Motor Control Library Help

Table of Contents

1 Motor Control Library

Files

Name	Description
motor_control.h	This header file lists all the interfaces used by the Motor Control library.

Description

1.1 Introduction

Motor Control Library

for

Microchip Microcontrollers

This library is a collection of optimized functions commonly used in Motor Control applications.

Description

The Motor Control library contains function blocks that are optimized for the dsPIC33F and dsPIC33E family of Digital Signal Controllers (DSC). All functions in this Motor Control library have input(s) and output(s), but do not access any of the DSC peripherals. The library functions are designed to be used within an application framework for realizing an efficient and flexible way of implementing a Motor Control application.

The block diagram in Figure-1 shows a typical usage scenario. The user-developed Motor Control application interfaces to the DSC peripherals while using function calls into this library to perform majority of the time-critical operations.

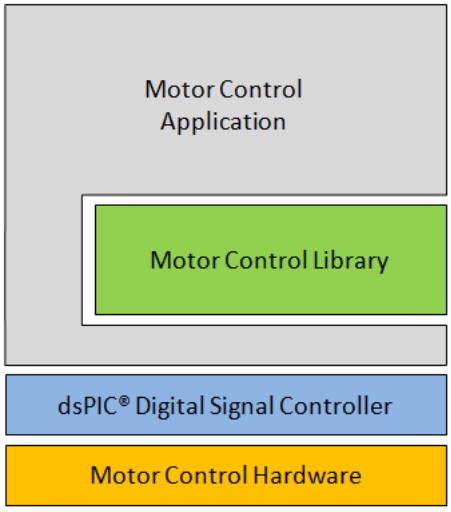


Figure-1: Block diagram of a typical library usage scenario.

1.2 Release Notes

Motor Control Library Version : 0.10 Release Date: December 11th, 2013

This is the first release of the library. The interface can change in the beta and\or 1.0 release.

New:

None.

Changes:

None.

Fixes:

None.

Known Issues:

None.

Development Tools:

This version of the library is tested to be compatible with the following:

- XC16 v1.11 compiler (only)
- MPLAB X IDE v1.90 and later

Performance and functional correctness of the library cannot be guaranteed if this version of the library is used with versions of the development tools other than those listed above.

Other Notes:

The "inline" keyword will be recognized by the compiler only if compiler optimizations are enabled. While using compiler optimization level -O1 and -Os, check the "Do not override 'inline'" option to inline the library function calls. Keyword "inline" is only a high-level suggestion to the compiler and certain usage cases or explicit options can disable inlining. Refer to the compiler documentation for more information on this topic.

1.3 SW License Agreement

1.4 Library Overview

This topic describes the basic architecture of the Motor Control Library and provides information and examples on how to use it.

The Motor Control Library hosts functions in two implementation variants:

1. C - functions are declared with static and inline keywords.
2. Assembly - functions are defined in a C-callable archive file with function interfaces defined in the motor_control.h file.

Library users may choose to use one or both or a mixture of these two implementation variants. Unused implementation variants of the library will not consume data or program memory on the target device.

Since the C implementation of library functions are declared "inline", the compiler will *attempt* to integrate the library function's code into the code for its callers. This usually makes execution faster by eliminating the function-call overhead. Refer to the compiler documentation for more information on function inlining.

Interface Header File: motor_control.h

The interfaces to the Motor Control library are defined in the "motor_control.h" header file. Any C language source (.c) file that uses the Motor Control library should include the "motor_control.h".

Library Files: libmotor_control_dspic33e-elf.a and libmotor_control_dspic33f-elf.a

The Motor Control library archive (.a) files installed with the library release. The prototypes for library functions hosted by the archive files are described in the motor_control.h file. Both of these archive files released with the library are built using the ELF-type of Object Module Format (OMF).

Inline C Definitions Header File: motor_control_inline_dspic.h

This header file hosts the C language definitions of the library functions. The "motor_control_inline_dspic.h" is automatically included when a C language source (.c) file includes the "motor_control.h" file.

Mapping Header File: motor_control_mapping.h

This header file defines a short function name for each of the library functions and maps it, by default, to the prototype of one of the implementation variants. The "motor_control_mapping.h" is automatically included when a C language source (.c) file includes the "motor_control.h" file.

1.4.1 Library Sections

The library interface routines are divided into three sub-sections. Each sub-section addresses one of the classes of operation in the Motor Control library.

Library Interface Section	Description
Calculate	Functions performing mathematical operations on a set of numerical inputs.
Transform	Functions for transforming inputs from one reference frame to another.

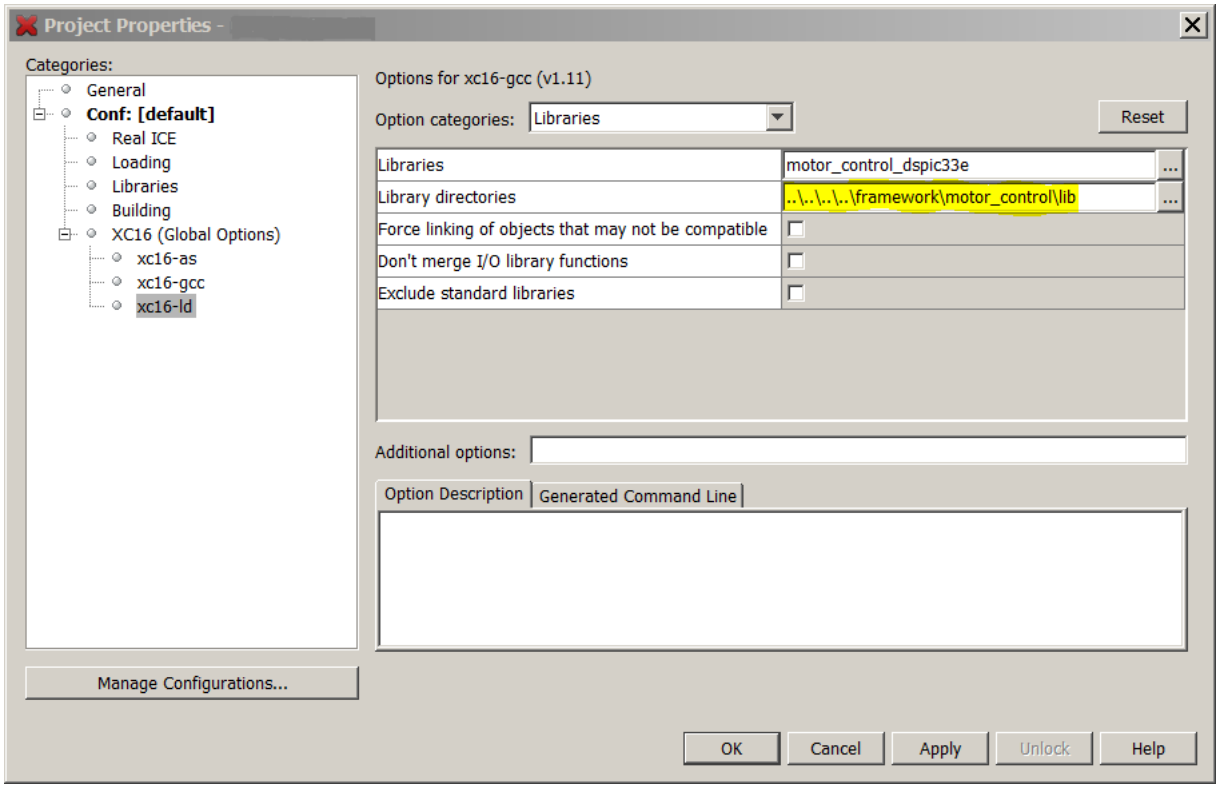
Controller	Functions related to the PI/PID controller.

1.4.2 Library Usage Model

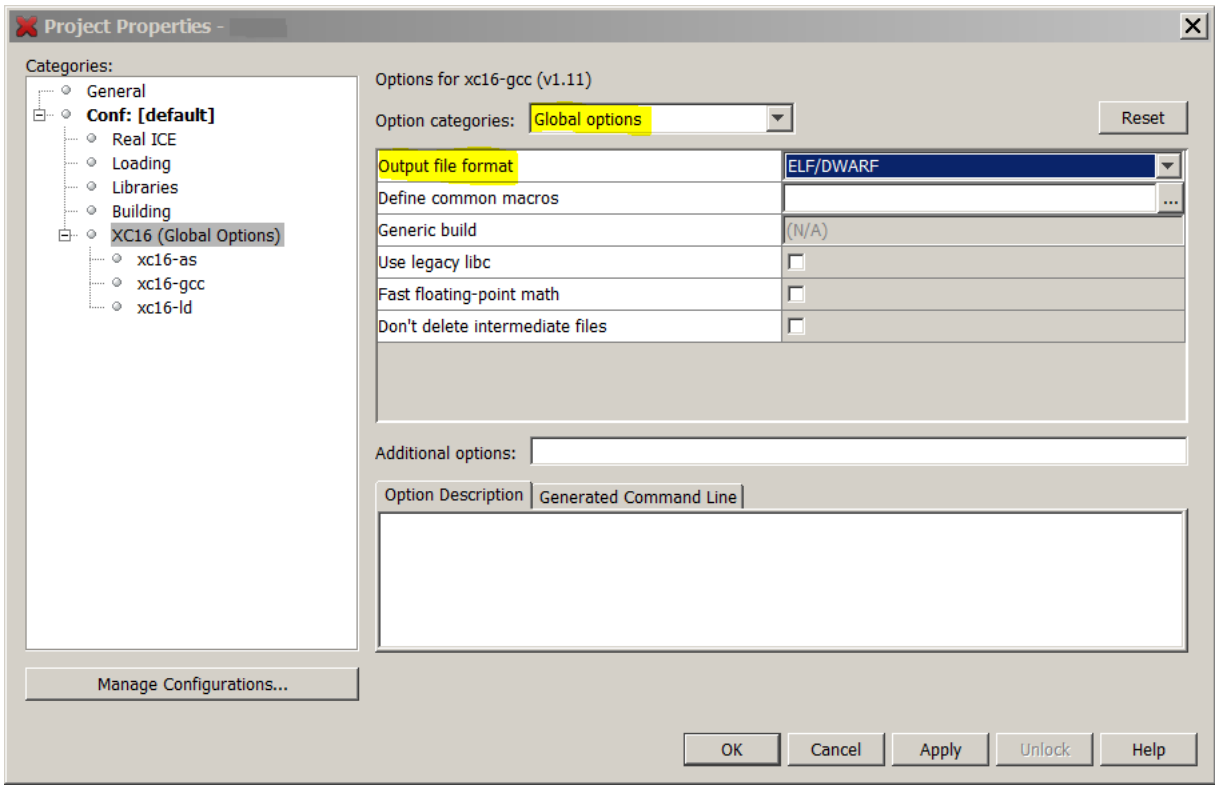
This topic describes the typical usage model for this library.

In order to use the library in the user application:

1. Include the library archive file into the application project. Add the library archive directory into the Project Properties -> xc16-ld -> (Option categories) Libraries field as shown below.



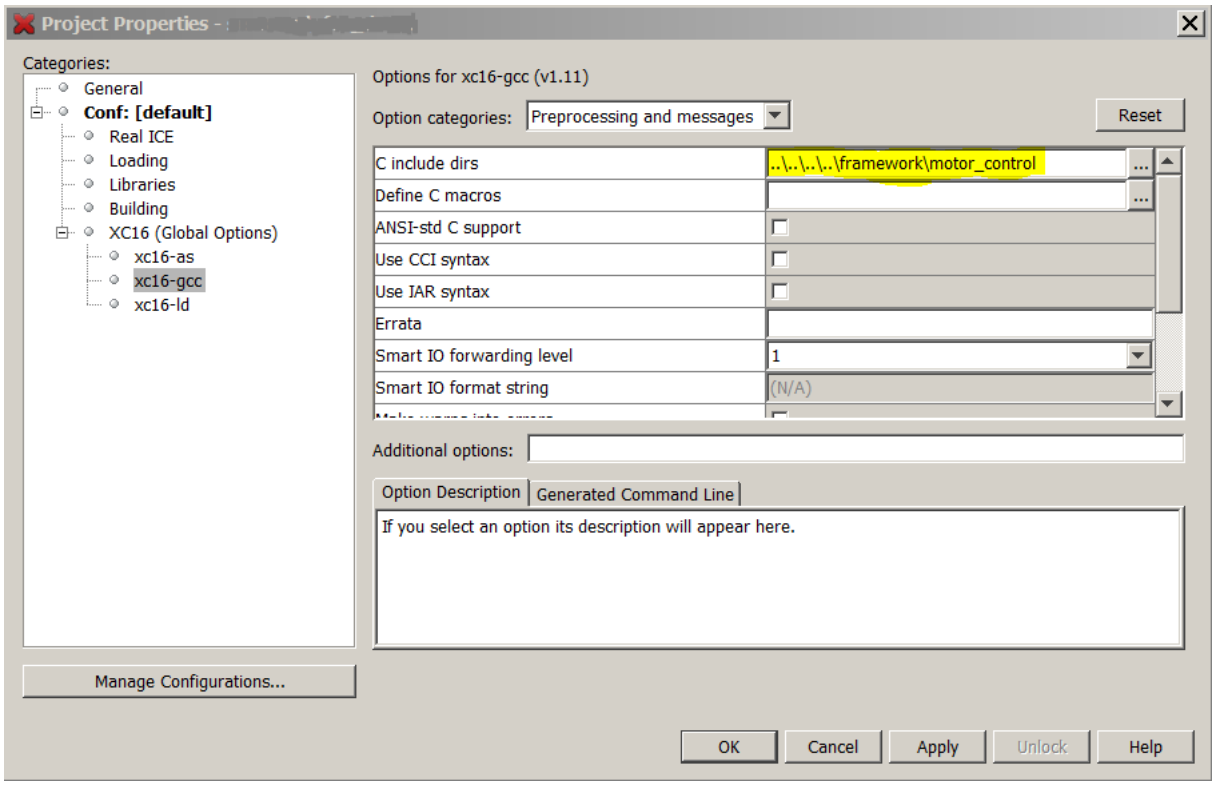
2. Ensure that the application project is configured to use ELF/DWARF type of output file format.



3. Include the motor_control.h file in all C language source (.c) file that use the Motor Control library.

```
#include "motor_control.h"
```

4. Add the library path to the C include directory field in Project Properties -> xc16-gcc -> (Option categories) Preprocessing and messages -> C include dirs.



5. Define variable/structures using type-defines listed in the motor_control.h file.

```
MC_SINCOS_T mcSineCos;
```

6. Initialize these structures as required.

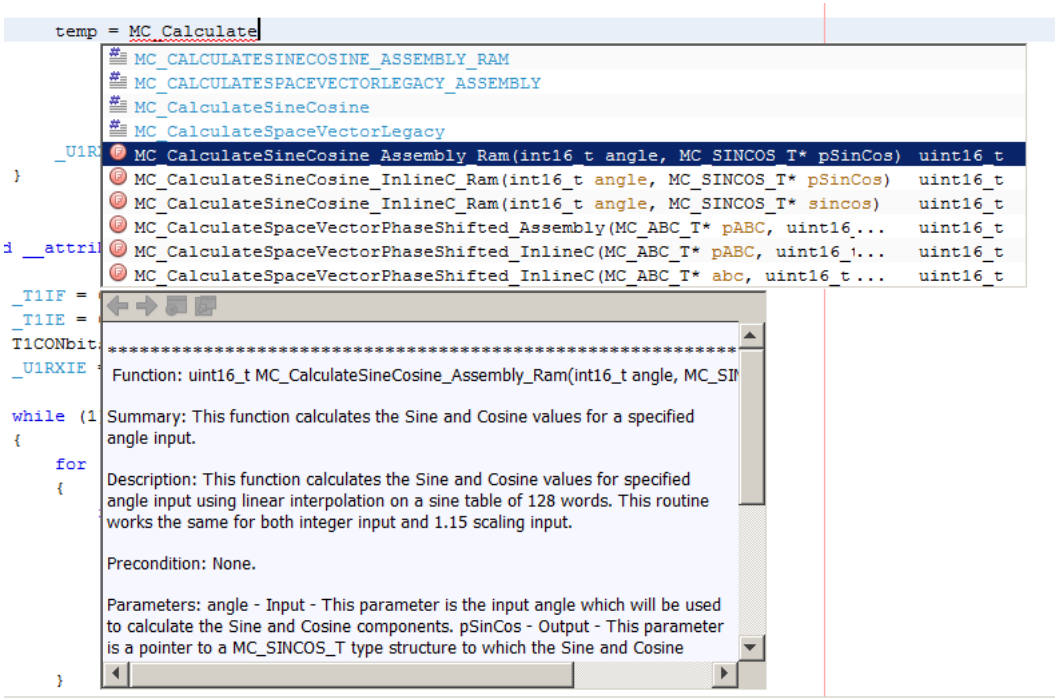
```
mcSineCos.cos = 0;
mcSineCos.sin = 0;
```

7. Where required, call the appropriate library function with the necessary arguments.

```
temp = MC_CalculateSineCosine_Assembly_Ram(angle, &mcSineCos);
```

Notes:

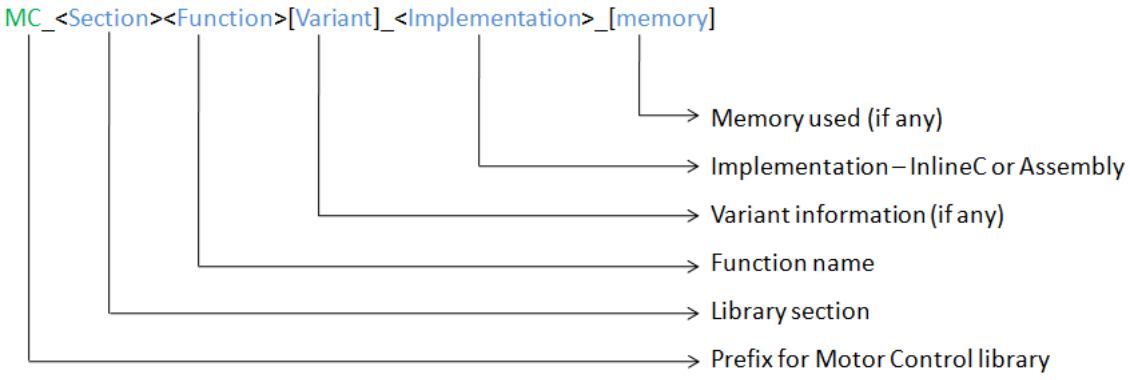
- While typing the library function name, using Ctrl+Space keys in the MPLAB X IDE will bring up a list of relevant function names to choose from, as shown below (code completion feature of the MPLAB X IDE).



- If the user application project files are within a folder structure which will not move relative to the library path, it is recommended to quote relative folder paths for the library header and archive directories.

1.4.3 Function Naming

Library function names have been organized in a specific order with underscore characters and camel case to work with the MPLAB X IDE code completion feature. Figure below describes the library function naming convention.



Example:

MC_CalculateSineCosine_Assembly_Ram

Calculate – Library section

SineCosine – Function name

Assembly – Implementation

Ram – Memory where the sine table is located

1.5 Configuring the Library Mapping

The library provides short function names for each of the library functions.

Each short function name maps into a default library function prototype as listed in the motor_control_mapping.h file. If required, users can add configuration define statements in the motor_control_mapping.h file to change the default mapping.

For instance, the CalculateSineCosine function has a short name MC_CalculateSineCosine. By default, MC_CalculateSineCosine short name maps to MC_CalculateSineCosine_Assembly_Ram. However, adding the following configuration define in the motor_control_mapping.h file:

```
#define MC_CONFIG_CALCULATESINECOSINE_INLINEC_RAM
```

will automatically map the MC_CalculateSineCosine short name to the function prototype MC_CalculateSineCosine_InlineC_Ram.

The default mappings are as shown in the table below.

Short function name	Library function prototype
MC_CalculateSineCosine	MC_CalculateSineCosine_Assembly_Ram
MC_TransformParkInverse	MC_TransformParkInverse_Assembly
MC_TransformClarkeInverse	MC_TransformClarkeInverseSwappedInput_Assembly
MC_CalculateSpaceVector	MC_CalculateSpaceVectorPhaseShifted_Assembly
MC_TransformClarke	MC_TransformClarke_Assembly
MC_TransformPark	MC_TransformPark_Assembly
MC_ControllerPIUpdate	MC_ControllerPIUpdate_Assembly

1.6 Modifying the Library

At release, the library includes two archive files:

1. libmotor_control_dspic33e-elf.a - To be used with dsPIC33E family of devices.
2. libmotor_control_dspic33f-elf.a - To be used with dsPIC33F family of devices.

Both of these archive files released with the library are built using the ELF type Object Module Format (OMF). The source (.s) files that are used to build these archive files are also provided with the library, in the /src folder. These source files are provided for reference and need not be used directly in a typical library usage scenario.

Users may also utilize the flexibility provided by the library to modify the source files and re-build their own archive files. In order to help users to get started, two library projects have been included in the /mplabx folder of the library:

1. libmotor_control_dspic33e-elf.X - To be used with dsPIC33E family of devices.
2. libmotor_control_dspic33f-elf.X - To be used with dsPIC33F family of devices.

These library projects assemble the source files from the /src folder using an assembly API file, /src/mc_interfaces_dspic.inc, and then archive the assembled output object files into a binary archive. The binary archive is, by default, saved in the /mplabx/libmotor_control_dspic33<e/f>-elf.X/dist/default/production folder.

1.7 Performance

The following table lists the approximate number of instruction cycles required to:

1. Save the arguments
2. Call into the library function
3. Return from the library function
4. Save the return value

Function Name	Instruction Cycle Usage (Minimum)	Instruction Cycle Usage (Maximum)
MC_TransformParkInverse_Assembly	33	33
MC_TransformClarkeInverseSwappedInput_Assembly	34	34
MC_CalculateSpaceVectorPhaseShifted_Assembly	51	59
MC_TransformClarke_Assembly	29	29
MC_TransformPark_Assembly	33	33
MC_ControllerPIUpdate_Assembly	53	55
MC_CalculateSineCosine_Assembly_Ram	34	49

It was observed that C variant of the library functions, when inlined by the compiler at optimization level -O2, required significantly lesser number of instruction cycles compared to their equivalent assembly-based variants. This relative improvement in performance is due to the absence of function call overhead when the C functions are inlined by the compiler into the application code.

Note:

The above performance numbers were measured on the current/latest version of the library.

1.8 Register Usage

The register usage and handling behavior of the library functions are as described below.

1. Assembly implementation: Register W0 - W7 are caller saved. The calling function must preserve these values before the library function call if their value is required subsequently from the library function call. The stack is a good place to preserve these values.
2. Assembly implementation: Register W8 - W14 are saved by the library function if they are used within the library function.
3. Register W0 - W7 may be used for argument transmission.
4. Accumulator (A and B) registers are not saved by any of the library functions. If the calling function requires the accumulator registers to be unchanged after the library function call, the calling function will have to save the accumulator registers before the library function call.
5. Core Control Register (CORCON): certain library functions require CORCON register to be setup in a certain state in order to operate correctly. Due to this requirement, these library functions save the CORCON register on the stack in the beginning of the function and restore it before the function return. After saving the CORCON register, library functions write to all bits of the CORCON register. Thus, for the brief duration when these library functions are executing, the state of CORCON register may be different from its state as set by the function caller. This may temporarily change the CPU core behavior with respect to exception processing latency, DO loop termination, CPU interrupt priority level and DSP-engine behavior.

1.9 Library Interface

1.9.1 Types

1.9.1.1 MC_SINCOS_T Structure

C

```
typedef struct {
  int16_t cos;
  int16_t sin;
} MC_SINCOS_T;
```

Description

Sine-Cosine data type

This structure will host parameters related to Sine and Cosine components of the motor angle.

Members

Members	Description
int16_t cos;	Cosine component
int16_t sin;	Sine component

1.9.1.2 MC_DQ_T Structure

C

```
typedef struct {
  int16_t d;
  int16_t q;
} MC_DQ_T;
```

Description

D-Q reference frame data type

This structure will host parameters related to D-Q reference frame.

Members

Members	Description
int16_t d;	D-axis component
int16_t q;	Q-axis component

1.9.1.3 MC_ALPHABETA_T Structure

C

```
typedef struct {
  int16_t alpha;
  int16_t beta;
} MC_ALPHABETA_T;
```

Description

Alpha-Beta reference frame data type

This structure will host parameters related to Alpha-Beta reference frame.

Members

Members	Description
int16_t alpha;	Alpha component
int16_t beta;	Beta component

1.9.1.4 MC_ABC_T Structure

C

```
typedef struct {
  int16_t a;
  int16_t b;
  int16_t c;
} MC_ABC_T;
```

Description

ABC reference frame data type

This structure will host parameters related to ABC reference frame.

Members

Members	Description
int16_t a;	Phase A component
int16_t b;	Phase B component
int16_t c;	Phase C component

1.9.1.5 MC_DUTYCYCLEOUT_T Structure

C

```
typedef struct {
  uint16_t dutycycle1;
  uint16_t dutycycle2;
  uint16_t dutycycle3;
} MC_DUTYCYCLEOUT_T;
```

Description

Duty-cycle data type

This structure will host parameters related to PWM module Duty Cycle values.

Members

Members	Description
uint16_t dutycycle1;	Duty cycle for phase #1
uint16_t dutycycle2;	Duty cycle for phase #2
uint16_t dutycycle3;	Duty cycle for phase #3

1.9.1.6 MC_PISTATE_T Structure

C

```
typedef struct {
  int32_t integrator;
```

```
  int16_t kp;
  int16_t ki;
  int16_t kc;
  int16_t outMax;
  int16_t outMin;
} MC_PISTATE_T;
```

Description

PI Controller State data type

This structure will host parameters related to the PI Controller state.

Members

Members	Description
int32_t integrator;	Integrator sum
int16_t kp;	Proportional gain co-efficient term
int16_t ki;	Integral gain co-efficient term
int16_t kc;	Excess gain co-efficient term
int16_t outMax;	Maximum output limit
int16_t outMin;	Minimum output limit

1.9.1.7 MC_PIPARMIN_T Structure

C

```
typedef struct {
  MC_PISTATE_T piState;
  int16_t inReference;
  int16_t inMeasure;
} MC_PIPARMIN_T;
```

Description

PI Controller Input data type

This structure will host parameters related to the PI Controller input. PI controller state is a part of the PI Controller input.

Members

Members	Description
MC_PISTATE_T piState;	PI state as input parameter to the PI controller
int16_t inReference;	Input reference to the PI controller
int16_t inMeasure;	Input measured value

1.9.1.8 MC_PIPARMOUT_T Structure

C

```
typedef struct {
  int16_t out;
} MC_PIPARMOUT_T;
```

Description

PI Controller Output data type

This structure will host parameters related to the PI Controller output.

Members

Members	Description
int16_t out;	Output of the PI controller

1.9.2 Calculate Functions

1.9.2.1 MC_CalculateSineCosine_Assembly_Ram Function

C

```
uint16_t MC_ATTRB MC_CalculateSineCosine_Assembly_Ram(
    int16_t angle,
    MC_SINCOS_T * pSinCos
);
```

Description

This function calculates the Sine and Cosine values for specified angle input using linear interpolation on a sine table of 128 words. This routine works the same for both integer input and 1.15 scaling input.

Preconditions

None.

Parameters

Parameters	Description
angle	Input - This parameter is the input angle which will be used to calculate the Sine and Cosine components.
pSinCos	Output - This parameter is a pointer to a MC_SINCOS_T type structure to which the Sine and Cosine components of the angle are written.

Returns

Unsigned integer value '1' for direct look up and '2' for interpolation.

Remarks

For integer scaling the Angle is scaled such that 0 <= Angle < 2*pi corresponds to 0 <= Ang < 0xFFFF. The resulting Sine and Cosine values are returned scaled to -32769 -> 32767 i.e. (0x8000 -> 0x7FFF). For 1.15 scaling the Angle is scaled such that -pi <= Angle < pi corresponds to -1 -> 0.9999 i.e. (0x8000 <= Ang < 0x7FFF). The resulting Sine and Cosine values are returned scaled to -1 -> 0.9999 i.e. (0x8000 -> 0x7FFF).

Example

```
uint16_t temp;
int16_t angle;
MC_SINCOS_T mcSinCos;
temp = MC_CalculateSineCosine_Assembly_Ram(angle, &mcSinCos);
```

1.9.2.2 MC_CalculateSineCosine_InlineC_Ram Function

C

```
static inline uint16_t MC_ATTRB MC_CalculateSineCosine_InlineC_Ram(
    int16_t angle,
    MC_SINCOS_T * pSinCos
);
```

Description

This function calculates the Sine and Cosine values for specified angle input using linear interpolation on a sine table of 128 words. This routine works the same for both integer input and 1.15 scaling input.

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own

assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
angle	Input - This parameter is the input angle which will be used to calculate the Sine and Cosine components.
pSinCos	Output - This parameter is a pointer to a MC_SINCOS_T type structure to which the Sine and Cosine components of the angle are written.

Returns

Unsigned integer value '1' for direct look up and '2' for interpolation.

Remarks

This function uses the sine-table included within the library archive file. Hence, the library archive file must be included in the application project for this function to link correctly.

For integer scaling the Angle is scaled such that 0 <= Angle < 2*pi corresponds to 0 <= Ang < 0xFFFF. The resulting Sine and Cosine values are returned scaled to -32769 -> 32767 i.e. (0x8000 -> 0x7FFF). For 1.15 scaling the Angle is scaled such that -pi <= Angle < pi corresponds to -1 -> 0.9999 i.e. (0x8000 <= Ang < 0x7FFF). The resulting Sine and Cosine values are returned scaled to -1 -> 0.9999 i.e. (0x8000 -> 0x7FFF).

Example

```
uint16_t temp;
int16_t angle;
MC_SINCOS_T mcSinCos;
temp = MC_CalculateSineCosine_InlineC_Ram(angle, &mcSinCos);
```

1.9.2.3 MC_CalculateSpaceVectorPhaseShifted_Assembly Function

C

```
uint16_t MC_ATTRB MC_CalculateSpaceVectorPhaseShifted_Assembly(
    MC_ABC_T * pABC,
    uint16_t iPwmPeriod,
    MC_DUTYCYCLEOUT_T * pDutyCycleOut
);
```

Description

This function calculates the duty cycle values based on the three scaled reference vectors in the a-b-c reference frame and the PWM period value.

This function is designed to work with the TransformClarkeInverseSwappedInput() in order to simplify the calculation of three-phase duty cycle values from a given set of inputs in the alpha-beta reference frame. This function uses a reference axis that is phase shifted by 30 degrees relative to the standard Space Vector Modulation reference axis. This phase-shifted reference axis is accomodated by using reference vector inputs from a modified version of the inverse Clarke transform which swaps the alpha-beta values at its input.

Preconditions

None.

Parameters

Parameters	Description
pABC	Input - This parameter is a pointer to a MC_ABC_T type structure.
iPwmPeriod	Input - This parameter is an unsigned integer value of the PWM period.

pDutyCycleOut	Output - This parameter is a pointer to a MC_DUTYCYCLEOUT_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ABC_T mcVabc;
uint16_t iPwmPeriod;
MC_DUTYCYCLEOUT_T mcDutyCycleOut;
temp = MC_CalculateSpaceVectorPhaseShifted_Assembly(&mcVabc, iPwmPeriod, &mcDutyCycleOut);
```

1.9.2.4 MC_CalculateSpaceVectorPhaseShifted_InlineC Function

C

```
static inline uint16_t MC_ATTRB MC_CalculateSpaceVectorPhaseShifted_InlineC(
    MC_ABC_T * pABC,
    uint16_t iPwmPeriod,
    MC_DUTYCYCLEOUT_T * pDutyCycleOut
);
```

Description

This function calculates the duty cycle values based on the three scaled reference vectors in the a-b-c reference frame and the PWM period value.

This function is designed to work with the TransformClarkeInverseSwappedInput() in order to simplify the calculation of three-phase duty cycle values from a given set of inputs in the alpha-beta reference frame. This function uses a reference axis that is phase shifted by 30 degrees relative to the standard Space Vector Modulation reference axis. This phase-shifted reference axis is accomodated by using reference vector inputs from a modified version of the inverse Clarke transform which swaps the alpha-beta values at its input.

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
pABC	Input - This parameter is a pointer to a MC_ABC_T type structure.
iPwmPeriod	Input - This parameter is an unsigned integer value of the PWM period.
pDutyCycleOut	Output - This parameter is a pointer to a MC_DUTYCYCLEOUT_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ABC_T mcVabc;
uint16_t iPwmPeriod;
```

```
MC_DUTYCYCLEOUT_T mcDutyCycleOut;
temp = MC_CalculateSpaceVectorPhaseShifted_InlineC(&mcVabc, iPwmPeriod, &mcDutyCycleOut);
```

1.9.3 Controller Functions

1.9.3.1 MC_ControllerPIUpdate_Assembly Function

C

```
uint16_t MC_ATTRB MC_ControllerPIUpdate_Assembly(
    int16_t inReference,
    int16_t inMeasure,
    MC_PISTATE_T * pPIState,
    int16_t * pPIParmOutput
);
```

Description

This function calculates a PI correction output from a given measured input and a reference. The equation for PI output is:

```
out = Kp*(inReference-inMeasure) + Ki*Integral[inReference-inMeasure, dt] - Kc*Excess
```

Where, out = Fractional 1.15 output, is limited to between outMax and outMin. Kp = Proportional gain co-efficient term Ki = Integral gain co-efficient term Kc = Excess gain co-efficient term Excess = Excess error after "out" is limited to between outMax and outMin. This implementation includes an anti-windup term to limit the integral windup.

Preconditions

None.

Parameters

Parameters	Description
inReference	Input - This parameter is a 1.15 fractional format reference input.
inMeasure	Input - This parameter is a 1.15 fractional format measured input.
pPIState	Input/Output - This paramater is a pointer to a MC_PISTATE_T type structure.
pPIParmOutput	Output - This paramater is a pointer to a signed integer type variable.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format, except for Kp which is in 1.11 format. The constant Kp is scaled so it can be represented in 1.15 format by adjusting the constant by a power of 2.

Example

```
MC_PIPARMIN_T mcPIParmInput;
MC_PIPARMOUT_T mcPIParmOutput;
temp = MC_ControllerPIUpdate_Assembly(mcPIParmInput.inReference, mcPIParmInput.inMeasure,
&mcPIParmInput.piState, &mcPIParmOutput.out);
```

1.9.3.2 MC_ControllerPIUpdate_InlineC Function

C

```
static inline uint16_t MC_ATTRB MC_ControllerPIUpdate_InlineC(
    int16_t inReference,
    int16_t inMeasure,
    MC_PISTATE_T * pPIState,
    int16_t * pPIParmOutput
);
```

Description

This function calculates a PI correction output from a given measured input and a reference. The equation for PI output is:

```
out = Kp*(inReference-inMeasure) + Ki*Integral[inReference-inMeasure, dt] - Kc*Excess
```

Where, out = Fractional 1.15 output, is limited to between outMax and outMin. Kp = Proportional gain co-efficient term Ki = Integral gain co-efficient term Kc = Excess gain co-efficient term Excess = Excess error after "out" is limited to between outMax and outMin. This implementation includes an anti-windup term to limit the integral windup.

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
inReference	Input - This parameter is a 1.15 fractional format reference input.
inMeasure	Input - This parameter is a 1.15 fractional format measured input.
pPIState	Input/Output - This paramater is a pointer to a MC_PISTATE_T type structure.
pPIParmOutput	Output - This paramater is a pointer to a signed integer type variable.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format, except for Kp which is in 1.11 format. The constant Kp is scaled so it can be represented in 1.15 format by adjusting the constant by a power of 2.

Example

```
MC_PIPARMIN_T mcPIParmInput;
MC_PIPARMOUT_T mcPIParmOutput;
temp = MC_ControllerPIUpdate_InlineC(mcPIParmInput.inReference, mcPIParmInput.inMeasure,
&mcPIParmInput.piState, &mcPIParmOutput.out);
```

1.9.4 Transform Functions

1.9.4.1 MC_TransformClarke_Assembly Function

C

```
uint16_t MC_ATTRB MC_TransformClarke_Assembly(
    MC_ABC_T * pABC,
    MC_ALPHABETA_T * pAlphaBeta
);
```

Description

This function transforms inputs in an a-b-c reference frame to an alpha-beta reference frame using the equation:

```
alpha = a
beta = a*(1/sqrt(3)) + 2*b*(1/sqrt(3))
```

Preconditions

None.

Parameters

Parameters	Description
pABC	Input - This parameter is a pointer to a MC_ABC_T type structure.
pAlphaBeta	Output - This parameter is a pointer to a MC_ALPHABETA_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ABC_T mcIabc;
MC_ALPHABETA_T mcIAlphaBeta;
temp = MC_TransformClarke_Assembly(&mcIabc, &mcIAlphaBeta);
```

1.9.4.2 MC_TransformClarke_InlineC Function

C

```
static inline uint16_t MC_ATTRB MC_TransformClarke_InlineC(
    MC_ABC_T * pABC,
    MC_ALPHABETA_T * pAlphaBeta
);
```

Description

This function transforms inputs in an a-b-c reference frame to an alpha-beta reference frame using the equation:

```
alpha = a
beta = a*(1/sqrt(3)) + 2*b*(1/sqrt(3))
```

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
pABC	Input - This parameter is a pointer to a MC_ABC_T type structure.
pAlphaBeta	Output - This parameter is a pointer to a MC_ALPHABETA_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ABC_T mcIabc;
MC_ALPHABETA_T mcIAlphaBeta;
temp = MC_TransformClarke_InlineC(&mcIabc, &mcIAlphaBeta);
```

1.9.4.3 MC_TransformClarkeInverseSwappedInput_Assembly Function

C

```
uint16_t MC_ATTRB MC_TransformClarkeInverseSwappedInput_Assembly(
    MC_ALPHABETA_T * pAlphaBeta,
    MC_ABC_T * pABC
);
```

Description

This function calculates the scaled reference vectors in an a-b-c reference frame using inputs from an alpha-beta reference frame, as described by the equation:

```
a = beta
b = -beta/2 + (sqrt(3)/2) * alpha
c = -beta/2 - (sqrt(3)/2) * alpha
```

This is a modified variant of the inverse Clarke transformation where alpha & beta are swapped compared to the normal inverse Clarke transformation. This function is designed to work with the CalculateSpaceVectorPhaseShifted() in order to simplify the calculation of three-phase duty cycle values from a given set of inputs in the alpha-beta reference frame.

Preconditions

None.

Parameters

Parameters	Description
pAlphaBeta	Input - This parameter is a pointer to a MC_ALPHABETA_T type structure.
pABC	Output - This parameter is a pointer to a MC_ABC_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ALPHABETA_T mcVAlphaBeta;
MC_ABC_T mcVabc;
temp = MC_TransformClarkeInverseSwappedInput_Assembly(&mcVAlphaBeta, &mcVabc);
```

1.9.4.4 MC_TransformClarkeInverseSwappedInput_InlineC Function

C

```
static inline uint16_t MC_ATTRB MC_TransformClarkeInverseSwappedInput_InlineC(
    MC_ALPHABETA_T * pAlphaBeta,
    MC_ABC_T * pABC
);
```

Description

This function calculates the scaled reference vectors in an a-b-c reference frame using inputs from an alpha-beta reference frame, as described by the equation:

```
a = beta
b = -beta/2 + (sqrt(3)/2) * alpha
c = -beta/2 - (sqrt(3)/2) * alpha
```

This is a modified variant of the inverse Clarke transformation where alpha & beta are swapped compared to the normal

inverse Clarke transformation. This function is designed to work with the CalculateSpaceVectorPhaseShifted() in order to simplify the calculation of three-phase duty cycle values from a given set of inputs in the alpha-beta reference frame.

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
pAlphaBeta	Input - This parameter is a pointer to a MC_ALPHABETA_T type structure.
pABC	Output - This parameter is a pointer to a MC_ABC_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ALPHABETA_T mcVAlphaBeta;
MC_ABC_T mcVabc;
temp = MC_TransformClarkeInverseSwappedInput_InlineC(&mcVAlphaBeta, &mcVabc);
```

1.9.4.5 MC_TransformPark_Assembly Function

C

```
uint16_t MC_ATTRB MC_TransformPark_Assembly(
    MC_ALPHABETA_T * pAlphaBeta,
    MC_SINCOS_T * pSinCos,
    MC_DQ_T * pDQ
);
```

Description

This function transforms inputs in an alpha-beta reference frame to a stationary d-q reference frame using the equation:

```
d = alpha*cos + beta*sin
q = -alpha*sin + beta*cos
```

Preconditions

None.

Parameters

Parameters	Description
pAlphaBeta	Input - This parameter is a pointer to a MC_ALPHABETA_T type structure.
pSinCos	Input - This parameter is a pointer to a MC_SINCOS_T type structure.
pDQ	Output - This parameter is a pointer to a MC_DQ_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ALPHABETA_T mcIAlphaBeta;
MC_SINCOS_T mcSinCos;
MC_DQ_T mcIDQ;
temp = MC_TransformPark_Assembly(&mcIAlphaBeta, &mcSinCos, &mcIDQ);
```

1.9.4.6 MC_TransformPark_InlineC Function

C

```
static inline uint16_t MC_ATTRB MC_TransformPark_InlineC(
    MC_ALPHABETA_T * pAlphaBeta,
    MC_SINCOS_T * pSinCos,
    MC_DQ_T * pDQ
);
```

Description

This function transforms inputs in an alpha-beta reference frame to a stationary d-q reference frame using the equation:

```
d = alpha*cos + beta*sin
q = -alpha*sin + beta*cos
```

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
pAlphaBeta	Input - This parameter is a pointer to a MC_ALPHABETA_T type structure.
pSinCos	Input - This parameter is a pointer to a MC_SINCOS_T type structure.
pDQ	Output - This parameter is a pointer to a MC_DQ_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_ALPHABETA_T mcIAlphaBeta;
MC_SINCOS_T mcSinCos;
MC_DQ_T mcIDQ;
temp = MC_TransformPark_InlineC(&mcIAlphaBeta, &mcSinCos, &mcIDQ);
```

1.9.4.7 MC_TransformParkInverse_Assembly Function

C

```
uint16_t MC_ATTRB MC_TransformParkInverse_Assembly(
    MC_DQ_T * pDQ,
    MC_SINCOS_T * pSinCos,
    MC_ALPHABETA_T * pAlphaBeta
);
```

Description

This function calculates the inverse Park transform on a pair of stationary reference frame inputs. Inverse park

transformation is performed as described by the equation:

```
alpha = d*cos - q*sin
beta = d*sin + q*cos
```

Preconditions

None.

Parameters

Parameters	Description
pDQ	Input - This parameter is a pointer to a MC_DQ_T type structure.
pSinCos	Input - This parameter is a pointer to a MC_SINCOS_T type structure.
pAlphaBeta	Output - This parameter is a pointer to a MC_ALPHABETA_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_DQ_T mcVDQ;
MC_SINCOS_T mcSinCos;
MC_ALPHABETA_T mcVAlphaBeta;
temp = MC_TransformParkInverse_Assembly(&mcVDQ, &mcSinCos, &mcVAlphaBeta);
```

1.9.4.8 MC_TransformParkInverse_InlineC Function

C

```
static inline uint16_t MC_ATTRB MC_TransformParkInverse_InlineC(
    MC_DQ_T * pDQ,
    MC_SINCOS_T * pSinCos,
    MC_ALPHABETA_T * pAlphaBeta
);
```

Description

This function calculates the inverse Park transform on a pair of stationary reference frame inputs. Inverse park transformation is performed as described by the equation:

```
alpha = d*cos - q*sin
beta = d*sin + q*cos
```

Inline keyword has been added to the function declaration so that, when appropriate, the compiler may inline the function call, thus eliminating the function call overhead. Static keyword has been added to the function declaration so that, if all calls to the function are integrated into the caller and when the function's address is never used, then the function's own assembler code is never referenced and unless specified in the command-line option, the compiler will not actually output assembler code for the function.

Preconditions

None.

Parameters

Parameters	Description
pDQ	Input - This parameter is a pointer to a MC_DQ_T type structure.
pSinCos	Input - This parameter is a pointer to a MC_SINCOS_T type structure.
pAlphaBeta	Output - This parameter is a pointer to a MC_ALPHABETA_T type structure.

Returns

Unsigned integer value '1'.

Remarks

This routine requires inputs in the 1.15 format.

Example

```
MC_DQ_T mcVDQ;
MC_SINCOS_T mcSinCos;
MC_ALPHABETA_T mcVAlphaBeta;
temp = MC_TransformParkInverse_InlineC(&mcVDQ, &mcSinCos, &mcVAlphaBeta);
```

1.10 Files

Files

Name	Description
motor_control.h	This header file lists all the interfaces used by the Motor Control library.

1.10.1 motor_control.h

Motor Control Library Interface Header File

This header file lists the type defines for structures used by the Motor Control library. Library function definitions are also listed along with information regarding the arguments of each library function. This header file also includes another header file that hosts inline definitions of certain library functions.

Functions

	Name	Description
	MC_CalculateSineCosine_Assembly_Ram	This function calculates the Sine and Cosine values for a specified angle input.
	MC_CalculateSineCosine_InlineC_Ram	This function calculates the Sine and Cosine values for a specified angle input.
	MC_CalculateSpaceVectorPhaseShifted_Assembly	This function calculates the duty cycle values based on the three scaled reference vectors in the a-b-c reference frame and the PWM period value.
	MC_CalculateSpaceVectorPhaseShifted_InlineC	This function calculates the duty cycle values based on the three scaled reference vectors in the a-b-c reference frame and the PWM period value.
	MC_ControllerPIUpdate_Assembly	This function calculates the PI correction.
	MC_ControllerPIUpdate_InlineC	This function calculates the PI correction.
	MC_TransformClarke_Assembly	This function calculates the Clarke transformation.
	MC_TransformClarke_InlineC	This function calculates the Clarke transformation.
	MC_TransformClarkeInverseSwappedInput_Assembly	This function calculates the scaled reference vectors using inputs in an alpha-beta reference frame.
	MC_TransformClarkeInverseSwappedInput_InlineC	This function calculates the scaled reference vectors using inputs in an alpha-beta reference frame.
	MC_TransformPark_Assembly	This function calculates the Park transformation.
	MC_TransformPark_InlineC	This function calculates the Park transformation.
	MC_TransformParkInverse_Assembly	This function calculates the inverse Park transformation.
	MC_TransformParkInverse_InlineC	This function calculates the inverse Park transformation.

Structures

	Name	Description
	MC_ABC_T	ABC reference frame data type This structure will host parameters related to ABC reference frame.
	MC_ALPHABETA_T	Alpha-Beta reference frame data type This structure will host parameters related to Alpha-Beta reference frame.
	MC_DQ_T	D-Q reference frame data type This structure will host parameters related to D-Q reference frame.

	MC_DUTYCYCLEOUT_T	Duty-cycle data type This structure will host parameters related to PWM module Duty Cycle values.
	MC_PIPARMIN_T	PI Controller input type define
	MC_PIPARMOUT_T	PI Controller Output data type This structure will host parameters related to the PI Controller output.
	MC_PISTATE_T	PI Controller State data type This structure will host parameters related to the PI Controller state.
	MC_SINCOS_T	Sine-Cosine data type This structure will host parameters related to Sine and Cosine components of the motor angle.

File Name

motor_control.h

Index