the Kinetis KV10Z32 MCU and their use by the PMSM sensorless vector control application.

Table 1. Kinetis KV10Z32 Peripherals Overview

Kinetis KV10 peripherals			Used in the application	Purpose
Group	**Module**	**Number of modules or channels**		
Analog	ADC0	11 channels single ended, 2 of them differential pairs	3 channels	DC-bus voltage and motor phase currents sensing
	ADC1	11 channels single ended, 2 of them differential pairs	2 channels	
	Comparators	2 modules, each 7 channels	—	—
	DAC	1 module	—	—
Communications	SPI	1 module, 4 chip select signals	1 module	MOSFET driver configuration
	UART	2 modules	1 module	FreeMASTER communication
	I2C	1	—	—
Timers	FlexTimer	6 channels	6 channels	Generation 6-channels PWM for motor control
		2 channels	—	—
		2 channels	—	—
	PDB	2 channels for ADC triggering	2 channels	DC-bus voltage and phase current sampling initiation
		2 channels for DAC triggering	—	—
	LPT	1 module	—	—
Other	eDMA	4 channels	—	—

4.1 FlexTimer0 configuration to generate a 6-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. On the Kinetis KV10 there are three instances of FTM. One FTM has 6 channels, the other two FTMs have 2 channels.

The procedure of the FlexTimer configuration for generating a center-aligned PWM with dead time insertion is described in the application note titled *Using FlexTimer in ACIM/PMSM Motor Control Applications* (document AN3729).

Because AN3729 supports an earlier version (1.0) of the FlexTimer implemented on the ColdFire V1, and with regards to the hardware used (TWR-MC-LV3PH), there are a few differences in the configuration, as follows:

- It is necessary to enable the system clock for the FlexTimer module in the Clock Gating Control register:

  SIM→SCGC6 |= SIM_SCGC6_FTM0_MASK;
- It is necessary to disable the write protection of some registers before they can be updated:

  FTM0→MODE |= FTM_MODE_WPDIS_MASK;
- It is recommended to enable the internal FlexTimer counter to run in the debug mode:

  FTM0→CONF |= FTM_CONF_BDMMODE(3);
- When the HW debugging interface (jLink, Multilink,…) is connected to the microcontroller, the MCU is in debug mode. This is not dependent on running code containing breakpoints.
- The PWM signals generated by the FlexTimer0 are directly connected to the MOSFET driver. Because of safety reasons, the input signals for the top transistors on the MOSFET driver used on the Tower low voltage power stage have inverse polarity. Therefore, it is also necessary to set the right polarity of the PWM signals:

  FTM0→POL = FTM_POL_POL0_MASK |

  FTM_POL_POL2_MASK |

  FTM_POL_POL4_MASK;
- The duty cycle is changed by changing 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 also by confirming the change by setting the Load Enable (LDOK) bit. This ensures that all values are updated at the same instance:

  FTM0→PWMLOAD = FTM_PWMLOAD_LDOK_MASK;

  It is necessary to write the LDOK bit every time the value registers are changed, so 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.
- Within the application, hardware triggering of the AD converter is employed. The Initialization Trigger signal from the FlexTimer is used as the primary triggering signal which is fed into the Programmable Delay Block that services the timing of the AD conversion initiation.

  FTM0→EXTTRIG |= FTM_EXTTRIG_INITTRIGEN_MASK;

- Finally, the output pins of the MCU must be configured, to bring out the signals from the chip. The assignment of signals to output pins is set in the Pin Control Register. The available signals are listed in the *Signal Multiplexing* chapter of *A New Starting Method of BLDC Motors Without Position Sensor* and are package dependent. (See Section 12, "References")

```
PORTC->PCR[1] = PORT_PCR_MUX(4);  // FTM0 CH0
PORTE->PCR[25] = PORT_PCR_MUX(3); // FTM0 CH1
PORTC->PCR[3] = PORT_PCR_MUX(4);  // FTM0 CH2
PORTC->PCR[4] = PORT_PCR_MUX(4);  // FTM0 CH3
PORTD->PCR[4] = PORT_PCR_MUX(4);  // FTM0 CH4
PORTD->PCR[5] = PORT_PCR_MUX(4);  // FTM0 CH5
```

The port settings implemented in the application code reflect the hardware solution built on the Tower system modules.

4.2 ADC and PDB modules configuration

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 closely cooperates with the ADC and triggers the hardware for sampling.

It is required to perform a self-calibrating procedure of the ADC module before it is used in the application to obtain the specified accuracy. The calibration process also requires a programmer's intervention—that is, to generate the plus-side and minus-side gain calibration results and store them in the ADC plus-side and minus-side gain registers after the calibration function completes. The calibration must be performed for both ADC modules. After calibration, the ADC modules are configured to a 12-bit accuracy. The CPU frequency is set to 75 MHz, so by using available prescaler values the input clock to the ADC module is set to 12.5 MHz. That setting yields a conversion time of 2.64 µs (33 ADC clock cycles). Finally, the hardware trigger must be enabled in the Status and Control Register 2.

The Programmable Delay Block (PDB) provides controllable delays from either an internal or an 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. Because the input trigger periodically comes from 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 KV10 MCU allows 15 different input trigger sources. They are listed in the *Chip configuration* chapter, in section *PDB Configuration* in the *KV10 Sub-Family Reference Manual* (document KV10P48M75RM). Similarly, as for FTM0, the LDOK bit must be set to acknowledge the changes in the delay registers.

On the KV10 MCU clock source for PDB is different than to FTM – Bus Clock for PDB and ADC, and System Clock for FTM.

4.3 ADC conversion timing, currents and voltage sampling

The FlexTimer0 is configured to trigger an internal hardware signal when its counter is reset after overflow to the initialization value. This signal is fed into the Programmable Delay Block (PDB) that consequently triggers the AD conversion of the voltage and currents with a predefined delay. On the Kinetis KV10

75 MHz MCU, two ADC modules are implemented. 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. So 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 3 conversions need to be triggered without CPU intervention (two motor phase currents and the DC-Bus voltage). Figure 5 timing diagram shows the module interconnections and the ADC interrupt generation.

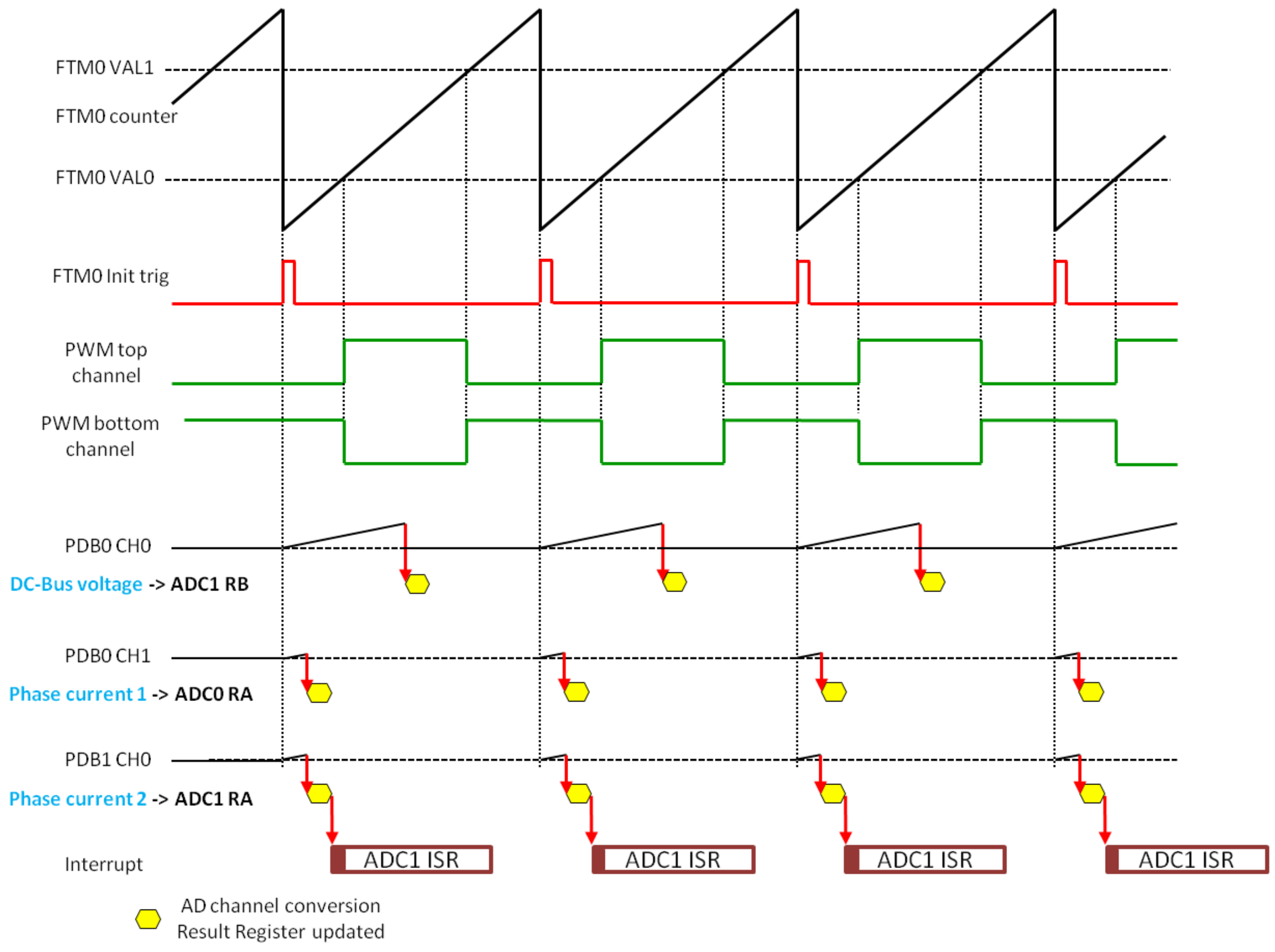


Figure 5. ADC conversion timing diagram

4.4 Current measurement

Closely related to the ADC conversion trigger timing is the assignment of the ADC channels to the measured analog signals. For computation of the fast (current) control loop of the FOC, it is necessary to know the values of all three motor phase currents. Because there are only two ADC modules, it is possible to sample only two analog quantities in one instance. Assuming the motor represents a symmetrical 3-phase system, the sum of all three instantaneous phase currents is zero.

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

Because the phase currents are measured the instance when the bottom transistors are conducting, in the case of high duty cycle ratios (current value is in the area of the maximum of the sine curve), the time when the current can be measured is too short. 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 determined based on the section when 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 is easily calculated according to Equation 2.

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

Figure 6 illustrates two cases (case I at 30°, case II at 60°) which explain why calculating the third current is necessary.

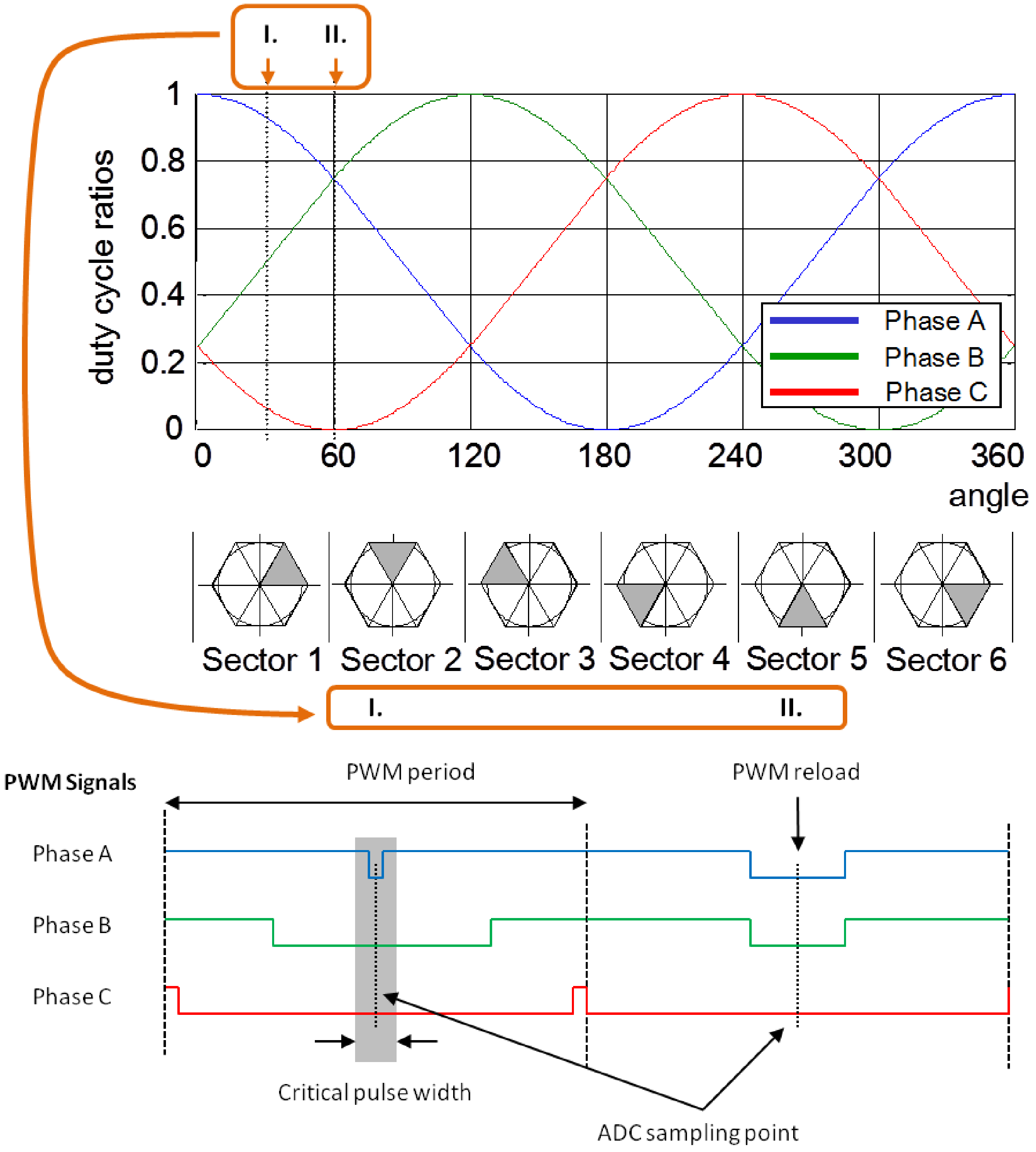


Figure 6. Current Sensing

As apparent from the picture, at 60° we can sample all three currents, because as mentioned previously, the currents are sampled when the bottom transistors are turned-on. Therefore, the pulse width is sufficient to stabilize the current and to perform the acquisition of the signal value by the AD converter. At 30°, the pulse is too short, so the current of Phase A cannot be sampled.

4.5 SPI configuration

The SPI interface is used in the application for communication between the intelligent MOSFET gate driver MC33937 and the KV10 MCU. The MC33937 gate driver is placed on the Tower low voltage power module and serves for driving the high-side and low-side MOSFET transistors of the 3-phase inverter. In the application, the initialization of the MC33937 must be performed, mainly in setting the dead time. During the motor run there is also 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 use. The exact timing of the SPI signals is listed in *Three Phase Field Effect Transistor Pre-driver Data Sheet* (document MC33937).

4.6 SCI (UART) configuration

The SCI is used in the application for the communication between the master system and the embedded application. A master system is the notebook or the PC where the FreeMASTER software is installed in order to control the application and visualization of its state. On the Kinetis KV10, there are two UART modules implemented. Because the hardware solution is based on the Tower modules, the UART1 is used. The communication speed is set to 19200 Bd and is limited by use of the OpenSDA - CDC serial communication driver. The use of direct RS232 connection between the PC and the embedded side enables an increase of the communication speed to 115200 Bd.

5 Interrupts

Because of the MCU’s ability of the hardware to trigger the AD conversion, the application requires a minimum number of interrupts.

5.1 ADC1 interrupt

This interrupt request is triggered when the conversion of channel A of the ADC1 module is completed and has the highest priority. During interrupt generation, there are sampled values of physical quantities ready: in the Result Registers A of the ADC0 (motor phase current 1), Result Register A of the ADC1 module (motor phase current 2), and Result Register B of the ADC1 (DC bus voltage). The interrupt is always enabled only for one module, otherwise two interrupt requests would be generated at the same time because the triggers for motor phase currents are generated in the same instance. In the beginning of the ADC1 ISR execution an application state machine function is called. If the application is in the Run state, then it is followed by the execution of the fast (current) control loop of the PMSM vector control algorithm, 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 of the result register of the ADC channel that triggered the interrupt. Therefore, at the beginning of each particular state machine function, the results of AD conversion are read, even though the values are not used until later during program execution.

The flow chart depicted on the Figure 7 provides an overview of the program flow during execution of the ADC interrupt service routine when the application is in Run state and Spin sub-state.

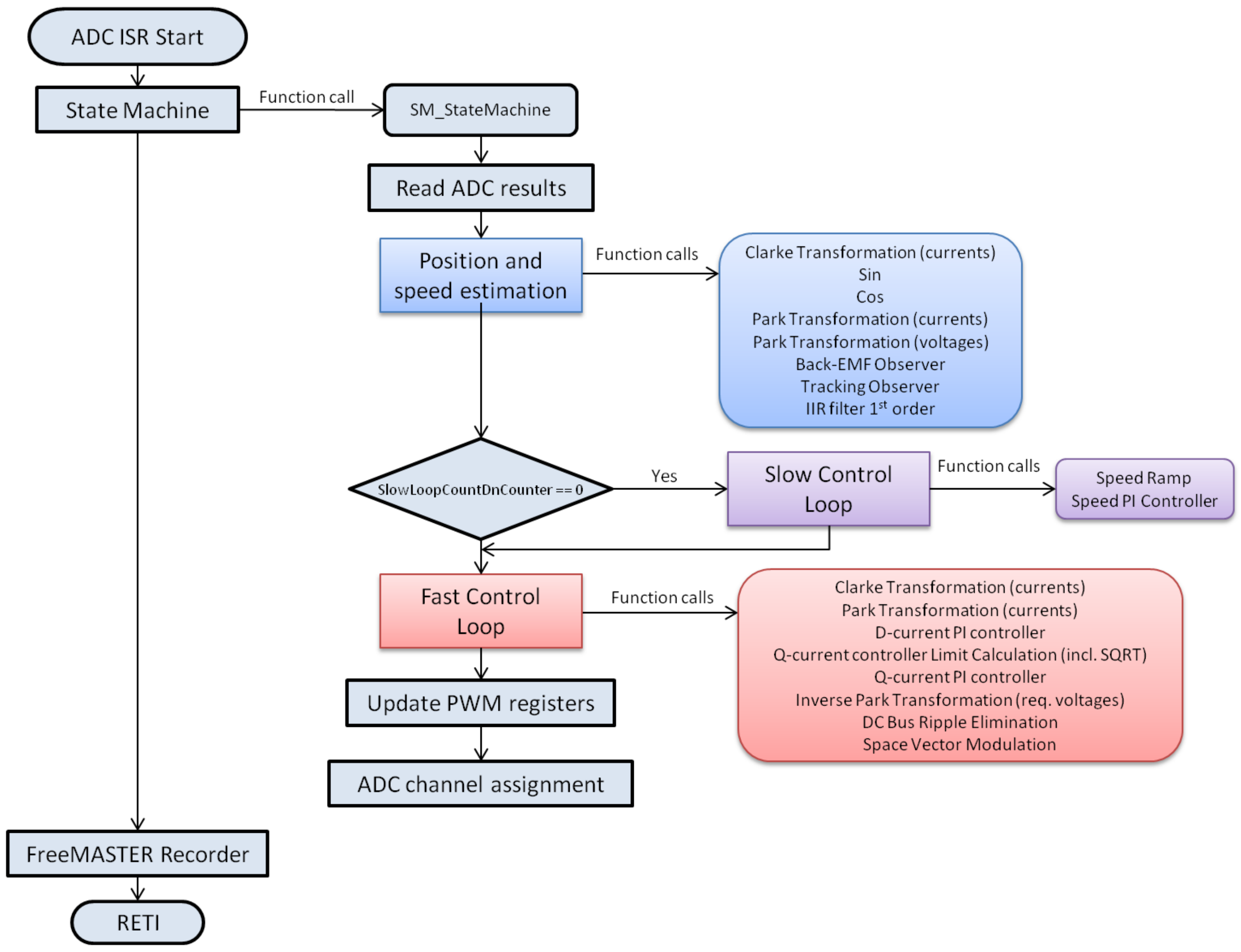


Figure 7. ADC ISR flow chart

5.2 Ports interrupt

Handling of the user's button on the KV10 tower board is performed in the ISR associated with the Ports interrupt, generated whenever the SW2 button is pressed. At the beginning, the interrupt flag is cleared. Pressing the SW2 button enables/disables the Demo mode. The first pressing of the SW2 button puts the application state to RUN mode and the required speed is changed in several steps according to a predefined profile. The subsequent press of the SW2 button returns the application to STOP mode.

More about the application control is available in Section 9, "Application operation." Using the FreeMASTER control interface enables enhanced control and diagnostic.

5.3 PDB error interrupt

The PDB error ISR serves to clear the sequence error fault that is generated in situations when PDB initiates the sampling of the AD converter even though the COCO flag in the specific **ADC*x*_SC1*n*** register of the ADC module was not cleared. The values from the result registers are not read, due to the reasons described as follows.

- One scenario is when the PDB counter stops working and an interrupt is asserted. The ISR in the PDB module is then re-initiated. The PDB generates a trigger signal with the same period as the ADC conversion complete interrupt which is also the same as the PWM period. If the user places an interrupt in the code, this stops execution. However, the PDB generates a trigger for the next conversion, even when the program execution stops. The COCO flags are not cleared and the PDB generates a sequence error.
- Another scenario is when the execution of the ADC conversion complete interrupt (when the fast control loop is calculated) extends over one period of the PWM. This occurs if the user enters additional tasks into the ADC conversion complete interrupt. In addition to the generation of PDB sequence error, the 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 to extend the sampling period. The real-time control application must be designed in such a way that this situation never occurs.

6 Project file structure

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

The main project folder is divided into three directories:

- **build\iar\kv10\PMSM_Sensorless—**contains the 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_SAC_SENSORLESS.eww located in the directory \build\iar\ launches the IAR IDE.
- **freemaster\PMSM_Sensorless—**contains the FreeMASTER configuration file PMSM_FOC_KV1x.pmp and supporting files (control page in HTML format and the binary file with addresses of the variables)**.**

  It also contains the FreeMASTER project for the Motor Control Application Tuning Tool PMSM_FOC_KV1x_MCAT.pmp.
- **src—**contains the project source and header files, and its contents will be described in the following text.**:**

  Files in the **src\projects\kv10\PMSM_Sensorless** folder:

  — **main.c** and **main.h** contain the basic application initialization (enabling interrupts), subroutines to access the MCU peripherals, and interrupt service routines. In the background infinite loop the FreeMASTER communication is performed.

  — **state_machine.c** and **state_machine.h** contain the application state machine structure definition and handles the switching between the application states and application states' transitions.

  — **motor_structure.c** and **motor_structure.h** contain the structure definitions and subroutines dedicated to perform the motor control algorithm (vector control algorithm, position and speed estimation algorithm, speed control loop).

  — **M1_statemachine.c** and **M1_statemachine.h** contain the software routines that are executed when the application is in the particular state or state transition.

  — **freemaster_cfg.h** configuration file for the FreeMASTER interface.

- **PMSMFOC_appconfig.h** contains the constant 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, this file is generated by the tool at the end of the tuning process.
- **\peripherals\** folder contains the important files for static configuration of the peripherals used in the application (FlexTimers, ADC, PDB, SPI, PIT).

Subdirectories in the src\ folder:

- **\common\** and **\cpu\** folders containing CPU initialization routines.
- **\cpu\isr.h** an important file with definitions of the peripherals interrupt service routines assigned to the interrupt vectors. In this file, the user can add the definition of an ISR for an additional peripheral interrupt.
- **\drivers\** files in the subdirectories contain generic source and header files for UART and watchdog configuration, as well as the CPU clock settings routines.
- **\platforms\tower.h** contains the Kinetis Tower card definitions (CPU speed and UART parameters).

Files in the **src\cpu\headers** folder:

- **MKV10Z7.h** is the header file containing macro definitions of all the MCU registers and register bits.

Files in the src\MMCLIB\ folder:

- **CM0+_MMCLIB_IAR.a** software library containing motor control, general math and filter algorithms. Additional files in the folder and subfolders are associated header files, each one for a particular function of the library.
- **ACLIB\CM0+_ACLIB_IAR_r0.2.a** contains the advanced control algorithms for rotor position and speed estimation (Back-EMF observer and Tracking observer).

Other subdirectories in the src\ folders:

- **\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 only via **freemaster_cfg.h** file located in src\projects\kv10\PMSM_Sensorless folder.
- \SAC\ folder (Sensor and Actuator Components) contains routines to access 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).

7 Memory usage

Table 2 summarizes the chip memories usage.

Table 2. Memory Usage, Values in Bytes

Memory	Total Available on the Kinetis MKV10Z32	Used by the Application
Program Flash (application code)	32 KB	20 460 B
Data Flash (application constants)		672 B
Data RAM (application variables)	8 KB	2 356 B

8 Hardware setup

The Tower modular development system is used as the hardware platform for the PMSM sensorless on Kinetis KV10. It consists of the following modules:

- Tower elevator modules (TWR-ELEV)
- Kinetis K60 tower module (TWR-KV10Z32)
- 3-phase low voltage power module (TWR-MC-LV3PH) with included motor

All modules of the Tower system are available for order through 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 set-up 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 the mini USB connector, power supplies and the switch); see the following Figure 8.

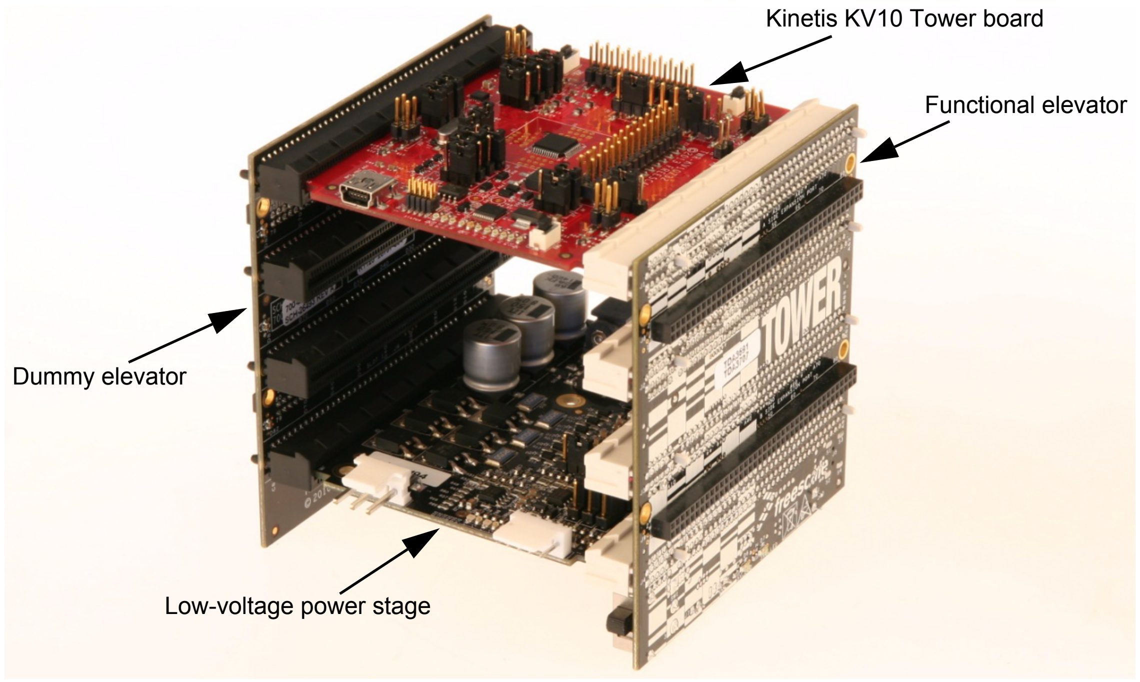


Figure 8. Hardware built on the modules of the Tower system

It is necessary to configure jumpers on both the Tower KV10 MCU module and the Tower 3-phase low-voltage power stage.

The jumper settings for the Tower KV10 board are listed in the Table 3. For detailed jumper descriptions refer to the TWR-KV10Z32 User Guide.

Table 3. Jumper Settings of TWR-KV10Z32 Board

Jumper #	Setting	Jumper #	Setting
J1	2-3	J14	open
J2	1-2	J18	2-3
J3	2-3	J19	2-3
J4	1-2	J20	2-3
J5	1-2	J21	3-4
J7	1-2	J22	3-4
J8	2-3	J25	open
J9	1-2	J26	1-2
J10	2-3	J27	1-2
J11	open	J28	1-2
J12	open	J29	1-2
J13	open	—	—

The jumper settings are listed in the Table 4 and the jumper positions are highlighted in Figure 9. See also the *TWR-MC-LV3PH User's Guide* (document TWRMCLV3PHUG) for more details (for example hardware overcurrent threshold setting) of the Tower low voltage power stage.

Table 4. Jumper Settings of TWR-MC-LV3PH Board

Jumper #		Setting	Note
J2	VDDA Source Select	1-2	Internal analog power supply
J3	VSSA Source Select	1-2	Internal analog power supply
J10	AN6 Signal Select	1-2	Phase C current signal
J11	AN5 Signal Select	1-2	Phase B current signal
J12	AN2 Signal Select	1-2	Phase A current signal

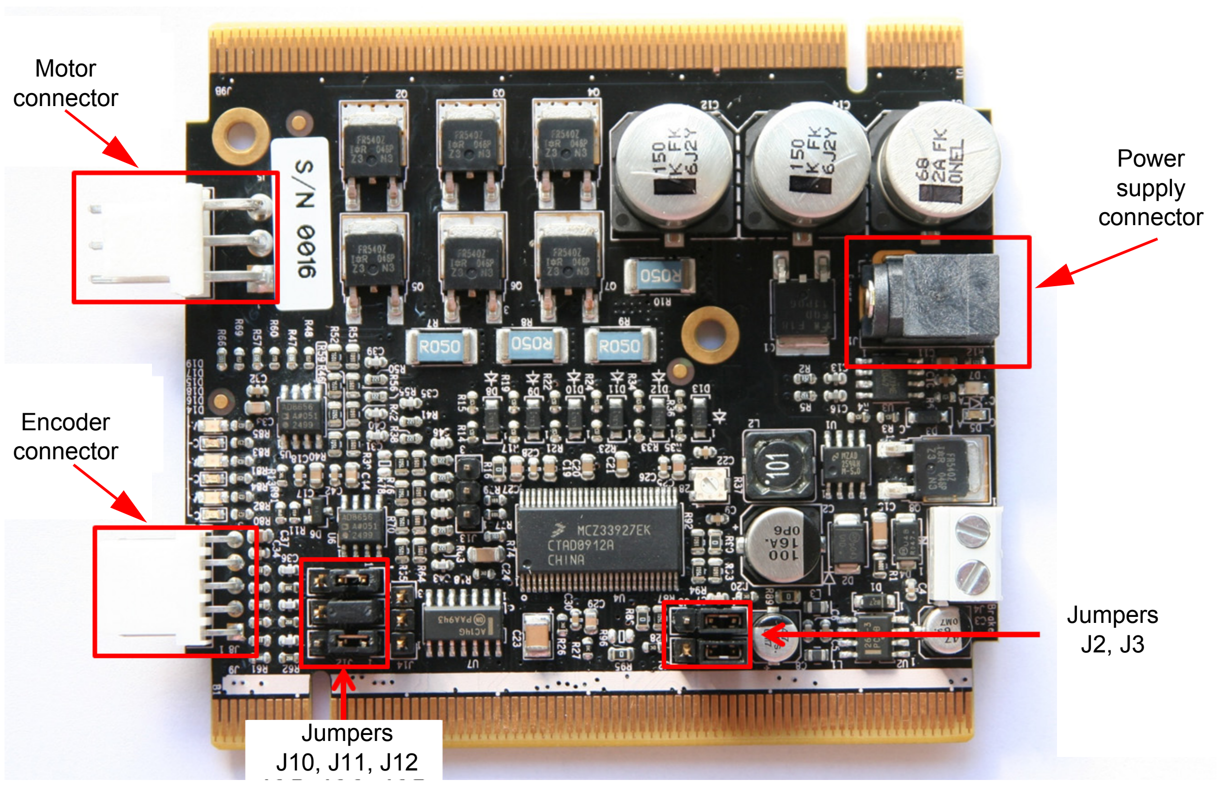


Figure 9. Jumpers and connectors positions on the TWR-MC-LV3PH

Table 5 shows the assignment of the signals of the motor connector of the TWR-MC-LV3PH.

Table 5. Motor Connector on the TWR-MC-LV3PH

Connector	Pin#	Description
Motor connector J5	1	Motor Phase A
	2	Motor Phase B
	3	Motor Phase C

The motor used in the reference design is part of the TWR-MC-LV3PH kit. It is a BLDC motor with the trapezoidal shape of the back-EMF voltage with salient poles on the stator. This shape is different than the PM synchronous motor which has distributed winding on the stator and forms a sinusoidal shape of the magnetic field. The construction of a rotor is same for both types of motors (salient poles on the shaft). The vector control algorithm which was originally developed for the PM synchronous motor assumes the sinusoidal shape of the magnetic field. It is possible to employ the same control strategy for a BLDC motor. The performance will not be optimal, but the drive will possess less audible noise compared to traditional six-step commutation control. The main benefit is that the customer can learn and adopt sensorless vector control on cost effective hardware solution.

Table 6 provides the motor specifications.

Table 6. Specification of the Motor

Motor Specifications	
Overall Motor Specification	
Manufacturer name	Linix
Type	45ZWN24-40
Nominal Voltage (line-to-line)	24V DC
Nominal Speed	4000 rpm
Rated power	40 W
Specific Model Specifications	
Stator Winding Resistance (line-to-line)	1 Ohm
Stator Winding Inductance d axis	426 μH
Stator Winding Inductance q axis	460 μH
Number of Pole-Pairs	2
Back-EMF constant k_e	0,01456 $V.s.rad^{-1}$

NOTE

The application parameters (speed PI controller and value of the start-up current) are set for the motor that has a plastic circle (part of the kit) mounted on the shaft, otherwise speed oscillation might occur.

9 Application operation

The application can be operated either using the user's buttons on the TWR-KV10 board (as mentioned in the Section 5.2, "Ports interrupt" or through the FreeMASTER software, which also enables visualizing 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 passed through the RS232 interface.

9.1 FreeMASTER installation on a PC or on a notebook

The FreeMASTER PC application can be downloaded from the Freescale webpage: www.freescale.com/freemaster, from the *Download* tab select *FreeMASTER 1.4 Application Installation*. Because downloading the FreeMASTER application requires registration, it is necessary to create an account before you can log-in. After you log into the system, the license agreement appears. You should read the license agreement and then you have to accept the agreement by clicking the *I Accept* button. If you are using Internet Explorer, then at the top of the web page you will see a bar asking to authorize the file download. Click on the bar and select *Download File*. A dialog box appears where you can choose to either Run or Save. In both cases, the installation archive will be stored on your machine. By selecting the *Save*, you have the option to select your preferred location to save the installation archive, otherwise it will be saved to a temporary folder created by your system. The library installation archive will be now downloaded to your computer.

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

9.2 Establishing the connection between the PC and the embedded application

The FreeMASTER enables the use of multiple communication interfaces between the embedded application and the PC or notebook (UART (RS232), CAN, Ethernet, USB, BDM, and so on). For this application the RS232 was used because the software overhead for the data transfer represents the lowest additional load on the CPU. Today, notebooks are not equipped with a COM port, so for this purpose the Kinetis KV10 tower module has in place a USB-to-RS232 interface (OpenSDA - CDC Serial Port). By connecting the Kinetis KV10 tower module with a notebook through the USB cable, a virtual serial port will be established in the Windows system.

9.3 Application operation using FreeMASTER

To run the FreeMASTER application double click on the **PMSM_FOC_KV1x.pmp** file located in the *\freemaster* folder. The FreeMASTER application starts and the environment will be automatically created, as it is defined in the *.pmp file.

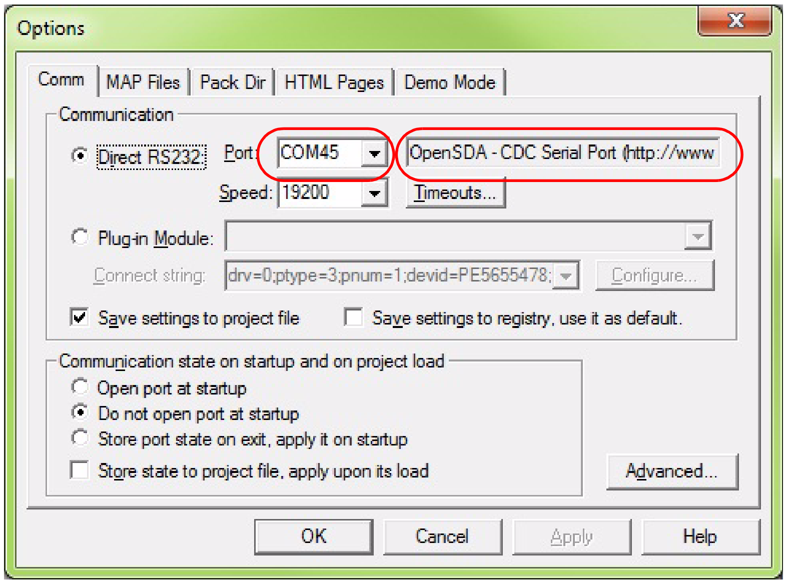
9.3.1 Setting up the communication

When the notebook is connected through the USB cable with the Kinetis KV10 tower module, the operating system assigns the number of the COM port to the OpenSDA - CDC Serial Port. This number is assigned randomly, therefore it is necessary to set the right communication port each time the connection is established (re-plugging the USB cable might cause a different port number assignment).

To set the port number, click the menu item ***Project \ Options …***

Then assign the port number in the **Comm** tab of the opened window.

The correct port number selection is confirmed by the text *OpenSDA – CDC Serial Port* next to the list box with available serial port numbers.


Options

Comm | MAP Files | Pack Dir | HTML Pages | Demo Mode

Communication

[✓ Direct RS232] Port: COM45 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...

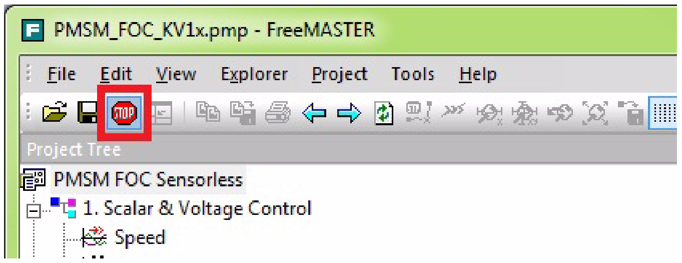
OK Cancel Apply Help

Figure 10. FreeMASTER communication settings

9.3.2 Application operation

9.3.2.1 Start/Stop the communication

When the communication settings are established, the communication between the PC and the embedded application can be initiated. Click the STOP button in the FreeMASTER toolbar as shown in Figure 11.


PMSM_FOC_KV1x.pmp - FreeMASTER

File Edit View Explorer Project Tools Help

Project Tree

PMSM FOC Sensorless

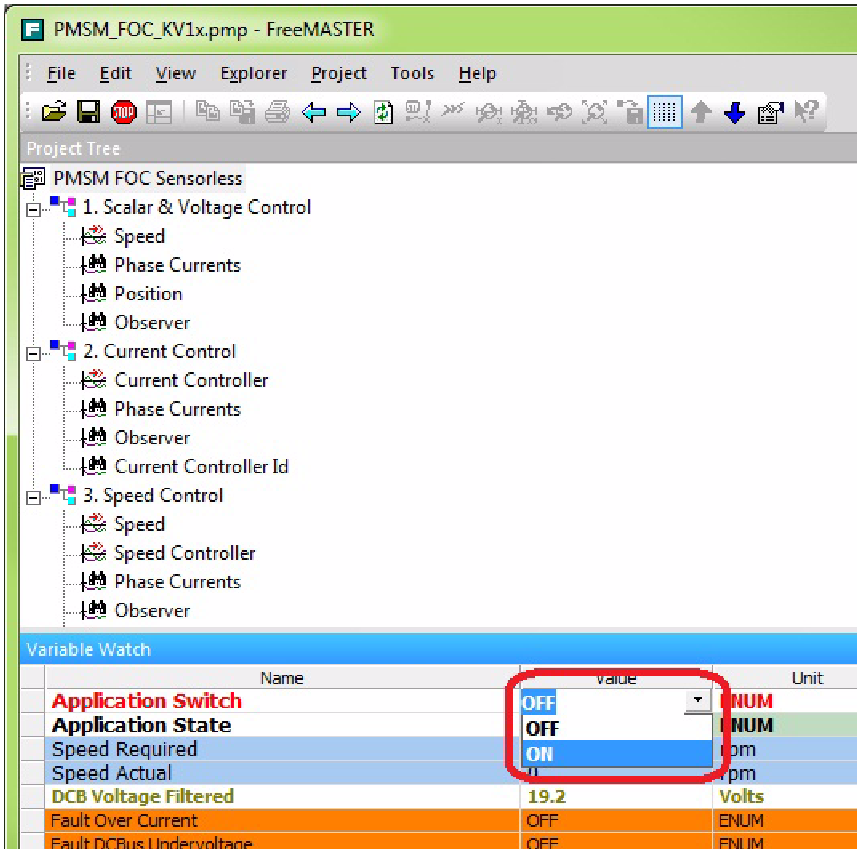
1. Scalar & Voltage Control

Speed

Figure 11. Initiating communication with the embedded side

9.3.2.2 Start/Stop the application, required speed setting

The next step is to switch the application to the RUN state. In the FreeMASTER window in the Variable Watch grid click on the drop-down list next to the *Application Switch* variable name, and select ON as Figure 12 illustrates:



Name	Value	Unit
Application Switch	OFF	ENUM
Application State		ENUM
Speed Required		rpm
Speed Actual	0	rpm
DCB Voltage Filtered	19.2	Volts
Fault Over Current	OFF	ENUM
Fault DCBus Undervoltage	OFF	ENUM

Figure 12. Start the application

After the application is set to the Run state, the required speed can now be changed to some 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.

9.3.2.3 Operation of the application from the control page

The application can also be operated from the control page, see the description in Figure 13.

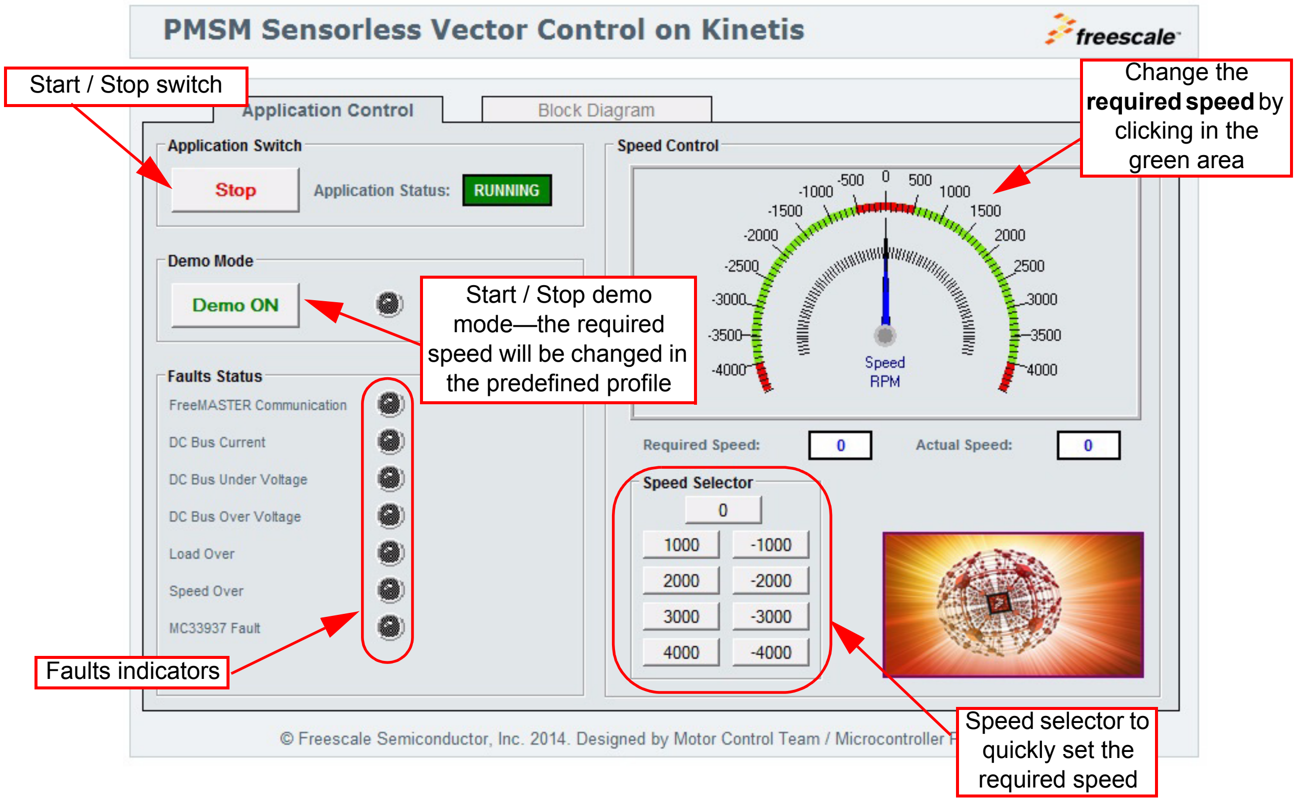


Figure 13. PMSM sensorless vector control page

9.3.2.4 Control mode selector

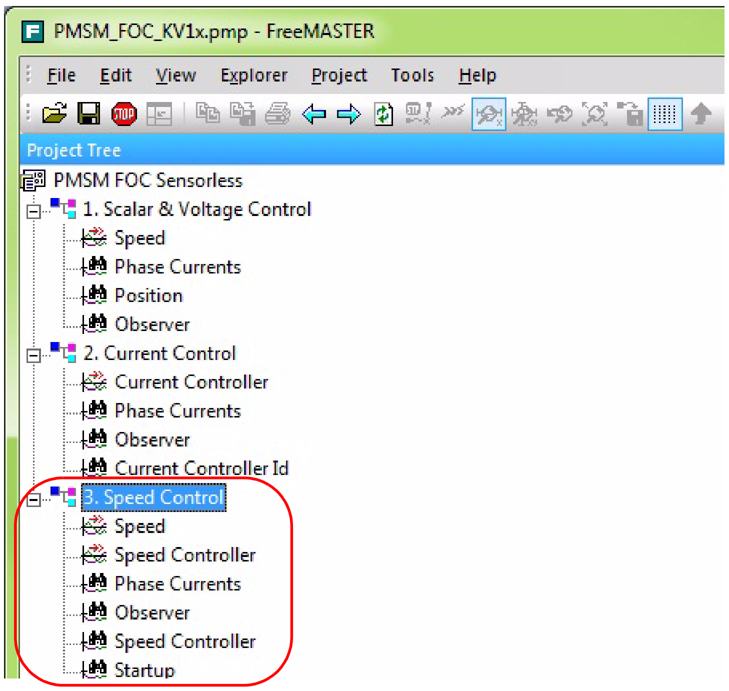
The application enables tuning of the PMSM sensorless application to any motor. For this purpose, the vector control application was divided into four parts (modes) that enables tuning the application in several steps, where in each step only a small number of unknown parameters are adjusted. To switch between these modes, a variable named *MCAT Control* is used. For more information about tuning the application to any motor, see *Tuning 3-Phase PMSM Sensorless Control Application Using MCAT Tool* (document AN4912).

The position detection algorithm is enabled only in *Speed FOC* control mode. In other control modes the initial position is determined by alignment. The calibration of the current channels offset is performed before alignment sub-state.

9.3.2.5 Scopes and Recorders

One of the main benefits of the FreeMASTER application is to visualize the values of the variables in real time. For this purpose, there are two possibilities. The user can select between the Scope and the Recorder. While the Scope feature downloads a stream of the data continuously in real time, the Recorder stores the data in a buffer located in the RAM of the embedded MCU and, after a trigger condition is met, the data is transferred in blocks through the communication interface and visualized in the FreeMASTER window. The sampling period of the Scope is limited by the speed of the communication interface, and therefore is used to slowly change quantities such as the speed. The sampling period of the oscilloscope-like Recorder is in the microseconds range to enable visualization of quickly changing quantities, such as the phase currents or the duty cycles. In this application, the recorder buffer is updated each time the fast control loop is executed, that is, every 100 µs.

Figure 14 shows the Recorders and Scopes that can be used when the application is running in the *Speed FOC* control mode.


PMSM_FOC_KV1x.pmp - FreeMASTER
File Edit View Explorer Project Tools Help
Project Tree

PMSM FOC Sensorless
- 1. Scalar & Voltage Control
  - Speed
  - Phase Currents
  - Position
  - Observer
- 2. Current Control
  - Current Controller
  - Phase Currents
  - Observer
  - Current Controller Id
- 3. Speed Control
  - Speed
  - Speed Controller
  - Phase Currents
  - Observer
  - Speed Controller
  - Startup

Figure 14. Scopes and recorders used in Speed FOC control mode

10 Measurement results

10.1 CPU load and the execution time

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 loops of the PMSM vector control are performed.

The complete ADC1 ISR requires 5209 (state machine and fast control loop) to 6036 (with the slow control loop calculation) machine cycles. The ADC1 interrupt is generated periodically with the same frequency as the PWM reload 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 a 10 kHz of PWM frequency. At 75 MHz on the Kinetis KV10 device, it consumes 69–80% of CPU performance.

10.2 FreeMASTER results

Figure 15 provides screenshots to show how the initial position detection method works. Separate figures with different values of the initial angle are presented. The FreeMASTER screenshots represent the maximum values of the phase currents as they were measured in each sector.

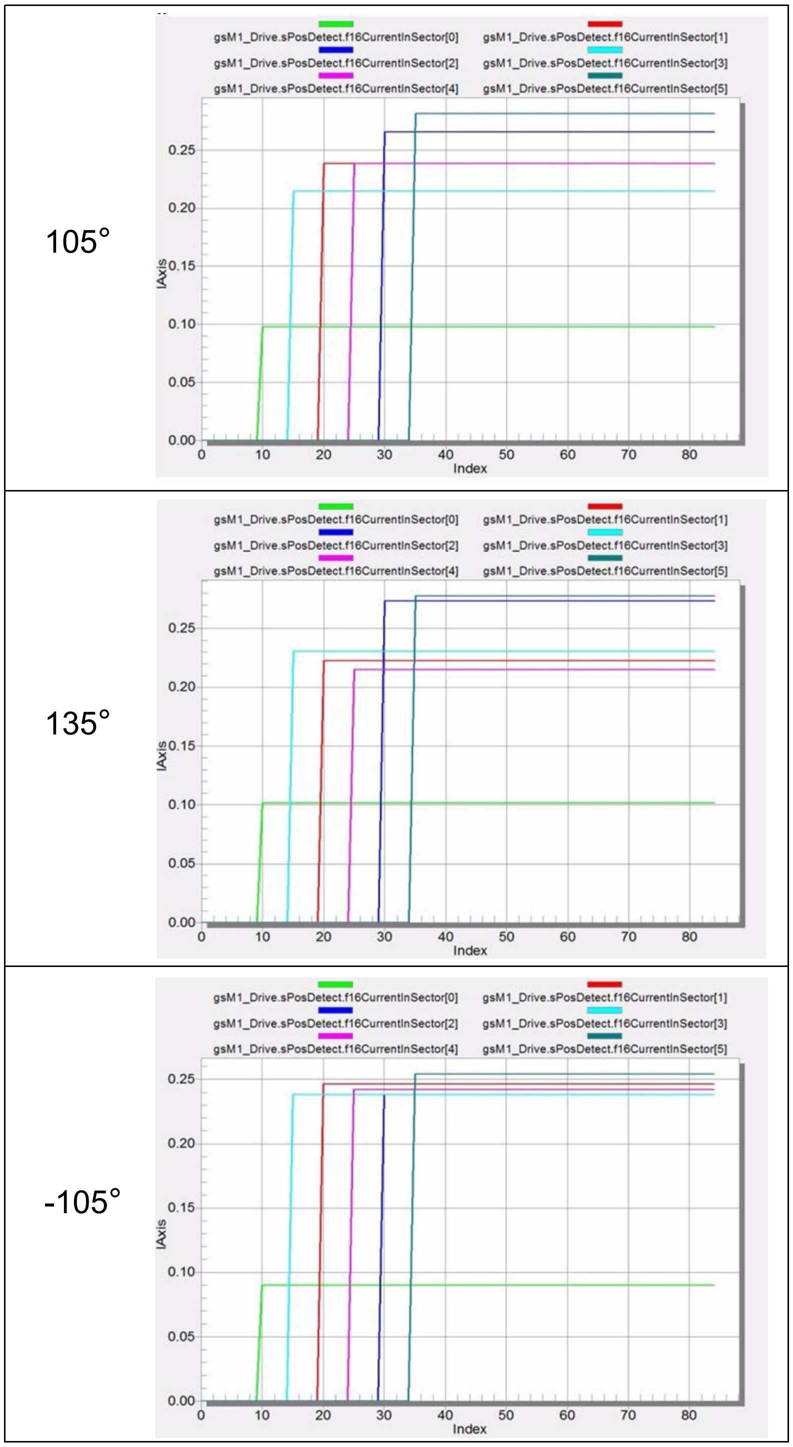


Figure 15. Maximum values of the phase currents during initial position detection obtained via the Recorder

11 Conclusion

The results of the execution time measurement show that the Kinetis KV10 microcontroller can be used to drive the PMSM sensorless vector control algorithm only for low-dynamic applications, as there is not enough CPU power to perform additional tasks. The presented solution offers an anti-wind algorithm, together with initial rotor position detection, that ideally suits ceiling fan applications.

12 References

A New Starting Method of BLDC Motors Without Position Sensor, by Wook-Jin Lee, Seung-Ki Sul, IEEE Trans. on Ind. Electronic, Vol. 42, no. 6, November / December 2006, pages 1532–1538

Sensorless PMSM Field-Oriented Control, Freescale Semiconductor (document DRM148)

Kinetis V Series KV10, 32/16 KB Flash Data Sheet, Freescale Semiconductor (document KV10P48M75)

KV10 Sub-Family Reference Manual, Freescale Semiconductor (document KV10P48M75RM)

Three Phase Field Effect Transistor Pre-driver Data Sheet, Freescale Semiconductor (document MC33937)

TWR-MC-LV3PH User's Guide, Freescale Semiconductor (document TWRMCLV3PHUG)

Tuning 3-Phase PMSM Sensorless Control Application Using MCAT Tool, Freescale Semiconductor (document AN4912)

Freescale documents are available online at www.freescale.com.

13 Acronyms and abbreviated terms

Table 7 contains the abbreviated terms used in this document.

Table 7. Acronyms and Abbreviated Terms

Term	Meaning
ADC	Analog to Digital Converter
Back-EMF	Back electromotive force—a voltage generated by rotating motor
FOC	Field Oriented Control—synonym for Vector Control
PMSM	Permanent Magnet Synchronous Motor
PWM	Pulse-Width Modulation
SVM	Space Vector Modulation—the algorithm used to generate the signals for PWM output

14 Revision history

Table 8. Revision history

Revision number	Revision date	Description of changes
0	05/2014	Initial release.
1	09/2014	Replaced Figure 6 and changed explanatory sentence that precedes it. Changed jumper number in Figure 9.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4935
Rev. 1
09/2014