Freescale Semiconductor
Application Note

Document Number: AN4254
Rev. 0, 5/2011

Motor Control Under the Freescale MQX Operating System

by: Libor Prokop
Czech Republic
Roznov

Contents

1 Introduction..1
2 Typical Motor Control Application...........................3
3 Integration of motor control in MQX.......................10
4 Demonstrating a BLDC Motor Control Application under MQX 20
5 BLDC Motor Control under MQX – Code Examples 20
6 Bibliography...23
7 Definitions and Acronyms.......................................23

1 Introduction

This application note explains the challenges of motor control in an operating system (OS). It evaluates the possible implementation approaches, and guidance for writing motor control applications in the Freescale operating system MQX.

1.1 Motor control and MQX

Embedded applications are becoming more complex and is putting more pressure on embedded system software programmers. In a complex system, a number of tasks must run in parallel in real time in an operating system. Examples include Ethernet, USB, SDHC, and so on. One such task is an electrical motor control, like DC, Brushless DC, stepper, or even a 3-phase sinusoidal motors such as the PMSM, or AC induction motors. The motor control algorithm requires precise timing of the control tasks, such as the output signal generation, which is based on the scheduled tasks of the opposing rotor position in the operating system (OS), and with significant latency. Therefore, the inclusion of motor control tasks in an operating system requires special care. Motor control in this document means a process that controls electrical motors such as AC induction, BLDC, PM synchronous, or DC MQX.

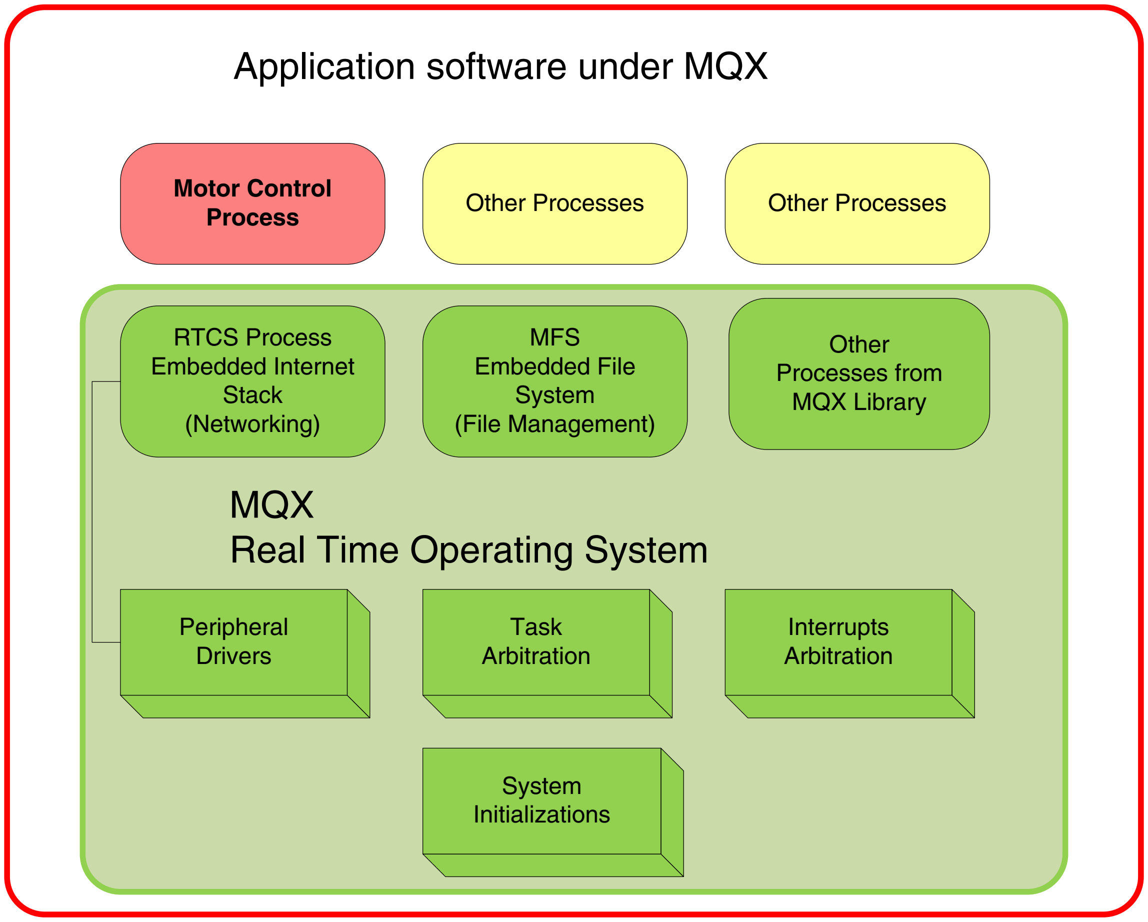
Application software under MQX
Motor Control Process
Other Processes
Other Processes
RTCS Process Embedded Internet Stack (Networking)
MFS Embedded File System (File Management)
Other Processes from MQX Library
MQX
Real Time Operating System
Peripheral Drivers
Task Arbitration
Interrupts Arbitration
System Initializations

Figure 1. MQX and other applications with motor control in the MQX

1.2 MQX real time operating system for high-end microcontrollers

Here are basic features of the MQX operating system. MQX is a run-time library of functions that programs use to become real-time multi-tasking applications. The main features are its scalable size, component-oriented architecture, and easy use. The MQX supports multi-processor applications and can be used with flexible embedded I/O products for networking, data communications, file management, and control. The main MQX application area is for large controller devices like the Kinetis (ARM®Cortex™-M4) or MCF5441x (ColdFire®) families with peripheries for Ethernet, USB, SDHC, and other support. Some of those devices are equipped with a PWM module and other peripherals designed or suitable for motor control. MQX is not exclusively a motor control dedicated operating system, but using MQX operating system for motor control brings some benefits to applications that combine motor control with other significant processes.

The Freescale MQX RTOS comes with the support of a complete software stack combined with basic core drivers, class drivers, and plenty of sample programs that can be used to achieve the desired target product. The MQX Real-Time Operating System from the MQX Embedded has been designed for a uni-processor, multi-processor, and distributed-processor embedded real-time systems. To leverage the success of the MQX operating system, Freescale Semiconductor adopted this software platform for its ColdFire, PowerPC™ and ARM Cortex families of microprocessors. Comparing to the original MQX distributions, the Freescale MQX distribution was made simpler to configure and use. One single release now contains the MQX operating system plus all the other software components supported for a given microprocessor part.

2 Typical Motor Control Application

There are several motor types that differ in construction and also in the control approach. This must be reflected in the implementation under the MQX OS.

2.1 Motor types

The most common electrical motor control applications according to motor type are:

- DC motors

  DC (direct current) motor control

- Commutating motors

  Brush-less DC (BLDC) motor control
  Stepper motor control

- Stepper motor control

  Permanent Magnet Sinusoidal motor (PMSM) control
  AC induction motor control
  Stepper motor control

2.2 Motor control techniques

The key motor control applications according to the control techniques.

2.2.1 According to the sensor

- Sensored control

  A sensor is used for rotor position and speed estimation. Most common sensor types are:
  — Hall sensor
  — Incremental encoder
  — Sin cos sensor
  — Tachogenerator

- Sensorless control

  — The rotor position and speed is estimated from the motor current and voltage without using other sensors

2.2.2 According to the control signal

- Sinusoidal scalar control
  - Classical control technique for sinusoidal PMSM or AC induction motors

- Vector control (FOC)

    - Advanced control technique for sinusoidal PMSM or AC induction motors for high dynamic and precision drives

- Commutation control
    - BLDC motor commutation control
    - Stepper motor stepping control

- Other control techniques

2.3 Motor application characteristic requirements

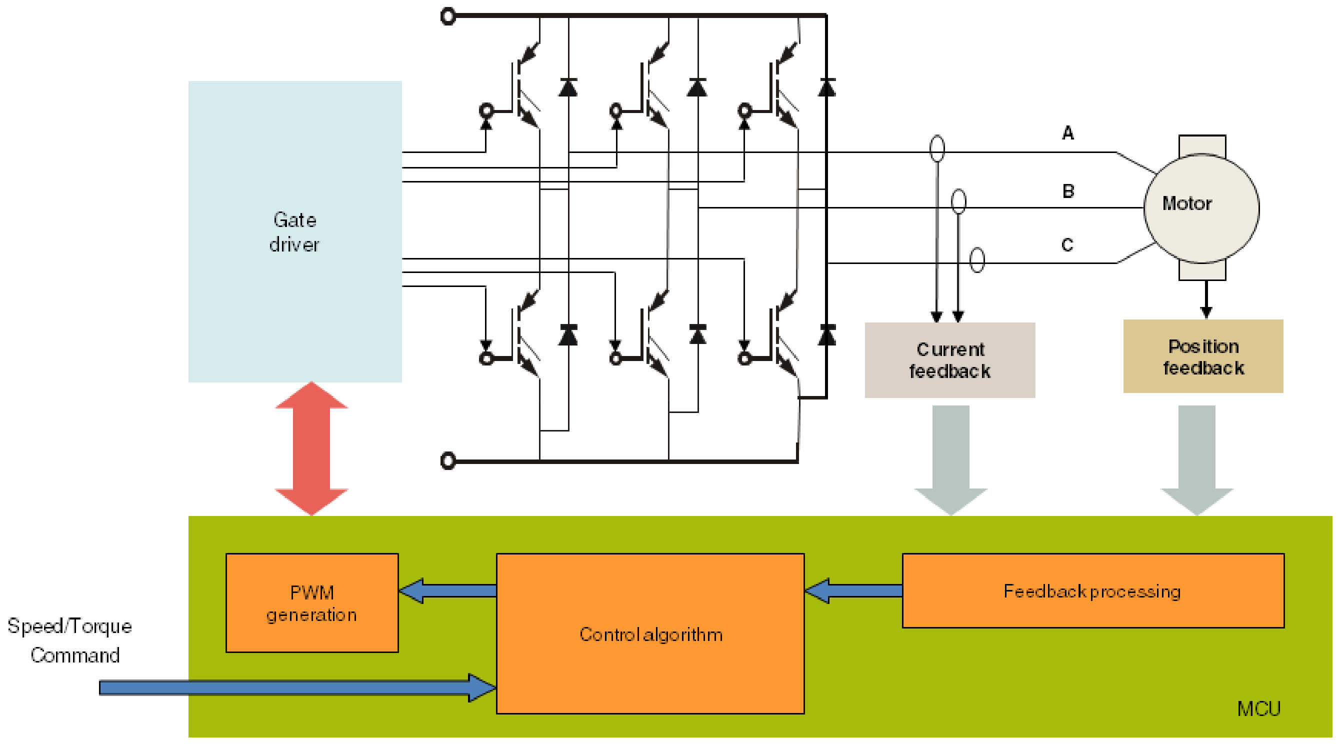

Figure 2. General motor control topology

Figure 2 shows a general 3-phase motor control topology. The topology is based on a 3-phase power stage with drivers that are controlled by the MCU. The MCU inputs are current, voltage, and positional feedback. Those signals are processed by the control software. The PWM generates the required signals for the 3-phase power stage. Motor control application characteristic requirements are elaborated in the next sections.

2.3.1 Motor control process in terms of time execution

The motor control process consists of synchronous and asynchronous tasks. These events are physically determined. The time duration between the events depends on the system time constants.

- Motor electrical time constants (winding) are usually tens of microseconds
- Mechanical (rotor mechanical inertia)

Physically determined asynchronous events must be serviced within one to tens of microseconds.

- Low interrupt latency
- High priority interrupts

2.3.2 Motor control process in terms of algorithms complexity

The motor control processes discuss various complexities of algorithms depending on the kind of control application. For example:

- Simple read -> modify -> write port
    - BLDC motor commutation according to Hall sensors

- Low complexity algorithms such as the PI speed controller
- Medium complexity algorithms such as the back
- EMF observer for sensorless control -Complex algorithms such as sensorless non-linear AC-induction motor control

See Tables 1,2,3 and 4 for details.

2.4 Typical motor control examples

There are many types of motor control algorithms, the focus here is on two typical algorithms:

- BLDC motor control—Represents the commutation type with asynchronous commutation events
- PMSM motor control—Represents the advanced control technique with periodic execution of control tasks independent of motor speed

2.4.1 Low complexity application example — BLDC motor control with Hall sensors

One typical low complexity application is the BLDC motor control with Hall sensors. The BLDC motor uses a rotor with a permanent magnet. The motor rotation is provided by a 6-step commutation of the stator flux vectors. This is provided by the 3-phase voltage system displayed in Figure 3. The commutation period depends on the rotor speed and its duration may be as short as 200 μs. For example a 4-pole, 3-phase BLDC motor running at 10 000 rpm is commutating with a time period of 500 μs. The commutation instant is synchronized using Hall sensors. The voltage amplitude is controlled with the pulse width modulation (PWM) technique.

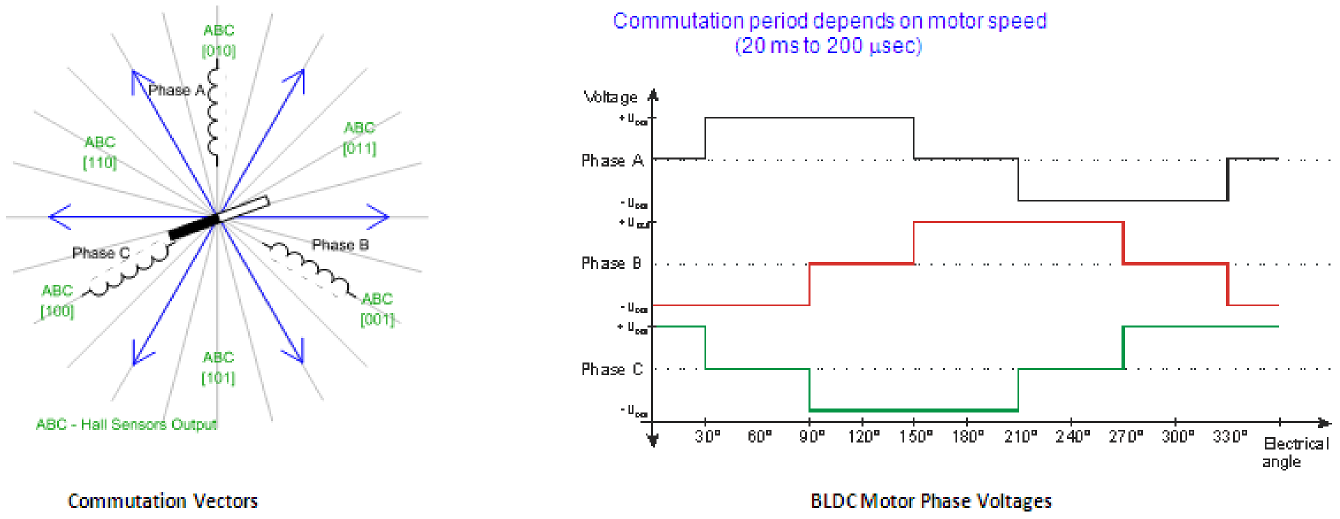

Figure 3. BLDC motor control commutations

The BLDC motor control system is displayed in Figure 4.The Hall sensors are read by a decoder. This is serviced in the commutation control block to provide a six step commutation of the PWM. This block also provides a commutation period for speed control. The required BLDC vector is sent to the PWM block. The motor speed and torque limitation is controlled with a controller. The required PWM duty cycle is sent to the PWM block. The PWM block sets the required phase signals according to the required BLDC vector and the PWM duty cycle. The motor DC bus current needs to be sampled synchronously with the PWM signal, therefore the PWM module is linked to the AD convertor. The sampled DC-bus current is used for the torque limitation.

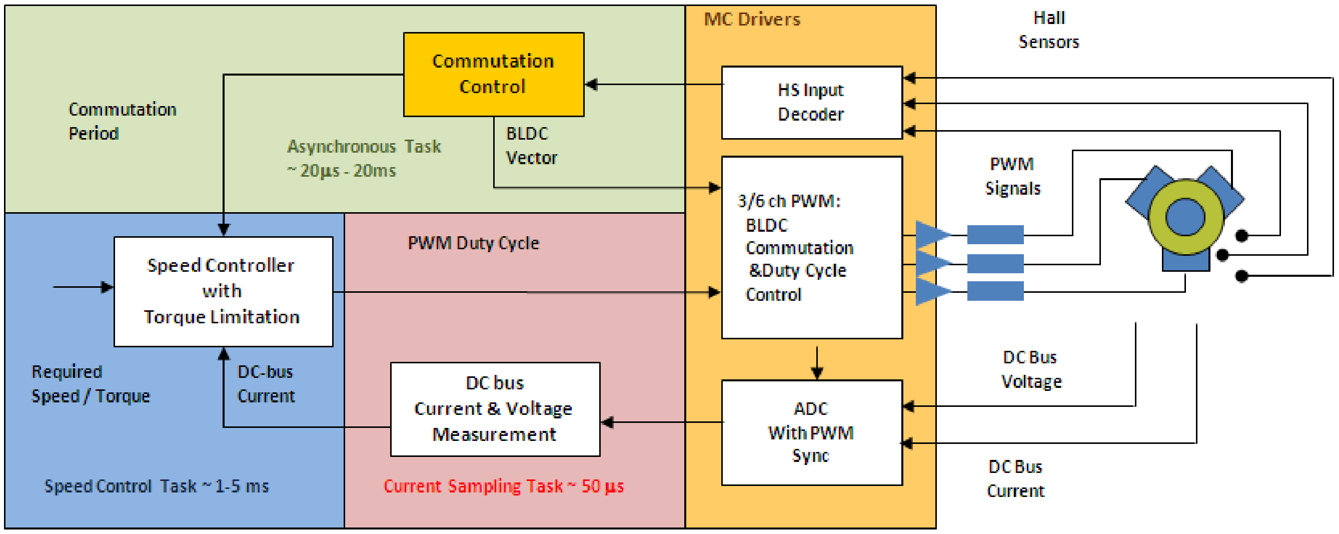
MC Drivers
Hall Sensors
Commutation Control
Commutation Period
Asynchronous Task ~ 20µs - 20ms
BLDC Vector
HS Input Decoder
PWM Signals
3/6 ch PWM: BLDC Commutation &Duty Cycle Control
PWM Duty Cycle
Speed Controller with Torque Limitation
Required Speed / Torque
DC-bus Current
DC bus Current & Voltage Measurement
ADC With PWM Sync
DC Bus Voltage
DC Bus Current
Speed Control Task ~ 1-5 ms
Current Sampling Task ~ 50 µs

Figure 4. BLDC motor control

The Hall sensor BLDC speed control application is based on three tasks:

- ADC—Current (and voltage) sampling—Periodical (a constant period, typically a 50 µs)
- Commutation—Synchronized with a Hall sensor change event (variable duration of 20 ms to 200 µs depending on the motor and speed)
- Speed control —Periodical (a constant period, typically a 1 ms to 5 ms period)

The most critical timing of the elaborated BLDC motor control application is the commutation and current sampling. While the speed control loop runs in the background, the highest execution frequency required is the current sampling. However this task execution duration is usually short (the current controller is usually not used). The commutation at a Hall sensor event has a variable frequency of calls. The duration of its execution (complexity) depends on the control technique and on the hardware support for the PWM module, but it is short due to the low complexity. The commutation task must be resolved within a short time response. Some Freescale devices have a hardware support to minimize the delay and simplify the control software.

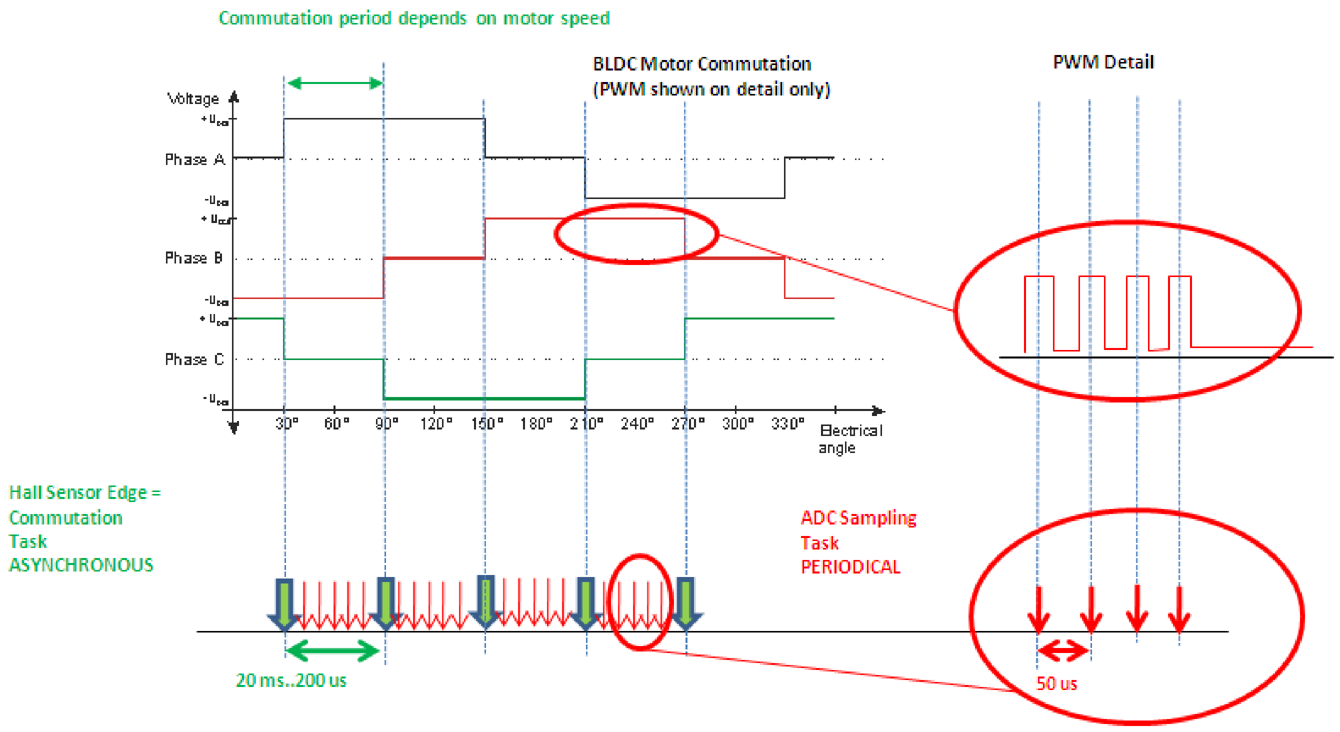

Figure 5. BLDC motor control ADC sampling and commutation timing

The speed controller task is called with a low execution frequency. This call frequency is constant. This constant period of calls is usually 1 ms to 10 ms according to the motor type.

2.4.2 High complexity application example – vector control

A typical high complexity motor control application is a vector control. This is used for sinusoidal motors (AC induction motor or permanent magnet sinusoidal motor). A typical vector control application is speed control with an internal current loop. See Figure 6.

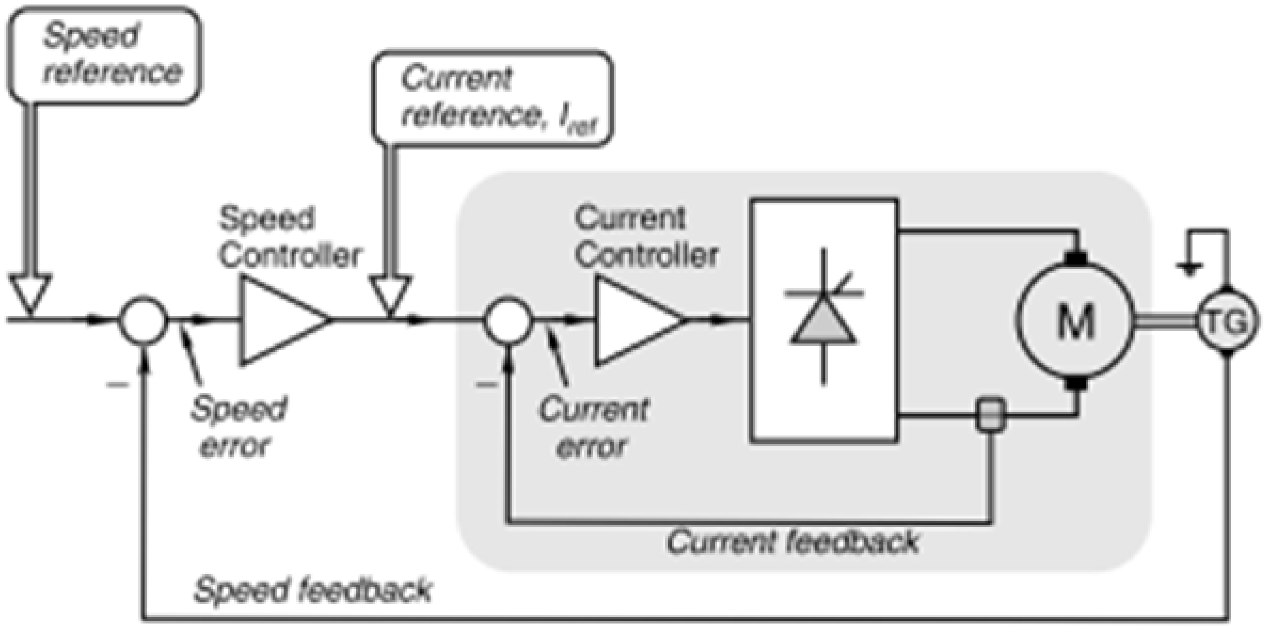

Figure 6. Speed control with an internal current loop

The application structure of a speed vector control application is displayed in detail in Figure 7. The motor is supplied by a sinusoidal PWM voltage. It has two software control loops:

- Slow outer (speed) control loop (1–5 msec)
- Fast inner (current) control loop (25–200 µsec)

Slow outer (speed) control loop and inner fast (current) control loop. The most critical part in terms of execution time is the current control loop, which is usually called with a period of 25 to 100 µs. This software control loop provides transformation from a 3-phase current system into the two-dimensional stator relative coordinate system α β. The inner loop also evaluates the rotor flux position and transforms the currents into the rotor flux relative system with d, q coordinates. The d and q axis currents are controlled by current controllers. After decoupling, it is backwardly transformed into 3-phase PWM signals. Therefore, the inner loop is a medium to high complex algorithm with a short period of calls. The outer loop with speed detection and a controller is much less sophisticated with a longer period of call. It is not time critical.

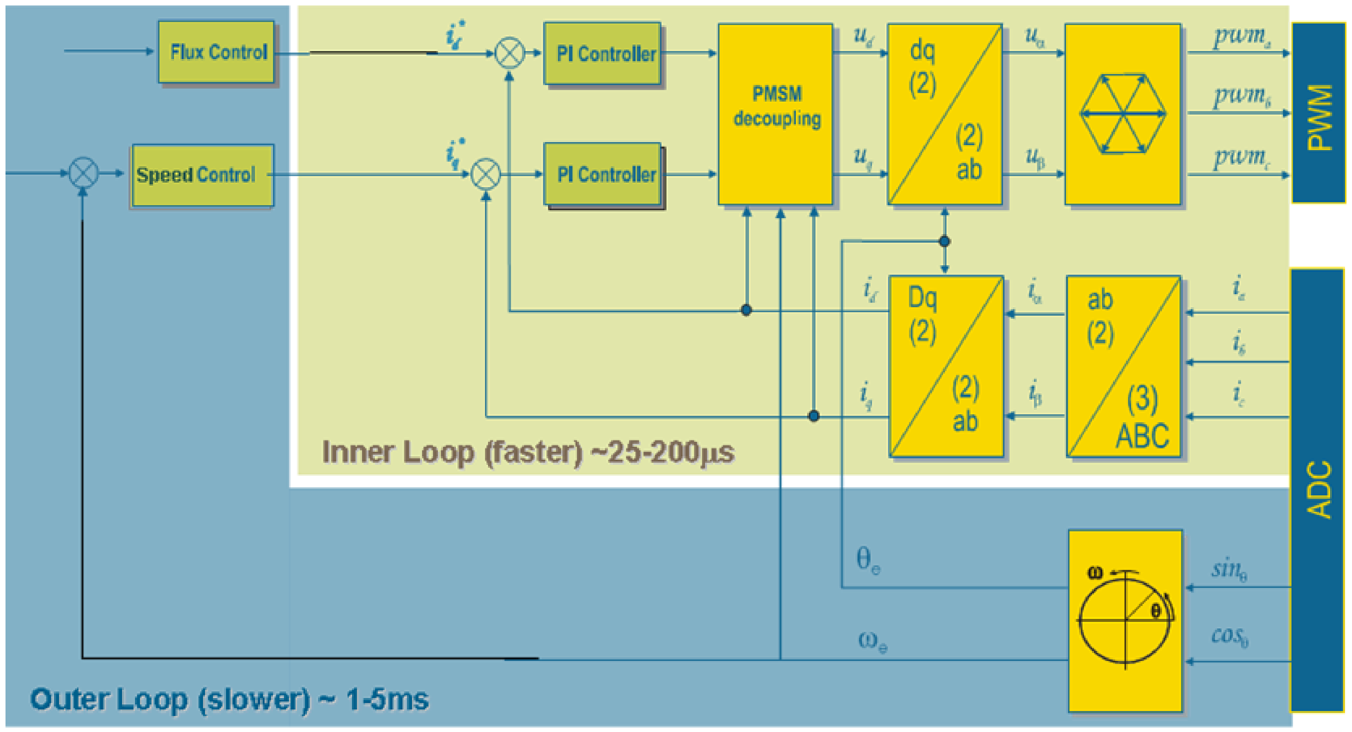

Figure 7. Sophisticated motor control — vector control

The fast loop timing for the 12 kHz PWM is displayed in Figure 8. The required currents can be processed once per PWM cycle, usually in the ADC interrupt subroutine. The main part of the timing is the fast control loop execution. The rest of the CPU time is used for a slow control loop and other tasks.

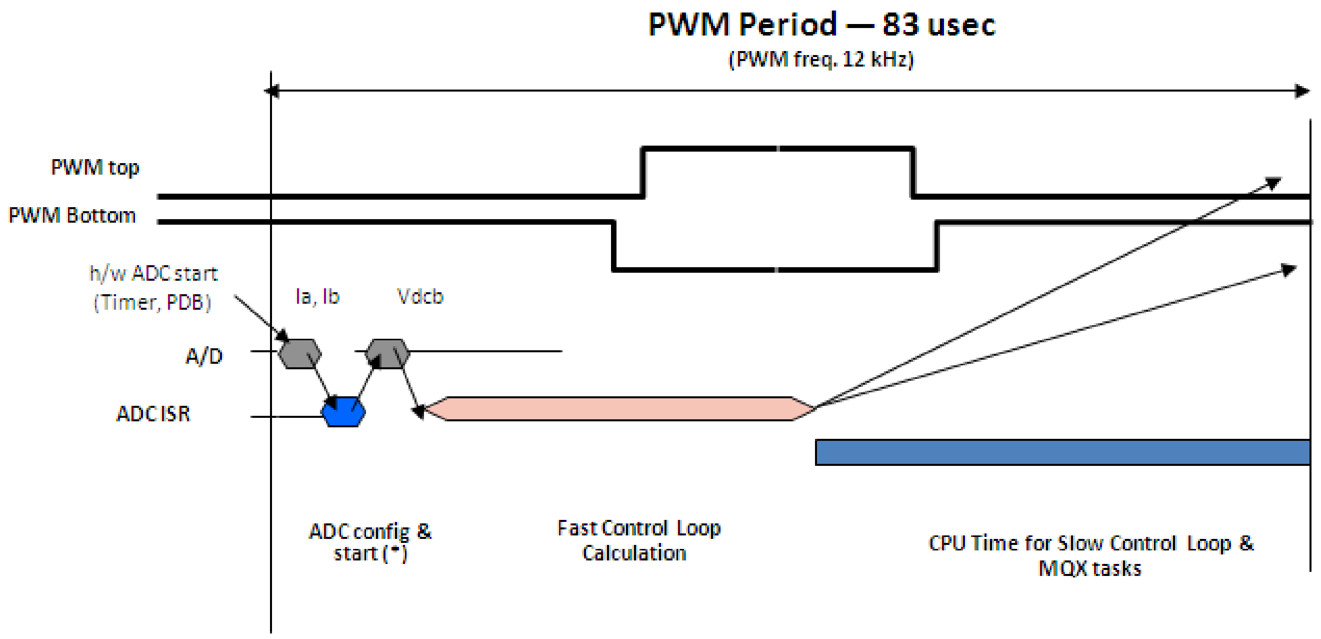
PWM Period — 83 usec
(PWM freq. 12 kHz)
PWM top
PWM Bottom
h/w ADC start (Timer, PDB)
A/D
ADC ISR
Ia, Ib
Vdcb
ADC config & start (*)
Fast Control Loop Calculation
CPU Time for Slow Control Loop & MQX tasks

Figure 8. Vector control – fast loop timing

3 Integration of motor control in MQX

This chapter will elaborate in the integration of a motor control application into MQX.

3.1 Typical system application example with motor control under the MQX

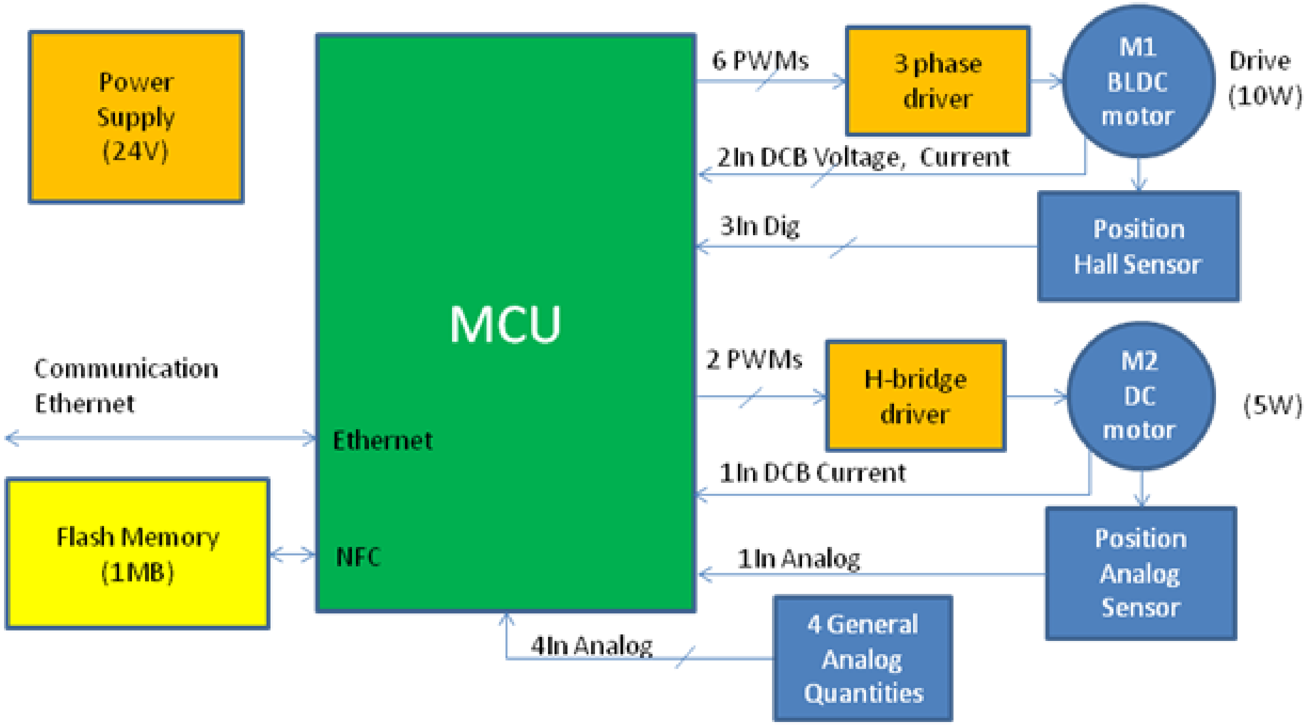

Figure 9. Web controlled application with two motors and analogue measurement

Figure 9 is an example of a motor control application typical for implementation under MQX. Motor control is usually one of the application functionalities (compared with pure motor control applications running on one CPU). The application controls one BLDC motor and one DC motor with dedicated sensors. Other application functionalities are communication using Ethernet and Analog quantities sampling and processing. The Ethernet task arbitration and other functionalities are supported by the MQX system. The entire application runs on a single MCU.

3.2 When to use motor control under MQX

MQX is not a typical operating system for motor control application. The MQX OS is suitable for large applications with advanced features, such as web control, USB, and SDHC card reading running on a dedicated device (usually one core). The primary strength of MQX is that it includes libraries for RTCS, Ethernet, USB communication, MFS file system, and many other applications.

In terms of the time scheduling, advanced motor control applications are naturally based on constant sampling (for example, ACIM and PMSM sinusoidal motor control) or asynchronous events (for example, BLDC motor commutation control) with a fast system response requirement. The required response of the most critical events is usually one to tens of microseconds.

The MQX is a complex system with dynamic allocations and POSIX scheduling. It has a system default tick duration of 5 ms. This is ideal for the majority of applications. However, this also means, that the MQX task time resolution is more than 1000 times longer when compared to the motor control requirements. Therefore, it is evident that the motor control process needs to be serviced with interrupts of a high priority. This can be provided with standard MQX interrupt routines. The time duration from the interrupt request to the service routines execution is usually units of a microsecond (depending on the CPU version and clock speed). And, if necessary, the motor control algorithms can be implemented using the kernel interrupts. The kernel interrupts are natural CPU interrupts with no MQX overhead and minimal execution duration. The disadvantage of the kernel interrupt is that no MQX functionalities such as events, or semaphores are supported.

3.3 Motor control under MQX – two approaches

There are two approaches to writing motor control software under MQX:

- **Dedicated motor control driver**
  The motor control process is provided by one or more kernel interrupts (MQX functionalities like events are not supported) or MQX highest priority interrupt tasks. The motor control process (task) software is then similar to a standard non-operating system approach. However, the software can possibly (though not necessarily) use some MQX device peripheral drivers (included in the MQX installation).
  In this approach, the MQX system is then used for:
  - Initialization of all tasks including motor control
  - No motor control related tasks

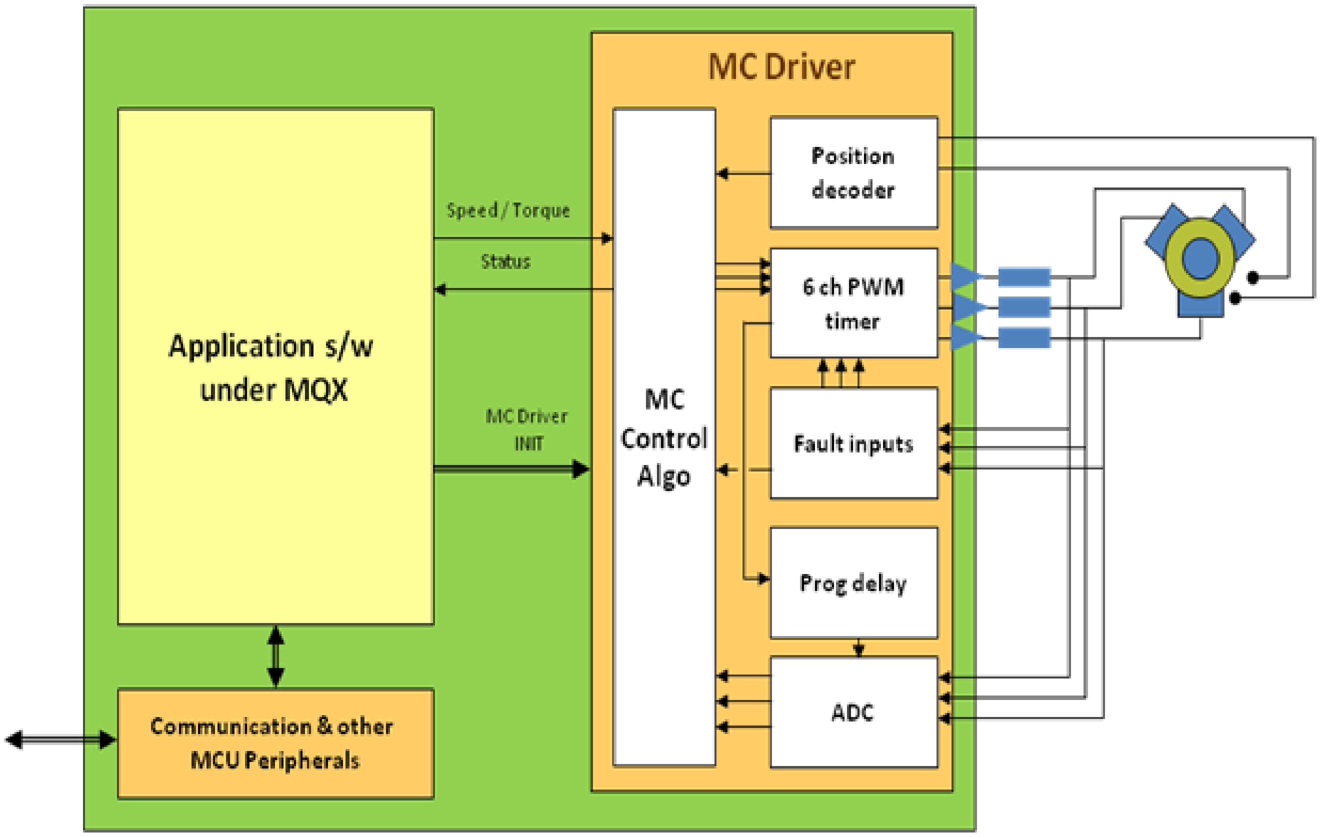

Figure 10. Dedicated motor control driver

- **True MQX approach**

The motor control application is provided by (usually more than one) MQX tasks and interrupts. The software usually uses some MQX or customer written device peripheral drivers. Interrupts service time critical events (such as commutation, PWM update, position sensor service), and MQX tasks service non-critical tasks where the MQX tick duration delay is not critical (possibly the speed control loop).

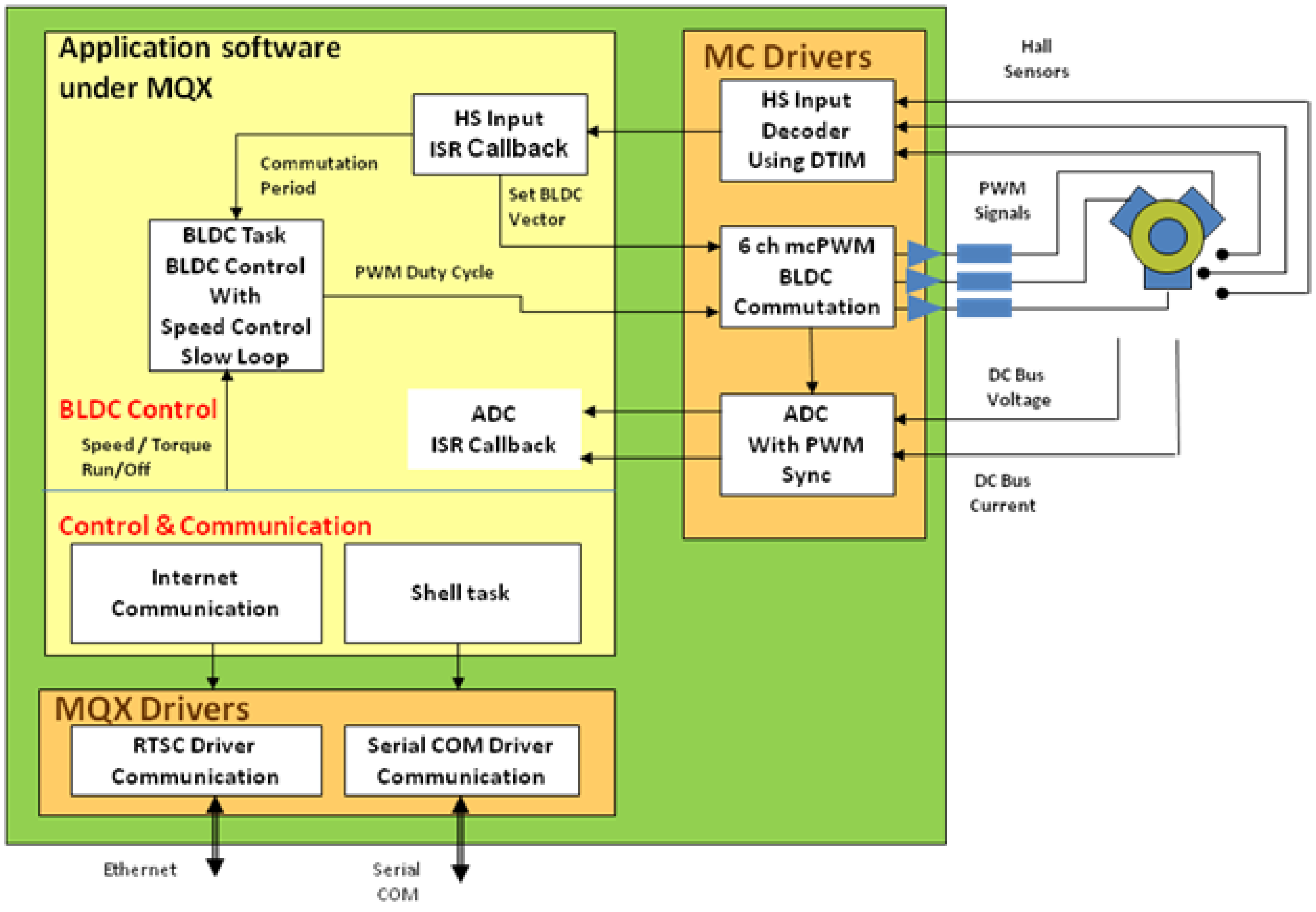

Figure 11. True MQX approach

3.4 Dedicated motor control driver example

An example of an application in MQX with a dedicated motor control driver and other application functionalities is shown in Figure 12. This application example is the PMSM vector control.

The application features are:
- PMSM vector control
- Ethernet communication
- USB interface
- Other MQX tasks

For the functional diagram, refer to Figure 7. The Ethernet based CGI interface, USB control task are running under MQX. The application control shell task is provided by MQX Task 1. The motor control upper software layer is realized by the MQX Task 2. The motor control process driver is provided as a set of ADC and timer peripheral callback functions. The interrupt callbacks may be initialized as kernel interrupts, that are independent of the MQX system. However, a standard MQX interrupt is possible for applications where the additional delay of MQX interrupt processing (typically 2 µs depending on the CPU speed) is not critical.

The benefit of a standard MQX interrupt is that the MQX events and flags can be used as an API for other MQX processes. The MQX system manages all the MCU registers and interrupt flag backups.

Dedicated motor control driver advantages:

- Full control of the motor control events timing — no delay caused by MQX arbitration
- Time critical motor control tasks can only be serviced using MQX or kernel interrupt subroutines

Disadvantages

- Software development is more complex (MQX arbitration is not used)
- Peripherals are fully assigned for the driver

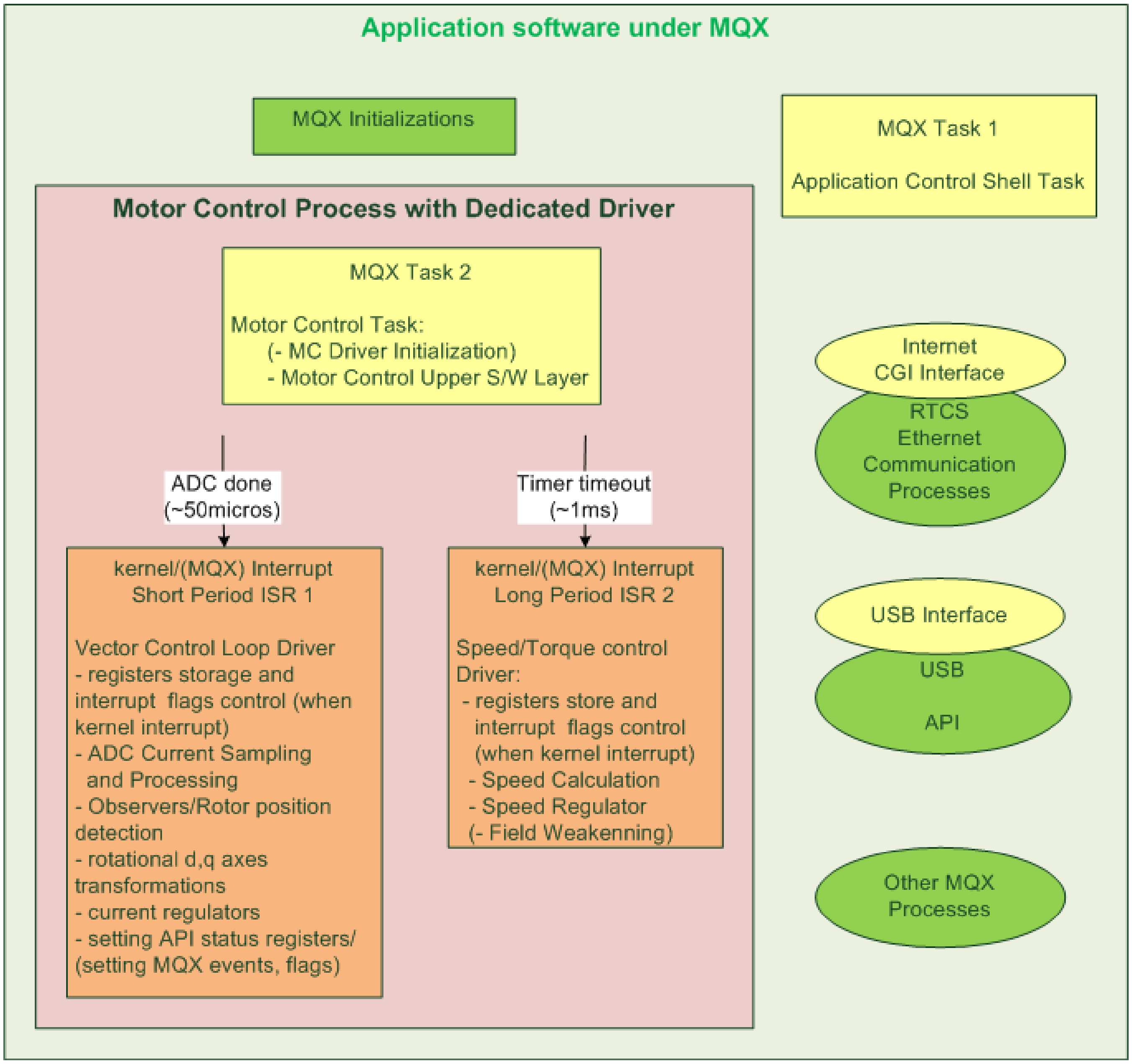
Application software under MQX
MQX Initializations
MQX Task 1
Application Control Shell Task
Motor Control Process with Dedicated Driver
MQX Task 2
Motor Control Task:
(- MC Driver Initialization)
- Motor Control Upper S/W Layer
ADC done (~50micros)
Timer timeout (~1ms)
kernel/(MQX) Interrupt Short Period ISR 1
Vector Control Loop Driver
- registers storage and interrupt flags control (when kernel interrupt)
- ADC Current Sampling and Processing
- Observers/Rotor position detection
- rotational d,q axes transformations
- current regulators
- setting API status registers/ (setting MQX events, flags)
kernel/(MQX) Interrupt Long Period ISR 2
Speed/Torque control Driver:
- registers store and interrupt flags control (when kernel interrupt)
- Speed Calculation
- Speed Regulator
(- Field Weakenning)
Internet CGI Interface
RTCS Ethernet Communication Processes
USB Interface
USB
API
Other MQX Processes

Figure 12. Motor control process under MQX with a dedicated driver

3.5 True MQX approach example

An example of an application under MQX with motor control and a true MQX approach is in Figure 13. This application example is BLDC motor control with Hall sensors.

The application features are:

- BLDC motor control with Hall sensors
- Ethernet communication
- USB interface
- Other MQX tasks

For a functional diagram, see fig 4 . The Ethernet based CGI interface, USB control task and other tasks are running under MQX. The application control shell task is provided by MQX Task 1.

The motor control process is provided using an MQX approach, using MQX Tasks 2, and 3, with the ADC and Hall sensor interrupt callback functions. The interrupt callbacks are initialized in MQX Task 2 as MQX system interrupts. The benefit of a standard MQX interrupt is that the MQX events and flags can be used as an API for other MQX tasks. The MQX system handles all MCU register and interrupt flags.

Dedicated motor control driver advantages:

- Software development is simpler (MQX task arbitration, event, and semaphores are used)

Disadvantages

- Time critical motor control tasks can not be serviced this way due to the MQX limitation (5 ms tick). Dedicated interrupts must be used
- Time delay of the motor control functions under MQX

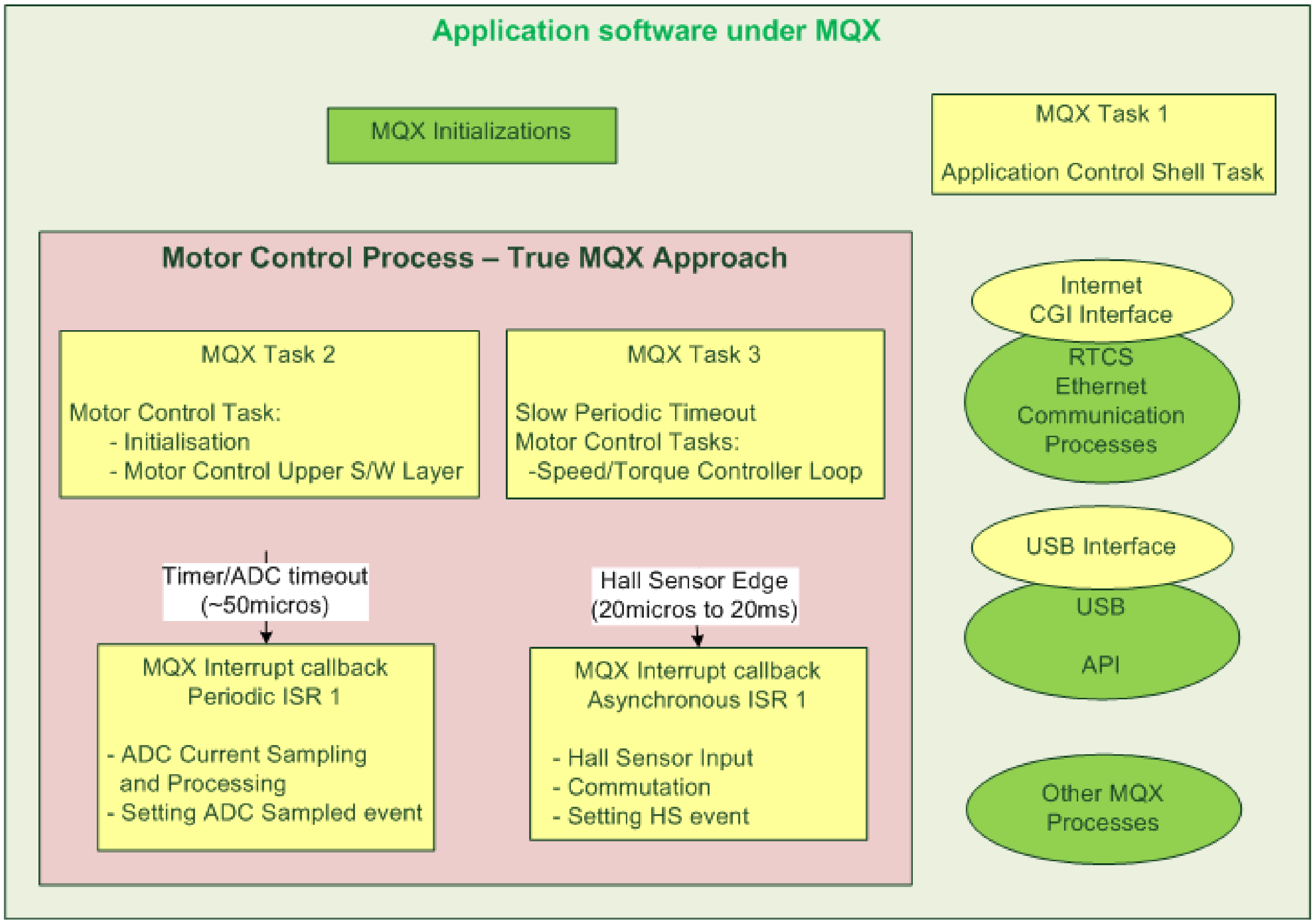

Figure 13. Motor control process under MQX – true MQX approach

3.6 Motor control applications and implementation under MQX

The actual implementation of motor control under MQX depends on the motor type, control algorithm, and application requirements. The following tables show the required tasks and MQX realization of the motor control process for a wide spread of motor versions. The required tasks of the dedicated motor control process depend on the control technique. Each motor control technique consists of periodical and asynchronous tasks. The tables show the period of the task calls and typical task complexity. The task complexity depends on the exact software implementation, so the value in a table is an example of a time duration on a 50 MHz clocked by 32-bit MCU. The function call is elaborated in Sections 3.3, 3.4, 3.5. The last columns show the recommended implementation under MQX and the MQX approach.

Table 1. DC Motor

Control Technique	Major Tasks Using the Control Technique	Task Period and Synchronization Event Example	Task Complexity Example (Duration at 50 MHz 32bit)	Task Recommended Implementation under MQX	Task Call under MQX	MC Application MQX Approach
Position Control	Periodic Task 1: — Position controller	Periodic 1ms	Low (approx. 5 µs)	MQX Task (or Interrupt MQX Callback) Controller algorithms with H-bridge Driver	MQX Task (/ MQX interrupt timer callback)	True MQX
	Periodic Task 2:	Periodic 50 µs	Low (approx. 3 µs)	Interrupt called driver for ADC sampling	MQX Interrupt callback at ADC conversion done	

Table 2. BLDC Motor

Control Technique	Major Tasks Using the Control Technique	Task Period and Synchronization Event Example	Task Complexity Example (Duration at 50MHz 32bit)	Task Recommended Implementation under MQX	Task Call under MQX	MC Application MQX Approach
BLDC Commutation with Hall Sensors	Periodic Task1: — DC-bus current and voltage sampling and processing	Periodic 50 µs synchronized with PWM => ADC conversion done	Low (approx. 2 µs)	Interrupt called driver for ADC sampling with PWM synchronisation and algorithms for further processing	MQX Interrupt callback at ADC conversion done	True MQX
	Periodic Task 2: — Speed/ torque controller	Periodic 1 ms	Low (approx. 5 µs)	MQX Task (or Interrupt MQX Callback) Controller algorithms with PWM duty cycle driver	MQX Task (time delay)	
	Asynchronous Task 3: — Hall sensor state read and commutation driver	Asynchronous 20 ms to 200 µs at Hall sensor change event	Low Read/Write PORT Registers	Interrupt called driver for timer (—Hall sensor) input and PWM commutation driver	MQX Interrupt callback at timer edge capture	

Table continues on the next page...

Table 2. BLDC Motor (continued)

Control Technique	Major Tasks Using the Control Technique	Task Period and Synchronization Event Example	Task Complexity Example (Duration at 50MHz 32bit)	Task Recommended Implementation under MQX	Task Call under MQX	MC Application MQX Approach
Sensorless BLDC Commutation	Periodic Task1: — DC-bus current and voltage sampling and processing	Periodic 50 µs synchronized with PWM => ADC conversion done	Low (approx.2 µs)	Interrupt called driver for ADC sampling with PWM synchronisation and algorithms for further processing	MQX Interrupt callback at ADC conversion done	Sensorless BLDC control Driver and (sub) task drivers (ADC sampling)
	Periodic Task 2: — Speed/ torque controller	Periodic 1 ms	Low (approx. 5 µs)	PI Controller algorithms with PWM duty cycle driver	MQX Interrupt callback at timer timeout	
	Asynchronous Task 3: — Back-EMF zero crossing — Commutation timing	Asynchronous — 20 ms to 200 µs at Back-EMF zero crossing event	Low to medium (25 µs)	Interrupt called Sensorless BLDC driver Part1: Back-EMF zero-crossing and commutation calculation and timer setting	MQX Interrupt callback Comparator -> timer input capture	
	Asynchronous Task 4: -BLDC motor commutation	Asynchronous 20 ms to 200 µs	Low (10 µs)	Interrupt Called Sensorless BLDC Driver Part2: PWM commutation driver	MQX Interrupt callback at timer output compare	

Table 3. PMSM Control

Control Technique	Major Tasks Using the Control Technique	Task Period and Synchronization Event Example	Task Complexity Example (Duration at 50MHz 32bit)	Task Recommended Implementation under MQX	Task Call under MQX	MC Application MQX Approach
Sinusoidal PMSM Control with Sensor	Periodic Task1: — DC-bus current and voltage sampling — 3phase sine generation	Periodic 50 µs synchronized with PWM => ADC conversion done	Low (10 µs)	Interrupt called driver for ADC sampling with PWM synchronisation —driver for 3phase sine generation	MQX Interrupt callback at ADC conversion done	True MQX with (sub)task drivers (ADC sampling, 3-phase sine generation)
	Asynchronous Task 2: — Rotor position recognition. — 3phase sine synchronisation	Asynchronous min 100 µs at sensor edge	Low (approx. 5 µs)	Interrupt called driver for timer edge input capture and algorithm for 3phase sine synchronisation	MQX Interrupt callback at timer edge input capture	
	Periodic Task 3: Speed/Torque Controller	Periodic 1m	Low (approx. 5 µs)	PID controller algorithm	Timer timeout MQX Interrupt callback	
PMSM Vector Control with Sensor (encoder, sin-cos or other)	Periodic Task1: — Current sampling — Position recognition — Transformations a,b,c->α, β->d,q, — Current controllers	Periodic 50 µs synchronized with PWM => ADC conversion done	Medium (20 µs)	Interrupt called PMSM Vector Control Driver	Interrupt callback at ADC conversion done	PMSM Vector Control driver and (sub) task drivers
	Periodic Task 2: Speed/Torque Controller	Periodic 1ms	Low (approx. µs)	PI controller algorithm	MQX Interrupt callback at timer timeout	

Table continues on the next page...

Table 3. PMSM Control (continued)

Control Technique	Major Tasks Using the Control Technique	Task Period and Synchronization Event Example	Task Complexity Example (Duration at 50MHz 32bit)	Task Recommended Implementation under MQX	Task Call under MQX	MC Application MQX Approach
Sensorless PMSM Vector Control	Periodic Task1: — Current sampling — Transformation a,b,c->α, β->d,q — Current controllers, — Observer calculation	Periodic 50 µs synchronized with PWM => ADC conversion done	Medium to High (40 µs)	Interrupt called Sensorless PMSM Vector Control Driver	Interrupt callback at PWM sync or ADC conversion done	Sensorless PMSM Vector Control driver and (sub) task drivers
	Periodic Task 2: — Speed/ Torque controller	Periodic 1m	Low (approx. 5 µs)	PID controller algorithm	MQX Interrupt callback at timer timeout	

Table 4. AC Induction motor control

Control Technique	Major Tasks Realizing the Control Technique	Task Period and Synchronisation Event Example	Task Complexity Example (Duration @50MHz 32bit)	Task Recommended Implementation under MQX	Task Call under MQX	MC Application MQX Approach
AC Induction Motor Vector Control with Sensor (encoder, sin-cos or other	Periodic Task1: — Current Sampling — Position Recognition — Transformations a,b,c->α, β->d,q, Flux estimator, — Current controllers	Periodic 50 µs synchronized with PWM => ADC conversion done	Medium (35 µs)	Interrupt called ACIM Vector Control Driver	Interrupt callback at ADC conversion done	ACIM Vector Control driver and (sub) task drivers
	Periodic Task 2: —Speed/ Torque controller	Periodic 1ms	Low (approx. 5 µs)	PID controller algorithm	MQX Interrupt callback at timer timeout	

4 Demonstrating a BLDC Motor Control Application under MQX

To demonstrate motor control under MQX a BLDC motor control application was designed.

4.1 BLDC motor control under MQX – application characteristics

The BLDC uses Freescale TOWER modular hardware with a 3-phase power stage board and an MCF5441x controller board. The application controls a 24 V BLDC motor with Hall sensors via the Ethernet. It is written under MQX . The Ethernet communication uses the MQX library. The BLDC motor control is provided using a true MQX approach. See Figure 14.

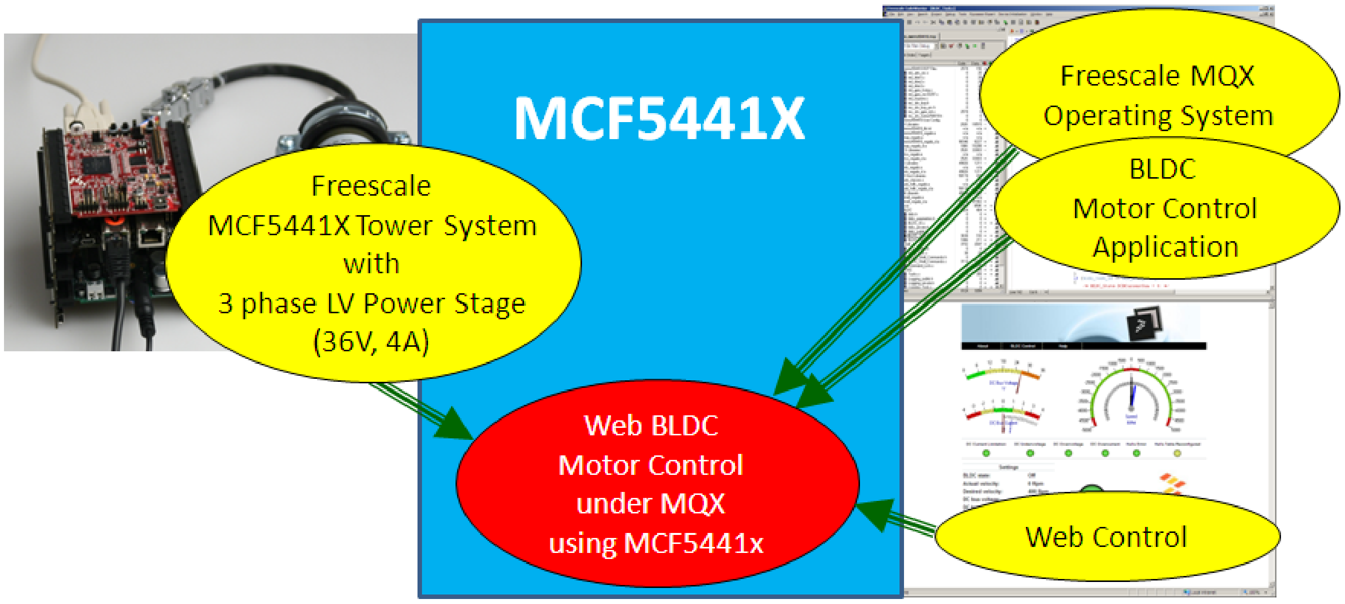
MCF5441X
Freescale MCF5441X Tower System with 3 phase LV Power Stage (36V, 4A)
Freescale MQX Operating System
BLDC Motor Control Application
Web BLDC Motor Control under MQX using MCF5441x
Web Control

Figure 14. Demonstrating an application of BLDC motor control under MQX

5 BLDC Motor Control under MQX – Code Examples

These are some examples of the code from the application described in Section4.1 BLDC motor control under MQX – application characteristics

5.1 Code examples

The application uses a true MQX approach with user defined interrupt functions and MQX interrupt callback, and MQX tasks. See Figure 14 for interrupts and MQX_Task reference.

A simplified structure of these examples is displayed in . The figure does not show the detailed application description, it shows some of the function calls and data structures that are evaluated in the code listing below. The data structures `BLDC_State`. `Event` is a variable for an MQX event. The `hsinput_dtim_info_ptr*` is a structure pointed to by the `hsinput_dtim_info_ptr` and includes Hall sensor input data that is used by the `_hs_input_dtim_a_isr` interrupt function and the HS_input_callback (see the code below). The `MQX_template_list` is an MQX structure that defines the MQX tasks (priority, type, and so on). Details are described in the MQX documentation see the Section 6. Bibliography number one and two.

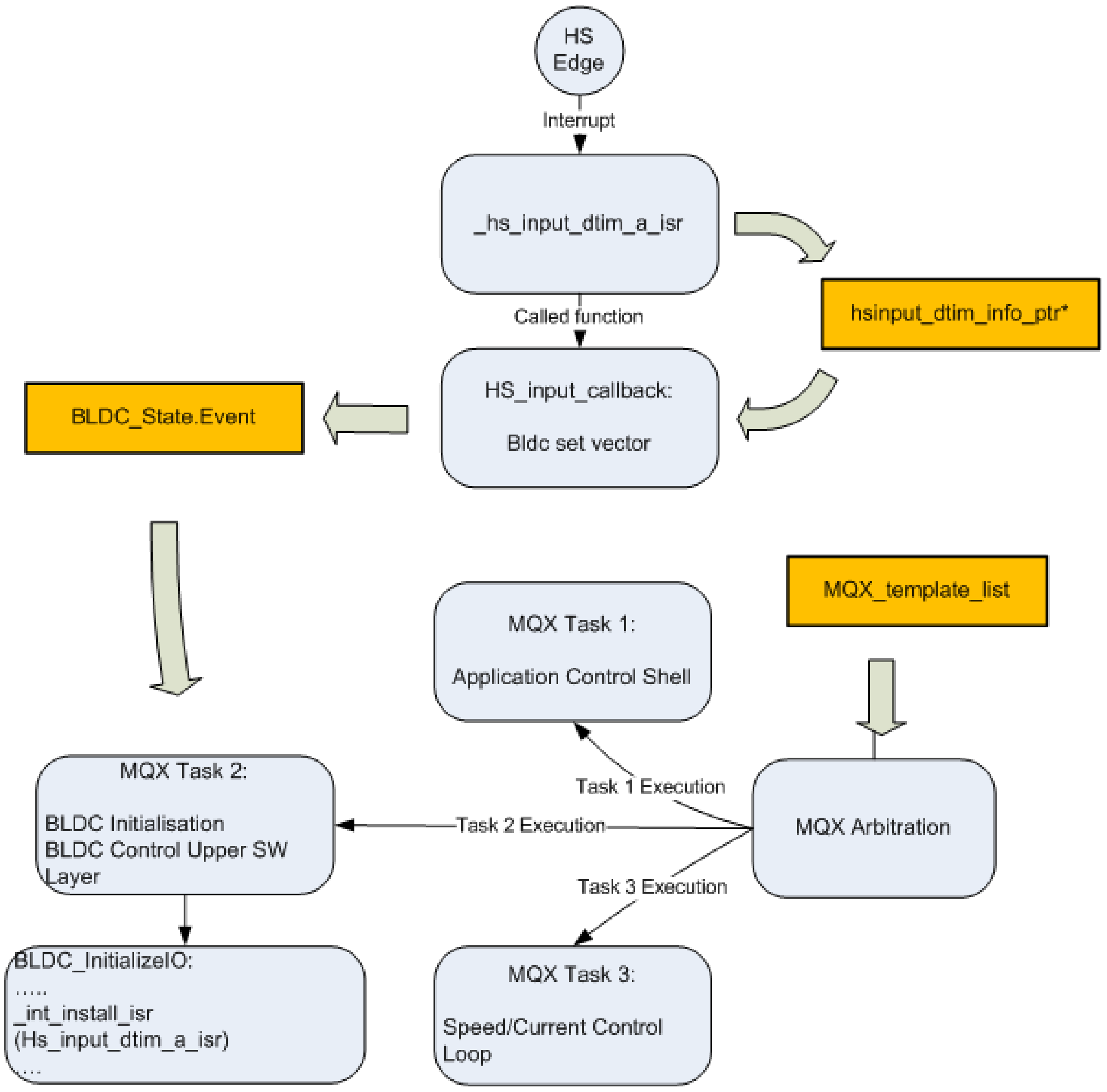

Figure 15. BLDC motor control under MQX – examples of function calls and data structures

The MQX interrupt callback installation is provided initialization of the application using the MQX library function:

`_int_install_isr.`

```
......
      _int_install_isr(dtim_init_ptr_a->VECTOR,
                       (void (_CODE_PTR_)(pointer))_hsinput_dtim_a_isr,
                       (pointer) hsinput_dtim_info_ptr);
    …
```

The Hall sensor inputs are executed by the dtim modules of the MCF5441x processor. The dtim driver for the Hall sensor input is serviced by the _hsinput_dtim_a_isr callback function. This function calculates the commutation period `hsinput_dtim_info_ptr->LAST_EDGE_PERIOD`, reads the Hall sensor input `hsinput_dtim_info_ptr->HS_STATE`, clears the peripheral pending flags, and finally calls the HSInputCallbackIsr algorithm: `uint_32 _hsinput_dtim_a_isr.`

```
(
      /* [IN] the address of the device specific information */
        MCF54XX_HSINPUT_DTIM_INFO_STRUCT _PTR_  hsinput_dtim_info_ptr,
    )
{
    /* calculation of the period between Hall Sensor Edges */
    hsinput_dtim_info_ptr->PREV_EDGE_TIME = hsinput_dtim_info_ptr->LAST_EDGE_TIME;
    hsinput_dtim_info_ptr->LAST_EDGE_TIME = hsinput_dtim_ptr->DTCR;
    hsinput_dtim_info_ptr->LAST_EDGE_PERIOD = hsinput_dtim_info_ptr->LAST_EDGE_TIME –
hsinput_dtim_info_ptr->PREV_EDGE_TIME;

     /* read & mask GPIO */
    hsinput_dtim_info_ptr->HS_STATE = (*(hsinput_dtim_info_ptr->GPIO_STR.GPIO_PPDSDR_PTR) & \
                                        hsinput_dtim_info_ptr->GPIO_STR.PIN_MASK)>> \
                                        hsinput_dtim_info_ptr->GPIO_STR.PIN_SHFT;
    /* clear pending flags */
    periphBitSet(MCF54XX_DTIM_DTER_CAP, &(hsinput_dtim_info_ptr->DTIM_A_PTR->DTER));

     /* call HS_input_callback  */
    if(hsinput_dtim_info_ptr->HSINP_CALLBACK_ISR != NULL)
    {
        hsinput_dtim_info_ptr->HSINP_CALLBACK_ISR (hsinput_dtim_info_ptr);
    }
    return( MQX_OK );
}
```

Interrupt callback algorithm HSInputCallbackIs is then called to provide PWM commutation according to the Hall sensor state. It also processes the commutation period and sets the MQX event `BLDC_HS_INPUT_CHANGED`:

```
void HS_input_callback (pointer parameter)
{
    MCF54XX_HSINPUT_DTIM_INFO_STRUCT_PTR  hsinput_dtim_info_ptr = parameter;
    /* set required sector_u8 variable according to table and Hall sensor input */
    sector_u8 = bldc_3pps_hs_table[hsinput_dtim_info_ptr->HS_STATE];
    /* commutate the pwm sector according to , sector_u8 variable */
    _3ppspwm_bldc_set_vector_sector_6s_ss_compl(pspwm_info_ptr, sector_u8);
    /* read Hall sensor period */
    BLDC_State.ActualPeriodSample = _hsinput_read_period(hsinput_info_ptr);
    /* set BLDC_HS_INPUT_CHANGED  event */
    _lwevent_set(&BLDC_State.Event, BLDC_HS_INPUT_CHANGED);
}
```

Task2 is an example of an MQX Task function, which in this case provides networking initialization, BLDC motor control application initialization, and the motor control upper software layer (see Figure 13, Task 2):

```
void Task2 (uint_32 data)
{
    /* Initialize ethernet networking */
    initialize_networking();
    /* Initialize operating parameters to default values */
    BLDC_InitializeParameters();
   /* create light weight event structure */
    _lwevent_create(&BLDC_State.Event, 0);
    /* Configure and reset outputs */
    BLDC_InitializeIO();
    while(TRUE)
    {
        /* Motor Control Upper S/W Layer */
        …
    }
```

Arbitration of the Task1 function is provided by the MQX . The tasks list is defined in the standard way in the MQX:

```
const TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
    /* Task Index,                       Function,         Stack,  Priority,  Name,
Attributes,                        Param,  Time Slice */
    {      TASK1,                          MQX_Task2      2000,         3,    "BLDC",
MQX_AUTO_START_TASK,    0,       0            },
    { BLDC_CONTROL_TASK, MQX_Task3,  1500,          4,      "BLDCCtrl",
0,                             0,      0           },
     { SHELL_TASK,                MQX_Task1,     2900,         12,    "Shell",
MQX_AUTO_START_TASK,      0,      0            },

{ ..........,                                                  ..,              .,         ..,
      ......,                                                 ....,         .,       ..
},
    {0}
};
```

6 Bibliography

1. Reference manual titled *Freescale FSLMQX™ RTOS* (document FSLMQXRM)
2. User guide titled *Freescale FSLMQXTM Real-Time Operating System* (document FSLMQXUG)
3. Design reference manual titled *3-Phase Sensorless BLDC Motor Control Using MC9S08MP16* (document DRM117) Freescale Semiconductor 2009
4. Design reference manual titled *Sensorless PMSM Vector Control with a Sliding Mode Observer for Compressors Using MC56F8013* (document DRM099) Freescale Semiconductor 2008
5. *3-Phase PM Synchronous Motor Vector Control using DSP56F80x*, by Prokop L., Grasblum P., AN1931, Motorola, 2002

7 Definitions and Acronyms

AN — Application Note

ACIM — Alternating Current Induction Motor

API — Application Interface

BLDC — Brush-less DC Motor

FOC — Field Oriented Control

Freescale MQX™ — Real-Time Operating System MQX adopted by Freescale Semiconductor

Motor control In this article, means a process which controls an electrical motor such as BLDC PMSM, AC-induction or other

MQX™ — Real-Time Operating System

Tick — Operating system time unit (also reflects the minimal time resolution)

OS — Operating System

POSIX — Portable Operating System Interface, produced by IEEE and standardized by ANSI and ISO. MQX conforms to POSIX.4 (real-time extensions), and POSIX.4a (threads extensions).

PMSM — Permanent Magnet Synchronous Motor

RTOS — Real Time Operating System

RTCS — Embedded Internet stack provides IP networking for the MQX platform. RTCS is provided with a rich assortment of TCP/IP networking application protocols and uses the MQX RTOS drivers for Ethernet or serial connectivity.

How to Reach Us:

Home Page:
www.freescale.com

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

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

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

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

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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4254
Rev. 0, 5/2011