Motor Control Library

User Reference Manual

56800E
Digital Signal Controller

56800E_MCLIB
Rev. 3
5/2011

freescale.com

freescale semiconductor

The following revision history table summarizes changes contained in this document.

Table 0-1. Revision History

Date	Revision Label	Description
	0	Initial release
	1	Reformatted and updated revision
	2	Arguments listing of MCLIB_ParkTrfInv fixed
	3	FSLESL 2.0

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)(l) 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 Motor Control Library for Freescale 56800E family of Digital Signal Controllers. This library contains optimized functions for 56800E family of controllers. The library is supplied in a binary form, which is unique by its simplicity to integrate with user application.

2.2 Supported Compilers

Motor Control Library (MCLIB) is written in assembly language with a C-callable interface. The library was built and tested using the following compiler:

- CodeWarrior™ Development Studio for Freescale™ DSC56800/E Digital Signal Controllers, version 8.3

The library is delivered in the *56800E_MCLIB.lib* library module. The interfaces to the algorithms included in this library have been combined into a single public interface include file, the *mclib.h*. This was done to reduce the number of files required for inclusion by the 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 the user wants to fully use this library, the CodeWarrior tools should be installed prior to Motor Control Library. In case that Motor Control Library tool is installed while CodeWarrior is not present, users can only browse the installed software package, but will not be able to build, download, and run the 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.

Each Motor Control Library release is installed in its own new folder, named *56800E_MCLIB_rX.X*, where *X.X* denotes the actual release number. This way of library installation allows the users to maintain older releases and projects and gives them a free choice to select the active library release.

To start the installation process, follow the following steps:

1. Execute the *56800E_FSLESL_rXX.exe* file.
2. Follow the FSLESL software installation instructions on your screen.

2.4 Library Integration

The Motor Control Library is added into a new CodeWarrior project by taking the following steps:

1. Create a new empty project.
2. Create *MCLIB* group in your new open project. Note that this step is not mandatory, it is mentioned here just for the purpose of maintaining file consistency in the CodeWarrior project window. In the CodeWarrior menu, choose Project > Create Group..., type MCLIB into the dialog window that pops up, and click <OK>.
3. Refer the *56800E_MCLIB.lib* file in the project window. This can be achieved by dragging the library file from the proper library subfolder and dropping it into the *MCLIB* group in the CodeWarrior project window. This step will automatically add the *MCLIB* path into the project access paths, such as the user can take advantage of the library functions to achieve flawless project compilation and linking.
4. It is similar with the reference file *mclib.h*. This file can be dragged from the proper library subfolder and dropped into the MCLIB group in the CodeWarrior project window.
5. The following program line must be added into the user-application source code in order to use the library functions.

```
#include "mclib.h"
```

6. Since Motor Control Library is not stand-alone, General Functions Library (GFLIB) must be installed and included in the application project prior to MCLIB.
7. Create *GFLIB* group in your new open project. Note that this step is not mandatory, it is mentioned here just for the purpose of maintaining file consistency in the CodeWarrior project window. In the CodeWarrior menu, choose Project > Create Group..., type GFLIB into the dialog window that pops up, and click <OK>.
8. Refer the *56800E_GFLIB.lib* file in the project window. This can be done by dragging the library file from the proper library subfolder and dropping it into the *GFLIB* group in the CodeWarrior project window. This step will automatically add the *GFLIB* path into the project access paths, such as the user can take advantage of the library functions to achieve flawless project compilation and linking.
9. It is similar with the reference file *gflib.h* in the project window. This can be achieved by dragging the file from the proper library subfolder and dropping it into the *GFLIB* group in the CodeWarrior project window.
10. The following program line must be added into the user application source code in order to use the library functions.

```
#include "gflib.h"
```

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. If the clock cycles have two numbers for instance

21/22, then the number 21 is measured on the MCF56F80xx core and the number 22 is measured on the MCF56F83xx core.

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:

$$-2^{[N-1]} \le SI \le [2^{[N-1]} - 1]$$ **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:

$$0 \le UI \le [2^{[N-1]} - 1]$$ **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.[N–1] format (one sign bit, N–1 fractional bits). The signed fractional numbers lie in the following range:

$$-1{,}0 \le SF \le 1{,}0 - 2^{-[N-1]} \quad \textbf{Eqn. 2-3}$$

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:

$$0{,}0 \le UF \le 2{,}0 - 2^{-[N-1]} \quad \textbf{Eqn. 2-4}$$

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 $\{1.0 + (1.0 - 2^{-[N-1]})\} = 1.99997$. The smallest unsigned fractional number is zero ($0000).

2.7 User Common Types

Table 2-1. User-Defined Typedefs in *56800E_types.h*

Mnemonics	Size — bits	Description
Word8	8	To represent 8-bit signed variable/value.
UWord8	8	To represent 16-bit unsigned variable/value.
Word16	16	To represent 16-bit signed variable/value.
UWord16	16	To represent 16-bit unsigned variable/value.
Word32	32	To represent 32-bit signed variable/value.
UWord32	32	To represent 16-bit unsigned variable/value.
Int8	8	To represent 8-bit signed variable/value.
UInt8	8	To represent 16-bit unsigned variable/value.
Int16	16	To represent 16-bit signed variable/value.
UInt16	16	To represent 16-bit unsigned variable/value.
Int32	32	To represent 32-bit signed variable/value.

Table 2-1. User-Defined Typedefs in *56800E_types.h* (continued)

UInt32	32	To represent 16-bit unsigned variable/value.
Frac16	16	To represent 16-bit signed variable/value.
Frac32	32	To represent 32-bit signed variable/value.
NULL	constant	Represents NULL pointer.
bool	16	Boolean variable.
false	constant	Represents false value.
true	constant	Represents true value.
FRAC16()	macro	Transforms float value from <–1, 1) range into fractional representation <–32768, 32767>.
FRAC32()	macro	Transforms float value from <–1, 1) range into fractional representation <–2147483648, 2147483648>.

Table 2-2. User-Defined Typedefs in *mclib_types.h*

Name	Structure Members	Description
MCLIB_3_COOR_SYST_T	Frac16 f16A Frac16 f16B Frac16 f16C	three phase system
MCLIB_2_COOR_SYST_T	Frac16 f16A Frac16 f16B	two phase system
MCLIB_2_COOR_SYST_ALPHA_BETA_T	Frac16 f16Alpha Frac16 f16Beta	two phase system — alpha/beta
MCLIB_2_COOR_SYST_D_Q_T	Frac16 f16D Frac16 f16Q	two phase system — generic DQ
MCLIB_ANGLE_T	Frac16 f16Sin Frac16 f16Cos	two phase system — sine and cosine components

2.8 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.

Chapter 3 FUNCTION API

3.1 API Summary

Table 3-1. API Functions Summary

Name	Arguments	Output	Description
MCLIB_ClarkTrf	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	void	This function calculates the Clarke transformation algorithm.
MCLIB_ClarkTrfInv	MCLIB_3_COOR_SYST_T *pudtAbc MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta	void	This function calculates the inverse Clarke transformation algorithm.
MCLIB_ParkTrf	MCLIB_2_COOR_SYST_D_Q_T *pudtDQ MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_ANGLE_T *pudtSinCos	void	This function calculates the Park transformation algorithm.
MCLIB_ParkTrfInv	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_2_COOR_SYST_D_Q_T *pudtDQ MCLIB_ANGLE_T *pudtSinCos	void	This function calculates the inverse Park transformation algorithm.
MCLIB_SvmStd	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	UWord16	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.
MCLIB_SvmU0n	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	UWord16	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.
MCLIB_SvmU7n	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	Uword16	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.

Table 3-1. API Functions Summary

MCLIB_SvmAlt	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	Uword16	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.
MCLIB_SvmSci	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	Uword16	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.
MCLIB_PwmIct	MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtAlphaBeta MCLIB_3_COOR_SYST_T *pudtAbc	Uword16	This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using the General Sinusoidal Modulation technique.
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	void	This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the control.
MCLIB_ElimDcBusRip	Frac16 f16InvModIndex, Frac16 f16DcBusMsr MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta	void	This function is used for elimination of the DC-bus voltage ripple.
MCLIB_ElimDcBusRipGen	Frac16 f16DcBusMsr MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtInAlphaBeta MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtOutAlphaBeta	void	This function is used for elimination of the DC-bus voltage ripple for the general cases of the modulation.
MCLIB_VectorLimit	MCLIB_2_COOR_SYST_T *pudtInVector MCLIB_2_COOR_SYST_T *pudtLimVector MCLIB_VECTOR_LIMIT_PARAMS_T *pudtParams	void	This function calculates the amplitude limitation of the input vector described by the dq components. Limitation is calculated to achieve the zero angle error.

Table 3-1. API Functions Summary

MCLIB_VectorLimit12	MCLIB_2_COOR_SYST_T *pudtInVector MCLIB_2_COOR_SYST_T *pudtLimVector MCLIB_VECTOR_LIMIT_PARAMS_T *pudtParams	void	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**.

3.2 MCLIB_ClarkTrf

This function calculates the Clarke transformation algorithm.

3.2.1 Synopsis

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

3.2.2 Prototype

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

3.2.3 Arguments

Table 3-2. Function Arguments

Name	In/Out	Format	Range	Description
*pudtAlphaBeta	Out	N/A	N/A	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.
*pudtAbc	In	N/A	N/A	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.

Table 3-3. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	Out	SF16	0x8000... 0x7FFF	Alpha component
	f16Beta	Out	SF16	0x8000... 0x7FFF	Beta component
MCLIB_3_COOR_SYST_T	f16A	In	SF16	0x8000... 0x7FFF	A component
	f16B	In	SF16	0x8000... 0x7FFF	B component
	f16C	In	SF16	0x8000... 0x7FFF	C component

3.2.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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:

$$alpha = a \qquad \textbf{Eqn. 3-1}$$

$$beta = \frac{1}{\sqrt{3}}a + \frac{2}{\sqrt{3}}b \qquad \textbf{Eqn. 3-2}$$

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

```
#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)
{
        /* Clark Transformation calculation */
        MCLIB_ClarkTrf(&mudtAlphaBeta, &mudtAbc);
```

```
}
```

3.2.10 See Also

See **MCLIB_ClarkTrfInv** for more information.

3.2.11 Performance

Table 3-4. Performance of the MCLIB_ClarkTrf Function

Code Size (bytes)	9	
Data Size (bytes)	0	
Execution Clock	Min.	21/22 cycles
	Max.	21/22 cycles

3.3 MCLIB_ClarkTrfInv

This function calculates the inverse Clarke transformation algorithm.

3.3.1 Synopsis

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

3.3.3 Arguments

Table 3-5. Function Arguments

Name	In/Out	Format	Range	Description
*pudtAlphaBeta	In	N/A	N/A	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.
*pudtAbc	Out	N/A	N/A	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.

Table 3-6. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	Alpha component
	f16Beta	In	SF16	0x8000... 0x7FFF	Beta component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A component
	f16B	Out	SF16	0x8000... 0x7FFF	B component
	f16C	Out	SF16	0x8000... 0x7FFF	C component

3.3.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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:

$$a = alpha \qquad \textbf{Eqn. 3-3}$$

$$b = -0.5 \times alpha + \frac{\sqrt{3}}{2} \times beta \qquad \textbf{Eqn. 3-4}$$

$$c = -(a + b) \qquad \textbf{Eqn. 3-5}$$

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

```
#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 */
```

```
        MCLIB_ClarkTrfInv(&mudtAbc, &mudtAlphaBeta);
}
```

3.3.10 See Also

See **MCLIB_ClarkTrf** for more information.

3.3.11 Performance

Table 3-7. Performance of the MCLIB_ClarkTrfInv Function

Code Size (bytes)	12	
Data Size (bytes)	0	
Execution Clock	Min.	24 cycles
	Max.	24 cycles

3.4 MCLIB_ParkTrf

This function calculates the Park transformation algorithm.

3.4.1 Synopsis

```
#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 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 3-8. Function Arguments

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

Table 3-9. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_D_Q_T	f16D	Out	SF16	0x8000... 0x7FFF	d component
	f16Q	Out	SF16	0x8000... 0x7FFF	q component
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	Alpha component
	f16Beta	In	SF16	0x8000... 0x7FFF	Beta component
MCLIB_ANGLE_T	f16Sin	In	SF16	0x8000... 0x7FFF	sine component of the angle
	f16Cos	In	SF16	0x8000... 0x7FFF	cosine component of the angle

3.4.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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:

$$d = alpha \times \cos(theta) + beta \times \sin(theta) \qquad \textbf{Eqn. 3-6}$$

$$q = beta \times \cos(theta) - alpha \times \sin(theta) \qquad \textbf{Eqn. 3-7}$$

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

```
#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)
{
         /* Alpha, Beta structure initialization */
         mudtAlphaBeta.f16Alpha = 0;
         mudtAlphaBeta.f16Beta = 0;

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

```
        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

Code Size (bytes)	9	
Data Size (bytes)	0	
Execution Clock	Min.	24 cycles
	Max.	24 cycles

3.5 MCLIB_ParkTrfInv

This function calculates the inverse Park transformation algorithm.

3.5.1 Synopsis

```
#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

```
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 3-11. Function Arguments

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

Table 3-12. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_D_Q_T	f16D	In	SF16	0x8000... 0x7FFF	d component
	f16Q	In	SF16	0x8000... 0x7FFF	q component
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	Out	SF16	0x8000... 0x7FFF	Alpha component
	f16Beta	Out	SF16	0x8000... 0x7FFF	Beta component
MCLIB_ANGLE_T	f16Sin	In	SF16	0x8000... 0x7FFF	sine component of the angle
	f16Cos	In	SF16	0x8000... 0x7FFF	cosine component of the angle

3.5.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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:

$$alpha = d \times \cos(theta) - q \times \sin(theta) \qquad \textbf{Eqn. 3-8}$$

$$beta = d \times \sin(theta) + q \times \cos(theta) \qquad \textbf{Eqn. 3-9}$$

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

```
#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 */
```

```
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

Code Size (bytes)	9	
Data Size (bytes)	0	
Execution Clock	Min.	24/25 cycles
	Max.	24/25 cycles

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

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

3.6.2 Prototype

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

3.6.3 Arguments

Table 3-14. Function Arguments

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

Table 3-15. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	d component
	f16Beta	In	SF16	0x8000... 0x7FFF	q component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A phase
	f16B	Out	SF16	0x8000... 0x7FFF	B phase
	f16C	Out	SF16	0x8000... 0x7FFF	C phase

3.6.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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.

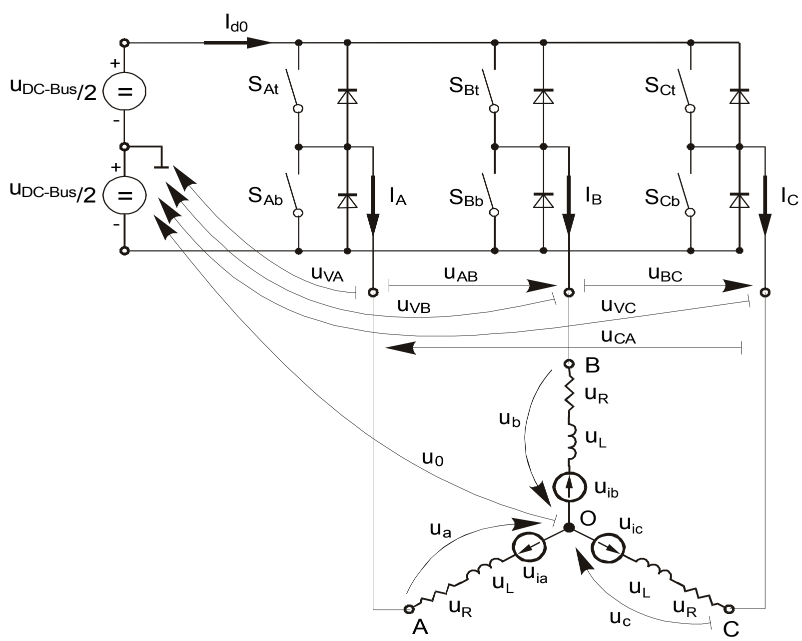

Figure 3-1. Power Stage Schematic Diagram

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:

$$\begin{bmatrix} U_{AB} \\ U_{BC} \\ U_{CA} \end{bmatrix} = U_{DCBus} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \textit{Eqn. 3-10}$$

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.

$$\begin{bmatrix} U_a \\ U_b \\ U_c \end{bmatrix} = \frac{U_{DCBus}}{3} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \textit{Eqn. 3-11}$$

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 3-16. Switching Patterns

A	B	C	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	$2U_{DCBus}/3$	$-U_{DCBus}/3$	$-U_{DCBus}/3$	U_{DCBus}	0	$-U_{DCBus}$	U_0
1	1	0	$U_{DCBus}/3$	$U_{DCBus}/3$	$-2U_{DCBus}/3$	0	U_{DCBus}	$-U_{DCBus}$	U_{60}
0	1	0	$-U_{DCBus}/3$	$2U_{DCBus}/3$	$-U_{DCBus}/3$	$-U_{DCBus}$	U_{DCBus}	0	U_{120}
0	1	1	$-2U_{DCBus}/3$	$U_{DCBus}/3$	$U_{DCBus}/3$	$-U_{DCBus}$	0	U_{DCBus}	U_{240}
0	0	1	$-U_{DCBus}/3$	$-U_{DCBus}/3$	$2U_{DCBus}/3$	0	$-U_{DCBus}$	U_{DCBus}	U_{300}
1	0	1	$U_{DCBus}/3$	$-2U_{DCBus}/3$	$U_{DCBus}/3$	U_{DCBus}	$-U_{DCBus}$	0	U_{360}
1	1	1	0	0	0	0	0	0	O_{111}

The quantities of the direct-α and the quadrature-β 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.

$$\begin{bmatrix} U_\alpha \\ U_\beta \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} U_a \\ U_b \\ U_c \end{bmatrix} \qquad \textit{Eqn. 3-12}$$

The three-phase stator voltages, U_a, U_b, and U_c, are transformed using the Clarke transformation into the direct-α and the quadrature-β 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

A	B	C	U_α	U_β	Vector
0	0	0	0	0	O_{000}
1	0	0	$2U_{DCBus}/3$	0	U_0
1	1	0	$U_{DCBus}/3$	$U_{DCBus}/\sqrt{3}$	U_{60}
0	1	0	$-U_{DCBus}/3$	$U_{DCBus}/\sqrt{3}$	U_{120}
0	1	1	$-2U_{DCBus}/3$	0	U_{240}
0	0	1	$-U_{DCBus}/3$	$-U_{DCBus}/\sqrt{3}$	U_{300}
1	0	1	$U_{DCBus}/3$	$-U_{DCBus}/\sqrt{3}$	U_{360}
1	1	1	0	0	O_{111}

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.

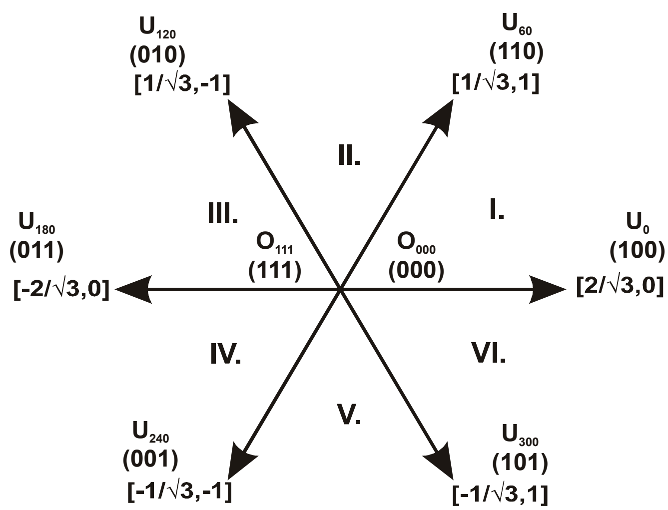

Figure 3-2. Basic Space Vectors

Referring to that principle, an objective of the standard space vector modulation is an approximation of the reference stator voltage vector US 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.

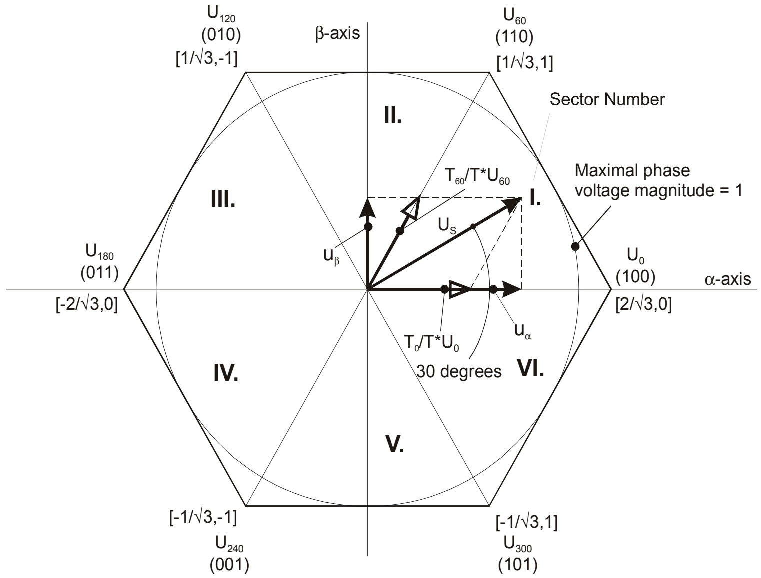

Figure 3-3. Projection of Reference Voltage Vector in Sector

The stator-reference voltage vector U_S is phase-advanced by 30° from the direct-α, 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-α and quadrature-β components for space vectors U_0 and U_{60}.

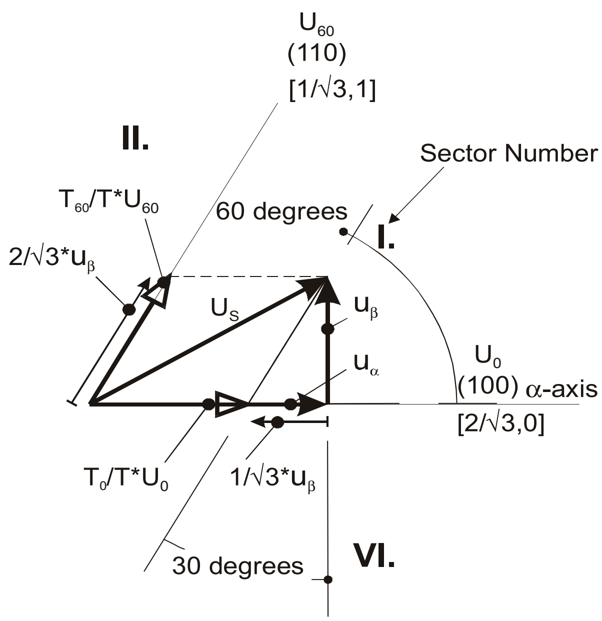

Figure 3-4. Detail of the Voltage Vector Projection in Sector

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:

$$T = T_{60} + T_0 + T_{null}$$ **Eqn. 3-13**

$$U_S = \frac{T_{60}}{T} \times U_{60} + \frac{T_0}{T} \times U_0$$ **Eqn. 3-14**

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:

$$u_\beta = \frac{T_{60}}{T} \times |U_{60}| \times \sin 60°$$ **Eqn. 3-15**

$$u_\alpha = \frac{T_0}{T} \times |U_0| + \frac{u_\beta}{\tan 60°}$$ **Eqn. 3-16**

Considering that normalized magnitudes of basic space vectors are |U60| = |U0| = $2/\sqrt{3}$, and by substitution of the trigonometric expressions sin 60° and tan 60° 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:

$$\frac{T_{60}}{T} = u_\beta$$ **Eqn. 3-17**

$$U_S = \frac{T_{120}}{T} \times U_{120} + \frac{T_{60}}{T} \times U_{60}$$ **Eqn. 3-18**

Sector II is depicted in Figure 3-5. In this particular case, the reference-stator voltage vector US 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:

$$T = T_{120} + T_{60} + T_{null}$$ **Eqn. 3-19**

$$U_S = \frac{T_{120}}{T} \times U_{120} + \frac{T_{60}}{T} \times U_{60}$$ **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.

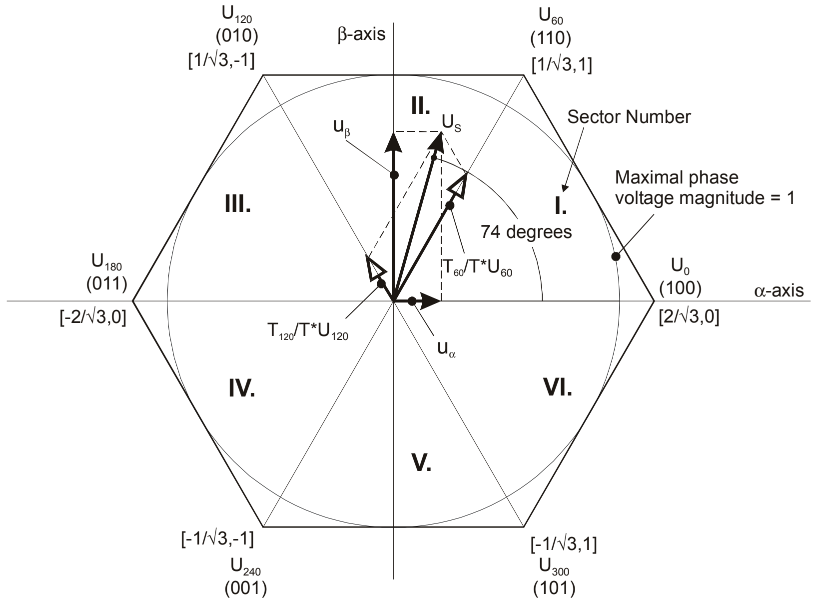

Figure 3-5. Projection of the Reference Voltage Vector in Sector

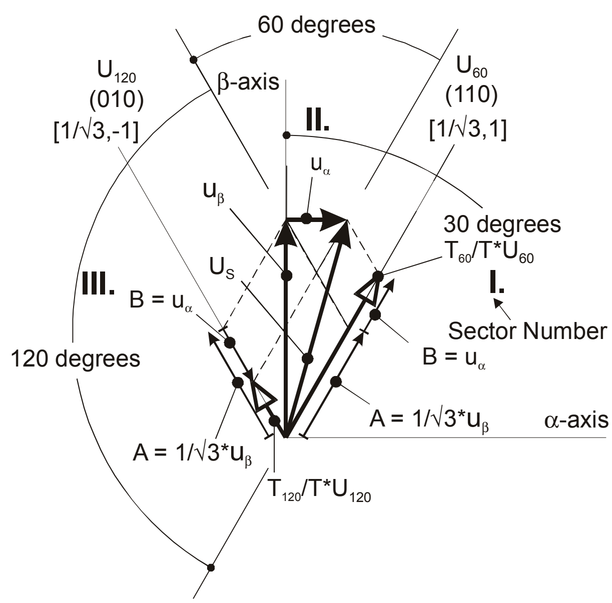

Figure 3-6. Detail of the Voltage-Vector Projection in Sector

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30°}{\sin 120°} = \frac{A}{u_\beta} \qquad \textit{Eqn. 3-21}$$

$$\frac{\sin 60°}{\sin 60°} = \frac{B}{u_\alpha} \qquad \textit{Eqn. 3-22}$$

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°, sin 120°, and sin 60°, by their numerical representations 1/2, $\sqrt{3}/2$, and $1/\sqrt{3}$, respectively.

$$A = \frac{1}{\sqrt{3}} \times u_\beta \qquad \textbf{Eqn. 3-23}$$

$$B = u_\alpha \qquad \textbf{Eqn. 3-24}$$

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 Equation 3-25 and Equation 3-26 as follows:

$$\frac{T_{120}}{T} \times |U_{120}| = (A - B) \qquad \textbf{Eqn. 3-25}$$

$$\frac{T_{60}}{T} \times |U_{60}| = (A + B) \qquad \textbf{Eqn. 3-26}$$

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:

$$\frac{T_{120}}{T} = \frac{1}{2} \times (u_\beta - \sqrt{3} \times u_\alpha) \qquad \textbf{Eqn. 3-27}$$

$$\frac{T_{60}}{T} = \frac{1}{2} \times (u_\beta + \sqrt{3} \times u_\alpha) \qquad \textbf{Eqn. 3-28}$$

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:

  $X = u\beta$

  $Y = 1/2 \times (u\beta + \sqrt{3} \times u\alpha)$

  $Z = 1/2 \times (u\beta - \sqrt{3} \times u\alpha)$
- 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 3-18. Determination of t_1 and t_2 Expressions

Sectors	U_0,U_{60}	U_{60},U_{120}	U_{120},U_{180}	U_{180},U_{240}	U_{240},U_{300}	U_{300},U_0
t_1	X	Y	–Y	Z	–Z	–X
t_2	–Z	Z	X	–X	–Y	Y

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-α and quadrature-β 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.

$$u_{ref1} = u_\beta \qquad \textbf{Eqn. 3-29}$$

$$u_{ref2} = \frac{-u_\beta + \sqrt{3} \times u_\alpha}{2} \qquad \textbf{Eqn. 3-30}$$

$$u_{ref3} = \frac{-u_\beta - \sqrt{3} \times u_\alpha}{2} \qquad \textbf{Eqn. 3-31}$$

The modified inverse Clarke transformation projects the quadrature-u_β 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_α component into u_{ref1}.

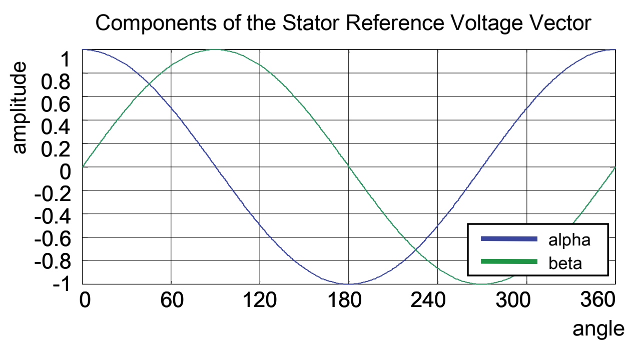

Figure 3-7. Direct-u_a and Quadrature-u_b Components of Stator Reference Voltage

Figure 3-7 depicts the direct-u_α and quadrature-u_β components of the stator reference voltage vector U_S that were calculated by the equations $u_\alpha = \cos\vartheta$ and $u_\beta = \sin\vartheta$, respectively.

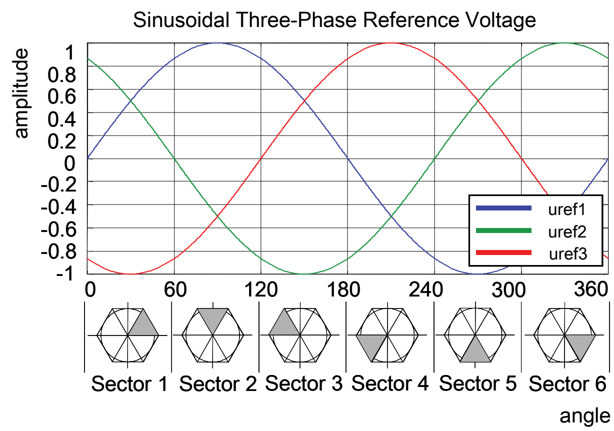

Figure 3-8. Reference Voltages Uref1, Uref2, and Uref3

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

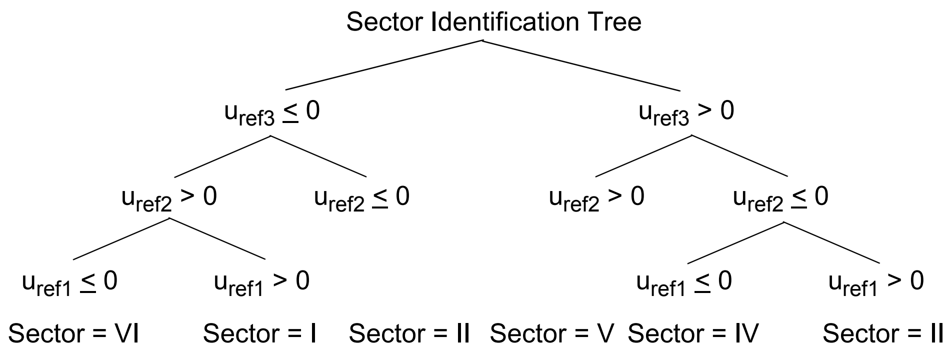

Figure 3-9. Identification of the Sector Number

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 α-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:

$$t_1 = \frac{T - t\_1 - t\_2}{2} \qquad \textbf{\textit{Eqn. 3-32}}$$

$$t_2 = t_1 + t\_1 \qquad \textbf{\textit{Eqn. 3-33}}$$

$$t_3 = t_2 + t\_2 \qquad \textbf{\textit{Eqn. 3-34}}$$

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 3-19. Assignment of the Duty-Cycle Ratios to Motor Phases

Sectors	U_0,U_{60}	U_{60},U_{120}	U_{120},U_{180}	U_{180},U_{240}	U_{240},U_{300}	U_{300},U_0
pwm_a	t_3	t_2	t_1	t_1	t_2	t_3
pwm_b	t_2	t_3	t_3	t_2	t_1	t_1
pwm_c	t_1	t_1	t_2	t_3	t_3	t_2

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 US. 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.

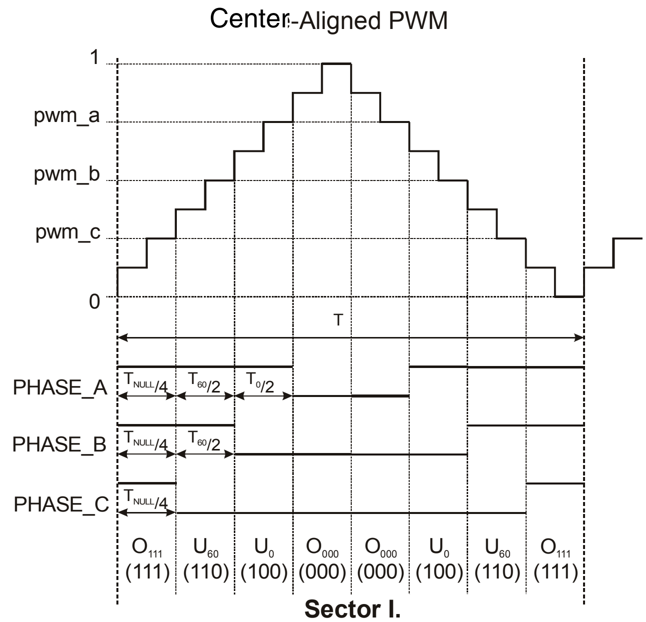

Figure 3-10. Standard Space Vector Modulation Technique — Center-Aligned PWM

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

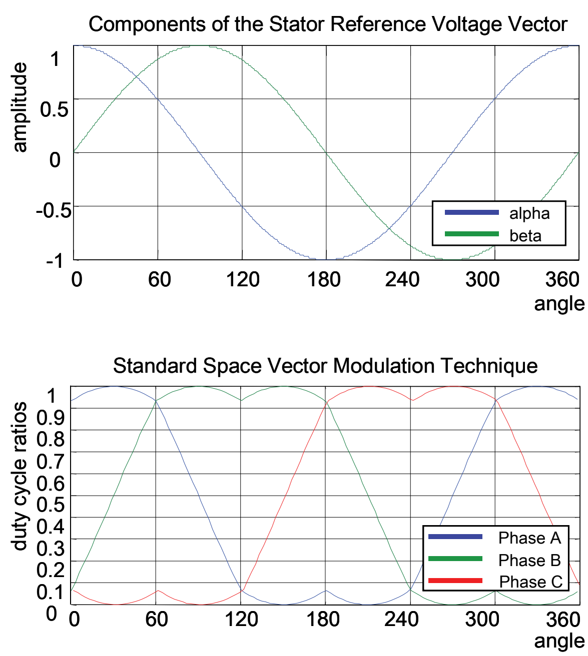

Figure 3-11. Standard Space Vector Modulation Technique

3.6.7 Returns

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

3.6.8 Range Issues

To provide an accurate calculation of the duty-cycle ratios, direct-α and quadrature-β 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 $\sqrt{\alpha^2 + \beta^2} \le 1$ 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

```
#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

Code Size (bytes)	119	
Data Size (bytes)	0	
Execution Clock	Min.	72/72 cycles
	Max.	82/84 cycles

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 O_{000} 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 3-21. Function Arguments

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

Table 3-22. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	d component
	f16Beta	In	SF16	0x8000... 0x7FFF	q component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A phase
	f16B	Out	SF16	0x8000... 0x7FFF	B phase
	f16C	Out	SF16	0x8000... 0x7FFF	C phase

3.7.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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 O_{000} Nulls.

The derivation approach of the Space Vector Modulation technique with O_{000} 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 switching duty-cycle ratios of the respective phases:

$$t_1 = 0 \qquad \textbf{\textit{Eqn. 3-35}}$$

$$t_2 = t_1 + t\_1 \qquad \textbf{\textit{Eqn. 3-36}}$$

$$t_3 = t_2 + t\_2 \qquad \textbf{\textit{Eqn. 3-37}}$$

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.

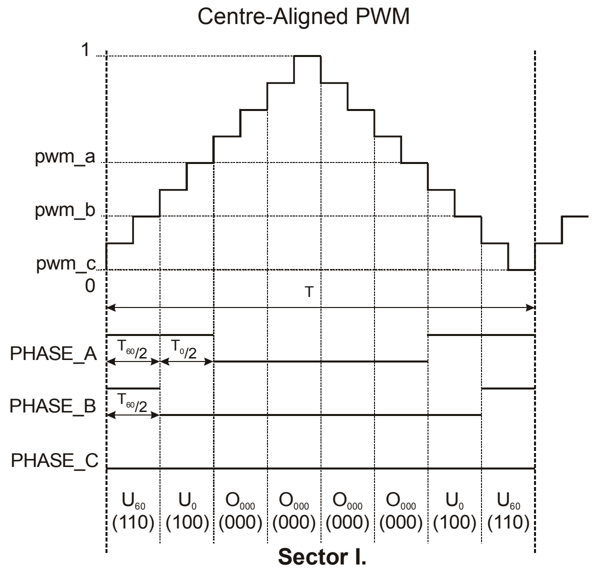

Figure 3-12. Space Vector Modulation Technique with O000 Nulls – Center-Aligned PWM

Figure 3-13 shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with O_{000} Nulls

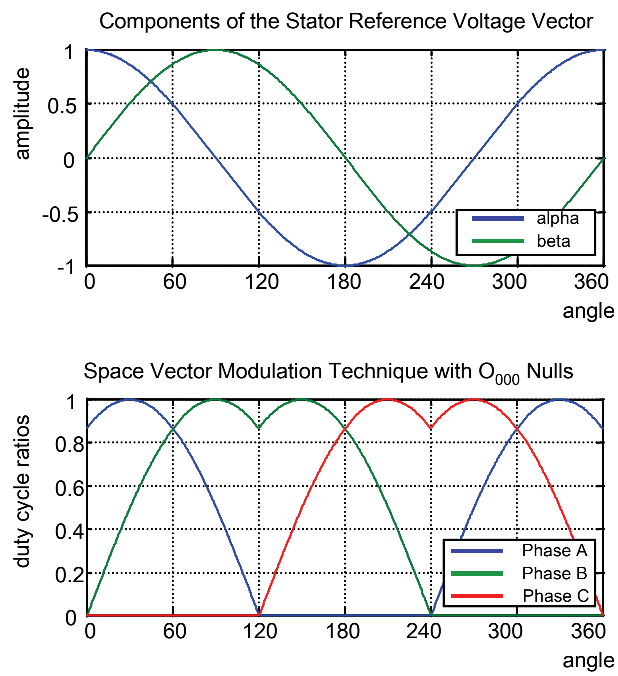

Figure 3-13. Space Vector Modulation Technique with O000 Nulls

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 $\sqrt{\alpha^2 + \beta^2} \le 1$ must be met.

3.7.9 Special Issues

The function **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 **MCLIB_SvmU0n** requires the saturation mode to be SET OFF!

3.7.10 Implementation

The **MCLIB_SvmU0n** function is implemented as a function call.

Example 3-6.

```
#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);
}
```

3.7.11 See Also

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

3.7.12 Performance

Table 3-23. Performance of MCLIB_SvmU0n Function

Code Size (words)	91 words	
Data Size (words)	0 words	
Execution Clock	Min	64/66 cycles
	Max	75/78 cycles

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

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

3.8.2 Prototype

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

3.8.3 Arguments

Table 3-24. Function Arguments

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

Table 3-25. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	d component
	f16Beta	In	SF16	0x8000... 0x7FFF	q component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A phase
	f16B	Out	SF16	0x8000... 0x7FFF	B phase
	f16C	Out	SF16	0x8000... 0x7FFF	C phase

3.8.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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 O_{111} Nulls.

The derivation approach of the Space Vector Modulation technique with O_{111} 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:

$$t_1 = T - t\_1 - t\_2 \qquad \textbf{\textit{Eqn. 3-38}}$$

$$t_2 = t_1 + t\_1 \qquad \textbf{\textit{Eqn. 3-39}}$$

$$t_3 = t_2 + t\_2 \qquad \textbf{\textit{Eqn. 3-40}}$$

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 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 3-14.

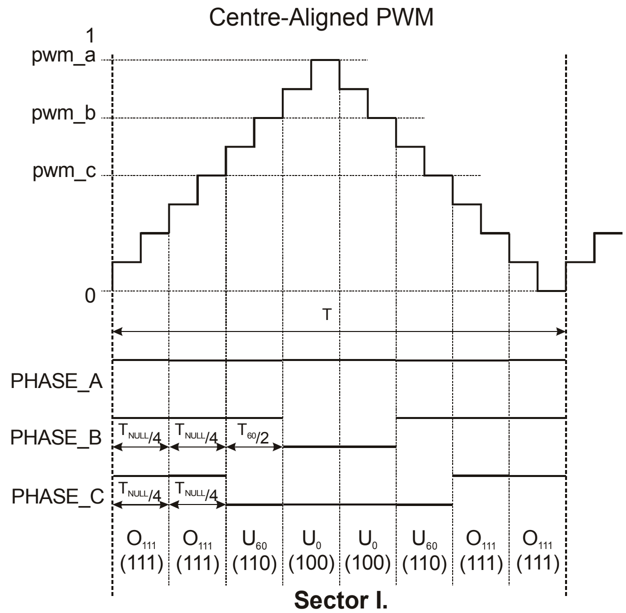

Figure 3-14. Space Vector Modulation Technique with O111 Nulls - Center-Aligned PWM

Figure 3-14 graphically shows calculated waveforms of the duty cycle ratios using Space Vector Modulation with O_{111} Nulls

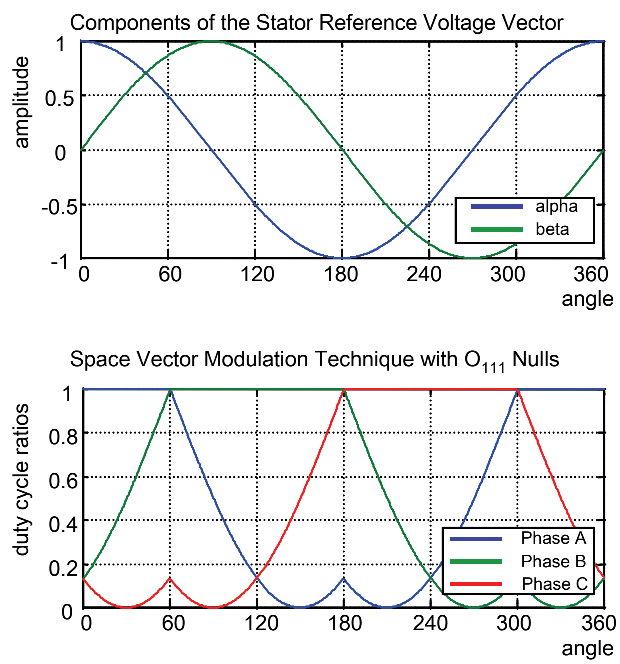

Figure 3-15. Space Vector Modulation with O111 Nulls

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 $\sqrt{\alpha^2 + \beta^2} \le 1$ 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

```
#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

Code Size (words)	99 words	
Data Size (words)	0 words	
Execution Clock	Min	66/66 cycles
	Max	76/78 cycles

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

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

3.9.2 Prototype

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

3.9.3 Arguments

Table 3-27. Function Arguments

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

Table 3-28. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	d component
	f16Beta	In	SF16	0x8000... 0x7FFF	q component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A phase
	f16B	Out	SF16	0x8000... 0x7FFF	B phase
	f16C	Out	SF16	0x8000... 0x7FFF	C phase

3.9.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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 O_{000} in the even sectors and state O_{111} 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".

$$t_1 = 0 \tag{Eqn. 3-41}$$

$$t_2 = t_1 + t\_1 \tag{Eqn. 3-42}$$

$$t_3 = t_2 + t\_2 \tag{Eqn. 3-43}$$

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".

$$t_1 = T - t\_1 - t\_2 \tag{Eqn. 3-44}$$

$$t_2 = t_1 + t\_1 \tag{Eqn. 3-45}$$

$$t_3 = t_2 + t\_2 \tag{Eqn. 3-46}$$

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

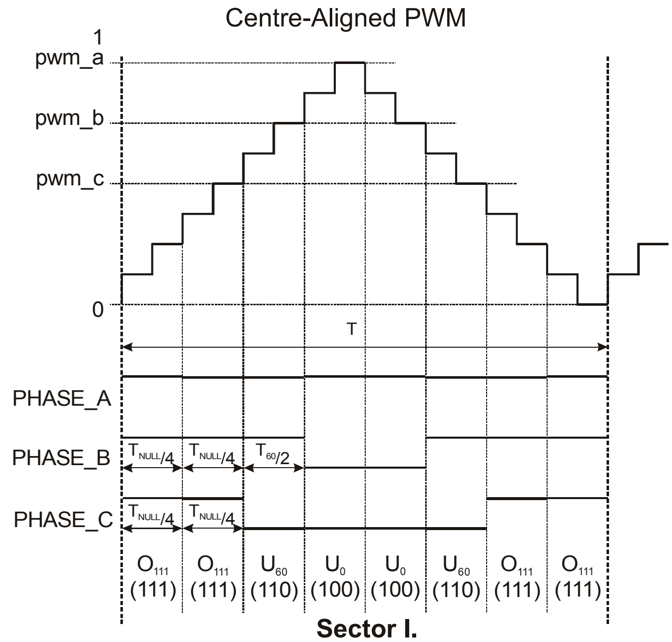

Figure 3-16. Space Vector Modulation Technique with Alternate Nulls – Center-Aligned PWM

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.

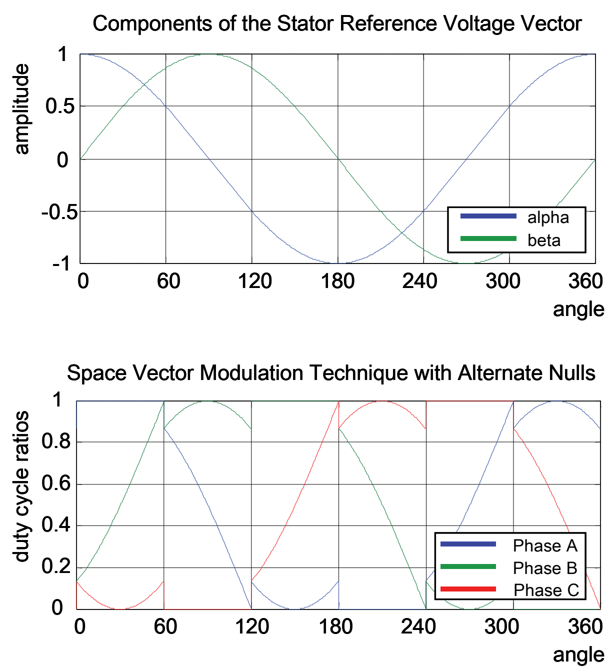

Figure 3-17. Space Vector Modulation Technique with Alternate Nulls

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

Example 3-8. 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_SvmAlt(&mudtAlphaBeta, &mudtAbc);
}
```

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

Code Size (words)	97 words	
Data Size (words)	0 words	
Execution Clock	Min	64/66 cycles
	Max	75/78 cycles

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 3-30. Function Arguments

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

Table 3-31. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	d component
	f16Beta	In	SF16	0x8000... 0x7FFF	q component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A phase
	f16B	Out	SF16	0x8000... 0x7FFF	B phase
	f16C	Out	SF16	0x8000... 0x7FFF	C phase

3.10.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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.

$$u_a = u_\alpha \qquad \textbf{\textit{Eqn. 3-47}}$$

$$u_b = \frac{-u_\alpha + \sqrt{3} \cdot u_\beta}{2} \qquad \textbf{\textit{Eqn. 3-48}}$$

$$u_c = \frac{-u_\alpha - \sqrt{3} \cdot u_\beta}{2} \qquad \textbf{\textit{Eqn. 3-49}}$$

2. An amplitude of the basic duty-cycle ratios u_a, u_b and u_c calculated by Equation 3-47, Equation 3-48and Equation 3-49 is in the range [-1, 1]. The basic duty-cycle ratios are then multiplied by the coefficient $2/(\sqrt{3})$

$$u'_a = \frac{2}{\sqrt{3}} \cdot u_a \qquad \textbf{\textit{Eqn. 3-50}}$$

$$u'_b = \frac{2}{\sqrt{3}} \cdot u_b \qquad \textbf{\textit{Eqn. 3-51}}$$

$$u'_c = \frac{2}{\sqrt{3}} \cdot u_c \qquad \textbf{\textit{Eqn. 3-52}}$$

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*.

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:

$$u_0 = \begin{cases} 1.0 - u'_a & \text{if } u'_a > 1.0 \\ -1.0 - u'_a & \text{if } u'_a < -1.0 \\ 0 & otherwise \end{cases} \qquad \textbf{Eqn. 3-53}$$

$$u_0 = \begin{cases} 1.0 - u'_b & \text{if } u'_b > 1.0 \\ -1.0 - u'_b & \text{if } u'_b < -1.0 \\ 0 & otherwise \end{cases} \qquad \textbf{Eqn. 3-54}$$

$$u_0 = \begin{cases} 1.0 - u'_c & \text{if } u'_c > 1.0 \\ -1.0 - u'_c & \text{if } u'_c < -1.0 \\ 0 & otherwise \end{cases} \qquad \textbf{Eqn. 3-55}$$

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:

$$pwm\_\mathrm{a} = \frac{1}{2} \cdot (u_0 + u'_a + 1) \qquad \textbf{Eqn. 3-56}$$

$$pwm\_\mathrm{b} = \frac{1}{2} \cdot (u_0 + u'_b + 1) \qquad \textbf{Eqn. 3-57}$$

$$pwm\_\mathrm{c} = \frac{1}{2} \cdot (u_0 + u'_c + 1) \qquad \textbf{Eqn. 3-58}$$

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

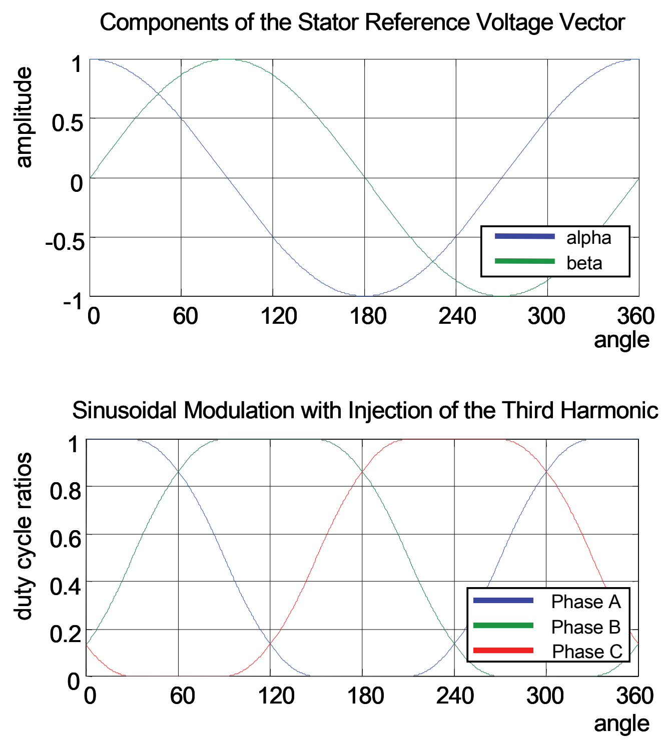

Figure 3-18. Sinusoidal Modulation with Injection of the Third Harmonic

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¨ $\sqrt{\alpha^2 + \beta^2} \le 1$ 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

```
#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

Code Size (words)	124 words	
Data Size (words)	7 words	
Execution Clock	Min	126/123 cycles
	Max	155/154 cycles

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

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

3.11.2 Prototype

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

3.11.3 Arguments

Table 3-33. Function Arguments

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

Table 3-34. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In	SF16	0x8000... 0x7FFF	d component
	f16Beta	In	SF16	0x8000... 0x7FFF	q component
MCLIB_3_COOR_SYST_T	f16A	Out	SF16	0x8000... 0x7FFF	A phase
	f16B	Out	SF16	0x8000... 0x7FFF	B phase
	f16C	Out	SF16	0x8000... 0x7FFF	C phase

3.11.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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:

$$u_{ref1} = u_{\beta} \qquad \textbf{Eqn. 3-59}$$

$$u_{ref2} = \frac{-u_{\beta} + \sqrt{3} \cdot u_{\alpha}}{2} \qquad \textbf{Eqn. 3-60}$$

$$u_{ref3} = \frac{-u_{\beta} - \sqrt{3} \cdot u_{\alpha}}{2} \qquad \textbf{Eqn. 3-61}$$

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:

$$u_0 = \left\{ \begin{bmatrix} 1.0 & \text{if} \quad u_{ref1} > 0 \\ 0 & otherwise \end{bmatrix} \right. \qquad \textbf{Eqn. 3-62}$$

$$u_0 = \left\{ \begin{bmatrix} 2.0 & \text{if} \quad u_{ref2} > 0 \\ 0 & otherwise \end{bmatrix} \right. \qquad \textbf{Eqn. 3-63}$$

$$u_0 = \left\{ \begin{bmatrix} 4.0 & \text{if} \quad u_{ref3} > 0 \\ 0 & otherwise \end{bmatrix} \right. \qquad \textbf{Eqn. 3-64}$$

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

Sector*	1	2	3	4	5	6
Sector	2	6	1	4	3	5

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 and u_c). The original equations of the Inverse Clark Transformation are scaled here to provide the duty-cycle ratios in the range 0 < pwm_x < 1, where x refers to the corresponding phases. These scaled duty-cycle ratios pwm_a, pwm_b and pwm_c might be used directly by the registers of the PWM block.

$$pwm\_a = 0.5 + \frac{u_\alpha}{2} \qquad \textbf{Eqn. 3-65}$$

$$pwm\_b = 0.5 + \frac{-u_\alpha + \sqrt{3} \cdot u_\beta}{4} \qquad \textbf{Eqn. 3-66}$$

$$pwm\_c = 0.5 + \frac{-u_\alpha - \sqrt{3} \cdot u_\beta}{4} \qquad \textbf{Eqn. 3-67}$$

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

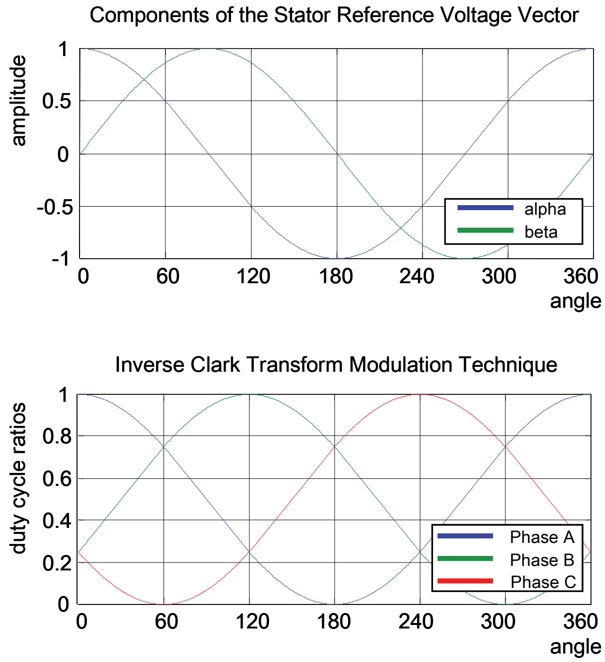

Figure 3-19. Inverse Clark Modulation Technique

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 $\sqrt{\alpha^2 + \beta^2} \le 1$ 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 3-36. Performance of MCLIB_PwmIct function

Code Size (words)	59 words	
Data Size (words)	7 words	
Execution Clock	Min	79/77 cycles
	Max	79/77 cycles

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

```
#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

```
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)
```

3.12.3 Arguments

Table 3-37. Function Arguments

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

Table 3-38. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_D_Q_T	f16D	In/out	SF16	0x8000... 0x7FFF	d component
	f16Q	In/out	SF16	0x8000... 0x7FFF	q component
MCLIB_DECOUPLING_PMSM_PARAM_T	f16Kd	In	SF16	0x8000... 0x7FFF	d axis stator inductance
	i16KdScale	In	SI16	–F...F	d axis stator inductance scale
	f16Kq	In	SF16	0x8000... 0x7FFF	q axis stator inductance
	i16KqScale	In	SI16	–F...F	q axis stator inductance scale

3.12.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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:

$$u_d = R_s \times i_d + L_d \times \frac{di_d}{dt} - \underline{L_q \times \omega_{el} \times i_q} \qquad \textbf{Eqn. 3-68}$$

$$u_q = R_s \times i_q + L_q \times \frac{di_q}{dt} + \underline{L_d \times \omega_{el} \times i_d} + \omega_{el} \times K \qquad \textbf{Eqn. 3-69}$$

where:

u_d, u_q — d,q voltage

R_s — stator winding resistance

L_d, L_q — stator winding inductance

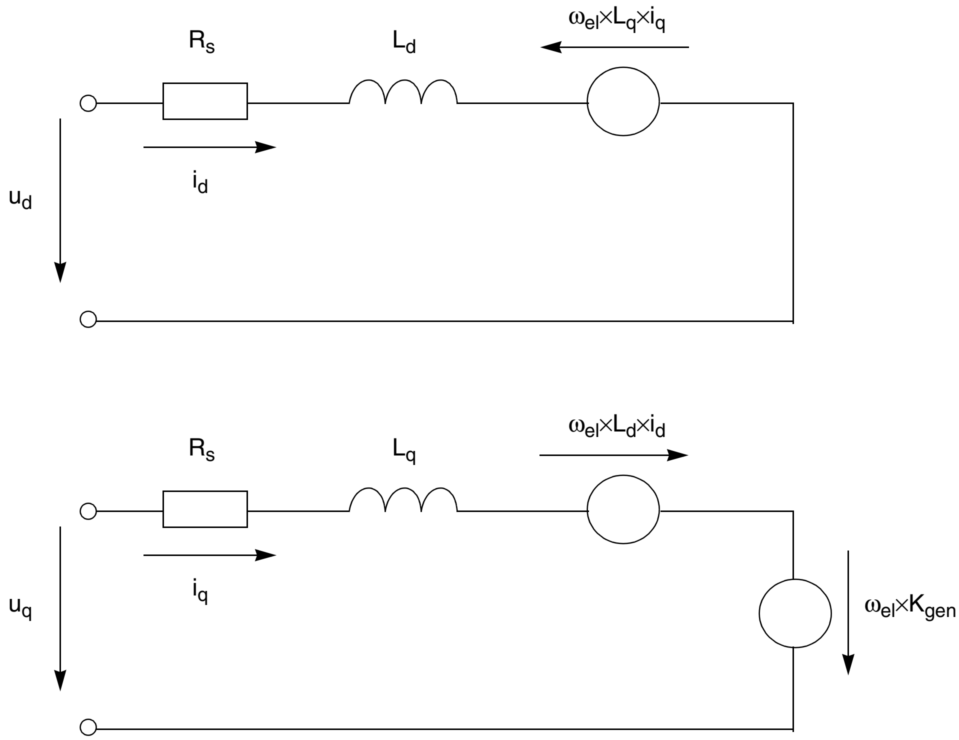

Figure 3-20. The d-q Model

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:

$$u_{ddec} = u_d - L_q \times \omega_{el} \times i_q \quad \textit{Eqn. 3-70}$$

$$u_{qdec} = u_q + L_d \times \omega_{el} \times i_d \quad \textit{Eqn. 3-71}$$

where:

u_{ddec}, u_{qdec} — decoupled d,q voltage output from the algorithm

The fractional representation of the d-component equation is:

$$u_{ddecs} = u_{ds} - \omega_{el} \times i_q \times \left(L_q \times \omega_{max} \times \frac{i_{max}}{u_{max}}\right) \quad \textit{Eqn. 3-72}$$

$$k_q = L_q \times \omega_{max} \times \frac{i_{max}}{u_{max}} \quad \textit{Eqn. 3-73}$$

$$u_{ddecs} = u_{ds} - \omega_{el} \times i_q \times k_q \quad \textit{Eqn. 3-74}$$

The fractional representation of the q-component equation is:

$$u_{qdecs} = u_{qs} + \omega_{el} \times i_d \times \left(L_d \times \omega_{max} \times \frac{i_{max}}{u_{max}}\right)$$ **Eqn. 3-75**

$$k_d = L_d \times \omega_{max} \times \frac{i_{max}}{u_{max}}$$ **Eqn. 3-76**

$$u_{qdecs} = u_{qs} + \omega_{el} \times i_d \times k_d$$ **Eqn. 3-77**

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

$$0{,}5 \le k_q \times 2^{-qsc} < 1$$ **Eqn. 3-78**

$$0{,}5 \le k_d \times 2^{-dsc} < 1$$ **Eqn. 3-79**

Then the scaled parameters can be defined as:

$$k_{qsc} = k_q \times 2^{-qsc}$$ **Eqn. 3-80**

$$k_{dsc} = k_d \times 2^{-dsc}$$ **Eqn. 3-81**

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

$$qsc \le \frac{\log k_q - \log 0{,}5}{\log 2}$$ **Eqn. 3-82**

$$qsc > \frac{\log k_q}{\log 2}$$ **Eqn. 3-83**

$$dsc \le \frac{\log k_d - \log 0{,}5}{\log 2}$$ **Eqn. 3-84**

$$dsc > \frac{\log k_d}{\log 2}$$ **Eqn. 3-85**

So the final fractional equations with scaling are:

$$u_{ddecs} = u_{ds} - (\omega_{el} \times i_q \times k_{qsc}) \times 2^{qsc}$$ **Eqn. 3-86**

$$u_{qdecs} = u_{qs} + (\omega_{el} \times i_d \times k_{dsc}) \times 2^{dsc}$$ **Eqn. 3-87**

The principle of the algorithm use is depicted in Figure 3-21 where:

i_{ddes}, i_{qdes} — 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_{ddec}, u_{qdec} — decoupled d, q voltages
ω_{el} — electrical angular velocity

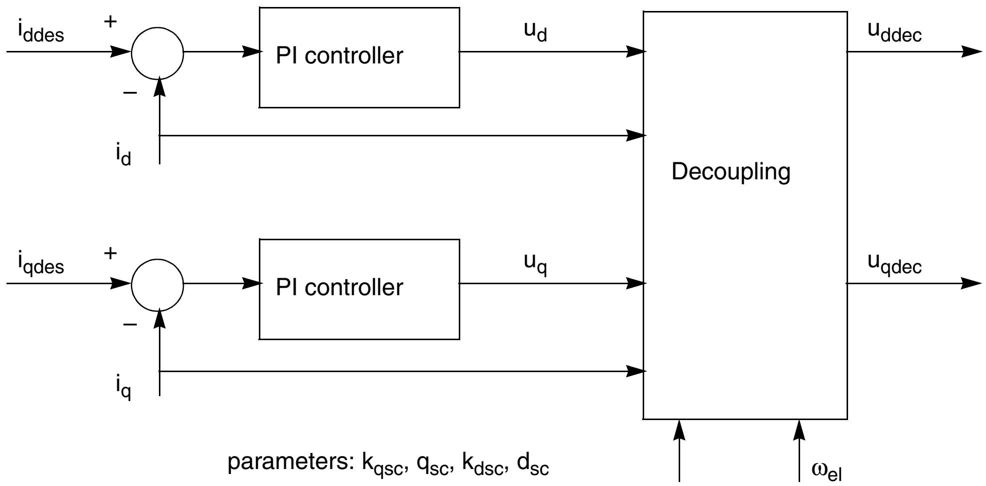

Figure 3-21. Algorithm Diagram

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.

Example 3-11. Implementation Code

```
#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)
```

```
{
        /* 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.f16Kd = FRAC16(0.8455);
        mudtDecouplingParam.i16KdScale = -5;
        mudtDecouplingParam.f16Kq = FRAC16(0.5095);
        mudtDecouplingParam.i16KqScale = -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

Code Size (bytes)	50	
Data Size (bytes)	0	
Execution Clock	Min.	61/62 cycles
	Max.	75/76 cycles

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 3-40. Function Arguments

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

Table 3-41. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In/out	SF16	0x8000... 0x7FFF	Alpha component
	f16Beta	In/out	SF16	0x8000... 0x7FFF	Beta component

3.13.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

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-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:

$$\text{f16Alpha*} = \begin{cases} \dfrac{\text{f16InvModIndex} \cdot \text{f16Alpha}}{\dfrac{\text{f16DcBusMsr}}{2}} & \text{if } |\text{f16InvModIndex} \cdot \text{f16Alpha}| < \dfrac{\text{f16DcBusMsr}}{2} \\ sgn(\text{f16Alpha}) \cdot 1.0 & \text{otherwise} \end{cases} \qquad \textbf{Eqn. 3-88}$$

$$\text{f16Beta*} = \begin{cases} \dfrac{\text{f16InvModIndex} \cdot \text{f16Beta}}{\dfrac{\text{f16DcBusMsr}}{2}} & \text{if } |\text{f16InvModIndex} \cdot \text{f16Beta}| < \dfrac{\text{f16DcBusMsr}}{2} \\ sgn(\text{f16Beta}) \cdot 1.0 & \text{otherwise} \end{cases} \qquad \textbf{Eqn. 3-89}$$

where y = sgn (x) function is defined as follows:

$$y = \begin{cases} 1.0 & \text{if } \ x \ge 0 \\ -1.0 & \text{otherwise} \end{cases} \qquad \textbf{Eqn. 3-90}$$

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.

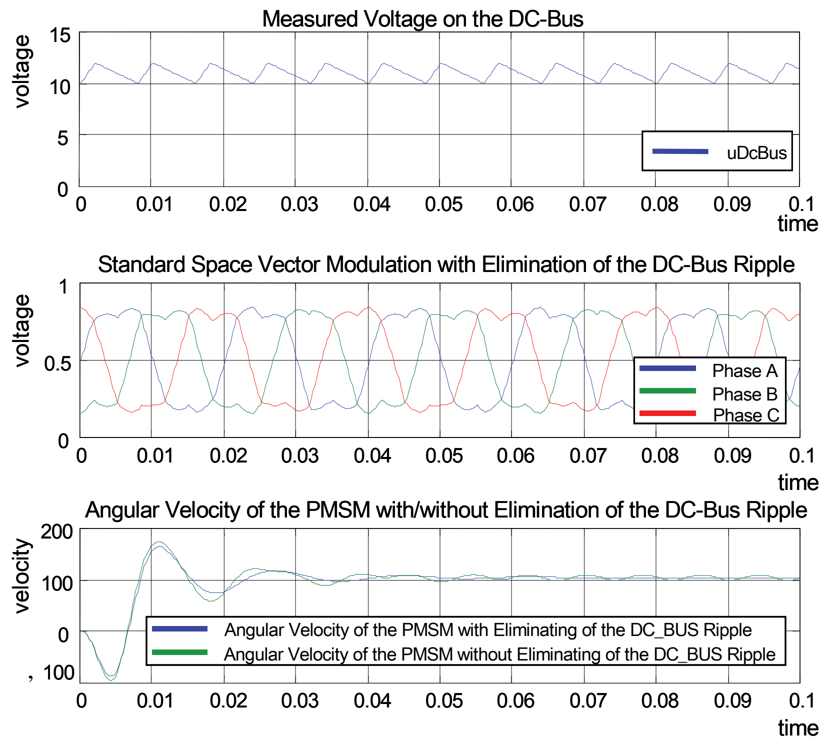

Figure 3-22. Results of the DC-Bus Voltage Ripple Elimination

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:

— -f16DcBusMsr / (2 · f16InvModIndex) < x < f16DcBusMsr / (2 · 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

```
#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

Code Size (bytes)	36	
Data Size (bytes)	0	
Execution Clock	Min.	68 cycles
	Max.	68 cycles

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 3-43. Function Arguments

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

Table 3-44. User Type Definitions

Typedef	Name	In/Out	Format	Range	Description
MCLIB_2_COOR_SYST_ALPHA_BETA_T	f16Alpha	In/out	SF16	0x8000... 0x7FFF	Alpha component
	f16Beta	In/out	SF16	0x8000... 0x7FFF	Beta component

3.14.4 Availability

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

This library module is targeted for the DSC 56F80xx platform.

3.14.5 Dependencies

List of all dependent files:

- MCLIB_ElimDcBusRipGenAsm.h
- MCLIB_types.h

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:

$$\text{f16Alpha*} = \begin{cases} \dfrac{\text{f16Alpha}}{\text{f16DcBusMsr}} & \text{if } |\text{f16Alpha}| < \dfrac{\text{f16DcBusMsr}}{2} \\ sgn(\text{f16Alpha}) \cdot 1.0 & \text{otherwise} \end{cases}$$

Eqn. 3-91

$$\text{f16Beta*} = \begin{cases} \dfrac{\text{f16Beta}}{\text{f16DcBusMsr}} & \text{if } |\text{f16Beta}| < \dfrac{\text{f16DcBusMsr}}{2} \\ sgn(\text{f16Beta}) \cdot 1.0 & \text{otherwise} \end{cases}$$

Eqn. 3-92

Eqn. 3-93

where y = sgn (x) function is defined as follows:

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

Eqn. 3-94

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

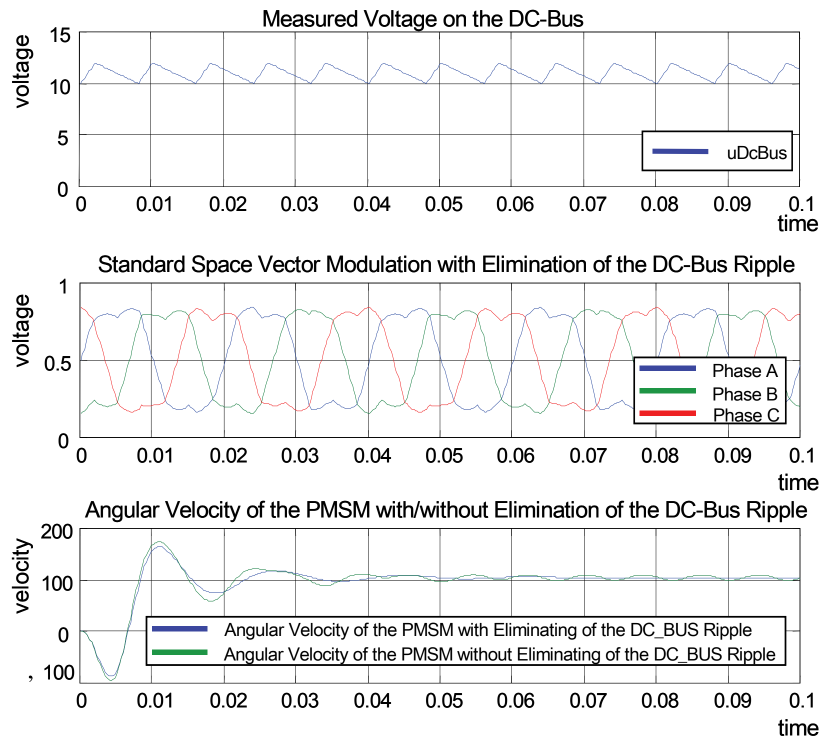

Figure 3-23. Results of the DC-Bus Voltage Ripple Elimination

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:

- 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:
  — -f16DcBusMsr / 1.73 < x < f16DcBusMsr / 1.73 in case of SVM with the 3rd harmonic injection, and/or -f16DcBusMsr / 2 < x < f16DcBusMsr / 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

```
#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

Code Size (bytes)	32	
Data Size (bytes)	0	
Execution Clock	Min.	63 cycles
	Max.	63 cycles

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

```
#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 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 3-46. Function Arguments

Name	In/Out	Format	Range	Description
*pudtInVector	In	N/A	N/A	Pointer to a structure containing input vectors.
*pudtLimVector	In	N/A	N/A	Pointer to a structure containing output vectors.
*pudtParams	In	N/A	N/A	Pointer to a structure containing the f16Lim and blnLimFlag.

Table 3-47. User Type Definitions

<table>
<tr><th>Typedef</th><th>Name</th><th>In/Out</th><th>Format</th><th>Range</th><th>Description</th></tr>
<tr><td rowspan="2">MCLIB_VECTOR_LIMIT_PARAMS_T</td><td>f16Lim</td><td>In</td><td>SF16</td><td>0x8000...
0x7FFF</td><td>Value of the input vector amplitude limit.</td></tr>
<tr><td>blnLimFlag</td><td>Out</td><td>SI16</td><td>0x8000...
0x7FFF</td><td>True/False flag describing the status of the limiter;
true — signal is limited
false — signal is not limited</td></tr>
</table>

3.15.4 Availability

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

This library module is targeted for the MCF51xx platform.

3.15.5 Dependencies

List of all dependent files:

- 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:

$$\mathrm{mod}^2 = \mathrm{pudtInVector.f16A}^2 + \mathrm{pudtInVector.f16B}^2 \qquad \textit{Eqn. 3-94}$$

$$\mathrm{pudtLimVector.f16A} = \begin{cases} \dfrac{\mathrm{f16Lim} \cdot \mathrm{mod}}{\mathrm{pudtInVector.f16A}} & \text{if } \mathrm{mod}^2 > \mathrm{f16Lim}^2 \\ \mathrm{pudtInVector.f16A} & \text{if } \mathrm{mod}^2 \le \mathrm{f16Lim}^2 \end{cases} \qquad \textit{Eqn. 3-95}$$

$$\mathrm{pudtLimVector.f16B} = \begin{cases} \dfrac{\mathrm{f16Lim} \cdot \mathrm{mod}}{\mathrm{pudtInVector.f16B}} & \text{if } \mathrm{mod}^2 > \mathrm{f16Lim}^2 \\ \mathrm{pudtInVector.f16B} & \text{if } \mathrm{mod}^2 \le \mathrm{f16Lim}^2 \end{cases} \qquad \textit{Eqn. 3-96}$$

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

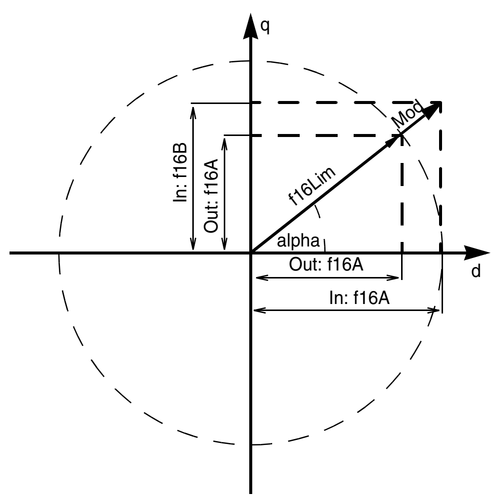

Figure 3-24. Input and Limited Output Vectors Relationship

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.

3.15.7 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

```
#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

Code Size (bytes)	51 + 65 (GFLIB_SqrtPoly)	
Data Size (bytes)	0 + 34 (GFLIB_SqrtPoly)	
Execution Clock	Min.	35/35 cycles
	Max.	165/155 cycles

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

```
#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 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

Name	In/Out	Format	Range	Description
*pudtInVector	In	N/A	N/A	Pointer to a structure containing input vectors.
*pudtLimVector	In	N/A	N/A	Pointer to a structure containing output vectors.
*pudtParams	In	N/A	N/A	Pointer to a structure containing the f16Lim and blnLimFlag.

Table 3-50. User Type Definitions

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

3.16.4 Availability

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

This library module is targeted for the MCF51xx platform.

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:

$$\mathrm{mod}^2 = \mathrm{pudtInVector.f16A}^2 + \mathrm{pudtInVector.f16B}^2 \qquad \textit{Eqn. 3-97}$$

$$\mathrm{pudtLimVector.f16A} = \begin{cases} \dfrac{\mathrm{f16Lim} \cdot \mathrm{mod}}{\mathrm{pudtInVector.f16A}} & \text{if } \mathrm{mod}^2 > \mathrm{f16Lim}^2 \\ \mathrm{pudtInVector.f16A} & \text{if } \mathrm{mod}^2 \le \mathrm{f16Lim}^2 \end{cases} \qquad \textit{Eqn. 3-98}$$

$$\mathrm{pudtLimVector.f16B} = \begin{cases} \dfrac{\mathrm{f16Lim} \cdot \mathrm{mod}}{\mathrm{pudtInVector.f16B}} & \text{if } \mathrm{mod}^2 > \mathrm{f16Lim}^2 \\ \mathrm{pudtInVector.f16B} & \text{if } \mathrm{mod}^2 \le \mathrm{f16Lim}^2 \end{cases} \qquad \textit{Eqn. 3-99}$$

The relationship between the input and limited output vectors is obvious from Figure 3-25.

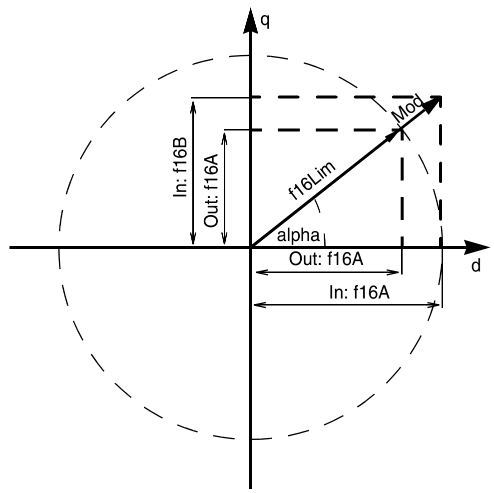

Figure 3-25. Input and Limited Output Vectors Relationship

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 **MCLIB_VectorLimit12** function uses the function GFLIB_SqrtPoly from GFLIB.

The **MCLIB_VectorLimit12** function requires the saturation mode to be set.

3.16.9 Implementation

The **MCLIB_VectorLimit12** function is implemented as a function call.

Example 3-15. Implementation Code

```
#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_VectorLimit12(&mudtVector, &mudtLimitedVector,
&mudtVectorLimitParam);
}
```

3.16.10 See Also

See **MCLIB_VectorLimit** for more information.

3.16.11 Performance

Table 3-51. Performance of the MCLIB_VectorLimit12 Function

Code Size (bytes)	49 + 28 (GFLIB_SqrtIter)	
Data Size (bytes)	0	
Execution Clock	Min.	35/35 cycles
	Max.	140/135 cycles

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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

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

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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150

56800E_MCLIB
Rev. 3, 5/2011