Motor Control Library

User Reference Manual

56800E, 56800Ex
Digital Signal Controller

56800Ex_MCLIB
Rev. 0
02/2014

freescale.com
Chapter 1  License Agreement

FREESCALE SEMICONDUCTOR SOFTWARE LICENSE AGREEMENT.
This is a legal agreement between you (either as an individual or as an authorized
representative of your employer) and Freescale Semiconductor, Inc.
("Freescale"). It concerns your rights to use this file and any accompanying
written materials (the "Software"). In consideration for Freescale allowing you to
access the Software, you are agreeing to be bound by the terms of this Agreement.
If you do not agree to all of the terms of this Agreement, do not download the
Software. If you change your mind later, stop using the Software and delete all
copies of the Software in your possession or control. Any copies of the Software
that you have already distributed, where permitted, and do not destroy will
continue to be governed by this Agreement. Your prior use will also continue to
be governed by this Agreement.

OBJECT PROVIDED, OBJECT REDISTRIBUTION LICENSE GRANT.
Freescale grants to you, free of charge, the non-exclusive, non-transferable right
(1) to reproduce the Software, (2) to distribute the Software, and (3) to sublicense
to others the right to use the distributed Software. The Software is provided to you
only in object (machine-readable) form. You may exercise the rights above only
with respect to such object form. You may not translate, reverse engineer,
decompile, or disassemble the Software except to the extent applicable law
specifically prohibits such restriction. In addition, you must prohibit your
sublicensees from doing the same. If you violate any of the terms or restrictions
of this Agreement, Freescale may immediately terminate this Agreement, and
require that you stop using and delete all copies of the Software in your
possession or control.

COPYRIGHT. The Software is licensed to you, not sold. Freescale owns the
Software, and United States copyright laws and international treaty provisions
protect the Software. Therefore, you must treat the Software like any other
copyrighted material (e.g. a book or musical recording). You may not use or copy
the Software for any other purpose than what is described in this Agreement.
Except as expressly provided herein, Freescale does not grant to you any express
or implied rights under any Freescale or third-party patents, copyrights,
trademarks, or trade secrets. Additionally, you must reproduce and apply any
copyright or other proprietary rights notices included on or embedded in the
Software to any copies or derivative works made thereof, in whole or in part, if
any.

SUPPORT. Freescale is NOT obligated to provide any support, upgrades or new
releases of the Software. If you wish, you may contact Freescale and report
problems and provide suggestions regarding the Software. Freescale has no
obligation whatsoever to respond in any way to such a problem report or
suggestion. Freescale may make changes to the Software at any time, without any
obligation to notify or provide updated versions of the Software to you.
NO WARRANTY. TO THE MAXIMUM EXTENT PERMITTED BY LAW, FREESCALE EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. YOU ASSUME THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE, OR ANY SYSTEMS YOU DESIGN USING THE SOFTWARE (IF ANY). NOTHING IN THIS AGREEMENT MAY BE CONSTRUED AS A WARRANTY OR REPRESENTATION BY FREESCALE THAT THE SOFTWARE OR ANY DERIVATIVE WORK DEVELOPED WITH OR INCORPORATING THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES.

INDEMNITY. You agree to fully defend and indemnify Freescale from any and all claims, liabilities, and costs (including reasonable attorney's fees) related to (1) your use (including your sublicensee's use, if permitted) of the Software or (2) your violation of the terms and conditions of this Agreement.

LIMITATION OF LIABILITY. IN NO EVENT WILL FREESCALE BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS; EXPORT RESTRICTIONS. You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

GOVERNMENT USE. Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Freescale Semiconductor, Inc., 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES. You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by Freescale for incorporation into products intended for use or resale in on-line control
equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems, in which the failure of products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). You specifically represent and warrant that you will not use the Software or any derivative work of the Software for High Risk Activities.

CHOICE OF LAW; VENUE; LIMITATIONS. You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING. You are not authorized to use any Freescale trademarks, brand names, or logos.

ENTIRE AGREEMENT. This Agreement constitutes the entire agreement between you and Freescale regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and Freescale.

SEVERABILITY. If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or Freescale of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER. The waiver by Freescale of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.
Chapter 2  INTRODUCTION

2.1  Overview

This reference manual describes the Motor Control Library (MCLIB) for the Freescale 56F800E(X) family of Digital Signal Controllers. This library contains optimized functions.

2.2  Supported Compilers

Motor Control Library (MCLIB) is written in assembly language with C-callable interface. The library was built and tested using the CodeWarrior™ Development Studio version 10.3.

The library is delivered in library module 56800Ex_MCLIB.lib and is intended for use in small data memory model projects. The interfaces to the algorithms included in this library have been combined into a single public interface include file, gdflib.h. This was done to simplify the number of files required for inclusion by application programs. Refer to the specific algorithm sections of this document for details on the software Application Programming Interface (API), defined and functionality provided for the individual algorithms.

2.3  Installation

If user wants to fully use this library, the CodeWarrior™ Development Studio should be installed prior to the Motor Control Library. In case that Motor Control Library is installed while CodeWarrior™ Development Studio is not present, users can only browse the installed software package, but will not be able to build, download and run code. The installation itself consists of copying the required files to the destination hard drive, checking the presence of CodeWarrior and creating the shortcut under the Start->Programs menu.

The Motor Control Library release is installed in its own folder named 56800Ex_MCLIB.

To start the installation process, perform the following steps:

1. Execute 56800Ex_FSLESL_rXX.exe.
2. Follow the FSLESL software installation instructions on your screen.

2.4  Library Integration

The library integration is described in AN4586 which can be downloaded from www.freescale.com.
2.5 API Definition

The description of each function described in this Motor Control Library user reference manual consists of a number of subsections:

Synopsis
This subsection gives the header files that should be included within a source file that references the function or macro. It also shows an appropriate declaration for the function or for a function that can be substituted by a macro. This declaration is not included in your program; only the header file(s) should be included.

Prototype
This subsection shows the original function prototype declaration with all its arguments.

Arguments
This optional subsection describes input arguments to a function or macro.

Description
This subsection is a description of the function or macro. It explains algorithms being used by functions or macros.

Return
This optional subsection describes the return value (if any) of the function or macro.

Range Issues
This optional subsection specifies the ranges of input variables.

Special Issues
This optional subsection specifies special assumptions that are mandatory for correct function calculation; for example saturation, rounding, and so on.

Implementation
This optional subsection specifies, whether a call of the function generates a library function call or a macro expansion. This subsection also consists of one or more examples of the use of the function. The examples are often fragments of code (not completed programs) for illustration purposes.

See Also
This optional subsection provides a list of related functions or macros.

Performance
This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

### 2.6 Data Types

The 16-bit DSC core supports four types of two’s-complement data formats:
- Signed integer
- Unsigned integer
- Signed fractional
- Unsigned fractional

Signed and unsigned integer data types are useful for general-purpose computation; they are familiar with the microprocessor and microcontroller programmers. Fractional data types allow powerful numeric and digital-signal-processing algorithms to be implemented.

#### 2.6.1 Signed Integer (SI)

This format is used for processing data as integers. In this format, the N-bit operand is represented using the N.0 format (N integer bits). The signed integer numbers lie in the following range:

\[
-2^{N-1} \leq SI \leq \left[2^{N-1} - 1 \right]
\]

Eqn. 2-1

This data format is available for bytes, words, and longs. The most negative, signed word that can be represented is –32,768 ($8000), and the most negative, signed long word is –2,147,483,648 ($80000000).

The most positive, signed word is 32,767 ($7FFF), and the most positive signed long word is 2,147,483,647 ($7FFFFFFF).

#### 2.6.2 Unsigned Integer (UI)

The unsigned integer numbers are positive only, and they have nearly twice the magnitude of a signed number of the same size. The unsigned integer numbers lie in the following range:

\[
0 \leq UI \leq \left[2^{N-1} - 1 \right]
\]

Eqn. 2-2

The binary word is interpreted as having a binary point immediately to the right of the integer’s least significant bit. This data format is available for bytes, words, and long words. The most positive, 16-bit, unsigned integer is 65,535 ($FFFF), and the most positive, 32-bit, unsigned integer is 4,294,967,295 ($FFFFFFFF). The smallest unsigned integer number is zero ($0000), regardless of size.
2.6.3 Signed Fractional (SF)

In this format, the N-bit operand is represented using the $1.\left[N^{-1}\right]$ format (one sign bit, $N-1$ fractional bits). The signed fractional numbers lie in the following range:

$$-1.0 \leq SF \leq 1.0 - 2^{-\left[N^{-1}\right]}$$ \hspace{1cm} Eqn. 2-3

This data format is available for words and long words. For both word and long-word signed fractions, the most negative number that can be represented is $-1.0$; its internal representation is $8000$ (word) or $80000000$ (long word). The most positive word is $7FFF$ ($1.0 - 2^{-15}$); its most positive long word is $7FFFFFFF$ ($1.0 - 2^{-31}$).

2.6.4 Unsigned Fractional (UF)

The unsigned fractional numbers can be positive only, and they have nearly twice the magnitude of a signed number with the same number of bits. The unsigned fractional numbers lie in the following range:

$$0.0 \leq UF \leq 2.0 - 2^{-\left[N^{-1}\right]}$$ \hspace{1cm} Eqn. 2-4

The binary word is interpreted as having a binary point after the MSB. This data format is available for words and longs. The most positive, 16-bit, unsigned number is $FFFF$, or $\{1.0 + (1.0 - 2^{-\left[16^{-1}\right]}{\}} = 1.99997$. The smallest unsigned fractional number is zero ($0000$).

2.7 User Common Types

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Size — bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word8</td>
<td>8</td>
<td>To represent 8-bit signed variable/value.</td>
</tr>
<tr>
<td>UWord8</td>
<td>8</td>
<td>To represent 16-bit unsigned variable/value.</td>
</tr>
<tr>
<td>Word16</td>
<td>16</td>
<td>To represent 16-bit signed variable/value.</td>
</tr>
<tr>
<td>UWord16</td>
<td>16</td>
<td>To represent 16-bit unsigned variable/value.</td>
</tr>
<tr>
<td>Word32</td>
<td>32</td>
<td>To represent 32-bit signed variable/value.</td>
</tr>
<tr>
<td>UWord32</td>
<td>32</td>
<td>To represent 16-bit unsigned variable/value.</td>
</tr>
<tr>
<td>Int8</td>
<td>8</td>
<td>To represent 8-bit signed variable/value.</td>
</tr>
<tr>
<td>UInt8</td>
<td>8</td>
<td>To represent 16-bit unsigned variable/value.</td>
</tr>
<tr>
<td>Int16</td>
<td>16</td>
<td>To represent 16-bit signed variable/value.</td>
</tr>
<tr>
<td>UInt16</td>
<td>16</td>
<td>To represent 16-bit unsigned variable/value.</td>
</tr>
<tr>
<td>Int32</td>
<td>32</td>
<td>To represent 32-bit signed variable/value.</td>
</tr>
</tbody>
</table>
2.8 V2 and V3 Core Support

The library has been written to support both 56800E (V2) and 56800Ex (V3) cores. The V3 core offers new set of math instructions which can simplify and accelerate the algorithm runtime. Therefore certain algorithms can have two prototypes.

If the library is used on the 56800Ex core, the V3 algorithms use is recommended because:

- the code is shorter
- the execution is faster
- the precision of 32-bit calculation is higher

The final algorithm is selected by a define. To select the correct algorithm implementation the user has to set up a define: OPTION_CORE_V3. If this define is not defined, it is automatically set up as 0. If its value is 0, the V2 algorithms are used. If its value is 1, the V3 algorithms are used.

The best way is to define this define is in the project properties (see Figure 2-1):

1. In the left hand tree, expand the C/C++ Build node
2. Click on the Settings node
3. Under the Tool Settings tab, click on the DSC Compiler/Input node
4. In the Defined Macros dialog box click on the first icon (+) and type the following: OPTION_CORE_V3=1
5. Click OK
6. Click OK on the Properties dialog box

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UInt32</td>
<td>32 To represent 16-bit unsigned variable/value.</td>
</tr>
<tr>
<td>Frac16</td>
<td>16 To represent 16-bit signed variable/value.</td>
</tr>
<tr>
<td>Frac32</td>
<td>32 To represent 32-bit signed variable/value.</td>
</tr>
<tr>
<td>NULL</td>
<td>constant Represents NULL pointer.</td>
</tr>
<tr>
<td>bool</td>
<td>16 Boolean variable.</td>
</tr>
<tr>
<td>false</td>
<td>constant Represents false value.</td>
</tr>
<tr>
<td>true</td>
<td>constant Represents true value.</td>
</tr>
<tr>
<td>FRAC16()</td>
<td>macro Transforms float value from (-1, 1) into fractional representation (-32768, 32767).</td>
</tr>
<tr>
<td>FRAC32()</td>
<td>macro Transforms float value from (-1, 1) into fractional representation (-2147483648, 2147483648).</td>
</tr>
</tbody>
</table>
2.9 Special Issues

All functions in the Motor Control Library are implemented without storing any of the volatile registers (refer to the compiler manual) used by the respective routine. Only non-volatile registers (C10, D10, R5) are saved by pushing the registers on the stack. Therefore, if the particular registers initialized before the library function call are to be used after the function call, it is necessary to save them manually.
# 3.1 API Summary

Table 3-1. API Functions Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_ClarkTrf</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>void</td>
<td>This function calculates the Clarke transformation algorithm.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_ClarkTrfInv</td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td>void</td>
<td>This function calculates the inverse Clarke transformation algorithm.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_ParkTrf</td>
<td>MCLIB_2_COOR_SYST_D_Q_T *pudtDQ MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_ANGLE_T *pudtSinCos</td>
<td>void</td>
<td>This function calculates the Park transformation algorithm.</td>
</tr>
<tr>
<td>MCLIB_ParkTrfInv</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>void</td>
<td>This function calculates the inverse Park transformation algorithm.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_D_Q_T *pudtDQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLIB_ANGLE_T *pudtSinCos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_SvmStd</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>UWord16</td>
<td>This function calculates the appropriate duty-cycle ratios, which are needed for generating the given stator-reference voltage vector using a special standard space vector modulation technique.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_SvmU0n</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>UWord16</td>
<td>This function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses the special Space Vector Modulation technique, termed Space Vector Modulation with O000 Nulls.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_SvmU7n</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>UWord16</td>
<td>This function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses the special Space Vector Modulation technique, termed Space Vector Modulation with O111 Nulls.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-1. API Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_SvmAlt</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>Uword16</td>
<td>This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Standard Space Vector Modulation technique.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_SvmSci</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>Uword16</td>
<td>This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using the General Sinusoidal Modulation with an injection of the third harmonic.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_PwmIct</td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta</td>
<td>Uword16</td>
<td>This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using the General Sinusoidal Modulation technique.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_3_COOR_SYST_T *pudtAbc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_DecouplingPMSM</td>
<td>MCLIB_2_COOR_SYST_D_Q_T *pudtUs</td>
<td>void</td>
<td>This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the control.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_D_Q_T *pudtIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frac16 f16AngularVelocity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLIB_DECOUPLING_PMSM_PARAM_T *pudtDecParam</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_D_Q_T *pudtUsDec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_ElimDcBusRip</td>
<td>Frac16 f16InvModIndex, Frac16 f16DcBusMsr</td>
<td>void</td>
<td>This function is used for elimination of the DC-bus voltage ripple.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_ElimDcBusRipGen</td>
<td>Frac16 f16DcBusMsr</td>
<td>void</td>
<td>This function is used for elimination of the DC-bus voltage ripple for the general cases of the modulation.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCLIB_VectorLimit</td>
<td>MCLIB_2_COOR_SYST_T *pudtInVector MCLIB_2_COOR_SYST_T *pudtLimVector</td>
<td>void</td>
<td>This function calculates the amplitude limitation of the input vector described by the dq components. Limitation is calculated to achieve the zero angle error.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_VECTOR_LIMIT_PARAMS_T *pudtParams</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3-1. API Functions Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_VectorLimit12</td>
<td>MCLIB_2_COOR_SYST_T *pudtInVector</td>
<td>This function calculates the amplitude limitation of the input vector described by the dq components. Limitation is calculated to achieve the zero angle error. This function is quicker with reduced precision in comparison to MCLIB_VectorLimit.</td>
</tr>
<tr>
<td></td>
<td>MCLIB_2_COOR_SYST_T *pudtLimVector</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MCLIB_VECTOR_LIMIT_PARAMS_T *pudtParams</td>
<td></td>
</tr>
</tbody>
</table>
3.2 MCLIB_ClarkTrf

This function calculates the Clarke transformation algorithm.

3.2.1 Synopsis

```c
#include "mclib.h"
void MCLIB_ClarkTrf(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
                    MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.2.2 Prototype

```c
asm void MCLIB_ClarkTrfFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
                            *pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.2.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a two-phase rotating orthogonal system, the MCLIB_2_COOR_SYST_ALPHA_BETA_T data type is defined in the header file MCLIB_types.h.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a three-phase rotating system, the MCLIB_3_COOR_SYST_T data type is defined in the header file MCLIB_types.h.</td>
</tr>
</tbody>
</table>

3.2.4 Availability

This library module is available in the C-callable interface assembly format.

This library module is targeted for the 56800E and 56800Ex platforms.
3.2.5 Dependencies

List of all dependent files:
- MCLIB_CPTrfAsm.h
- MCLIB_types.h

3.2.6 Description

The MCLIB_ClarkTrf function calculates the Clarke transformation, which is used to transform values (flux, voltage, current) from the three-phase rotating coordinate system to the alpha-beta rotating orthogonal coordinate system, according to these functions:

\[
alpha = a \quad \text{Eqn. 3-1}
\]

\[
beta = \frac{1}{\sqrt{3}}a + \frac{2}{\sqrt{3}}b \quad \text{Eqn. 3-2}
\]

3.2.7 Range Issues

This function works with the 16-bit signed fractional values in the range \((-1, 1)\).

3.2.8 Special Issues

The function MCLIB_ClarkTrf is the saturation mode independent.

3.2.9 Implementation

Example 3-1. Implementation Code

```c
#include "mclib.h"

static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* ABC structure initialization */
    mudtAbc.f16A = 0;
    mudtAbc.f16B = 0;
    mudtAbc.f16C = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Clarke Transformation calculation */
    MCLIB_ClarkTrf(&mudtAlphaBeta, &mudtAbc);
}
```

Motor Control Library, Rev. 0
3.2.10 **See Also**

See `MCLIB_ClarkTrfInv` for more information.

3.2.11 **Performance**

Table 3-4. Performance of the `MCLIB_ClarkTrf` Function

<table>
<thead>
<tr>
<th>Code Size (bytes)</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (bytes)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td>Min. 26 cycles</td>
</tr>
</tbody>
</table>
3.3 MCLIB_ClarkTrfInv

This function calculates the inverse Clarke transformation algorithm.

3.3.1 Synopsis

```c
#include "mclib.h"

void MCLIB_ClarkTrfInv(MCLIB_3_COOR_SYST_T *pudtAbc,
                        MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta)
```

3.3.2 Prototype

```asm
asm void MCLIB_ClarkTrfInvFAsm(MCLIB_3_COOR_SYST_T *pudtAbc,
                               MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta)
```

3.3.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a two-phase rotating orthogonal system, the MCLIB_2_COOR_SYST_ALPHA_BETA_T data type is defined in the header file MCLIB_types.h.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a three-phase rotating system, the MCLIB_3_COOR_SYST_T data type is defined in the header file MCLIB_types.h.</td>
</tr>
</tbody>
</table>

Table 3-6. User Type Definitions

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T</td>
<td>f16Alpha</td>
<td>In</td>
<td>SF16</td>
<td>0x8000... 0x7FFF</td>
<td>Alpha component</td>
</tr>
<tr>
<td></td>
<td>f16Beta</td>
<td>In</td>
<td>SF16</td>
<td>0x8000... 0x7FFF</td>
<td>Beta component</td>
</tr>
<tr>
<td>MCLIB_3_COOR_SYST_T</td>
<td>f16A</td>
<td>Out</td>
<td>SF16</td>
<td>0x8000... 0x7FFF</td>
<td>A component</td>
</tr>
<tr>
<td></td>
<td>f16B</td>
<td>Out</td>
<td>SF16</td>
<td>0x8000... 0x7FFF</td>
<td>B component</td>
</tr>
<tr>
<td></td>
<td>f16C</td>
<td>Out</td>
<td>SF16</td>
<td>0x8000... 0x7FFF</td>
<td>C component</td>
</tr>
</tbody>
</table>

3.3.4 Availability

This library module is available in the C-callable interface assembly format.
This library module is targeted for the 56800E and 56800Ex platforms.
3.3.5 Dependencies

List of all dependent files:

- MCLIB_CPTrfAsm.h
- MCLIB_types.h

3.3.6 Description

The MCLIB_ClarkTrfInv function calculates the inverse Clarke transformation, which transforms values (flux, voltage, current) from the alpha-beta rotating orthogonal coordination system to the three-phase rotating coordination system, according to these equations:

\[ a = \alpha \]  \hspace{1cm}  \text{Eqn. 3-3}

\[ b = -0.5 \times \alpha + \frac{\sqrt{3}}{2} \times \beta \]  \hspace{1cm}  \text{Eqn. 3-4}

\[ c = -(a + b) \]  \hspace{1cm}  \text{Eqn. 3-5}

3.3.7 Range Issues

This function works with the 16-bit signed fractional values in the range \((-1, 1)\).

3.3.8 Special Issues

The function MCLIB_ClarkTrfInv is the saturation mode independent.

3.3.9 Implementation

Example 3-2. Implementation Code

```c
#include "mclib.h"

static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Inverse Clark Transformation calculation */
```
3.3.10 See Also

See MCLIB_ClarkTrf for more information.

3.3.11 Performance

Table 3-7. Performance of the MCLIB_ClarkTrfInv Function

<table>
<thead>
<tr>
<th>Code Size (bytes)</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (bytes)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td>Min.</td>
<td>29 cycles</td>
</tr>
<tr>
<td>Max.</td>
<td>29 cycles</td>
</tr>
</tbody>
</table>
3.4 MCLIB_ParkTrf

This function calculates the Park transformation algorithm.

3.4.1 Synopsis

```c
#include "mclib.h"
void MCLIB_ParkTrf(MCLIB_2_COOR_SYST_D_Q_T *pudtDQ,
                   MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta, MCLIB_ANGLE_T
                   *pudtSinCos)
```

3.4.2 Prototype

```asm
asm void MCLIB_ParkTrfFAsm(MCLIB_2_COOR_SYST_D_Q_T *pudtDQ,
                           MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta, MCLIB_ANGLE_T
                           *pudtSinCos)
```

3.4.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtDQ</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of the dq coordinate of a two-phase stationary orthogonal system.</td>
</tr>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a two-phase rotating orthogonal system.</td>
</tr>
<tr>
<td>*pudtSinCos</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure, where the values of sine and cosine are stored.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_2_COOR_SYST_D_Q_T</td>
<td>f16D</td>
<td>Out</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>d component</td>
</tr>
<tr>
<td></td>
<td>f16Q</td>
<td>Out</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>q component</td>
</tr>
<tr>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T</td>
<td>f16Alpha</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>Alpha component</td>
</tr>
<tr>
<td></td>
<td>f16Beta</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>Beta component</td>
</tr>
<tr>
<td>MCLIB_ANGLE_T</td>
<td>f16Sin</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>sine component of the angle</td>
</tr>
<tr>
<td></td>
<td>f16Cos</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>cosine component of the angle</td>
</tr>
</tbody>
</table>
3.4.4 Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.

3.4.5 Dependencies

List of all dependent files:
- MCLIB_CPTrfAsm.h
- MCLIB_types.h

3.4.6 Description

The **MCLIB_ParkTrf** function calculates the Park transformation, which transforms values (flux, voltage, current) from the alpha-beta rotating orthogonal coordinate system to the d-q stationary orthogonal coordinate system, according to these equations:

\[
\begin{align*}
    d &= \alpha \times \cos(\theta) + \beta \times \sin(\theta) \quad \text{Eqn. 3-6} \\
    q &= \beta \times \cos(\theta) - \alpha \times \sin(\theta) \quad \text{Eqn. 3-7}
\end{align*}
\]

3.4.7 Range Issues

This function works with the 16-bit signed fractional values in the range \((-1, 1)\).

3.4.8 Special Issues

The function **MCLIB_ParkTrf** is the saturation mode independent.

3.4.9 Implementation

**Example 3-3. Implementation Code**

```c
#include "mclib.h"

static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_2_COOR_SYST_D_Q_T mudtDQ;
static MCLIB_ANGLE_T mudtAngle;
void Isr(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;

    /* Angle structure initialization */
    mudtAngle.f16Sin = 0;

    // Motor Control Library, Rev. 0
```

```c
}
```
mudtAngle.f16Cos = FRAC16(1.0);
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Park Transformation calculation */
    MCLIB_ParkTrf(&mudtDQ, &mudtAlphaBeta, &mudtAngle);
}

3.4.10 See Also

See MCLIB_ParkTrfInv for more information.

3.4.11 Performance

Table 3-10. Performance of the MCLIB_ParkTrf Function

<table>
<thead>
<tr>
<th>Code Size (bytes)</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (bytes)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td>Min. 25 cycles</td>
</tr>
<tr>
<td></td>
<td>Max. 25 cycles</td>
</tr>
</tbody>
</table>
3.5 MCLIB_ParkTrfInv

This function calculates the inverse Park transformation algorithm.

3.5.1 Synopsis

```c
#include "mclib.h"
void MCLIB_ParkTrfInv(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
MCLIB_2_COOR_SYST_D_Q_T *pudtDQ, MCLIB_ANGLE_T *pudtSinCos)
```

3.5.2 Prototype

```c
asm void MCLIB_ParkTrfInvFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
MCLIB_2_COOR_SYST_D_Q_T *pudtDQ, MCLIB_ANGLE_T *pudtSinCos)
```

3.5.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a two-phase rotating orthogonal system.</td>
</tr>
<tr>
<td>*pudtDQ</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the data of a d-q coordinate two-phase stationary orthogonal system.</td>
</tr>
<tr>
<td>*pudtSinCos</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure, where the values of sine and cosine are stored.</td>
</tr>
</tbody>
</table>

3.5.4 Availability

This library module is available in the C-callable interface assembly format.

Motor Control Library, Rev. 0
This library module is targeted for the 56800E and 56800Ex platforms.

### 3.5.5 Dependencies

List of all dependent files:
- MCLIB_CPTrfAsm.h
- MCLIB_types.h

### 3.5.6 Description

The MCLIB_ParkTrfInv function calculates the inverse Park transformation, which transforms values (flux, voltage, current) from the d-q stationary orthogonal coordinate system to the alpha-beta rotating orthogonal coordinate system, according to these equations:

\[
\begin{align*}
\alpha &= d \times \cos(\theta) - q \times \sin(\theta) \\
\beta &= d \times \sin(\theta) + q \times \cos(\theta)
\end{align*}
\]

**Eqn. 3-8**

**Eqn. 3-9**

### 3.5.7 Range Issues

This function works with the 16-bit signed fractional values in the range \(-1, 1\).

### 3.5.8 Special Issues

The function MCLIB_ParkTrfInv is saturation mode independent.

### 3.5.9 Implementation

**Example 3-4. Implementation Code**

```c
#include "mclib.h"

static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_2_COOR_SYST_D_Q_T mudtDQ;
static MCLIB_ANGLE_T mudtAngle;
void Isr(void);

void main(void)
{
    /* D, Q structure initialization */
    mudtDQ.f16D = 0;
    mudtDQ.f16Q = 0;

    /* Angle structure initialization */
    mudtAngle.f16Sin = 0;
    mudtAngle.f16Cos = FRAC16(1.0);
}

/* Periodical function or interrupt */
```

Motor Control Library, Rev. 0
void Isr(void)
{
   /* Inverse Park Transformation calculation */
   MCLIB_ParkTrfInv(&mudtAlphaBeta, &mudtDQ, &mudtAngle);
}

3.5.10 See Also

See MCLIB_ParkTrf for more information.

3.5.11 Performance

Table 3-13. Performance of the MCLIB_ParkTrfInv Function

<table>
<thead>
<tr>
<th>Code Size (bytes)</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (bytes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Clock</td>
<td>Min.</td>
</tr>
<tr>
<td></td>
<td>Max.</td>
</tr>
</tbody>
</table>
3.6 MCLIB_SvmStd

This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator-reference voltage vector using a special standard space vector modulation technique.

3.6.1 Synopsis

```c
#include "mclib.h"
UWord16 MCLIB_SvmStd(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.6.2 Prototype

```c
asm UWord16 MCLIB_SvmStdFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.6.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator voltage vector.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing calculated duty-cycle ratios of the three-phase system.</td>
</tr>
</tbody>
</table>

3.6.4 Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.
3.6.5 Dependencies

List of all dependent files:
- MCLIB_SvmAsm.h
- MCLIB_types.h

3.6.6 Description

The MCLIB_SvmStd function for calculating duty-cycle ratios is widely-used in modern electric drives. This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator-reference voltage vector using a special space vector modulation technique, termed standard space vector modulation.

The basic principle of the standard space vector modulation technique can be explained with the help of the power stage diagram shown in Figure 3-1.

![Power Stage Schematic Diagram](image)

*Figure 3-1. Power Stage Schematic Diagram*

Top and bottom switches are working in a complementary mode; for example if the top switch $S_{At}$ is on, then the corresponding bottom switch $S_{Ab}$ is off and vice versa. Considering that the value one is assigned to the on state of the top switch, and value zero is assigned to the on state of the bottom switch, the switching vector $[a, b, c]^{T}$ can be defined. Creating such a vector allows numerical definition of all possible switching states. Phase-to-phase voltages can be then expressed in terms of these states:
where $U_{DCBus}$ is the instantaneous voltage measured on the DC-bus.

Assuming that the motor is ideally symmetrical, it’s possible to write a matrix equation that expresses the motor phase voltages shown in Equation 3-10.

$$
\begin{bmatrix}
U_{a} \\
U_{b} \\
U_{c}
\end{bmatrix}
= \frac{U_{DCBus}}{3}
\begin{bmatrix}
2 & -1 & -1 \\
-1 & 2 & -1 \\
-1 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c
\end{bmatrix}
\tag{Eqn. 3-11}
$$

In a three-phase power stage configuration (as shown in Figure 3-1), eight possible switching states (detailed in Figure 3-2) are feasible. These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in Table 3-16.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$U_a$</th>
<th>$U_b$</th>
<th>$U_c$</th>
<th>$U_{AB}$</th>
<th>$U_{BC}$</th>
<th>$U_{CA}$</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-U_{DCBus}</td>
<td>-U_{DCBus}</td>
<td>O_{000}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2U_{DCBus}/3</td>
<td>-U_{DCBus}/3</td>
<td>-U_{DCBus}/3</td>
<td>U_{DCBus}</td>
<td>0</td>
<td>0</td>
<td>U_{0}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>U_{DCBus}/3</td>
<td>U_{DCBus}/3</td>
<td>-2U_{DCBus}/3</td>
<td>0</td>
<td>U_{DCBus}</td>
<td>-U_{DCBus}</td>
<td>U_{60}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-U_{DCBus}/3</td>
<td>2U_{DCBus}/3</td>
<td>-U_{DCBus}/3</td>
<td>-U_{DCBus}</td>
<td>U_{DCBus}</td>
<td>0</td>
<td>U_{120}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-2U_{DCBus}/3</td>
<td>U_{DCBus}/3</td>
<td>U_{DCBus}/3</td>
<td>-U_{DCBus}</td>
<td>0</td>
<td>U_{DCBus}</td>
<td>U_{240}</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-U_{DCBus}/3</td>
<td>-U_{DCBus}/3</td>
<td>2U_{DCBus}/3</td>
<td>0</td>
<td>-U_{DCBus}</td>
<td>U_{DCBus}</td>
<td>U_{300}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>U_{DCBus}/3</td>
<td>-2U_{DCBus}/3</td>
<td>U_{DCBus}/3</td>
<td>U_{DCBus}</td>
<td>-U_{DCBus}</td>
<td>0</td>
<td>U_{360}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>O_{111}</td>
</tr>
</tbody>
</table>

The quantities of the direct-$\alpha$ and the quadrature-$\beta$ components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke transformation, arranged in a matrix form.

$$
\begin{bmatrix}
U_{\alpha} \\
U_{\beta}
\end{bmatrix}
= \frac{2}{3}
\begin{bmatrix}
1 & -1 & -1 \\
2 & 2 & 2
\end{bmatrix}
\begin{bmatrix}
U_{a} \\
U_{b} \\
U_{c}
\end{bmatrix}
\tag{Eqn. 3-12}
$$

The three-phase stator voltages, $U_a$, $U_b$, and $U_c$, are transformed using the Clarke transformation into the direct-$\alpha$ and the quadrature-$\beta$ components of the
two-phase orthogonal coordinate system. The transformation results are listed in Table 3-17.

Table 3-17. Switching Patterns and Space Vectors

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$U_\alpha$</th>
<th>$U_\beta$</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>O000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2$U_{DCBus}/3$</td>
<td>0</td>
<td>U0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$U_{DCBus}/3$</td>
<td>$U_{DCBus}/\sqrt{3}$</td>
<td>U60</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$-U_{DCBus}/3$</td>
<td>$U_{DCBus}/\sqrt{3}$</td>
<td>U120</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$-2U_{DCBus}/3$</td>
<td>0</td>
<td>U240</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$-U_{DCBus}/3$</td>
<td>$-U_{DCBus}/\sqrt{3}$</td>
<td>U300</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$U_{DCBus}/3$</td>
<td>$-U_{DCBus}/\sqrt{3}$</td>
<td>U360</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>O111</td>
</tr>
</tbody>
</table>

Figure 3-2 graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors, $U_0$, $U_{60}$, $U_{120}$, $U_{180}$, $U_{240}$, $U_{300}$, and two zero vectors, $O_{111}$, $O_{000}$, usable for switching. Therefore, the principle of the standard space vector modulation resides in applying the appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

Referring to that principle, an objective of the standard space vector modulation is an approximation of the reference stator voltage vector $U_S$ with an appropriate combination of the switching patterns, composed of basic space vectors. The graphical explanation of this objective is shown in Figure 3-3 and Figure 3-4.
The stator-reference voltage vector $U_S$ is phase-advanced by $30^\circ$ from the direct-$\alpha$, and thus might be generated with an appropriate combination of the adjacent basic switching states $U_0$ and $U_{60}$. These figures also indicate resultant direct-$\alpha$ and quadrature-$\beta$ components for space vectors $U_0$ and $U_{60}$.

In this case, the reference-stator voltage vector $U_S$ is located in sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states $U_{60}$ and $U_0$. The principal equations concerning this vector location are:

$$
\begin{align*}
U_{60} &= (110) \\
&= (1/\sqrt{3},1) \\
U_0 &= (100) \\
&= (2/\sqrt{3},0) \\
T_{60}/T^*U_{60} &= 60 \text{ degrees} \\
T_{0}/T^*U_0 &= 30 \text{ degrees}
\end{align*}
$$
where $T_{60}$ and $T_0$ are the respective duty-cycle ratios, for which the basic space vectors $U_{60}$ and $U_0$ should be applied within the time period $T$. $T_{null}$ is the course of time, for which the null vectors $O_{000}$ and $O_{111}$ are applied. Those duty-cycle ratios can be calculated using the following equations:

$$u_\beta = \frac{T_{60}}{T} \times |U_{60}| \times \sin 60^\circ$$  \hspace{1cm} \text{Eqn. 3-15}

$$u_\alpha = \frac{T_0}{T} \times |U_0| + \frac{u_\beta}{\tan 60^\circ}$$  \hspace{1cm} \text{Eqn. 3-16}

Considering that normalized magnitudes of basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{3}$, and by substitution of the trigonometric expressions $\sin 60^\circ$ and $\tan 60^\circ$ by their quantities $2/\sqrt{3}$ and $\sqrt{3}$, respectively, Equation 3-15 and Equation 3-16 can be rearranged for the unknown duty-cycle ratios $T_{60}/T$ and $T_0/T$ as follows:

$$T_{60} = u_\beta$$  \hspace{1cm} \text{Eqn. 3-17}

$$U_S = \frac{T_{120}}{T} \times U_{120} + \frac{T_{60}}{T} \times U_{60}$$  \hspace{1cm} \text{Eqn. 3-18}

Sector II is depicted in Figure 3-5. In this particular case, the reference-stator voltage vector $U_S$ is generated by the appropriate duty-cycle ratios of the basic switching states $U_{60}$ and $U_{120}$. The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{null}$$  \hspace{1cm} \text{Eqn. 3-19}

$$U_S = \frac{T_{120}}{T} \times U_{120} + \frac{T_{60}}{T} \times U_{60}$$  \hspace{1cm} \text{Eqn. 3-20}

where $T_{120}$ and $T_{60}$ are the respective duty-cycle ratios, for which the basic space vectors $U_{120}$ and $U_{60}$ should be applied within the time period $T$. $T_{null}$ is the course of time, for which the null vectors $O_{000}$ and $O_{111}$ are applied. These resultant duty-cycle ratios are formed from the auxiliary components termed A and B. The graphical representation of the auxiliary components is shown in Figure 3-6.
The equations describing those auxiliary time-duration components are:

\[
\frac{\sin 30^\circ}{\sin 120^\circ} = \frac{A}{u_\beta}
\]

Eqn. 3-21

\[
\frac{\sin 60^\circ}{\sin 60^\circ} = \frac{B}{u_\alpha}
\]

Eqn. 3-22

Equation 3-21 and Equation 3-22 have been formed using the sine rule.
These equations can be rearranged for the calculation of the auxiliary
time-duration components A and B. This is done simply by substituting the
trigonometric terms \( \sin 30^\circ \), \( \sin 120^\circ \), and \( \sin 60^\circ \), by their numerical
representations \( 1/2 \), \( \sqrt{3}/2 \), and \( 1/\sqrt{3} \), respectively.

\[
A = \frac{1}{\sqrt{3}} \times u_\beta \\
B = u_\alpha 
\]

\textit{Eqn. 3-23}  \\
\textit{Eqn. 3-24}

The resultant duty-cycle ratios, \( T_{120}/T \) and \( T_{60}/T \), are then expressed in terms of
the auxiliary time-duration components defined by \textit{Equation 3-25} and
\textit{Equation 3-26} as follows:

\[
\frac{T_{120}}{T} \times |U_{120}| = (A - B) \\
\frac{T_{60}}{T} \times |U_{60}| = (A + B)
\]

\textit{Eqn. 3-25}  \\
\textit{Eqn. 3-26}

With the help of these equations, and also considering the normalized magnitudes
of the basic space vectors to be \( |U_{120}| = |U_{60}| = 2/\sqrt{3} \), the equations expressed for
the unknown duty-cycle ratios of basic space vectors \( T_{120}/T \) and \( T_{60}/T \) can be
written as follows:

\[
\frac{T_{120}}{T} = \frac{1}{2} \times (u_\beta - \sqrt{3} \times u_\alpha) \\
\frac{T_{60}}{T} = \frac{1}{2} \times (u_\beta + \sqrt{3} \times u_\alpha)
\]

\textit{Eqn. 3-27}  \\
\textit{Eqn. 3-28}

The duty-cycle ratios in the remaining sectors can be derived using the same
approach. The resulting equations will be similar to those derived for sector I and
sector II.

To depict the duty-cycle ratios of the basic space vectors for all sectors, we
define:

- Three auxiliary variables:
  
  \[
  X = u_\beta \\
  Y = \frac{1}{2} \times (u_\beta + \sqrt{3} \times u_\alpha) \\
  Z = \frac{1}{2} \times (u_\beta - \sqrt{3} \times u_\alpha)
  \]
  
- Two expressions:
  
  \[
  t_1 \\
  t_2
  \]
which generally represent the duty-cycle ratios of the basic space vectors in the respective sector; for example, for the first sector, \( t_1 \) and \( t_2 \) represent duty-cycle ratios of the basic space vectors \( U_{60} \) and \( U_0 \); for the second sector, \( t_1 \) and \( t_2 \) represent duty-cycle ratios of the basic space vectors \( U_{120} \) and \( U_{60} \), and so on.

For each sector, the expressions \( t_1 \) and \( t_2 \), in terms of auxiliary variables \( X \), \( Y \), and \( Z \), are listed in Table 3-18.

<table>
<thead>
<tr>
<th>Sectors</th>
<th>( U_{0,60} )</th>
<th>( U_{60,120} )</th>
<th>( U_{120,180} )</th>
<th>( U_{180,240} )</th>
<th>( U_{240,300} )</th>
<th>( U_{300,0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( X )</td>
<td>( Z )</td>
<td>( -X )</td>
<td>( Z )</td>
<td>( -Z )</td>
<td>( Y )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( -Z )</td>
<td>( Y )</td>
<td>( Z )</td>
<td>( -X )</td>
<td>( -Y )</td>
<td>( -X )</td>
</tr>
</tbody>
</table>

For the determination of auxiliary variables \( X \), \( Y \), and \( Z \), the sector number is required. This information can be obtained through several approaches. One approach discussed here requires the use of modified inverse Clarke transformation to transform the direct-\( \alpha \) and quadrature-\( \beta \) components into a balanced three-phase quantity \( u_{ref1} \), \( u_{ref2} \), and \( u_{ref3} \), used for straightforward calculation of the sector number, to be shown later.

\[
\begin{align*}
 u_{ref1} &= u_\beta \\
 u_{ref2} &= \frac{-u_\beta + \sqrt{3} \times u_\alpha}{2} \\
 u_{ref3} &= \frac{-u_\beta - \sqrt{3} \times u_\alpha}{2}
\end{align*}
\]

The modified inverse Clarke transformation projects the quadrature-\( u_\beta \) component into \( u_{ref1} \), as shown in Figure 3-7 and Figure 3-8, whereas voltages generated by the conventional inverse Clarke transformation project the direct-\( u_\alpha \) component into \( u_{ref1} \).

![Components of the Stator Reference Voltage Vector](image-url)
Figure 3-7 depicts the direct-\(u_\alpha\) and quadrature-\(u_\beta\) components of the stator reference voltage vector \(U_S\) that were calculated by the equations \(u_\alpha = \cos \theta\) and \(u_\beta = \sin \theta\), respectively.

The sector identification tree, shown in Figure 3-9, can be a numerical solution of the approach shown in Figure 3-8.

![Sinusoidal Three-Phase Reference Voltage](image)

**Figure 3-8. Reference Voltages Uref1, Uref2, and Uref3**

It should be pointed out that in the worst case three simple comparisons are required to precisely identify the sector of the stator-reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 3-3, the stator-reference voltage vector is phase-advanced by 30° from the direct \(\alpha\)-axis, which results in the positive quantities of \(u_{ref1}\) and \(u_{ref2}\) and the negative quantity of \(u_{ref3}\); refer to Figure 3-8. If these quantities are used as the inputs to the sector identification tree, the product of those comparisons will be sector I. Using the same approach identifies the sector II, if the stator-reference voltage vector is located according to the one shown in Figure 3-5. The variables \(t_1\), \(t_2\), and \(t_3\), representing switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

\[
t_1 = \frac{T - t_2}{2}
\]

**Eqn. 3-32**
where $T$ is the switching period, $t_1$ and $t_2$ are the duty-cycle ratios of the basic space vectors, given for the respective sector; see Table 3-18. Equation 3-12, Equation 3-33, and Equation 3-34, are specific solely to the standard space vector modulation technique; consequently, other space vector modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, $t_1$, $t_2$, and $t_3$, to the respective motor phases. This is a simple task, accomplished in a view of the position of the stator-reference voltage vector; see Table 3-19.

<table>
<thead>
<tr>
<th>Sectors</th>
<th>$U_{0,60}$</th>
<th>$U_{60,120}$</th>
<th>$U_{120,180}$</th>
<th>$U_{180,240}$</th>
<th>$U_{240,300}$</th>
<th>$U_{300,0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwm_a</td>
<td>$t_3$</td>
<td>$t_2$</td>
<td>$t_1$</td>
<td>$t_1$</td>
<td>$t_2$</td>
<td>$t_3$</td>
</tr>
<tr>
<td>pwm_b</td>
<td>$t_2$</td>
<td>$t_3$</td>
<td>$t_3$</td>
<td>$t_2$</td>
<td>$t_1$</td>
<td>$t_1$</td>
</tr>
<tr>
<td>pwm_c</td>
<td>$t_1$</td>
<td>$t_1$</td>
<td>$t_2$</td>
<td>$t_3$</td>
<td>$t_3$</td>
<td>$t_2$</td>
</tr>
</tbody>
</table>

The principle of the space vector modulation technique consists of applying the basic voltage vectors $U_{XXX}$ and $O_{XXX}$ for the certain time in such a way that the mean vector, generated by the pulse width modulation approach for the period $T$, is equal to the original stator-reference voltage vector $U_S$. This provides a great variability of the arrangement of the basic vectors during the PWM period $T$. Those vectors might be arranged either to lower the switching losses or to achieve diverse results, such as center-aligned PWM, edge-aligned PWM, or a minimal number of switching states. A brief discussion of the widely-used center-aligned PWM follows.

Generating the center-aligned PWM pattern is accomplished practically by comparing the threshold levels, $pwm_a$, $pwm_b$, and $pwm_c$, with a free-running up-down counter. The timer counts to 1 ($0x7FFF$), and then down to 0 ($0x0000$). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 3-10.
Figure 3-10. Standard Space Vector Modulation Technique — Center-Aligned PWM

Figure 3-11 graphically shows the calculated waveforms of duty-cycle ratios using standard space vector modulation.

3.6.7 Returns

This function returns an integer value representing the sector number, in which the instantaneous stator-reference voltage vector is located.

Motor Control Library, Rev. 0
### 3.6.8 Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-\( \alpha \) and quadrature-\( \beta \) components of the stator-reference voltage vector must be considered as SF16 fractional numbers with their magnitude within the unit circle; in other words, the assumption \( \sqrt{\alpha^2 + \beta^2} \leq 1 \) must be met.

### 3.6.9 Special Issues

The function **MCLIB_SvmStd** is intended for periodical use; i.e., it might be called from a timer interrupt or a PWM update interrupt.

The function **MCLIB_SvmStd** requires the saturation mode to be SET OFF!

### 3.6.10 Implementation

The **MCLIB_SvmStd** function is implemented as a function call.

#### Example 3-5. Implementation Code

```c
#include "mclib.h"

static UWord16 muw16Sector;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* SVM calculation */
    muw16Sector = MCLIB_SvmStd(&mudtAlphaBeta, &mudtAbc);
}
```

### 3.6.11 See Also

See **MCLIB_SvmU0n**, **MCLIB_SvmU7n**, **MCLIB_SvmAlt**, **MCLIB_SvmSci** and **MCLIB_PwmIct** for more information.
### 3.6.12 Performance

Table 3-20. Performance of the MCLIB\_SvmStd Function

<table>
<thead>
<tr>
<th>Code Size (words)</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (words)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td>Min.</td>
<td>90 cycles</td>
</tr>
<tr>
<td>Max.</td>
<td>118 cycles</td>
</tr>
</tbody>
</table>
3.7 MCLIB_SvmU0n

This function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses the special Space Vector Modulation technique, termed Space Vector Modulation with O000 Nulls.

3.7.1 Synopsis

    #include "mclib.h"
    UWord16 MCLIB_SvmU0n(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
                         MCLIB_3_COOR_SYST_T *pudtAbc)

3.7.2 Prototype

    asm UWord16 MCLIB_SvmU0nFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
                                   *pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)

3.7.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator voltage vector.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing calculated duty-cycle ratios of the three-phase system.</td>
</tr>
</tbody>
</table>

3.7.4 Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.
3.7.5 Dependencies

List of all dependent files:

- MCLIB_SvmAsm.h
- MCLIB_types.h

3.7.6 Description

The MCLIB_SvmU0n function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses a special Space Vector Modulation technique, termed Space Vector Modulation with O000 Nulls.

The derivation approach of the Space Vector Modulation technique with O000 Nulls is identical, in many aspects, to the approach presented in Section 3.6, “MCLIB_SvmStd. However, a distinct difference lies in the definition of the variables t1, t2 and t3 that represent switching duty-cycle ratios of the respective phases:

\[ t_1 = 0 \]  \hspace{1cm} Eqn. 3-35

\[ t_2 = t_1 + t_{-1} \]  \hspace{1cm} Eqn. 3-36

\[ t_3 = t_2 + t_{-2} \]  \hspace{1cm} Eqn. 3-37

where T is the switching period and t_{-1} and t_{-2} are duty-cycle ratios of basic space vectors that are defined for the respective sector in Table 3-18.

The generally-used center-aligned PWM is discussed briefly in the following sections. Generating the center-aligned PWM pattern is accomplished practically by comparing threshold levels pwm_a, pwm_b, and pwm_c with the free-running up-down counter. The timer counts up to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that, when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 3-12.
Figure 3-12. Space Vector Modulation Technique with O000 Nulls – Center-Aligned PWM

Figure 3-13 shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with O000 Nulls.

3.7.7 Returns

The function returns an integer value representing the sector number in which the instantaneous stator reference voltage vector is located.
3.7.8 Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-a and quadrature-b components of the stator reference voltage vector must be considered as Q15 fractional numbers with their magnitude within the unit circle; i.e., the assumption \( \sqrt{\alpha^2 + \beta^2} \leq 1 \) must be met.

3.7.9 Special Issues

The function \texttt{MCLIB\_SvmU0n} is intended for periodical use; i.e., it might be called from a timer interrupt or a PWM updates interrupt. Referring to that, this function was programmed using the assembler language with emphasis on maximizing the computational speed.

The function \texttt{MCLIB\_SvmU0n} requires the saturation mode to be SET OFF!

3.7.10 Implementation

The \texttt{MCLIB\_SvmU0n} function is implemented as a function call.

\begin{verbatim}
#include "mclib.h"

static UWord16 muw16Sector;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* SVM calculation */
    muw16Sector = MCLIB\_SvmU0n(&mudtAlphaBeta, &mudtAbc);
}
\end{verbatim}

3.7.11 See Also

See \texttt{MCLIB\_SvmStd}, \texttt{MCLIB\_SvmU7n}, \texttt{MCLIB\_SvmAlt}, \texttt{MCLIB\_SvmSci} and \texttt{MCLIB\_PwmIct} for more information.
### 3.7.12 Performance

#### Table 3-23. Performance of `MCLIB_SvmU0n` Function

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Size (words)</strong></td>
<td>99</td>
</tr>
<tr>
<td><strong>Data Size (words)</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Execution Clock</strong></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>83 cycles</td>
</tr>
<tr>
<td>Max</td>
<td>112 cycles</td>
</tr>
</tbody>
</table>
3.8 MCLIB_SvmU7n

This function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses a special Space Vector Modulation technique, termed Space Vector Modulation with O_111 Nulls.

3.8.1 Synopsis

```c
#include "mclib.h"
UWord16  MCLIB_SvmU7n(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.8.2 Prototype

```c
asm UWord16 MCLIB_SvmU7nFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.8.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator voltage vector.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing calculated duty-cycle ratios of the three-phase system.</td>
</tr>
</tbody>
</table>

3.8.4 Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.
3.8.5 Dependencies

List of all dependent files:
  • MCLIB_SvmAsm.h
  • MCLIB_types.h

3.8.6 Description

The **MCLIB_SvmU7n** function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses the special Space Vector Modulation technique, termed Space Vector Modulation with O₁₁₁ Nulls.

The derivation approach of the Space Vector Modulation technique with O₁₁₁ Nulls is identical, in many aspects, to the approach presented in Section 3.6, “MCLIB_SvmStd”. However, a distinct difference lies in the definition of the variables \( t_1 \), \( t_2 \) and \( t_3 \) that represent the switching duty-cycle ratios of the respective phases:

\[
\begin{align*}
  t_1 &= T - t_1 - t_2 \quad \text{Eqn. 3-38} \\
  t_2 &= t_1 + t_1 \quad \text{Eqn. 3-39} \\
  t_3 &= t_2 + t_2 \quad \text{Eqn. 3-40}
\end{align*}
\]

where \( T \) is the switching period, and \( t_1 \) and \( t_2 \) are the duty-cycles ratios of the space vectors that are defined for the respective sector in Table 3-18.

Generating the center-aligned PWM pattern is accomplished practically by comparing the threshold levels \( \text{pwm}_a \), \( \text{pwm}_b \), and \( \text{pwm}_c \) with the free-running up-down counter. The timer counts up to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 3-14.
Figure 3-14 graphically shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with O111 Nulls.

Figure 3-14. Space Vector Modulation Technique with O111 Nulls - Center-Aligned PWM

Figure 3-15. Space Vector Modulation with O111 Nulls
3.8.7 Returns

The function returns an integer value representing the sector number in which the instantaneous stator reference voltage vector is located.

3.8.8 Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-a and quadrature-b components of the stator reference voltage vector must be considered as Q15 fractional numbers with their magnitude within the unit circle; i.e., the assumption $\sqrt{\alpha^2 + \beta^2} \leq 1$ must be met.

3.8.9 Special Issues

The function MCLIB_SvmU7n is intended for the periodical use; i.e., it might be called from a timer interrupt or a PWM update interrupt. Referring to that, this function was programmed using the assembler language with emphasis on maximizing the computational speed.

The function MCLIB_SvmU7n requires the saturation mode to be SET OFF!

3.8.10 Implementation

The MCLIB_SvmU7n is implemented as a function call.

**Example 3-7. Implementation Code**

```c
#include "mclib.h"

static UWord16 muw16Sector;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* SVM calculation */
    muw16Sector = MCLIB_SvmU7n(&mudtAlphaBeta, &mudtAbc);
}
```
3.8.11 See Also

See MCLIB_SvmStd, MCLIB_SvmU0n, MCLIB_SvmAlt, MCLIB_SvmSci and MCLIB_PwmIct for more information.

3.8.12 Performance

Table 3-26. Performance of MCLIB_SvmU7n function

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size (words)</td>
<td>107</td>
</tr>
<tr>
<td>Data Size (words)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td></td>
<td>Max</td>
</tr>
</tbody>
</table>
3.9 MCLIB_SvmAlt

This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Alternating State Null Vector Space Vector Modulation technique.

3.9.1 Synopsis

```c
#include "mclib.h"
UWord16 MCLIB_SvmAlt(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.9.2 Prototype

```c
asm UWord16 MCLIB_SvmAltFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.9.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator voltage vector.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing calculated duty-cycle ratios of the three-phase system.</td>
</tr>
</tbody>
</table>

3.9.4 Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.
3.9.5 Dependencies

List of all dependent files:

- MCLIB_SvmAsm.h
- MCLIB_types.h

3.9.6 Description

The **MCLIB_SvmAlt** function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector. It uses a special Space Vector Modulation technique, termed Space Vector Modulation with states O000 in the even sectors and state O111 in the odd sectors.

The derivation approach of this Space Vector Modulation technique is identical, in many aspects, to the approach presented in Section 3.6, “MCLIB_SvmStd”. However, a distinct difference lies in the definition of the variables t1, t2 and t3 that represent the switching duty-cycle ratios of the respective phases. These variables are given for the even sectors (2, 4, 6) by the same equations as those defined in Section 3.7, “MCLIB_SvmU0n”.

\[
\begin{align*}
t_1 &= 0 & \text{Eqn. 3-41} \\
t_2 &= t_1 + t_1 & \text{Eqn. 3-42} \\
t_3 &= t_2 + t_2 & \text{Eqn. 3-43}
\end{align*}
\]

For the odd sectors (1, 3, 5), these variables are given by equations that are identical to those defined in Section 3.8, “MCLIB_SvmU7n”.

\[
\begin{align*}
t_1 &= T - t_1 - t_2 & \text{Eqn. 3-44} \\
t_2 &= t_1 + t_1 & \text{Eqn. 3-45} \\
t_3 &= t_2 + t_2 & \text{Eqn. 3-46}
\end{align*}
\]

where T is the switching period, t_1 and t_2 are duty-cycle ratios of the space vectors, which are defined for the respective sector in Table 3-18. Generating the center-aligned PWM pattern is accomplished practically by comparing the threshold levels pwm_a, pwm_b, and pwm_c with a free-running up-down counter. This timer counts up to 1 (0x7FFF) and then down to 0 (0x0000). When a threshold level is larger than the counter value, the respective PWM output is active. Otherwise, it is inactive; see Figure 3-16.
Figure 3-16. Space Vector Modulation Technique with Alternate Nulls – Center-Aligned PWM

Figure 3-17 shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with states O_{000} in the even sectors and state O_{111} in the odd sectors.

Figure 3-17. Space Vector Modulation Technique with Alternate Nulls
3.9.7 Returns

The function returns an integer value representing the Sector number in which the instantaneous stator reference voltage vector is located.

3.9.8 Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-a and quadrature-b components of the stator reference voltage vector must be considered as Q15 fractional numbers with their magnitude within the unit circle; i.e., the assumption $\sqrt{\alpha^2 + \beta^2} \leq 1$ must be met.

3.9.9 Special Issues

The function `MCLIB_SvmAlt` is intended for the periodical use; i.e., it might be called from a timer interrupt or a PWM update interrupt. Referring to that, this function was programmed using the assembler language with emphasis on maximizing the computational speed.

The function `MCLIB_SvmAlt` requires the saturation mode to be SET OFF!

3.9.10 Implementation

The `MCLIB_SvmAlt` function is implemented as a function call

```c
#include "mclib.h"

static UWord16 muw16Sector;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* SVM calculation */
    muw16Sector = MCLIB_SvmAlt(&mudtAlphaBeta, &mudtAbc);
}
```

Motor Control Library, Rev. 0
3.9.11  See Also

See MCLIB_SvmStd, MCLIB_SvmU0n, MCLIB_SvmU7n, MCLIB_SvmSci and MCLIB_PwmIct for more information.

3.9.12  Performance

Table 3-29. Performance of MCLIB_SvmAlt function

<table>
<thead>
<tr>
<th>Code Size (words)</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (words)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>87 cycles</td>
</tr>
<tr>
<td>Max</td>
<td>116 cycles</td>
</tr>
</tbody>
</table>
3.11  MCLIB_PwmIct

This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using the General Sinusoidal Modulation technique.

3.11.1  Synopsis

```c
#include "mclib.h"
UWord16 MCLIB_PwmIct(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.11.2  Prototype

```c
asm UWord16 MCLIB_PwmIctFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)
```

3.11.3  Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>components of the stator voltage vector.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing calculated duty-cycle ratios of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>three-phase system.</td>
</tr>
</tbody>
</table>

3.11.4  Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.
3.11.5 Dependencies

List of all dependent files:
- MCLIB_SvmAsm.h
- MCLIB_types.h

3.11.6 Description

The MCLIB_PwmIct function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector with the help of the conventional Inverse Clark transformation.

Finding the sector in which the reference stator voltage vector $U_S$ resides is similar to that discussed in Section 3.6, “MCLIB_SvmStd“. This is achieved by first converting the direct-a and the quadrature-b components of the reference stator voltage vector $U_S$ into the balanced three-phase quantities $u_{ref1}$, $u_{ref2}$ and $u_{ref3}$, using the modified Inverse Clark Transform:

\[ u_{ref1} = u_\beta \]
\[ u_{ref2} = \frac{-u_\beta + \sqrt{3} \cdot u_a}{2} \]  
\[ u_{ref3} = \frac{-u_\beta - \sqrt{3} \cdot u_a}{2} \]

The calculation of the sector number is based on comparing the three-phase reference voltages $u_{ref1}$, $u_{ref2}$ and $u_{ref3}$ with zero. This computation can be described by the following set of rules:

\[ u_0 = \begin{cases} 
1.0 & \text{if } u_{ref1} > 0 \\
0 & \text{otherwise}
\end{cases} \quad \text{Eqn. 3-62} \]
\[ u_0 = \begin{cases} 
2.0 & \text{if } u_{ref2} > 0 \\
0 & \text{otherwise}
\end{cases} \quad \text{Eqn. 3-63} \]
\[ u_0 = \begin{cases} 
4.0 & \text{if } u_{ref3} > 0 \\
0 & \text{otherwise}
\end{cases} \quad \text{Eqn. 3-64} \]

After passing these rules, modified sector numbers are then derived from the formula sector * = a + b + c.
The sector numbers determined by this formula must be further transformed to correspond to those which would be determined by the Sector Identification Tree. The transformation, which meets this requirement, is shown in Table 3-36.

**Table 3-35. Transformation of the Sectors**

<table>
<thead>
<tr>
<th>Sector*</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sector</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

The Inverse Clark Transformation might be used for transforming values such as flux, voltage and current from an orthogonal coordination system \((u_a, u_b)\) to a 3-phase rotating coordination system \((u_a, u_b, u_c)\). The original equations of the Inverse Clark Transformation are scaled here to provide the duty-cycle ratios in the range \(0 < \text{pwm}_x < 1\), where \(x\) refers to the corresponding phases. These scaled duty-cycle ratios \(\text{pwm}_a\), \(\text{pwm}_b\) and \(\text{pwm}_c\) might be used directly by the registers of the PWM block.

\[
\text{pwm}_a = 0.5 + \frac{u_a}{2} \quad \text{Eqn. 3-65}
\]

\[
\text{pwm}_b = 0.5 + \frac{-u_a + \sqrt{3} \cdot u_b}{4} \quad \text{Eqn. 3-66}
\]

\[
\text{pwm}_c = 0.5 + \frac{-u_a - \sqrt{3} \cdot u_b}{4} \quad \text{Eqn. 3-67}
\]

Figure 3-19 shows calculated waveforms of duty cycle ratios using the Inverse Clark Transformation.
3.11.7  Returns

The function returns an integer value representing the sector number in which the instantaneous stator reference voltage vector is located.

3.11.8  Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-a and quadrature-b components of the stator reference voltage vector must be considered as Q15 fractional numbers with their magnitude within the unit circle; i.e., the assumption $\sqrt{\alpha^2 + \beta^2} \leq 1$ must be met.

3.11.9  Special Issues

The function MCLIB_PwmIct is intended for the periodical use; i.e., it might be called from a timer interrupt or a PWM update interrupt. Referring to that, this function was programmed using assembler language with emphasis on maximizing the computational speed. The function MCLIB_PwmIct requires the saturation mode to be SET OFF!

3.11.10 Implementation

The MCLIB_PwmIct function is implemented as a function call.
Example 3-10. Implementation Code

```
#include "mclib.h"

static UWord16 muw16Sector;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* SVM calculation */
    muw16Sector = MCLIB_PwmIct(&mudtAlphaBeta, &mudtAbc);
}
```

### 3.11.11 See Also

See MCLIB_SvmStd, MCLIB_SvmU0n, MCLIB_SvmU7n, MCLIB_SvmAlt and MCLIB_SvmSci for more information.

### 3.11.12 Performance

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size (words)</td>
<td>59</td>
</tr>
<tr>
<td>Data Size (words)</td>
<td>7</td>
</tr>
<tr>
<td>Execution Clock</td>
<td>Min 82 cycles</td>
</tr>
<tr>
<td></td>
<td>Max 82 cycles</td>
</tr>
</tbody>
</table>
3.10 MCLIB_SvmSci

This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using the General Sinusoidal Modulation with an injection of the third harmonic.

3.10.1 Synopsis

#include "mclib.h"

UWord16 MCLIB_SvmSci(MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta,
MCLIB_3_COOR_SYST_T *pudtAbc)

3.10.2 Prototype

asm UWord16 MCLIB_SvmSciFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtAlphaBeta, MCLIB_3_COOR_SYST_T *pudtAbc)

3.10.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator voltage vector.</td>
</tr>
<tr>
<td>*pudtAbc</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing calculated duty-cycle ratios of the three-phase system.</td>
</tr>
</tbody>
</table>

3.10.4 Availability

This library module is available in the C-callable interface assembly format.
This library module is targeted for the 56800E and 56800Ex platforms.
3.10.5 Dependencies

List of all dependent files:
- MCLIB_SvmAsm.h
- MCLIB_types.h

3.10.6 Description

The **MCLIB_SvmSci** function calculates the appropriate duty-cycle ratios, needed for generating the given stator reference voltage vector with the help of the sinusoidal modulation with Sine-Cap Injection algorithm.

Finding the sector in which the reference stator voltage vector $U_S$ resides is similar to that discussed in Section 3.6, “MCLIB_SvmStd“.

The balanced 3-Phase duty-cycle ratios may be calculated based on Sine Cap Injection algorithm in the following stages:

1. The calculation of the basic duty-cycle ratios using the Inverse Clarke Transformation.

   $$ u_a = u_\alpha $$  \hspace{1cm}  \text{Eqn. 3-47}

   $$ u_b = -u_\alpha + \frac{\sqrt{3} \cdot u_\beta}{2} $$  \hspace{1cm}  \text{Eqn. 3-48}

   $$ u_c = u_\alpha - \frac{\sqrt{3} \cdot u_\beta}{2} $$  \hspace{1cm}  \text{Eqn. 3-49}

2. An amplitude of the basic duty-cycle ratios $u_a$, $u_b$ and $u_c$ calculated by Equation 3-47, Equation 3-48 and Equation 3-49 is in the range $[-1, 1]$.

   The basic duty-cycle ratios are then multiplied by the coefficient $2/(\sqrt{3})$

   $$ u'_a = \frac{2}{\sqrt{3}} \cdot u_a $$  \hspace{1cm}  \text{Eqn. 3-50}

   $$ u'_b = \frac{2}{\sqrt{3}} \cdot u_b $$  \hspace{1cm}  \text{Eqn. 3-51}

   $$ u'_c = \frac{2}{\sqrt{3}} \cdot u_c $$  \hspace{1cm}  \text{Eqn. 3-52}

3. The values of these variables are within the range $-2/(\sqrt{3}) < u'x < 2/(\sqrt{3})$.

   Therefore, smart scaling of the fractional numbers must be utilized to provide fractional calculations with an adequate accuracy level. For more information about scaling, refer to the assembler source code of the described modulation function the in module `svm.c`. 

---

Motor Control Library, Rev. 0

Freescale Semiconductor
4. If the values of variables $u'_a$, $u'_b$, and $u'_c$ exceed the unity, they are stored in an auxiliary variable $u_0$. This variable is called the Sine Cap Voltage variable. The procedure to obtain this can be mathematically defined by a series of three formulas:

\[
\begin{align*}
  u_0 &= \begin{cases} 
    1.0 - u'_a & \text{if } u'_a > 1.0 \\
    -1.0 - u'_a & \text{if } u'_a < -1.0 \\
    0 & \text{otherwise}
  \end{cases} \quad \text{Eqn. 3-53} \\
  u_0 &= \begin{cases} 
    1.0 - u'_b & \text{if } u'_b > 1.0 \\
    -1.0 - u'_b & \text{if } u'_b < -1.0 \\
    0 & \text{otherwise}
  \end{cases} \quad \text{Eqn. 3-54} \\
  u_0 &= \begin{cases} 
    1.0 - u'_c & \text{if } u'_c > 1.0 \\
    -1.0 - u'_c & \text{if } u'_c < -1.0 \\
    0 & \text{otherwise}
  \end{cases} \quad \text{Eqn. 3-55}
\end{align*}
\]

5. Due to the 120° voltage phase shift, distinguishing for the balanced three-phase system, only one phase contributes to the building of Sine-Cap Voltage $u_0$ at each time point.

6. Final duty-cycle ratios are then calculated by the following equations:

\[
\begin{align*}
  \text{pwm}_a &= \frac{1}{2} \cdot (u_0 + u'_a + 1) \quad \text{Eqn. 3-56} \\
  \text{pwm}_b &= \frac{1}{2} \cdot (u_0 + u'_b + 1) \quad \text{Eqn. 3-57} \\
  \text{pwm}_c &= \frac{1}{2} \cdot (u_0 + u'_c + 1) \quad \text{Eqn. 3-58}
\end{align*}
\]

Figure 3-18 shows calculated waveforms of the duty cycle ratios using the sinusoidal modulation with Sine-Cap Injection algorithm.
3.10.7 Returns

The function returns an integer value representing the sector number in which the instantaneous stator reference voltage vector is located.

3.10.8 Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-a and quadrature-b components of the stator reference voltage vector must be considered as Q15 fractional numbers with their magnitude within the unit circle; i.e., the assumption $\sqrt{\alpha^2 + \beta^2} \leq 1$ must be met.

3.10.9 Special Issues

The MCLIB_SvmSci function function is intended for the periodical use; i.e., it might be called from a timer interrupt or a PWM update interrupt. Referring to that, this function was programmed using the assembler language with emphasis on maximizing the computational speed.

The MCLIB_SvmSci function requires the saturation mode to be SET OFF!
3.10.10 Implementation

The **MCLIB_SvmSci** function is implemented as a function call.

### Example 3-9. Implementation Code

```c
#include "mclib.h"

static UWord16 muw16Sector;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtAlphaBeta;
static MCLIB_3_COOR_SYST_T mudtAbc;

void Isr(void);

void main(void)
{
    /* Alpha, Beta structure initialization */
    mudtAlphaBeta.f16Alpha = 0;
    mudtAlphaBeta.f16Beta = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* SVM calculation */
    muw16Sector = MCLIB_SvmSci(&mudtAlphaBeta, &mudtAbc);
}
```

3.10.11 See Also

See **MCLIB_SvmStd**, **MCLIB_SvmU0n**, **MCLIB_SvmU7n**, **MCLIB_SvmAlt** and **MCLIB_PwmIct** for more information.

3.10.12 Performance

### Table 3-32. Performance of MCLIB_SvmSci function

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size (words)</td>
<td>130</td>
</tr>
<tr>
<td>Data Size (words)</td>
<td>7</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>139 cycles</td>
</tr>
<tr>
<td>Max</td>
<td>167 cycles</td>
</tr>
</tbody>
</table>
3.12 MCLIB_DecouplingPMSM

This function calculates the cross-coupling voltages to eliminate the d-q axis coupling, causing non-linearity of the control.

3.12.1 Synopsis

```c
#include "mclib.h"
void MCLIB_DecouplingPMSM(MCLIB_2_COOR_SYST_D_Q_T *pudtUs,
MCLIB_2_COOR_SYST_D_Q_T *pudtIs, Frac16 f16AngularVelocity,
MCLIB_DECOUPLING_PMSM_PARAM_T *pudtDecParam, MCLIB_2_COOR_SYST_D_Q_T
*pudtUsDec)
```

3.12.2 Prototype

```c
asm void MCLIB_DecouplingPMSMFAsm(MCLIB_2_COOR_SYST_D_Q_T *pudtUs,
MCLIB_2_COOR_SYST_D_Q_T *pudtIs, Frac16 f16AngularVelocity,
MCLIB_DECOUPLING_PMSM_PARAM_T *pudtDecParam, MCLIB_2_COOR_SYST_D_Q_T
*pudtUsDec)
```

V3 core version:

```c
asm void MCLIB_V3DecouplingPMSMFAsm(MCLIB_2_COOR_SYST_D_Q_T *pudtUs,
MCLIB_2_COOR_SYST_D_Q_T *pudtIs, Frac16 f16AngularVelocity,
MCLIB_DECOUPLING_PMSM_PARAM_T *pudtDecParam, MCLIB_2_COOR_SYST_D_Q_T
*pudtUsDec)
```

3.12.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtUs</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator-voltage vector.</td>
</tr>
<tr>
<td>*pudtIs</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the stator-current vector.</td>
</tr>
<tr>
<td>f16AngularVelocity</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>angular velocity in rad/s</td>
</tr>
<tr>
<td>*pudtDecParam</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the stator inductances in the d, q axes and their scale parameters.</td>
</tr>
<tr>
<td>*pudtUsDec</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing direct (alpha/a) and quadrature (beta/b) components of the decoupled stator voltage vector.</td>
</tr>
</tbody>
</table>
### 3.12.4 Availability

This library module is available in the C-callable interface assembly format.

This library module is targeted for the 56800E and 56800Ex platforms.

### 3.12.5 Dependencies

List of all dependent files:
- MCLIB_DecouplingAsm.h
- MCLIB_types.h

### 3.12.6 Description

The d-q model of the motor contains cross-coupling voltage that causes non-linearity of the control. Figure 3-20 represents the d-q model of the motor that can be described using these equations, where the underlined portion is the cross-coupling voltage:

\[
\begin{align*}
ud &= R_s \times id + L_d \times \frac{di_d}{dt} - L_q \times \omega_{el} \times iq \\
uq &= R_s \times iq + L_q \times \frac{di_q}{dt} + L_d \times \omega_{el} \times id + \omega_{el} \times K
\end{align*}
\]

Eqn. 3-68

Eqn. 3-69

where:

\[ud, uq \text{ — d,q voltage}\]
Rs — stator winding resistance  
Ld, Lq — stator winding inductance  

![Figure 3-20. The d-q Model](image)

To eliminate this non-linearity, the cross-coupling voltage is calculated using the MCLIB_DecouplingPMSM algorithm and feedforwarded to the d and q voltages. The decoupling algorithm is calculated, according to the following equations:

\[ u_{ddec} = u_d - L_q \times \omega_{el} \times i_q \]  
\[ u_{qdec} = u_q + L_d \times \omega_{el} \times i_d \]  

where:

- \( u_{ddec}, u_{qdec} \) — decoupled d,q voltage output from the algorithm

The fractional representation of the d-component equation is:

\[ u_{ddec_s} = u_{ds} - \omega_{el} \times i_q \times \left( L_q \times \omega_{max} \times \frac{i_{max}}{u_{max}} \right) \]  

\[ k_q = L_q \times \omega_{max} \times \frac{i_{max}}{u_{max}} \]  

\[ u_{ddec_s} = u_{ds} - \omega_{el} \times i_q \times k_q \]
The fractional representation of the q-component equation is:

\[ u_{qdecs} = u_{qs} + \omega_{el} \times i_d \times \left( L_d \times \omega_{max} \times \frac{i_{max}}{u_{max}} \right) \]  \hspace{2cm} Eqn. 3-75

\[ k_d = L_d \times \omega_{max} \times \frac{i_{max}}{u_{max}} \]  \hspace{2cm} Eqn. 3-76

\[ u_{qdecs} = u_{qs} + \omega_{el} \times i_d \times k_d \]  \hspace{2cm} Eqn. 3-77

These two parameters have to be scaled to fit into the 16-bit fractional range. This condition has to be fulfilled:

\[ 0.5 \leq k_q \times 2^{-q_{sc}} < 1 \]  \hspace{2cm} Eqn. 3-78

\[ 0.5 \leq k_d \times 2^{-d_{sc}} < 1 \]  \hspace{2cm} Eqn. 3-79

Then the scaled parameters can be defined as:

\[ k_{qsc} = k_q \times 2^{-q_{sc}} \]  \hspace{2cm} Eqn. 3-80

\[ k_{dsc} = k_d \times 2^{-d_{sc}} \]  \hspace{2cm} Eqn. 3-81

where the scaling coefficients \( q_{sc} \) and \( d_{sc} \) have to fulfill this condition:

\[ q_{sc} \leq \frac{\log k_q - \log 0.5}{\log 2} \]  \hspace{2cm} Eqn. 3-82

\[ q_{sc} > \frac{\log k_q}{\log 2} \]  \hspace{2cm} Eqn. 3-83

\[ d_{sc} \leq \frac{\log k_d - \log 0.5}{\log 2} \]  \hspace{2cm} Eqn. 3-84

\[ d_{sc} > \frac{\log k_d}{\log 2} \]  \hspace{2cm} Eqn. 3-85

So the final fractional equations with scaling are:

\[ u_{ddecs} = u_{ds} - (\omega_{el} \times i_q \times k_{qsc}) \times 2^{q_{sc}} \]  \hspace{2cm} Eqn. 3-86

\[ u_{qdecs} = u_{qs} + (\omega_{el} \times i_d \times k_{dsc}) \times 2^{d_{sc}} \]  \hspace{2cm} Eqn. 3-87
The principle of the algorithm use is depicted in Figure 3-21 where:

- $i_{d\text{des}}, i_{q\text{des}}$ — desired d, q currents
- $i_d, i_q$ — measured d, q currents
- $u_d, u_q$ — d, q voltage output from the PI controller
- $u_{d\text{dec}}, u_{q\text{dec}}$ — decoupled d, q voltages
- $\omega_{el}$ — electrical angular velocity

3.12.7 Range Issues

This function works with the 16-bit signed fractional values in the range $(-1, 1)$. The range of the $q_{sc}$ and $d_{sc}$ parameters is $(-15, 15)$.

3.12.8 Special Issues

The function `MCLIB_DecouplingPMSM` is the saturation mode independent.

3.12.9 Implementation

The `MCLIB_DecouplingPMSM` function is implemented as a function call.

```
#include "mclib.h"

static MCLIB_2_COOR_SYST_D_Q_T mudtVoltageDQ;
static MCLIB_2_COOR_SYST_D_Q_T mudtCurrentDQ;
static Frac16 mf16AngularSpeed;
static MCLIB_DECOUPLING_PMSM_PARAM_T mudtDecouplingParam;
static MCLIB_2_COOR_SYST_D_Q_T mudtVoltageDQDecoupled;

void Isr(void);
void main(void)
```

`Motor Control Library, Rev. 0`

Freescale Semiconductor
/* Voltage D, Q structure initialization */
mudtVoltageDQ.f16D = 0;
mudtVoltageDQ.f16Q = 0;

/* Current D, Q structure initialization */
mudtCurrentDQ.f16D = 0;
mudtCurrentDQ.f16Q = 0;

/* Speed initialization */
mf16AngularSpeed = 0;

/* Motor parameters for decoupling */
mudtDecouplingParam.f16KdGain = FRAC16(0.8455);
mudtDecouplingParam.i16KdGainShift = -5;
mudtDecouplingParam.f16KqGain = FRAC16(0.5095);
mudtDecouplingParam.i16KqGainShift = -4;

} /* Periodical function or interrupt */
void Isr(void)
{
    /* Decoupling calculation */
    MCLIB_DecouplingPMSM(&mudtVoltageDQ, &mudtCurrentDQ,
mf16AngularSpeed, &mudtDecouplingParam, &mudtVoltageDQDecoupled);
}

3.12.10 Performance

Table 3-39. Performance of the MCLIB_DecouplingPMSM Function

<table>
<thead>
<tr>
<th>Code Size (words)</th>
<th>V2: 54, V3: 34</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (words)</td>
<td>0</td>
</tr>
</tbody>
</table>
3.13 MCLIB_ElimDcBusRip

This function is used for elimination of the DC-bus voltage ripple. The alpha, beta voltage scale is assumed to be the dc-bus voltage scale.

3.13.1 Synopsis

```
#include "mclib.h"
void MCLIB_ElimDcBusRip(Frac16 f16InvModIndex, Frac16 f16DcBusMsr,
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta,
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta)
```

3.13.2 Prototype

```
asm void MCLIB_ElimDcBusRipFAsm(Frac16 f16InvModIndex, Frac16
f16DcBusMsr, MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta,
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta)
```

3.13.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f16InvModIndex</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...</td>
<td>Inverse modulation index; depends on the selected modulation technique.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x7FFF</td>
<td></td>
</tr>
<tr>
<td>f16DcBusMsr</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...</td>
<td>measured DC-bus voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0x7FFF</td>
<td></td>
</tr>
<tr>
<td>*pudtInAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure with direct (alpha) and quadrature (beta) components of the stator-voltage vector.</td>
</tr>
<tr>
<td>*pudtOutAlphaBeta</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure with direct (alpha) and quadrature (beta) components of the stator-voltage vector.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_2_COOR_SYST_ALPHA_BETA_T</td>
<td>f16Alpha</td>
<td>In/out</td>
<td>SF16</td>
<td>0x8000...</td>
<td>Alpha component</td>
</tr>
<tr>
<td></td>
<td>f16Beta</td>
<td>In/out</td>
<td>SF16</td>
<td>0x8000...</td>
<td>Beta component</td>
</tr>
</tbody>
</table>

3.13.4 Availability

This library module is available in the C-callable interface assembly format. This library module is targeted for the 56800E and 56800Ex platforms.
3.13.5 Dependencies

List of all dependent files:

- MCLIB_ElimDcBusRipAsm.h
- MCLIB_types.h

3.13.6 Description

The **MCLIB_ElimDcBusRip** function may be used in general motor control applications, and provides elimination of the voltage ripple on the DC-bus of the power stage.

The **MCLIB_ElimDcBusRip** function compensates an amplitude of the direct-\( \alpha \) and the quadrature-\( \beta \) component of the stator-reference voltage vector \( U_S \) due to imperfections of the DC-bus voltage. These imperfections are eliminated by the formula shown in the following equations:

\[
\begin{align*}
    f16Alpha^* &= \begin{cases} 
        \frac{f16InvModIndex \cdot f16Alpha}{f16DcBusMsr} & \text{if } |f16InvModIndex \cdot f16Alpha| < \frac{f16DcBusMsr}{2} \\
        \text{sgn}(f16Alpha) \cdot 1.0 & \text{otherwise}
    \end{cases} \\
    f16Beta^* &= \begin{cases} 
        \frac{f16InvModIndex \cdot f16Beta}{f16DcBusMsr} & \text{if } |f16InvModIndex \cdot f16Beta| < \frac{f16DcBusMsr}{2} \\
        \text{sgn}(f16Beta) \cdot 1.0 & \text{otherwise}
    \end{cases}
\end{align*}
\]

where \( y = \text{sgn} \) (x) function is defined as follows:

\[
y = \begin{cases} 
    1.0 & \text{if } x \geq 0 \\
    -1.0 & \text{otherwise}
\end{cases}
\]

where alpha, beta are the input duty-cycle ratios and \( f16Alpha^* \), \( f16Beta^* \) are the output duty-cycle ratios. Note that the input duty-cycle ratios are referred with the pointer *pudtInAlphaBeta, and the output duty-cycle ratios are referred with *pudtOutAlphaBeta.

Figure 3-22 shows the results of the DC-bus ripple elimination, while compensating the ripples of rectified voltage using a three-phase uncontrolled rectifier.
3.13.7 Returns

This function returns an integer value representing the sector number, in which the instantaneous stator-reference voltage vector is located.

3.13.8 Range Issues

To achieve proper functionality, the arguments of this function must be within the specified limits:

- InvModIndex must be within the fractional range and positive:
  - $0 < f16InvModIndex < 1$. The value depends on the selected modulation technique; in other words for space vector modulation techniques and injection of the third harmonic, it is equal to 0.866025, and for the inverse Clarke transformation, it is equal to 1.0.

- $f16DcBusMsr$ must be within the fractional range and positive:
  - $0 < f16DcBusMsr < 1$ that is equal to 0 % – 100 % of the maximum DC-bus voltage.

- Alpha and beta components of the stator-reference voltage vector must be within the fractional range:

---

**Figure 3-22. Results of the DC-Bus Voltage Ripple Elimination**
— $-\frac{\text{f16DcBusMsr}}{2 \cdot \text{f16InvModIndex}} < x < \frac{\text{f16DcBusMsr}}{2 \cdot \text{f16InvModIndex}}$, where $x$ stands for alpha, beta. If the inputs are out of the specified range, then the respective outputs $\alpha^*$, $\beta^*$ will be saturated to their positive or negative maximal values, according to the sign of the input components.

### 3.13.9 Special Issues

The **MCLIB_ElimDcBusRip** function is the saturation mode independent.

### 3.13.10 Implementation

**Example 3-12. Implementation Code**

```c
#include "mclib.h"

static Frac16 mf16InvModeIndex;
static Frac16 mf16DCBusVoltage;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtVoltageAlphaBeta;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtVoltageAlphaBetaOut;

void Isr(void);

void main(void)
{
    /* Voltage Alpha, Beta structure initialization */
    mudtVoltageAlphaBeta.f16Alpha = 0;
    mudtVoltageAlphaBeta.f16Beta = 0;

    /* Inv. mode index */
    mf16InvModeIndex = FRAC16(0.866025);

    /* DC bus voltage initialization */
    mf16DCBusVoltage = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Ripple elimination calculation */
    MCLIB_ElimDcBusRip(mf16InvModeIndex, mf16DCBusVoltage,
                        &mudtVoltageAlphaBeta, &mudtVoltageAlphaBetaOut);
}
```

### 3.13.11 See Also

See **MCLIB_ElimDcBusRipGen** for more information.
### 3.13.12 Performance

Table 3-42. Performance of the MCLIB_ElimDcBusRip Function

<table>
<thead>
<tr>
<th>Code Size (words)</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (words)</td>
<td>0</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td>Min.</td>
<td>79 cycles</td>
</tr>
<tr>
<td>Max.</td>
<td>79 cycles</td>
</tr>
</tbody>
</table>
3.14 MCLIB_ElimDcBusRipGen

This function is used for elimination of the DC-bus voltage ripple for the general cases of alpha, beta voltage scale, i.e. the voltage scale depends on the modulation technique.

3.14.1 Synopsis

#include "mclib.h"
void MCLIB_ElimDcBusRipGen(Frac16 f16DcBusMsr, MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta, MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta)

3.14.2 Prototype

asm void MCLIB_ElimDcBusRipGenFAsm(Frac16 f16DcBusMsr, MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta, MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta)

3.14.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f16DcBusMsr</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...0x7FFF</td>
<td>measured DC-bus voltage</td>
</tr>
<tr>
<td>*pudtInAlphaBeta</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure with direct (alpha) and quadrature (beta) components of the stator-voltage vector.</td>
</tr>
<tr>
<td>*pudtOutAlphaBeta</td>
<td>Out</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure with direct (alpha) and quadrature (beta) components of the stator-voltage vector.</td>
</tr>
</tbody>
</table>

3.14.4 Availability

This library module is available in the C-callable interface assembly format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.14.5 Dependencies

List of all dependent files:
3.14.6 Description

The **MCLIB_ElimDcBusRipGen** function may be used in general motor control applications, and provides elimination of the voltage ripple on the DC-bus of the power stage.

The **MCLIB_ElimDcBusRipGen** function compensates an amplitude of the direct-α and the quadrature-β component of the stator-reference voltage vector \( U_s \) due to imperfections of the DC-bus voltage. These imperfections are eliminated by the formula shown in the following equations:

\[
\begin{align*}
\text{Eqn. 3-91} \\
\text{Eqn. 3-92} \\
\text{Eqn. 3-93}
\end{align*}
\]

\[
\begin{align*}
f_{16}\text{Alpha}^* &= \left\{ \begin{array}{ll}
\frac{f_{16}\text{Alpha}}{f_{16}\text{DcBusMsr}} & \text{if } |f_{16}\text{Alpha}| < \frac{f_{16}\text{DcBusMsr}}{2} \\
\text{sgn}(f_{16}\text{Alpha}) \cdot 1.0 & \text{otherwise}
\end{array} \right. \\
f_{16}\text{Beta}^* &= \left\{ \begin{array}{ll}
\frac{f_{16}\text{Beta}}{f_{16}\text{DcBusMsr}} & \text{if } |f_{16}\text{Beta}| < \frac{f_{16}\text{DcBusMsr}}{2} \\
\text{sgn}(f_{16}\text{Beta}) \cdot 1.0 & \text{otherwise}
\end{array} \right.
\end{align*}
\]

where \( y = \text{sgn} (x) \) function is defined as follows:

\[
\text{Eqn. 3-94}
\]

\[
y = \left\{ \begin{array}{ll}
1.0 & \text{if } x \geq 0 \\
-1.0 & \text{otherwise}
\end{array} \right.
\]

where alpha, beta are the input duty-cycle ratios and \( f_{16}\text{Alpha}^*, f_{16}\text{Beta}^* \) are the output duty-cycle ratios. Note that the input duty-cycle ratios are referred with the pointer \(*\text{pudtInAlphaBeta} \), and the output duty-cycle ratios are referred with \(*\text{pudtOutAlphaBeta} \).

**Figure 3-23** shows the results of the DC-bus ripple elimination, while compensating the ripples of rectified voltage using a three-phase uncontrolled rectifier.
3.14.7 Returns

This function returns an integer value representing the sector number, in which the instantaneous stator-reference voltage vector is located.

3.14.8 Range Issues

To achieve proper functionality, the arguments of this function must be within the specified limits:

- \( f_{16DcBusMsr} \) must be within the fractional range and positive:
  
  \[ 0 < f_{16DcBusMsr} < 1 \] that is equal to 0 % – 100 % of the maximum DC-bus voltage.

- Alpha and beta components of the stator-reference voltage vector must be within the fractional range:
  
  \[ -f_{16DcBusMsr}/1.73 < x < f_{16DcBusMsr}/1.73 \] in case of SVM with the 3rd harmonic injection, and/or \(-f_{16DcBusMsr}/2 < x < f_{16DcBusMsr}/2\) in case of the inverse Clarke transformation (where \( x \) stands for alpha, beta). If the inputs are out of the specified range, then the respective outputs alpha*, beta* will be saturated to their positive or negative maximal values, according to the sign of the input components.
3.14.9 Special Issues

The **MCLIB_ElimDcBusRipGen** function is the saturation mode independent.

3.14.10 Implementation

**Example 3-13. Implementation Code**

```c
#include "mclib.h"

static Frac16 mf16DCBusVoltage;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtVoltageAlphaBeta;
static MCLIB_2_COOR_SYST_ALPHA_BETA_T mudtVoltageAlphaBetaOut;

void Isr(void);

void main(void)
{
    /* Voltage Alpha, Beta structure initialization */
    mudtVoltageAlphaBeta.f16Alpha = 0;
    mudtVoltageAlphaBeta.f16Beta = 0;

    /* DC bus voltage initialization */
    mf16DCBusVoltage = 0;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Ripple elimination calculation */
    MCLIB_ElimDcBusRipGen(mf16DCBusVoltage, &mudtVoltageAlphaBeta,
                          &mudtVoltageAlphaBetaOut);
}
```

3.14.11 See Also

See **MCLIB_ElimDcBusRip** for more information.

3.14.12 Performance

**Table 3-45. Performance of the MCLIB_ElimDcBusRipGen Function**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Size (words)</strong></td>
<td>32</td>
</tr>
<tr>
<td><strong>Data Size (words)</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Execution Clock</strong></td>
<td></td>
</tr>
<tr>
<td>Min.</td>
<td>70 cycles</td>
</tr>
<tr>
<td>Max.</td>
<td>70 cycles</td>
</tr>
</tbody>
</table>

Motor Control Library, Rev. 0

Freescale Semiconductor
3.15 MCLIB_VectorLimit

This function calculates the amplitude limitation of the input vector described by the d and q components. The limitation is calculated to achieve the zero angle error.

3.15.1 Synopsis

```c
#include "mclib.h"
void MCLIB_VectorLimit(MCLIB_2_COOR_SYST_T *pudtInVector,
                       MCLIB_2_COOR_SYST_T *pudtLimVector, MCLIB_VECTOR_LIMIT_PARAMS_T
                       *pudtParams)
```

3.15.2 Prototype

```asm
asm void MCLIB_VectorLimitFAsm(MCLIB_2_COOR_SYST_T *pudtInVector,
                                MCLIB_2_COOR_SYST_T *pudtLimVector, MCLIB_VECTOR_LIMIT_PARAMS_T
                                *pudtParams)
```

3.15.3 Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtInVector</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing input vectors.</td>
</tr>
<tr>
<td>*pudtLimVector</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing output vectors.</td>
</tr>
<tr>
<td>*pudtParams</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the f16Lim and blnLimFlag.</td>
</tr>
</tbody>
</table>

3.15.4 Availability

This library module is available in the C-callable interface assembly format.

This library module is targeted for the 56800E and 56800Ex platforms.

3.15.5 Dependencies

List of all dependent files:

---

Motor Control Library, Rev. 0

Freescale Semiconductor 5-90
• MCLIB_VectorLimitAsm.h
• MCLIB_types.h
• GFLIB.h

3.15.6 Description

The **MCLIB_VectorLimit** function limits the amplitude of the input vector. The input vector components, `pudtInVector.f16A` and `pudtInVector.f16B`, are passed into the function as the input arguments. The resulting limited vector is transformed back into the components `pudtLimVector.f16A` and `pudtLimVector.f16B`. This function uses the GFLIB_SqrtPoly module of General Function Library to calculate the modulus of the input vector. The limitation is performed as follows:

\[
\text{mod}^2 = \text{pudtInVector}.f16A^2 + \text{pudtInVector}.f16B^2 \quad \text{Eqn. 3-95}
\]

\[
\text{pudtLimVector}.f16A = \begin{cases} 
\frac{\text{f16Lim} \cdot \text{mod}}{\text{pudtInVector}.f16A} & \text{if mod}^2 > \text{f16Lim}^2 \\
\text{pudtInVector}.f16A & \text{if mod}^2 \leq \text{f16Lim}^2
\end{cases} \quad \text{Eqn. 3-96}
\]

\[
\text{pudtLimVector}.f16B = \begin{cases} 
\frac{\text{f16Lim} \cdot \text{mod}}{\text{pudtInVector}.f16B} & \text{if mod}^2 > \text{f16Lim}^2 \\
\text{pudtInVector}.f16B & \text{if mod}^2 \leq \text{f16Lim}^2
\end{cases} \quad \text{Eqn. 3-97}
\]

The relationship between the input and limited output vectors is obvious from Figure 3-24.

![Figure 3-24. Input and Limited Output Vectors Relationship](image)
If the actual mod value is greater than the input fl6Lim value, the function calculates the value from the fl6Lim value. If the actual mod value is lower than the input fl6Lim value, the function copies the value from the actual value.

3.15.7 Range Issues

The input data value is in the range of $(-1, 1)$, and the output data values are in the range $(-1, 1)$.

3.15.8 Special Issues

The MCLIB_VectorLimit function uses the function GFLIB_SqrtPoly from GFLIB.

The MCLIB_VectorLimit function requires the saturation mode to be set.

3.15.9 Implementation

The MCLIB_VectorLimit function is implemented as a function call.

**Example 3-14. Implementation Code**

```c
#include "mclib.h"

static MCLIB_2_COOR_SYST_T mudtVector;
static MCLIB_2_COOR_SYST_T mudtLimitedVector;
static MCLIB_VECTOR_LIMIT_PARAMS_T mudtVectorLimitParam;

void main(void)
{
    /* Vector limit structure initialization */
    mudtVectorLimitParam.f16Lim = FRAC16(0.5);
    mudtVectorLimitParam.blnLimFlag = 0;

    /* Vector definition */
    mudtVector.f16A = FRAC16(0.8);
    mudtVector.f16B = FRAC16(0.7);

    /* Vector limitation */
    MCLIB_VectorLimit(&mudtVector, &mudtLimitedVector, &mudtVectorLimitParam);
}
```

3.15.10 See Also

See MCLIB_VectorLimit12 for more information.
### 3.15.11 Performance

Table 3-48. Performance of the MCLIB\_VectorLimit Function

<table>
<thead>
<tr>
<th>Code Size (words)</th>
<th>52 + 65 (GFLIB_SqrtPoly)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size (words)</td>
<td>0 + 34 (GFLIB_SqrtPoly)</td>
</tr>
<tr>
<td>Execution Clock</td>
<td></td>
</tr>
<tr>
<td>Min.</td>
<td>45 cycles</td>
</tr>
<tr>
<td>Max.</td>
<td>186 cycles</td>
</tr>
</tbody>
</table>
3.16 MCLIB_VectorLimit12

This function calculates the amplitude limitation of the input vector described by the d and q components. The limitation is calculated to achieve the zero angle error. This function uses the 12-bit precision square root calculation so it is quicker but with lower precision of calculation in comparison to MCLIB_VectorLimit.

3.16.1 Synopsis

```c
#include "mclib.h"
void MCLIB_VectorLimit12(MCLIB_2_COOR_SYST_T *pudtInVector, MCLIB_2_COOR_SYST_T *pudtLimVector, MCLIB_VECTOR_LIMIT_PARAMS_T *pudtParams)
```

3.16.2 Prototype

```asm
asm void MCLIB_VectorLimit12FAsm(MCLIB_2_COOR_SYST_T *pudtInVector, MCLIB_2_COOR_SYST_T *pudtLimVector, MCLIB_VECTOR_LIMIT_PARAMS_T *pudtParams)
```

3.16.3 Arguments

Table 3-49. Function Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*pudtInVector</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing input vectors.</td>
</tr>
<tr>
<td>*pudtLimVector</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing output vectors.</td>
</tr>
<tr>
<td>*pudtParams</td>
<td>In</td>
<td>N/A</td>
<td>N/A</td>
<td>Pointer to a structure containing the f16Lim and blnLimFlag.</td>
</tr>
</tbody>
</table>

Table 3-50. User Type Definitions

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Name</th>
<th>In/Out</th>
<th>Format</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLIB_VECTOR_LIMIT_PARAMS_T</td>
<td>f16Lim</td>
<td>In</td>
<td>SF16</td>
<td>0x8000...</td>
<td>Value of the input vector amplitude limit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x7FFF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>blnLimFlag</td>
<td>Out</td>
<td>SI16</td>
<td>0x8000...</td>
<td>True/False flag describing the status of the limiter;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x7FFF</td>
<td>true — signal is limited; false — signal is not limited</td>
</tr>
</tbody>
</table>

3.16.4 Availability

This library module is available in the C-callable interface assembly format.
This library module is targeted for the 56800E and 56800Ex platforms.

Motor Control Library, Rev. 0
3.16.5 Dependencies

List of all dependent files:

- MCLIB_VectorLimitAsm.h
- MCLIB_types.h
- GFLIB.h

3.16.6 Description

The MCLIB_VectorLimit12 function limits the amplitude of the input vector. The input vector components, pudtInVector.f16A and pudtInVector.f16B, are passed into the function as the input arguments. The resulting limited vector is transformed back into the components pudtLimVector.f16A and pudtLimVector.f16B. This function uses the GFLIB_SqrtPoly module of General Function Library to calculate the modulus of the input vector. The limitation is performed as follows:

\[
\text{mod}^2 = \text{pudtInVector.f16A}^2 + \text{pudtInVector.f16B}^2 \quad \text{Eqn. 3-98}
\]

\[
\text{pudtLimVector.f16A} = \begin{cases} 
    \frac{\text{f16Lim} \cdot \text{mod}}{\text{pudtInVector.f16A}} & \text{if } \text{mod}^2 > \text{f16Lim}^2 \\
    \text{pudtInVector.f16A} & \text{if } \text{mod}^2 \leq \text{f16Lim}^2 
\end{cases} \quad \text{Eqn. 3-99}
\]

\[
\text{pudtLimVector.f16B} = \begin{cases} 
    \frac{\text{f16Lim} \cdot \text{mod}}{\text{pudtInVector.f16B}} & \text{if } \text{mod}^2 > \text{f16Lim}^2 \\
    \text{pudtInVector.f16B} & \text{if } \text{mod}^2 \leq \text{f16Lim}^2 
\end{cases} \quad \text{Eqn. 3-100}
\]

The relationship between the input and limited output vectors is obvious from Figure 3-25.
If the actual mod value is greater than the input f16Lim value, the function calculates the value from the f16Lim value. If the actual mod value is lower than the input f16Lim value, the function copies the value from the actual value.

3.16.7 Range Issues

The input data value is in the range of \((-1, 1)\), and the output data values are in the range \((-1, 1)\).

3.16.8 Special Issues

The \texttt{MCLIB\_VectorLimit12} function uses the function \texttt{GFLIB\_SqrtPoly} from \texttt{GFLIB}.

The \texttt{MCLIB\_VectorLimit12} function requires the saturation mode to be set.

3.16.9 Implementation

The \texttt{MCLIB\_VectorLimit12} function is implemented as a function call.

\begin{verbatim}
#include "mclib.h"

static MCLIB_2_COOR_SYST_T mudtVector;
static MCLIB_2_COOR_SYST_T mudtLimitedVector;
static MCLIB_VECTOR_LIMIT_PARAMS_T mudtVectorLimitParam;

void main(void)
{
    /* Vector limit structure initialization */
    mudtVectorLimitParam.f16Lim = FRAC16(0.5);
    mudtVectorLimitParam.blnLimFlag = 0;

    /* Example 3-15. Implementation Code */

    /* Code implementation */
}
\end{verbatim}
MCLIB VectorLimit12

/* Vector definition */
mudtVector.f16A = FRAC16(0.8);
mudtVector.f16B = FRAC16(0.7);

/* Vector limitation */
MCLIB_VectorLimit12(&mudtVector, &mudtLimitedVector,
&mudtVectorLimitParam);
}

3.16.10 See Also

See MCLIB VectorLimit for more information.

3.16.11 Performance

<table>
<thead>
<tr>
<th>Table 3-51. Performance of the MCLIB VectorLimit12 Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Size (words)</strong></td>
</tr>
<tr>
<td><strong>Data Size (words)</strong></td>
</tr>
<tr>
<td><strong>Execution Clock</strong></td>
</tr>
<tr>
<td>Min.</td>
</tr>
<tr>
<td>Max.</td>
</tr>
</tbody>
</table>
## Appendix A  Revision History

Table 0-1. Revision history

<table>
<thead>
<tr>
<th>Revision number</th>
<th>Date</th>
<th>Subsequent changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02/2014</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and CodeWarrior are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2014 Freescale Semiconductor, Inc.