Three-Phase BLDC Sensorless Control Using the MKV10x

by: Petr Staszko

1 Introduction

This application note describes the implementation of a sensorless three-phase Brushless DC (BLDC) motor control application (described in *DRM144: Three-Phase BLDC Sensorless Motor Control Application design reference manual*) using the Freescale 32-bit Kinetis MKV10x devices. The application software provided uses the concept of an isolated algorithm software and hardware. This software approach enables easy porting of an application between other devices or platforms. The application software is divided in following parts:

- BLDC motor control algorithm process input variables to output variables and flags.
- MKV10x hardware and microprocessor serves as a bridge between hardware peripheral modules and BLDC motor control software algorithm.

This application note describes the concrete implementation to Kinetis MKV10x devices. It does not discuss the BLDC motor control algorithm portion of the code.

Contents

The purpose of this application note is to implement a sensorless BLDC motor control algorithm on a MKV10x device.

It serves as an example of a BLDC motor control design focused on a simple and easy to understand control approach to BLDC, using the MKV10x device's peripheral modules. The application is bare-metal type.

This application note includes a basic overview of MKV10x device, system design concept, and hardware implementation.

The hardware is built on the Freescale Tower rapid prototyping system and contains the following modules:

- TWR-Elevator
- TWR-KV10Z32
- TWR-MC-LV3PH

1.1 Advantages and features of MKV10x device

Kinetis KV1x subfamily represents very low-cost portfolio of ARM® Cortex®-M0+ MCUs with peripheral modules dedicated for motor control applications. Typical application segment includes BLDC sensorless, PMSM sensorless FOC (with low dynamic), ACIM V/Hz, and FOC (low dynamic) motor control applications.

Key features of MKV10x devices are as follows:
- 75 MHz ARM Cortex-M0+
- Hardware divide peripheral module
- Square-root peripheral module
- 4-channel eDMA
- 16 or 32 KB Flash memory
- 8 KB SRAM
- IIC, SPI, 2 x UART communication ports
- 2 x 8-channel 16-bit ADC modules
- 12-bit DAC module
- 2 x analog comparators with 6-bit DAC
- 1 x 6-channel FlexTimer module
- 2 x 2-channel FlexTimer with quadrature decoding feature
- Programmable delay block
- 32-Bit CRC
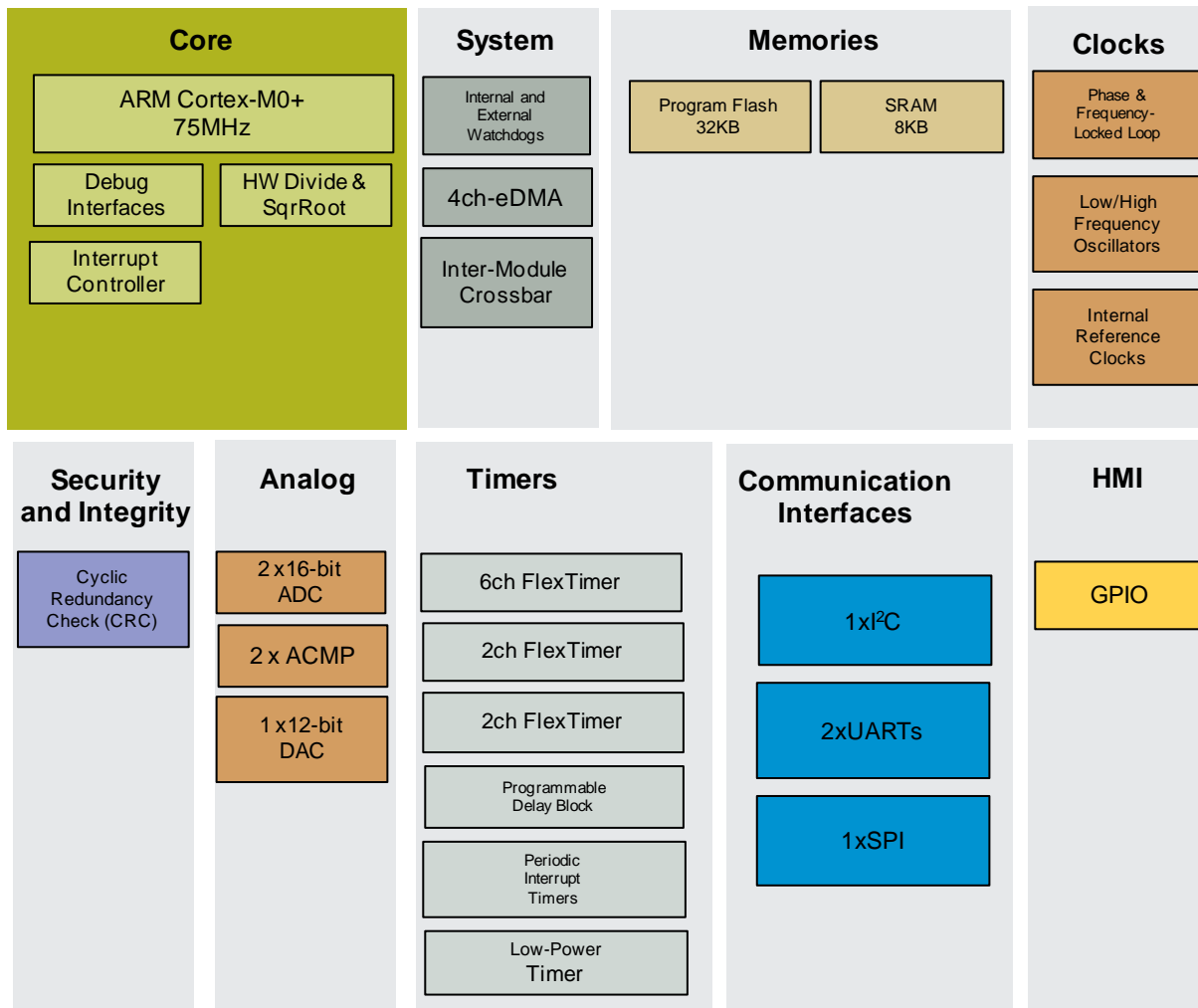- Up to 35 GPIOs
- 1.71–3.6 V, -40 to 105°C

Figure 1. MKV10x device overview

2 Application concept

This application is developed with a focus on easy porting of software across the devices or platforms. For this reason, the application software is divided into two separate parts of code:

- Hardware dependent code (dependent on the hardware boards used and the MCU device's peripheral modules). Includes CPU and peripheral modules initialization, I/O control drivers, and interrupt service routines handling.
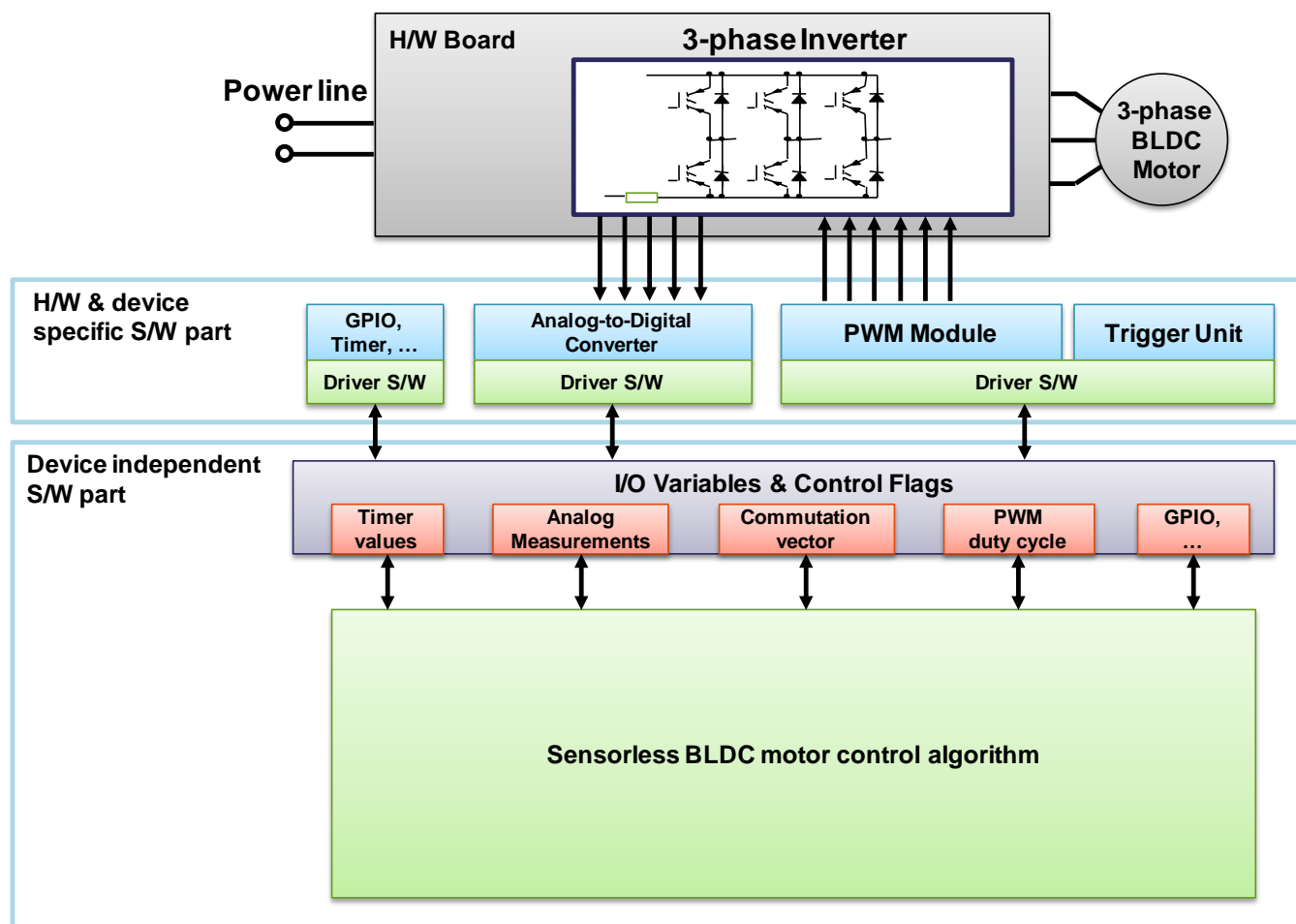- Hardware independent code (pure BLDC motor control application)

Figure 2. System concept overview

An overview of the system concept of application is shown in Figure 2. Input and output variables and output control flags are used as a bridge between the hardware specific part of code and motor control algorithm. Most of the application input and output variables are stored in a fractional number representation (in range from -1 to 1), scaling of these variable from/to the real values is calculated in drivers (hardware related part of code).

2.1 I/O values processing

This application uses the concept of a separate MCU specific part of code and hardware independent BLDC motor control algorithm. The hardware related code reads the input values from the peripheral modules (timer counter, ADC result, and others), scales them to the fractional number representation if needed and stores as the input variables; these input variables are then processed in the hardware independent BLDC motor control algorithm part of software. Similarly, the BLDC algorithm output variables are processed and updated into the respective peripheral modules in hardware related code. If there is a need to update output hardware parameter the corresponding *request flag* is set in the variable *uw16HwUpdateRequests* during the application control algorithm execution. Function

UpdateBldcHwModules() is used to check the status of *request flags* and perform the appropriate hardware modules update.

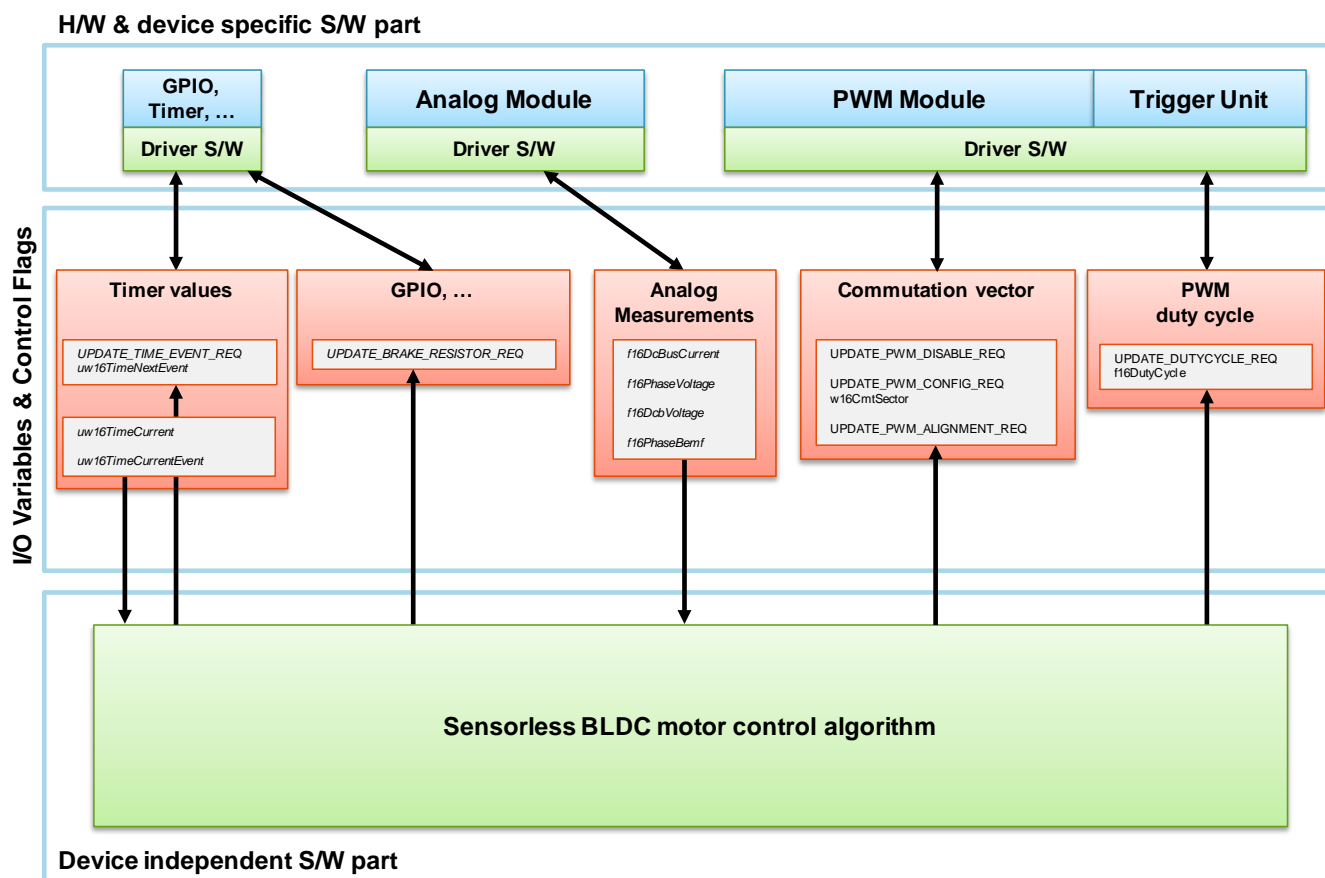The following figure illustrates the overview of application I/O data flow:

Figure 3. Application input/output data flow overview

2.2 Input application values

Input application variables are updated at the beginning of respective interrupt service routines:

- *uw16TimeCurrent* – actual time value. If timer is used for timing purpose, actual timer counter register value is stored in this variable.
- *uw16TimeCurrentEvent* – time of last *Time Event*. If timer is used for timing purpose, timer value register is stored in this variable.
- *f16DcBusCurrent* – DC bus current value in fractional number representation is stored in this 16-bit variable. -1 means the maximal negative DC Bus current; +1 means maximal positive DC Bus current.
- *f16DcBusCurrentOffset* – DC bus current offset value in fractional number representation is stored in this 16-bit variable. Value is measured during the calibration state after the application initialization.

- *f16DcbVoltage* – DC bus voltage value in fractional number representation is stored in this 16-bit variable. 0 means 0 Volts, +1 means maximal measurable DC bus voltage.
- *f16PhaseVoltage* – phase voltage value in fractional number representation is stored in this 16-bit variable. 0 means 0 Volts, +1 means maximal measurable DC bus voltage.
- *f16PhaseBemf* – calculated back-EMF voltage value in fractional number representation is stored in this 16-bit variable. The following equation is used to calculate the value (*f16DcbVoltage / 2* is considered as a middle point of the motor windings):

$$f16PhaseBemf = f16PhaseVoltage - \frac{f16DcbVoltage}{2}$$

2.3 Output application values and flags

There can be pending request for hardware output update after the execution of BLDC motor control algorithm. For this purpose *UpdateBldcHwModules()* function is executed. This function checks all the *request flags* stored in *uw16HwUpdateRequests* variable and do the appropriate action based on hardware configuration:

- *UPDATE_TIME_EVENT_REQ* – request to set new time of *Time Event*. In variable *uw16TimeNextEvent* there is stored new time to be set. If timer is used for timing purpose, timer value register is updated with the value of *uw16TimeNextEvent*.
- *UPDATE_PWM_DISABLE_REQ* – request to disable all PWM outputs. Powerstage outputs have to be switched off.
- *UPDATE_DUTYCYCLE_REQ* – request to update duty cycle generated with PWM module. In variable *f16DutyCycle* there is stored new value of duty cycle in the fractional number representation. Fractional number value in range <-1; +1) is then processed and scaled in appropriate PWM driver function to fit the actual PWM module configuration.
- *UPDATE_PWM_CONFIG_REQ* – request to update PWM outputs to set the new commutation vector stored in variable *w16CmtSector*. Integer value in *w16CmtSector* is in range from 0 to 5.
- *UPDATE_PWM_ALIGNMENT_REQ* – request to apply alignment vector on the PWM outputs. Alignment vector can be applied using special case of PWM module configuration.
- *UPDATE_BRAKE_RESISTOR_ON_REQ* – if this bit is set then output MOSFET for brake resistor control is turned on; if it is cleared the output MOSFET for brake resistor control is turned off.

2.4 Complementary bipolar switching PWM

Complementary bipolar switching PWM is used in the BLDC motor control application. With complementary bipolar switching, two phases are powered with the complementary PWM signals (Bottom MOSFET is switching in complement to the top MOSFET within a phase), one phase with the duty-cycle greater than 50%, second phase with complement duty-cycle value (less than 50%), as shown in Figure 4.
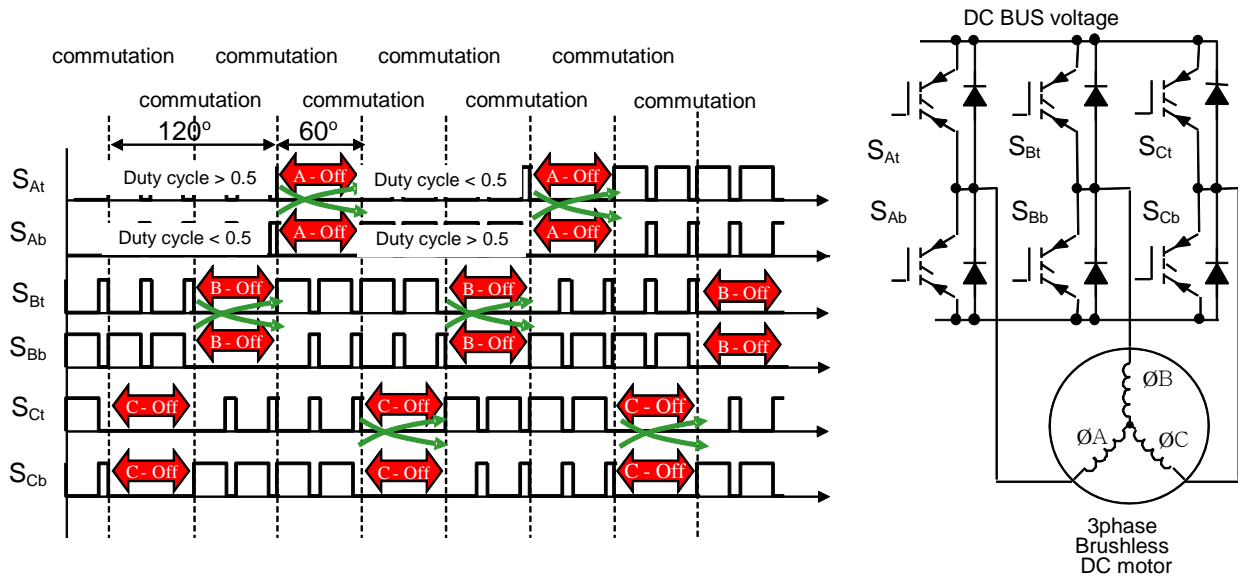
Figure 4. Complementary bipolar PWM switching

Advantage of complementary bipolar switching PWM technique is possibility to control motor in all four quadrants of operation. The bipolar PWM switching pattern requires that the top and bottom switch PWM signals need to be swapped at a commutation event. Another requirement is a dead-time insertion in the complementary top and bottom signals. All these requirements are well handled by FlexTimer on MKV10x device, will be described in FlexTimer registers update section.

3 Application details

This section describes MKV10x device peripheral modules configuration, interrupt service routines configuration, and BLDC motor commutation support using FlexTimer features.

3.1 MKV10x device configuration

Application uses following peripheral and external modules for sensorless BLDC motor control:
- Clock distribution module (MCG, SIM)
- Analog-to-digital converter (ADC0)
- FlexTimer modules (FTM1, FTM2, FTM3)
- Programmable delay block (PDB0)
- Serial peripheral interface (SPI0)
- Universal asynchronous receiver/transmitter (UART1)
- MC33927 external MOSFET driver (over SPI0)

3.1.1 Clock distribution modules (MCG, SIM)

MKV10Zx device uses two modules to configure and distribute clock across peripheral modules – MCG and SIM modules. MCG (Multipurpose Clock Generator) module provides several clock source choices

for the MCU. SIM module (System Integration Module) provides system control and chip configuration. The description of modules configuration is as follows:

MCG:
- Slow internal reference clock (32 768 Hz) used as a source
- FLL used to generate 72 MHz from input clock (FLL factor 2197)

SIM:
- Clock enabled to the all peripheral modules
- System clock (core clock) divider set to 1 => system clock set to 72 MHz
- Bus/Flash clock divider set to 3 => Bus clock set to 24 MHz

3.1.2 Analog-to-digital converter (ADC0)

ADC0 module is used for DC-bus voltage, DC-bus current, and phase voltages measurement; the module configuration is as follows:

- Input clock set to Bus clock / 1 = 24 MHz
- High speed mode (24 MHz)
- Short sample (6 ADCK cycles sample time)
- Single-ended input conversion mode
- 12-bit resolution
- Hardware trigger by PDB
- Conversion complete ISR, priority level 1 (high priority)

3.1.3 FlexTimer modules (FTM1, FTM2, FTM3)

FlexTimer modules are used for the output PWM generation (FTM0); commutation control, speed measurement, and application timing (FTM1); slow control loop (FTM2). The modules have the following configuration:

FTM0 – Output PWM generation:
- FTM in enhanced mode
- Input clock set to system clock / 1 = 72 MHz
- Output PWM frequency set to 20 kHz
- Center-aligned complementary PWM configured using combined (n) and (n+1) channels feature
- Complementary PWM generation
- Dead time set to 875 ns
- Duty-cycle updated using *LDOK* feature
- Output mask, inverting (swap), and software control registers updated using *software synchronization* feature (commutation)
- Reset FTM counter on commutation
- Trigger to PDB enabled

FTM1 – Commutation control:

- FTM in enhanced mode
- Free running timer
- Input clock set to system clock / 128 = 562 500 Hz
- Output compare mode
- ISR on output compare, priority level 1 (high priority)

FTM2 – Slow control loop:

- FTM in enhanced mode
- Free running timer
- Input clock set to system clock / 2 = 36 MHz
- Output compare mode
- ISR on output compare, priority level 3 (low priority)

3.1.4 Programmable delay block (PDB0)

PDB0 module is used to synchronize ADC0 measurement time within the PWM period. The module configuration is as follows:

- Input clock set to Bus clock / 1 = 24 MHz
- FTM0 external trigger (EXTRIG) selected as a source PDB0 trigger
- Double buffered mode selected for registers update
- Pre-trigger0 point set to the middle of the PWM pulse
- Back-to-back mode set for pre-trigger1 (Second ADC measurement)
- Sequence Error ISR Enabled, priority level 1 (high priority)

Note
The PDB clock is bus clock (24 MHz) and the FTM clock is the system clock (72 MHz).

3.1.5 Serial peripheral interface (SPI0)

SPI0 is a four-wire interface used for MC33927/33937 three-phase predriver configuration and status reading. The module configuration is as follows:

- Baud rate set to 1.28 MHz
- Falling-edge SCLK polarity
- Master SPI mode

3.1.6 Universal asynchronous receiver/transmitter (UART1)

UART1 is used for FreeMASTER communication between the MCU board and PC. The module configuration is as follows:

- Baud rate set to 9600 bps
- Enabled receiver and transmitter
- Other setting by default

3.1.7 MC33927 external MOSFET driver (over SPI0)

MC33927/MC33937 three-phase MOSFET driver is used for output control. SPI0 communication channel is used to configure and read status of the driver. Configuration of the driver is as follows:

- Desaturation protection enabled
- Interrupts disabled (pooling mode used instead)
- Dead-time set to zero (used dead-time on FlexTimer)

3.2 Application interrupt service routines

Application uses four interrupt services, three interrupts are part of the BLDC motor control algorithm and one interrupt is used for trigger error clearing.

- ADC0 ISR – Fast Control loop ISR, executed every PWM period (50 us)
- FTM1 ISR – Time Event ISR, executed according to the current application needs
- FTM2 ISR – Slow Control loop ISR, executed with 1 ms period
- PDB0 ISR – Trigger error fault clearing

Following functions (defined in arm_cm0.h file) can be used to enable, disable, or set priority of individual ISRs:

- disable_irq(IRQ_number)
- enable_irq(IRQ_number)
- set_irq_priority(IRQ_number, priority_level)

ISR numbers and priority levels are summarized in Table 1:

Table 1. Interrupt services list

Source	Module Function	Application Function	Vector number	ISR name	Level
ADC0	Conversion Complete	Fast Control loop	VECTOR_031	adc0_isr()	1
FTM1	Output Compare	Time Event	VECTOR_034	ftm1_isr()	1
FTM2	Output Compare	Slow Control loop	VECTOR_035	ftm2_isr()	3
PDB0	Trigger error	Trigger error	VECTOR_045	pdb0_isr()	1

As default, all interrupt vectors are set to dummy ISR function, *default_isr()*. To replace default ISR the following code is used in *hwconfig.h* file:

```
#undef  VECTOR_031
#define VECTOR_031 adc0_isr
```

```
#undef  VECTOR_034
#define VECTOR_034 ftm1_isr

#undef  VECTOR_035
#define VECTOR_035 ftm2_isr

#undef  VECTOR_045
#define VECTOR_045 pdb0_isr
```

3.2.1 ADC0 conversion complete ISR

ADC0 module is used for three analog measurements every 1 PWM period (DC-bus voltage, DC-bus current, and phase voltage on disconnected motor's phase). Because KV10x device has only two channel select and result registers per ADC module, only two sequential measurements can be done at one hardware trigger from PDB, then ADC0 conversion complete interrupt is generated. The third measurement is configured, started, and done within this ISR. ADC is configured to software trigger mode, conversion is started by writing a channel configuration register, algorithm waits for end of conversion, and then ADC0 is reconfigured back to hardware trigger mode. ADC conversion results are processed to left-aligned values. Fast control loop function is then executed. Interrupt request conversion complete flag (COCO) is cleared when the respective SC1n (channel select) register is written or when the respective Rn (result register) register is read.

3.2.2 FTM1 output compare ISR

FTM1 ISR is used for timing purpose and commutation timing during open-loop BLDC startup. FTM1 uses 562 500 Hz input clock, maximal period defined by 16-bit positive number is 58.25 ms (maximal commutation period). Interrupt flag needs to be cleared by writing zero on it.

3.2.3 FTM2 output compare ISR

FTM2 ISR is used for slow control loop function execution and MC33927 status read. This ISR is executed in every 1 ms. FTM2 uses 36 MHz input clock, channel value register is incremented by value of 36 000 every ISR. Interrupt flag needs to be cleared by writing zero on it.

3.2.4 PDB0 trigger error ISR

PDB0 trigger error interrupt is generated in case it is unable to trigger ADC module. Typically, it is caused if PDB trigger is generated while ADC conversion is still in progress, and therefore it is unable to start another ADC conversion (too short period between two PDB triggers). PDB module will stop its operation if there is trigger error flag set. To clear the error flag, PDB0 module needs to be disabled, then the error flag can be cleared by writing zero on it, then PDB0 module can be enabled.

3.3 FlexTimer registers update

FlexTimer in enhanced mode provides several options for double buffered registers update. There are two algorithm control loops present in BLDC motor control applications – output voltage control loop and commutation control loop. These two control loops are independent to each other and FlexTimer on

KV10x device provides configuration option for independent update of double buffered registers for output voltage control loop and for commutation control loop.

3.3.1 Output voltage control

In the slow control loop, which is executed in every 1 ms, the output voltage (PWM duty-cycle) is calculated and applied to achieve the required motor speed. PWM duty-cycle on all phases needs to be applied simultaneously to avoid any disturbances or hazards at the powerstage output lines. For this purpose FlexTimer module provides double buffered mechanism, where MOD (modulo), CNTIN (counter initial value), and CnV (channel n value) registers are loaded with their write buffers while LDOK bit in PWMLOAD register is set and new PWM cycle begins.

Example code for PWM output duty-cycle update using LDOK mechanism:

```
FTM0->CONTROLS[0].CnV = FirstEdge;
FTM0->CONTROLS[1].CnV = SecondEdge;
FTM0->CONTROLS[2].CnV = FirstEdge;
FTM0->CONTROLS[3].CnV = SecondEdge;
FTM0->CONTROLS[4].CnV = FirstEdge;
FTM0->CONTROLS[5].CnV = SecondEdge;
FTM0->PWMLOAD |= FTM_PWMLOAD_LDOK_MASK;
```

3.3.2 Commutation control

Time of commutation event depends on the actual rotor's position; therefore the commutation period depends on the actual motor speed. During the commutation the PWM output configuration needs to be updated simultaneously. FlexTimer on KV10x device supports simultaneous update of output control registers, at top of that FlexTimer supports also FTM counter initialization as well at the same time.

Following FlexTimer output registers are configured for simultaneous synchronization in the application:

- Software output control register (SWOCTRL) – This register enables software control of channel (n) output and defines the value forced to the channel (n) output.
- Output mask register (OUTMASK) – This register disables or enables output PWM generation on channel (n) output pin.
- Inverting control register (INVCTRL) – This registers swaps top and bottom channels within one PWM pair.

In this application software only OUTMASK and INVCTRL registers are used to perform commutation. SWOCTRL register should be useful in different PWM switching types (for example unipolar PWM type).

Application software uses array variable *bldcCommutationTableComp[8]* pointing to a table with predefined values for OUTMASK and INVCTRL registers according to the vector being applied to FlexTimer outputs. See Table 2 for details:

Table 2. Commutation table

OUTMASK	INVCTRL	Output	Phase A	Phase B	Phase C
0x30	0x02	vector 0	Normal	Inverted	Off
0x0C	0x04	vector 1	Normal	Off	Inverted
0x03	0x04	vector 2	Off	Normal	Inverted
0x30	0x01	vector 3	Inverted	Normal	Off
0x0C	0x01	vector 4	Inverted	Off	Normal
0x03	0x02	vector 5	Off	Inverted	Normal
0x00	0x06	alignment vector	Normal	Inverted	Inverted
0x3F	0x00	PWM Off	Off	Off	Off

Example code for FlexTimer output configuration and synchronization is as follows:

> FTM0->INVCTRL = bldcCommutationTableComp[sector].swap;
> FTM0->OUTMASK = bldcCommutationTableComp[sector].mask;
> FTM0->SYNC |= FTM_SYNC_SWSYNC_MASK;

To perform a commutation, FlexTimer output control registers (OUTMASK, INVCTRL, and SWOCTRL respectively) are loaded with the corresponding values; then SWSYNC bit is set in the SYNC register. In application with current FlexTimer configuration, OUTMASK, INVCTRL, and SWOCTRL registers will be applied simultaneously, and FlexTimer counter will be loaded with its initialization value (CNTIN). CNTIN, MOD, and CnV registers will not be affected using this synchronization as they are loaded from their write buffers using LDOK bit in PWMLOAD register.

4 User interface

The application can be controlled using two interfaces:

- Up / Down buttons on TWR-KV10Z32 board
- Remote control using FreeMASTER running on PC, connected to TWR-KV10Z32 via mini-USB port

4.1 Up/down buttons control

After power supply is plugged in TWR-MC-LV3PH, the motor is prepared to run.

- Pressing the Up button (SW1) increases the speed by 500 rpm. Motor will start rotating in the clockwise direction if it is not spinning, or decrease speed if the actual direction of rotation is counter-clockwise.
- Pressing the Down button (SW2) decreases the speed by 500 rpm. Motor will start rotating in the counter-clockwise direction if it is not spinning, or decrease speed if the actual direction of rotation is clockwise.
- Further pressing the buttons increases or decreases the required speed within the speed limit -5000 to 5000 rpm.

- If both buttons are pressed for more than 2 seconds then the demonstration mode is switched on (or demonstration mode is switched off if it is on)

4.2 Remote control using FreeMASTER

Remote operation can be provided by FreeMASTER software via the USB interface. For a correct FreeMASTER operation, follow the given steps:

1. Open FreeMASTER and go to Project->Options->Comm and set communication via to the Direct RS232.
2. Select the virtual COM port where the TWR-KV10Z32 board is connected - see System Properties->Device Manager->Ports and look for "OpenSDA – CDC Serial Port (http://www.pemicro.com/opensda) (COMxx)". Set communication speed to 9600 bps.
3. The next step is to toggle the communication button (Red STOP button). After that, in the bottom right-hand corner, there should be RS232;COMxx;9600, which means that communication has been established.
4. If not, toggle (STOP) the communication and unplug/plug the FreeMASTER USB cable. Then toggle (START) the communication button.

After launching the application and performing all necessary settings, click the "App Control" page in the main window (Motor Control Application Tuning Tool), as shown in Figure 5. In this view, variables used for the application state, speed, PI controller, and ramp settings are visible at the bottom window; most important variables and settings are displayed using the graphics representation. Application can be switched on or off using "App switch" button or selecting state of "Application Switch" variable in the variable list window. You can enter required speed (in range -5000 to 5000) or click on the speed tachymeter to set the required speed. DC Bus current limiting value can be also set by clicking on the Ampere meter. In case any fault is detected it has to be cleared manually by entering zero value to "Fault" variable; then the application can be switched on again.
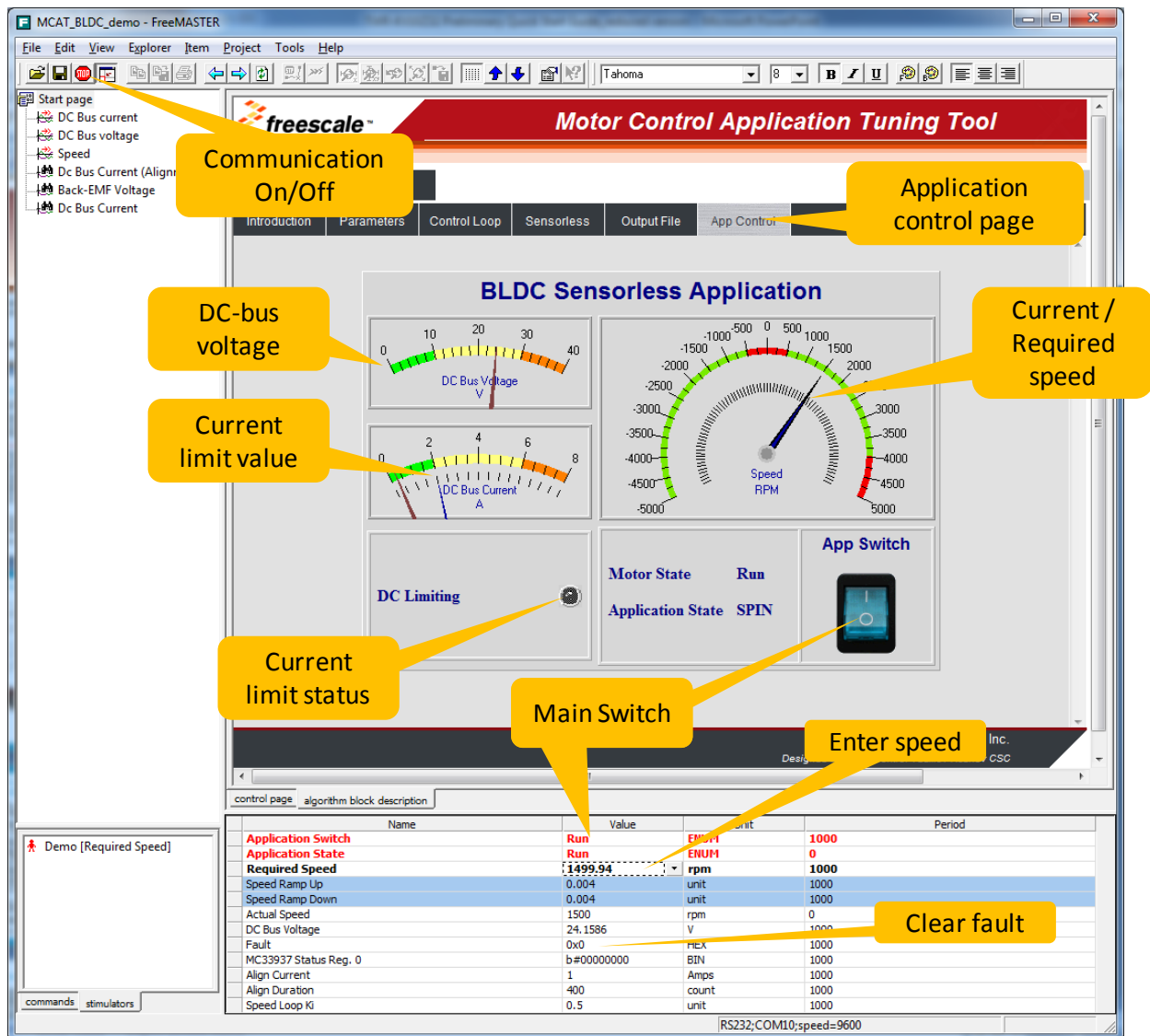
Figure 5. FreeMASTER project page

5 Hardware

The following hardware modules are required for this application to function properly:

- TWR-KV10Z32
- TWR-MC-LV3PH
- TWR-Elevator

5.1 TWR-MC-LV3PH

The 3-phase Low-Voltage Motor Control board (TWR-MC-LV3PH) is a peripheral Tower System Module, interchangeable across the Tower development platform. Phase voltage and current feedback signals are provided; that allow a variety of algorithms to control 3-phase PMSM and BLDC motors.

High level of board protection (overcurrent, undervoltage, overtemperature, and others) provided by MC33937 predriver.

The features of TWRMCLV3PH module are as follows:

- Power supply voltage input of 12-24 V DC, extended up to 50 V
- Output current up to 8 A
- Power supply reverse polarity protection circuitry
- 3-phase bridge inverter (6-MOSFETs)
- 3-phase MOSFET gate driver with overcurrent and undervoltage protection
- 3-phase and DC, bus-current-sensing shunts
- DC bus-voltage sensing
- 3-phase back-EMF voltage sensing circuitry
- Low-voltage on-board power supplies
- Encoder/Hall sensor sensing circuitry
- Motor power and signal connectors
- User LED, power-on LED, 6 PWM LED diodes

Jumpers' setting for BLDC application is listed in Table 3:

Table 3. TWR-MC-LV3PH jumper setting

Jumper	Position
J2	1-2
J3	1-2
J10	2-3
J11	2-3
J12	2-3
J13	2-3
J14	1-2
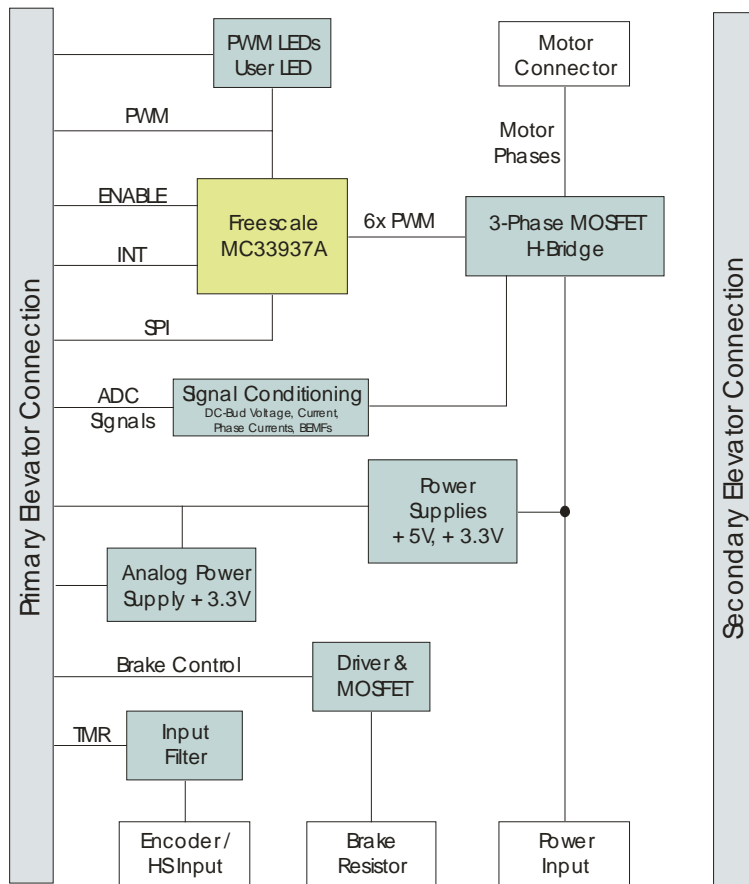
Figure 6. TWR-MC-LV3PH block diagram
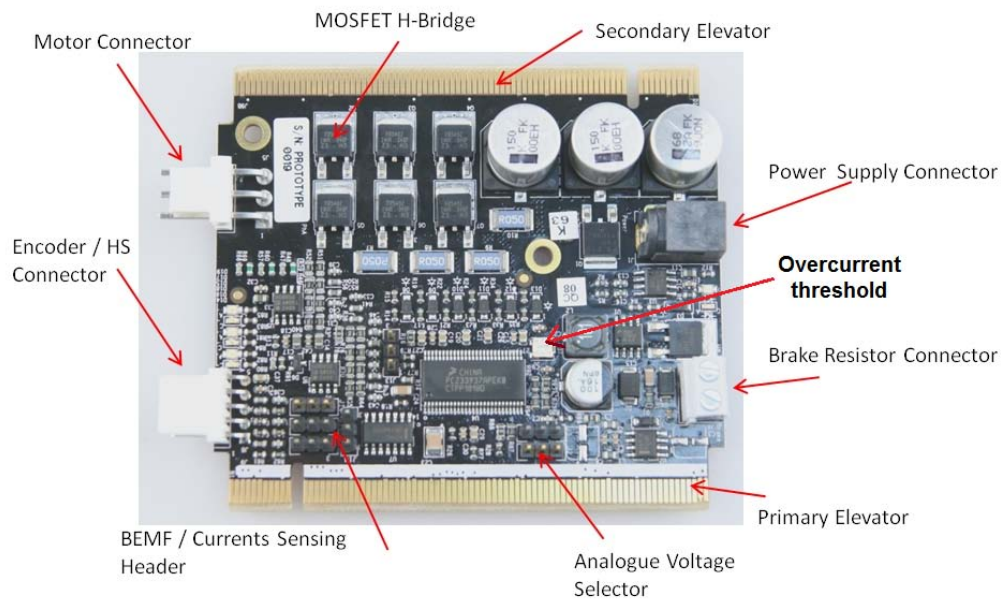
Figure 7. TWR-MC-LV3PH image

5.2 TWR–KV10Z32

The TWR-KV10Z32 microcontroller module is part of Freescale Tower System, a modular development platform that enables rapid prototyping and tool reuse through reconfigurable hardware.

- 3.3 V power supply regulator from USB
- Digital accelerometer
- 8 User LEDs
- 2 User push buttons, 1 reset button
- 4 thermistors
- Open SDA interface
- Motor Control Connector
- Tower signals compatible with TWR-MC-LV3PH power stage board
- 10 MHz crystal

Jumper settings for BLDC application is listed in Table 4:

Table 4. TWR-KV10Z32 jumper setting

Jumper	Position	Jumper	Position	Jumper	Position
J1	2-3	J10	1-2	J21	3-4
J2	short	J11	open	J22	3-4
J3	2-3	J12	open	J25	open
J4	short	J13	open	J26	short
J5	short	J14	open	J27	short
J7	1-2	J18	2-3	J28	short
J8	1-2	J19	2-3	J29	1-2
J9	1-2	J20	2-3	–	–
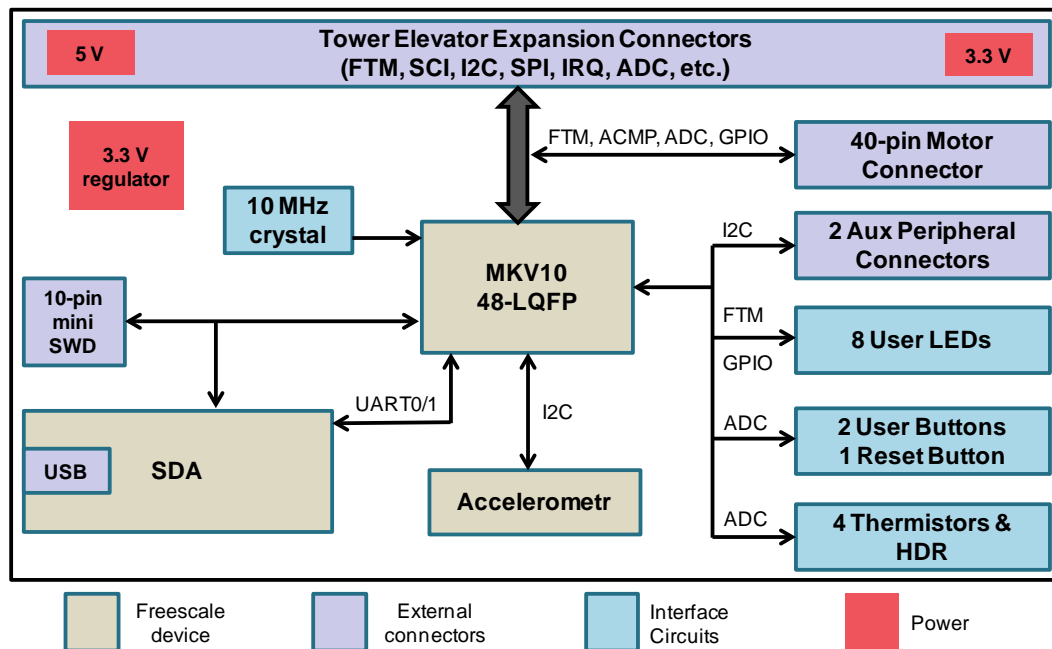
Figure 8. TWR-KV10Z32 block diagram
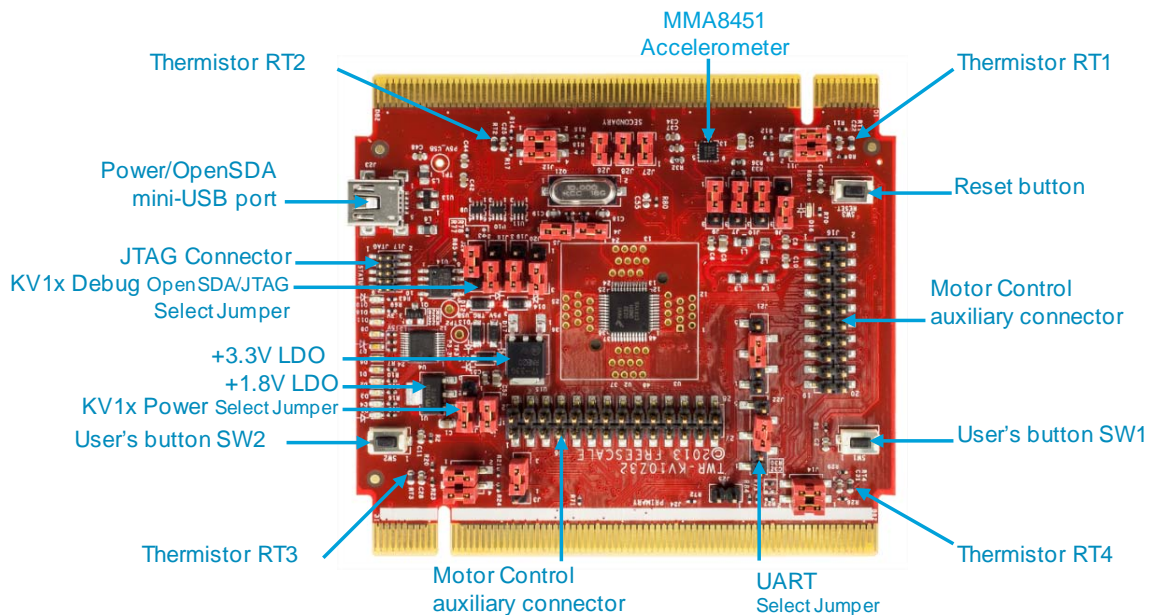
Figure 9. TWR-KV10Z32 image

5.3 Motor LINIX 45ZWN24-40

Motor described in Table 5 is used by the BLDC Sensorless application. Other motors can also be adapted to the application, just by defining and changing the motor related parameters.

Table 5. Motor parameters

Characteristic	Symbol	Value	Units
Rated Voltage	Vt	24	V
Rated Speed @ Vt	–	4000	RPM
Rated torque	T	0.0924	Nm
Rated power	P	40	W
Continuous Current	Ics	2.34	A
Number of Pole Pairs	PP	2	–

5.4 Microcontroller memory usage

Table 6 shows the memory resources used in the sensorless BLDC motor control application, including the FreeMASTER tool and demonstration code.

Table 6. Memory usage

Code type	Flash [bytes]	RAM [bytes]
Application	6212	239
Demonstration code	836	20
FreeMASTER	2528	2241
Total	9576	2500

6 Acronyms and abbreviations

Table 7. Acronyms

Term	Meaning
ACIM	Alternate current induction motor
ADC	Analog-to-digital converter.
BEMF	Back-electromotive force.
BLDC	Brushless DC motor.
CCW	Counter-clockwise direction.
CMP	Comparator.
CPU	Central processing unit.
CW	Clockwise direction.
DAC	Digital-to-analog converter.
DC	Direct current.
DMA	Direct memory access module.
DRM	Design reference manual.

Term	Meaning
FOC	Field oriented control.
FTM	FlexTimer module.
GPIO	General-purpose input/output.
I/O	Input/output interfaces between a computer system and the external world — a CPU reads an input to sense the level of an external signal and writes to an output to change the level of an external signal.
I²C	Inter-Integrated Circuit interface.
ISR	Interrupt Service Routine.
KV10x	Freescale 32-bit ARM based sub-family microcontroller.
MCU	Microcontroller.
PDB	Programmable delay block module.
PWM	Pulse-width modulation.
RPM	Revolutions per minute.
UART	Universal asynchronous receiver/transmitter.
SPI	Serial peripheral interface module.

7 References

Following references are available on freescale.com:

1. *K10 Sub-Family Reference Manual,* KV10P48M75RM, by Freescale Semiconductor, Inc., 2013.

2. *3-Phase BLDC Motor Sensorless Control using MC9S08AW60,* DRM086, by Freescale Semiconductor, Inc., 2005.

3. *3-Phase BLDC Motor Sensorless Control using MC9S08MP16*, DRM117, by Freescale Semiconductor, Inc., 2009.

8 Revision history

Table 8. Revision history

Revision number	Date	Substantial changes
0	02/2014	Initial release

Document Number: AN4862
Rev.0, February 2014

ARM
POWERED

freescale™