AN 2520 Sensorless FOC PLL Estimator Demonstration Help

AN2520 Sensorless FOC PLL Estimator Demonstration Help

Introduction

This document provides detailed instructions on how to run the demonstration for the Sensorless Field Oriented Control (FOC) of a PMSM motor using a PLL based estimator on the PIC32MK family of MCUs.

About the Demonstration

This section introduces the demonstration.

Description

This demonstration is designed to work with the PIC32MK 100-pin Motor Control PIM mounted on the dsPICDEM™ MCLV-2 Development Board for the Small Hurst Motor (DMB0224C10002) using a dual shunt configuration. This demo leverages the Floating Point Unit for handling all floating point arithmetic and therefore, does not need any tedious fixed point scaling. The variables containing physical values such as current, voltage, electrical speed, and rotor angle, are in real units unlike in a fixed point implementation where the physical values are in normalized Q31 units.

Please refer to the Microchip Application Note AN2520 Sensorless Field Oriented Control (FOC) for a Permanent Magnet Synchronous Motor (PMSM) Using a PLL Estimator and Flux Weakening (FW) for theoretical details about the algorithm.

Hardware and Software Requirements

This section outlines the required hardware and software tools to run the Demonstration.

Description

- Software tools
  - MPLAB X IDE v4.05 or higher
  - Compiler - XC32 v1.44or higher
- Hardware tools – available on www.microchipdirect.com
  - dsPICDEM MCLV-2 Development Board (DM330021-2)
  - PIC32MK 100 pin Motor Control Plug In Module (MA320024)
  - Debugger or Programmer – PICkit 3 or above, MPLAB ICD-3 or above, or MPLAB ICE
  - 24V Power Supply (AC002013)
  - Small Hurst Motor - DMB0224C10002 (AC300020)

Hardware Setup

This demonstration outlines how to configure the hardware.

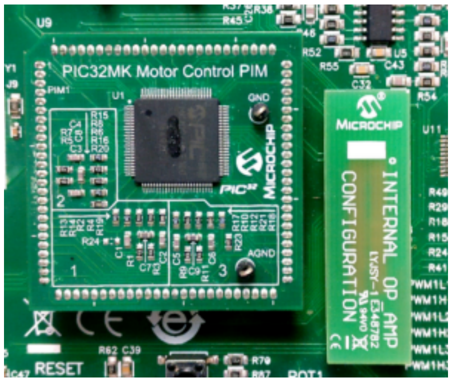
Description

Configure the hardware as outlined in the steps below to run the sensorless FOC demonstration based on AN2520. Attempt steps one through four while the board is de-energized.

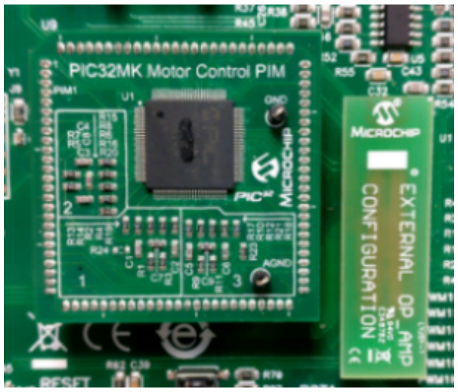
1. Mount the PIC32MK 100-pin Motor Control PIM on the dsPICDEM MCLV-2 Development Board.
2. This demo supports communication with the X2C-Scope plug-in, to monitor or plot any variables used in the application. The communication between the target and the X2C-Scope plug-in can be established using the RS-232 serial port (J10) or mini-USB port (J8). To use the RS-232 serial port, connect JP4 and JP5 to the UART position. To use the mini USB port, connect JP4 and JP5 to the USB position.
3. Connect the three-phase wires coming from P1 header of the small Hurst motor (i.e., Red, White, and Black to M1, M2, and M3 ports, respectively). Leave the green wire from the motor unconnected.
4. This demonstration allows the user to choose between using the external (on-board) op-amps, or internal (on-chip) op-amps for

signal conditioning of the current sense signal measured across the shunt resistor.

- For internal op-amps, connect the *Internal OP AMP Configuration* matrix board to header J14



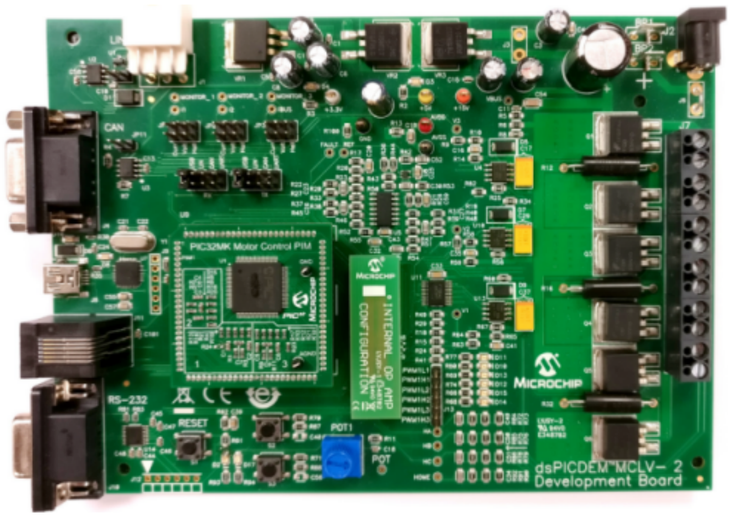
- For external op-amps, connect the *External OP AMP Configuration* matrix board to header J14



5. Connect the programmer and debugger using the J11 connector.
6. Apply 24V DC at BP1-BP2/J2.

Note:	Please refer to the dsPICDEM MCLV-2 Development Board User's Guide for safety and operating details of the development board.

PIC32MK PIM with dsPICDEM MCLV-2 Development Board Using Internal OPAMP Configuration



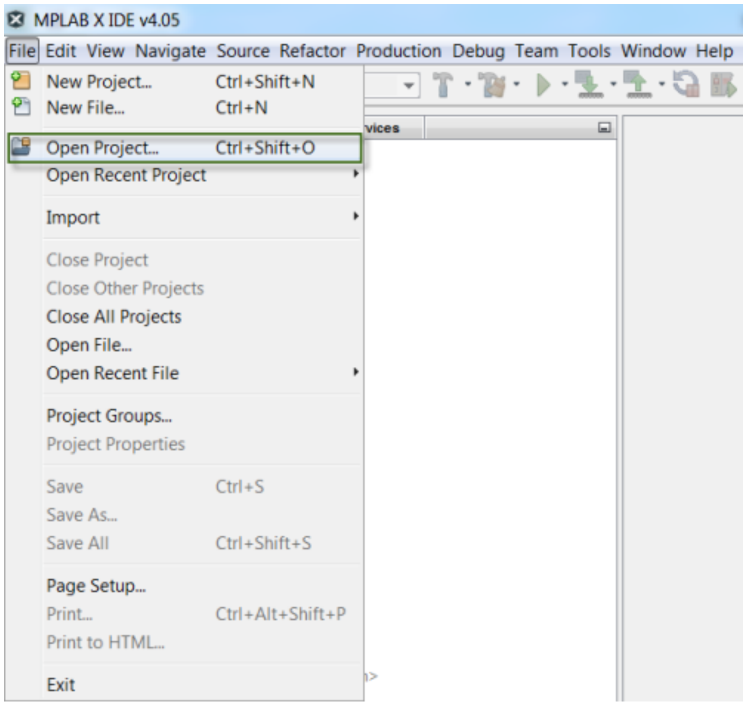
Software Setup

This section describes how to set up the software.

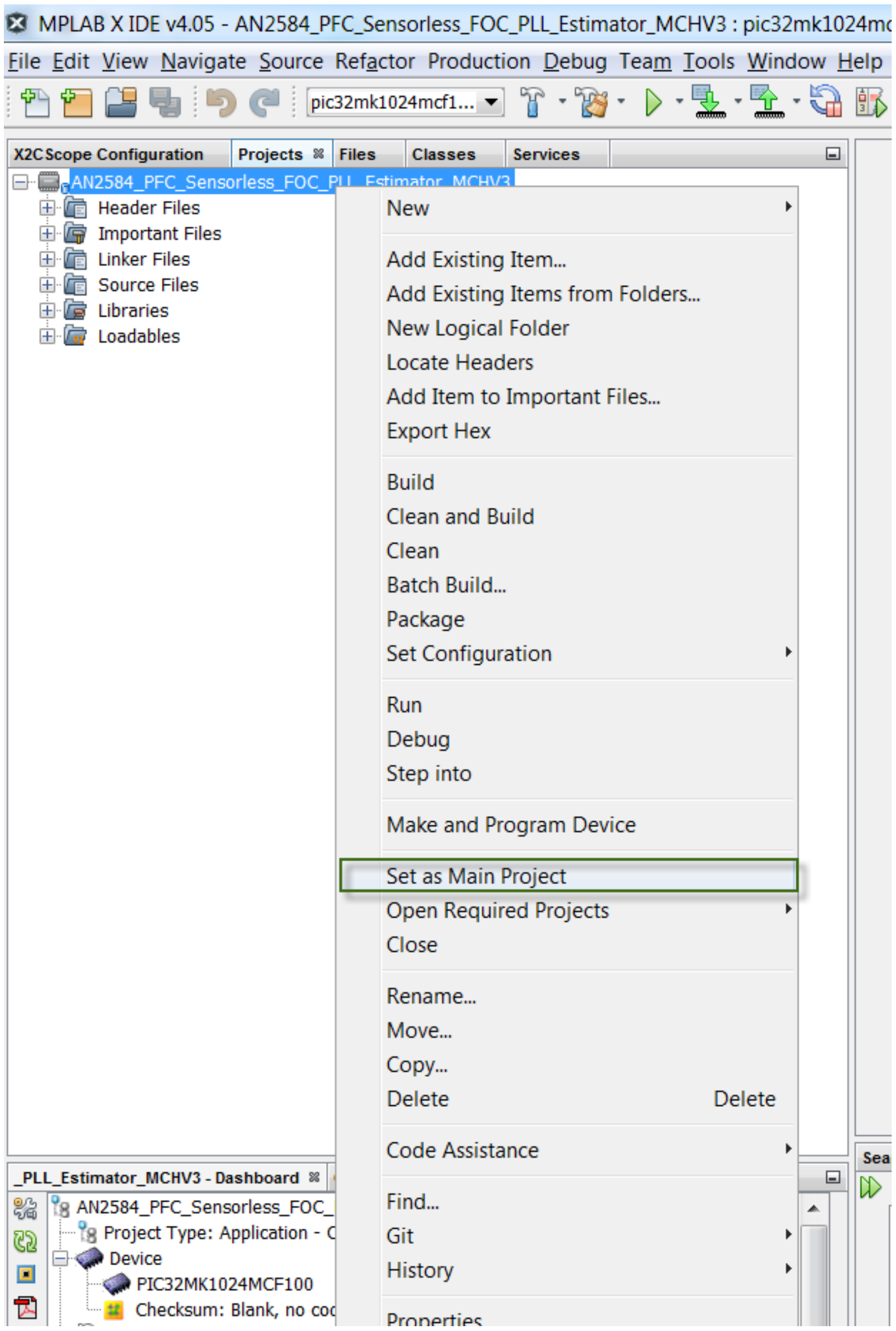
Description

To setup the software, use the following steps.

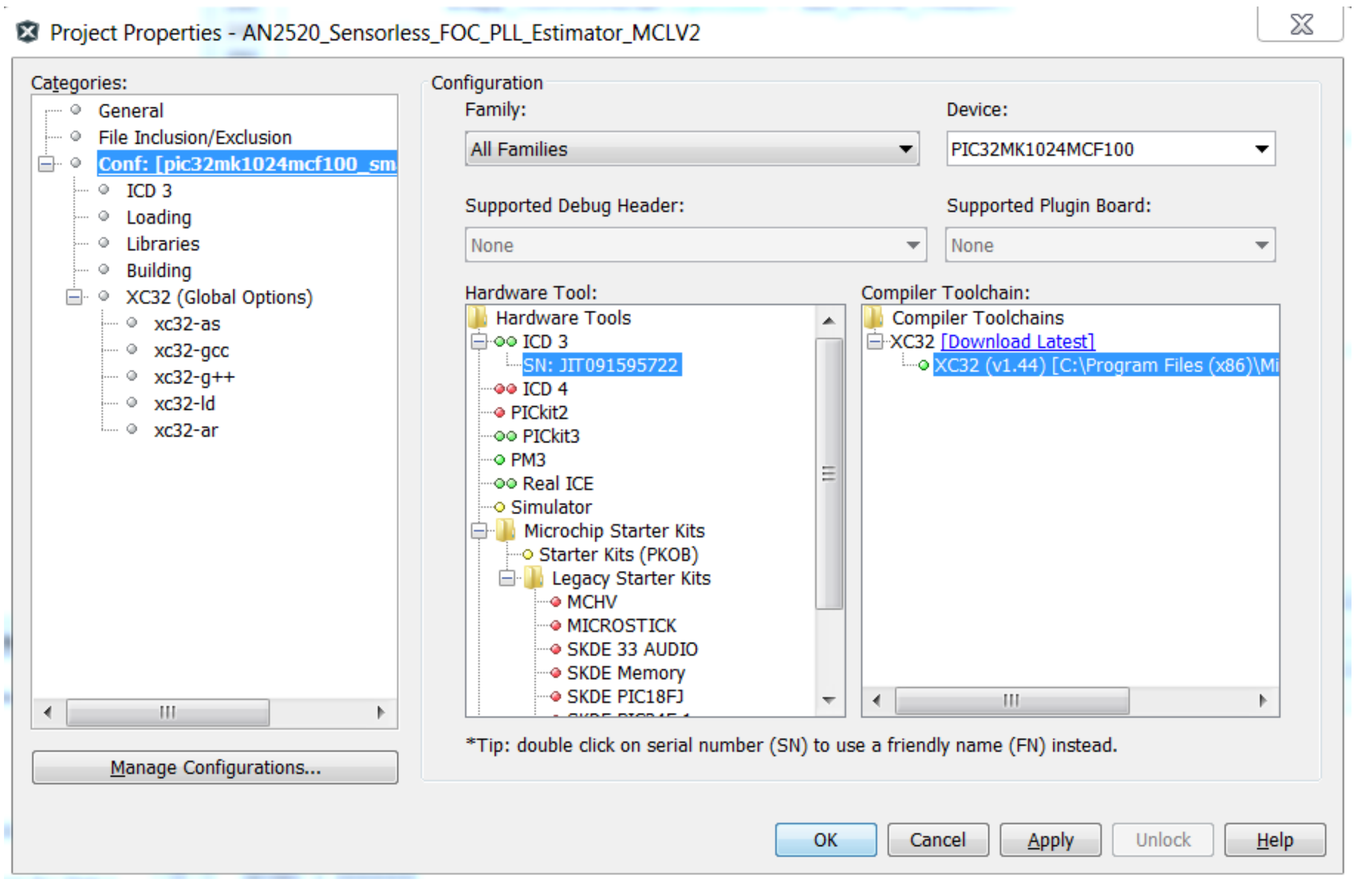
1. Start MPLAB X IDE and open the project AN2520_Sensorless_FOC_PLL_Estimator_MCLV2.x



2. Right click on AN2520_Sensorless_FOC_PLL_Estimator_MCLV2 project under *Projects* tab and click *Set as Main Project.* If the project is already selected as main project then skip this step.



3. Right click on AN2520_Sensorless_FOC_PLL_Estimator_MCLV2 project under the *Projects* tab and select properties.
4. Select the connected programmer or debugger, under *Hardware tools* and the respective XC32 compiler version under *Compiler Toolchains*. Click *Apply* and *OK*.


Project Properties - AN2520_Sensorless_FOC_PLL_Estimator_MCLV2

Categories:
- General
- File Inclusion/Exclusion
- Conf: [pic32mk1024mcf100_sm
  - ICD 3
  - Loading
  - Libraries
  - Building
  - XC32 (Global Options)
    - xc32-as
    - xc32-gcc
    - xc32-g++
    - xc32-ld
    - xc32-ar

Configuration

Family: All Families

Device: PIC32MK1024MCF100

Supported Debug Header: None

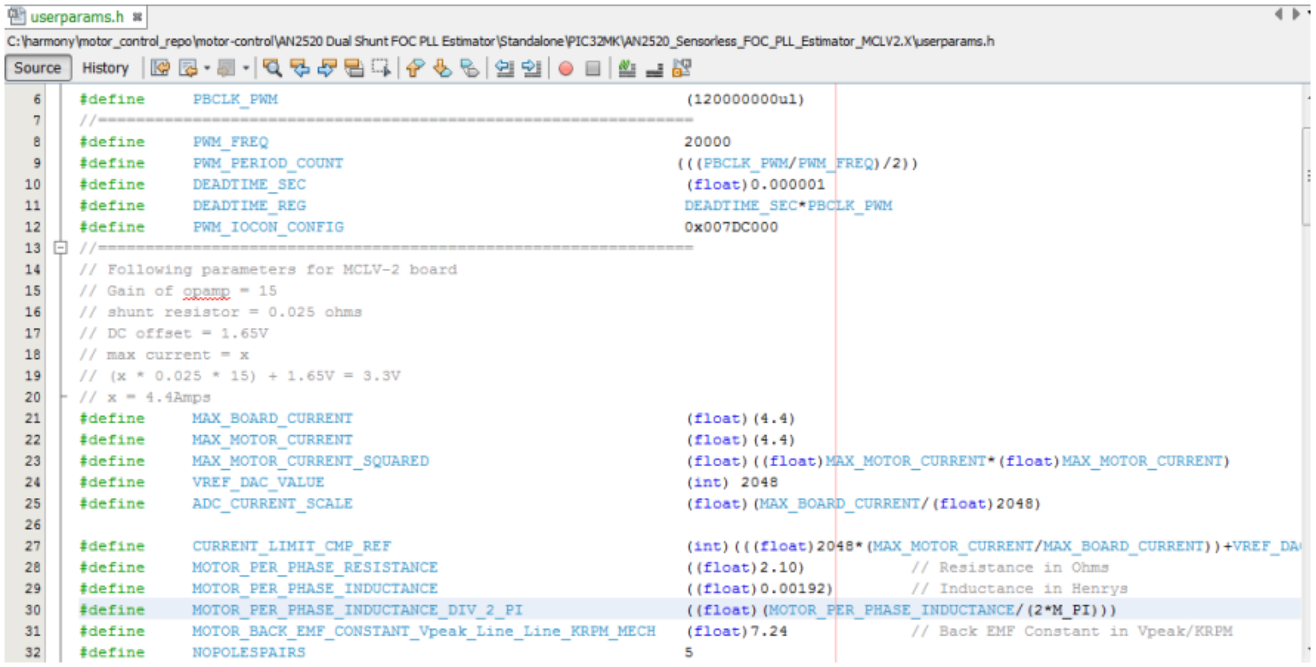
Supported Plugin Board: None

Hardware Tool:
- Hardware Tools
  - ICD 3
    - SN: JIT091595722
  - ICD 4
  - PICkit2
  - PICkit3
  - PM3
  - Real ICE
  - Simulator
  - Microchip Starter Kits
    - Starter Kits (PKOB)
    - Legacy Starter Kits
      - MCHV
      - MICROSTICK
      - SKDE 33 AUDIO
      - SKDE Memory
      - SKDE PIC18FJ

Compiler Toolchain:
- Compiler Toolchains
  - XC32 [Download Latest]
    - XC32 (v1.44) [C:\Program Files (x86)\Mi

*Tip: double click on serial number (SN) to use a friendly name (FN) instead.

5. Under *Header Files*, open the header file `userparams.h`. All the system level configuration parameters like PWM Frequency, motor resistance, motor inductance, etc. are defined as macros in this header file.


userparams.h

C:\harmony\motor_control_repo\motor-control\AN2520 Dual Shunt FOC PLL Estimator\Standalone\PIC32MK\AN2520_Sensorless_FOC_PLL_Estimator_MCLV2.X\userparams.h

```
#define     PBCLK_PWM                                   (120000000ul)
//==================================================================
#define     PWM_FREQ                                    20000
#define     PWM_PERIOD_COUNT                           (((PBCLK_PWM/PWM_FREQ)/2))
#define     DEADTIME_SEC                                (float)0.000001
#define     DEADTIME_REG                                DEADTIME_SEC*PBCLK_PWM
#define     PWM_IOCON_CONFIG                            0x007DC000
//==================================================================
// Following parameters for MCLV-2 board
// Gain of opamp = 15
// shunt resistor = 0.025 ohms
// DC offset = 1.65V
// max current = x
// (x * 0.025 * 15) + 1.65V = 3.3V
// x = 4.4Amps
#define     MAX_BOARD_CURRENT                           (float)(4.4)
#define     MAX_MOTOR_CURRENT                           (float)(4.4)
#define     MAX_MOTOR_CURRENT_SQUARED                   (float)((float)MAX_MOTOR_CURRENT*(float)MAX_MOTOR_CURRENT)
#define     VREF_DAC_VALUE                              (int) 2048
#define     ADC_CURRENT_SCALE                           (float)(MAX_BOARD_CURRENT/(float)2048)

#define     CURRENT_LIMIT_CMP_REF                       (int)(((float)2048*(MAX_MOTOR_CURRENT/MAX_BOARD_CURRENT))+VREF_DA
#define     MOTOR_PER_PHASE_RESISTANCE                  ((float)2.10)           // Resistance in Ohms
#define     MOTOR_PER_PHASE_INDUCTANCE                  ((float)0.00192)        // Inductance in Henrys
#define     MOTOR_PER_PHASE_INDUCTANCE_DIV_2_PI         ((float)(MOTOR_PER_PHASE_INDUCTANCE/(2*M_PI)))
#define     MOTOR_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH   (float)7.24     // Back EMF Constant in Vpeak/KRPM
#define     NOPOLESPAIRS                                5
```

The following table describes the macros which the user should modify to fine tune their application. The default values are tested to work with the dsPICDEM MCLV-2 development board and small hurst motor.

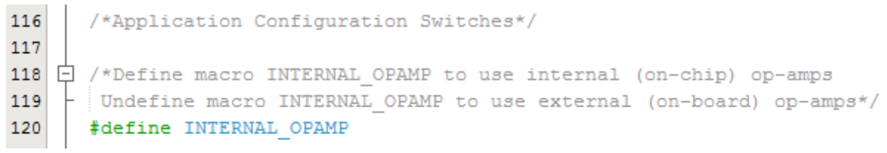
AN2520 Macro Definitions

Macro Name	Description	Units
PWM_FREQ	Motor Inverter PWM Frequency	Hz
DEADTIME_SEC	Dead-time	Sec

MAX_BOARD_CURRENT	Maximum Inverter DC bus current through the board that can be measured without saturating the ADC	A
MAX_MOTOR_CURRENT	Maximum Motor Phase Current	A
MOTOR_PER_PHASE_RESISTANCE	Motor per phase resistance	Ohms
MOTOR_PER_PHASE_INDUCTANCE	Motor per phase inductance	H
MOTOR_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH	Motor Back EMF constant	Vpeak (line-line)/ KRPM
NOPOLESPAIRS	Number of Motor Pole Pairs	-
MAX_ADC_COUNT	Full Scale ADC Count, $2^{n\ bit\ ADC}$ -1 (For 12 bit ADC, 2^{12}-1= 4095	-
MAX_ADC_INPUT_VOLTAGE	ADC Reference voltage	V
DCBUS_SENSE_TOP_RESISTOR	High side resistance of DC BUS Sense voltage divider	Ohms
DCBUS_SENSE_BOTTOM_RESISTOR	Low side resistance of DC BUS Sense voltage divider	Ohms
LOCK_TIME_IN_SEC	Rotor lock time to a forced rotor angle before spinning the motor	Sec
END_SPEED_RPM	Motor speed in open loop mode at which the algorithm switches to closed loop	RPM
RAMP_TIME_IN_SEC	Time to reach Open Loop End Speed (END_SPEED_RPM) during open loop operation	Sec
CL_RAMP_RATE_RPM_SEC	Speed Ramp Rate suring Closed Loop FOC operation	RPM/Sec
Q_CURRENT_REF_OPENLOOP	Q axis current during open loop operation	A
NOMINAL_SPEED_RPM	Maximum rated speed in constant torque mode (without field weakening)	RPM
MOVING_AVG_WINDOW_SIZE	Total number of current samples used to calculate moving average of the current to obtain ADC Current Offset = $2^{MOVING\_AVG\_WINDOW\_SIZE}$	-
CURRENT_OFFSET_MAX	Maximum limit of the ADC Current Offset	-
CURRENT_OFFET_MIN	Minimum limit of the ADC Current Offset	-
CURRENT_OFFSET_INIT	Initial value of the ADC Current Offset	-
KFILTER_ESDQ	First order low pass filter coefficient for Ed, Eq estimator parameters	-
KFILTER_VELESTIM	First order low pass filter coefficient for speed estimation	-
FW_SPEED_RPM	Maximum rated speed in Field Weakening Mode	RPM
MAX_FW_NEGATIVE_ID_REF	Maximum applicable negative D axis current	A
D_CURRCNTR_PTERM	D axis Current Controller Proportional Gain	-
D_CURRCNTR_ITERM	D axis Current Controller Integral Gain	-
D_CURRCNTR_CTERM	D axis Current Controller Anti-Windup Gain	-
D_CURRCNTR_OUTMAX	D axis Current Controller Saturation Limit	-
Q_CURRCNTR_PTERM	Q axis Current Controller Proportional Gain	-
Q_CURRCNTR_ITERM	Q axis Current Controller Integral Gain	-
Q_CURRCNTR_CTERM	Q axis Current Controller Anti-Windup Gain	-
Q_CURRCNTR_OUTMAX	Q axis Current Controller Saturation Limit	-
SPEEDCNTR_PTERM	Speed Controller Proportional Gain	-
SPEEDCNTR_ITERM	Speed Controller Integral Gain	-
SPEEDCNTR_CTERM	Speed Current Controller Anti-Windup Gain	-
SPEEDCNTR_OUTMAX	Speed Current Controller Saturation Limit	-

6. This demo allows the user to choose between using the external (on-board) op-amps, or internal (on-chip) op-amps for signal conditioning of the current sense signal measured across the shunt resistor.

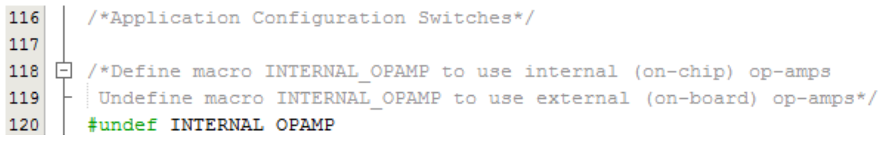
- For internal op-amps, define the macro INTERNAL_OPAMP.



```
/*Application Configuration Switches*/

/*Define macro INTERNAL_OPAMP to use internal (on-chip) op-amps
 Undefine macro INTERNAL_OPAMP to use external (on-board) op-amps*/
#define INTERNAL_OPAMP
```

- For external op-amps, undefine the macro INTERNAL_OPAMP.



```
/*Application Configuration Switches*/

/*Define macro INTERNAL_OPAMP to use internal (on-chip) op-amps
 Undefine macro INTERNAL_OPAMP to use external (on-board) op-amps*/
#undef INTERNAL_OPAMP
```

7. Under Menu go to *Production -> Clean and Build Main Project,* or click the *hammer & brush icon* which is a shortcut for the *Clean and Build* command.
8. Once the project is built successfully, download the hex file by clicking *Make and Program the device project*.

Running the Demonstration

This section details how to run the demonstration.

Description

1. Press the ‘S2’ switch to start spinning the motor.
2. Vary the potentiometer, P1, to change the motor speed.
3. Press the 'S1’ switch to stop the motor.

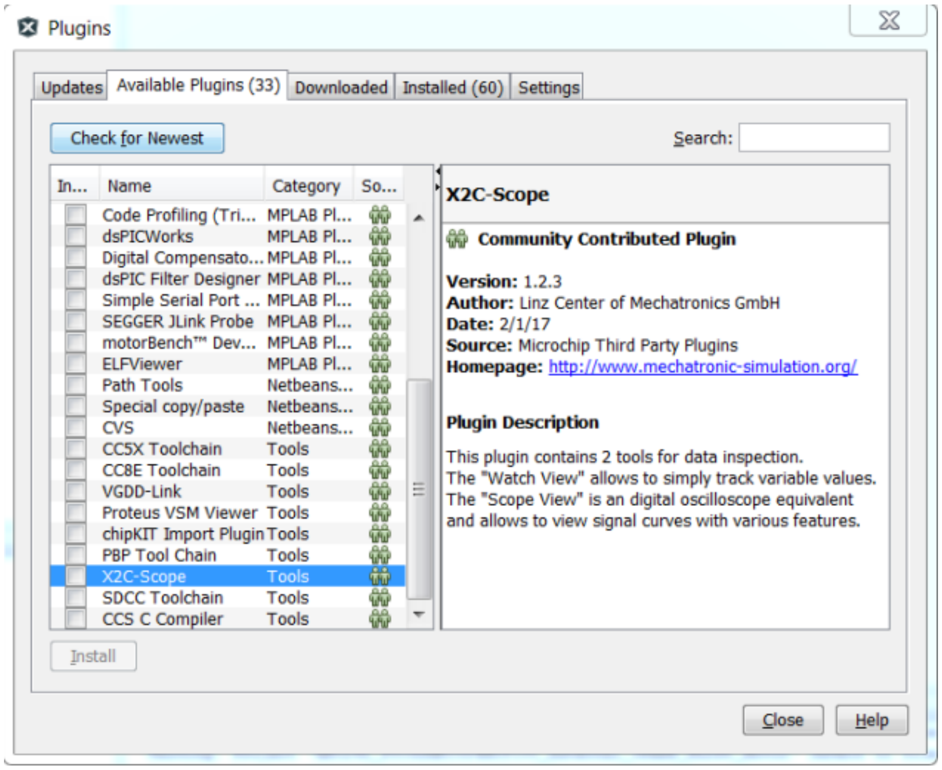
Using the X2C Scope

This section details how to use the X2C Scope.

Description

The MPLAB X IDE enables the use of the X2C Scope plug-in to read, write, and plot global variables in real time. In this demonstration, the X2C Scope communicates with the target using the UART Channel 2 at 38400 bps.

1. If not already installed, in MPLAB X IDE, select *Tools* > *Plugins* > *Available Plugins* > *Select X2C Scope*> *Install*.
2. Restart MPLAB X IDE to complete the plug-in installation.



In...	Name	Category	So...
☐	Code Profiling (Tri...	MPLAB Pl...	
☐	dsPICWorks	MPLAB Pl...	
☐	Digital Compensato...	MPLAB Pl...	
☐	dsPIC Filter Designer	MPLAB Pl...	
☐	Simple Serial Port ...	MPLAB Pl...	
☐	SEGGER JLink Probe	MPLAB Pl...	
☐	motorBench™ Dev...	MPLAB Pl...	
☐	ELFViewer	MPLAB Pl...	
☐	Path Tools	Netbeans...	
☐	Special copy/paste	Netbeans...	
☐	CVS	Netbeans...	
☐	CC5X Toolchain	Tools	
☐	CC8E Toolchain	Tools	
☐	VGDD-Link	Tools	
☐	Proteus VSM Viewer	Tools	
☐	chipKIT Import Plugin	Tools	
☐	PBP Tool Chain	Tools	
☐	X2C-Scope	Tools	
☐	SDCC Toolchain	Tools	
☐	CCS C Compiler	Tools	

X2C-Scope

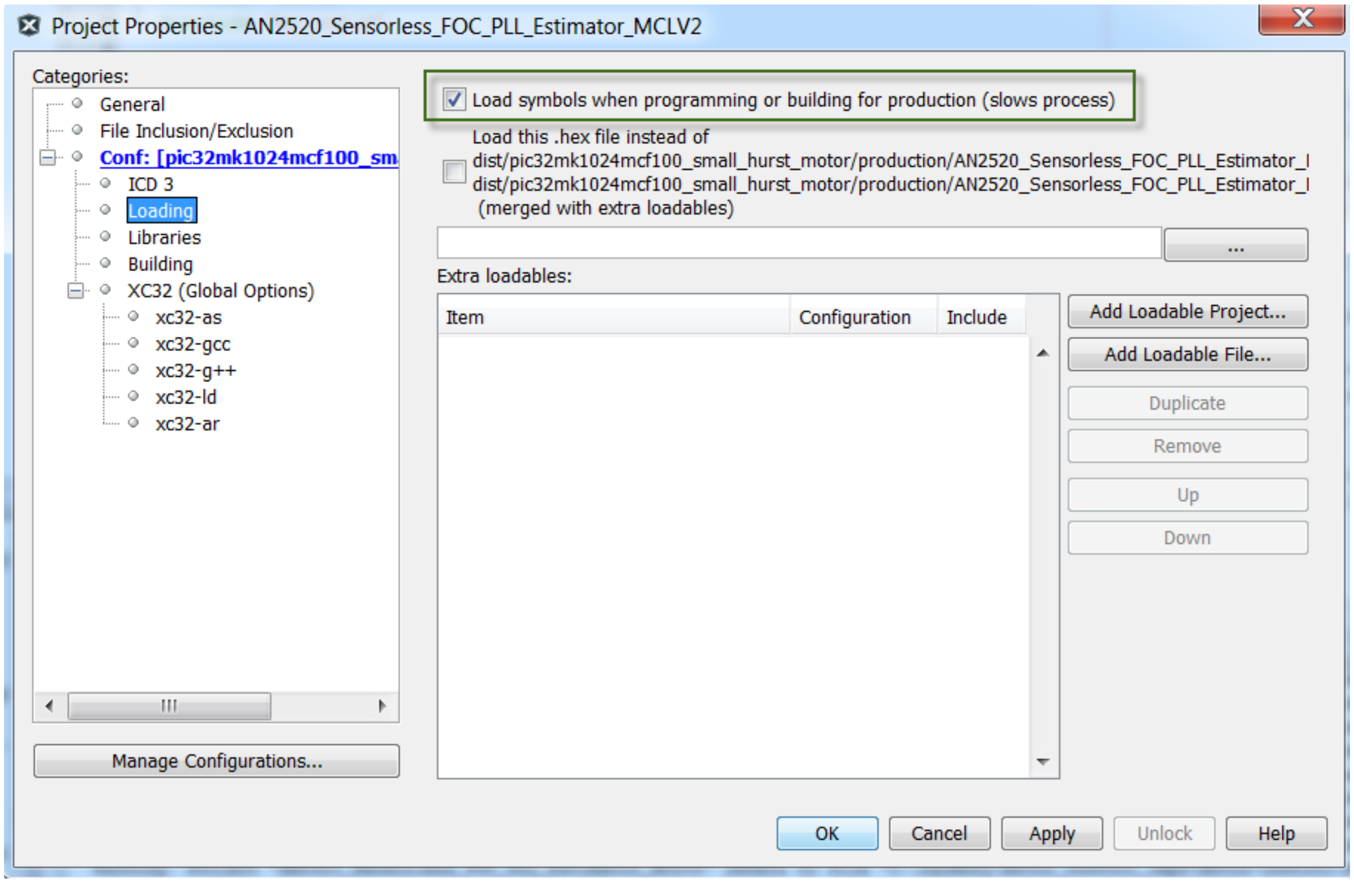
Community Contributed Plugin

Version: 1.2.3
Author: Linz Center of Mechatronics GmbH
Date: 2/1/17
Source: Microchip Third Party Plugins
Homepage: http://www.mechatronic-simulation.org/

Plugin Description

This plugin contains 2 tools for data inspection.
The "Watch View" allows to simply track variable values.
The "Scope View" is an digital oscilloscope equivalent and allows to view signal curves with various features.

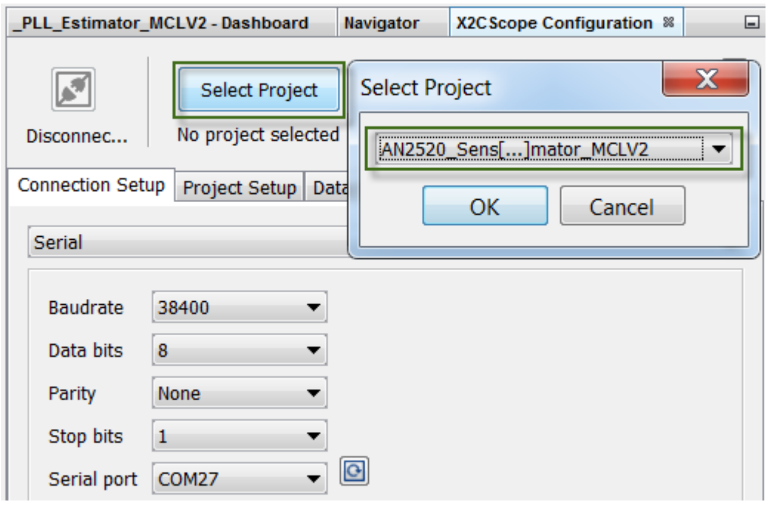
3. Open X2C Scope by selecting *Tools > Embedded > X2C Scope*
4. Ensure the symbols are loaded during the project build by selecting *Project Properties > Selected Configuration > Loading*, and ensure that the check box for *Load Symbols when programming or building for production (slows process)* is selected. If this check box was not selected, then re-build the project and re-program the device.



☑ Load symbols when programming or building for production (slows process)

☐ Load this .hex file instead of
dist/pic32mk1024mcf100_small_hurst_motor/production/AN2520_Sensorless_FOC_PLL_Estimator_I
dist/pic32mk1024mcf100_small_hurst_motor/production/AN2520_Sensorless_FOC_PLL_Estimator_I
(merged with extra loadables)

5. Under the *X2C Scope Configuration* tab, select the project.



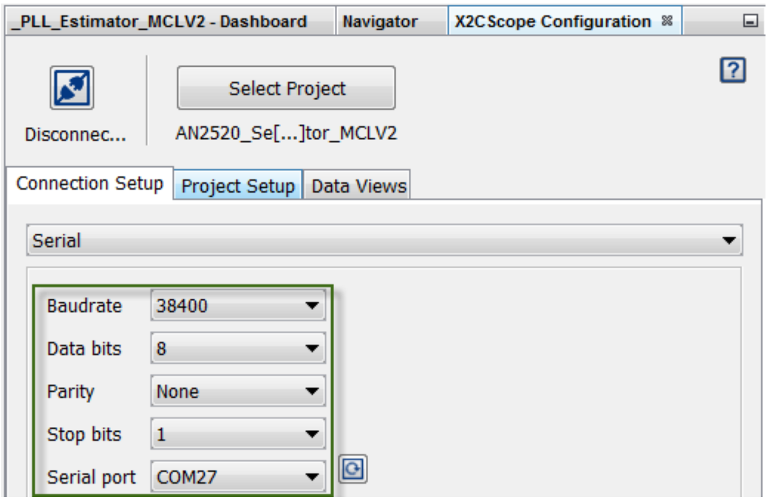
No project selected

Select Project: AN2520_Sens[...]mator_MCLV2

Serial

Baudrate 38400
Data bits 8
Parity None
Stop bits 1
Serial port COM27

6. Under *Connection Setup*, Set the Baud Rate = 38400, Data bits = 8, Parity = None, Stop Bits = 1 and select the associated COM port.

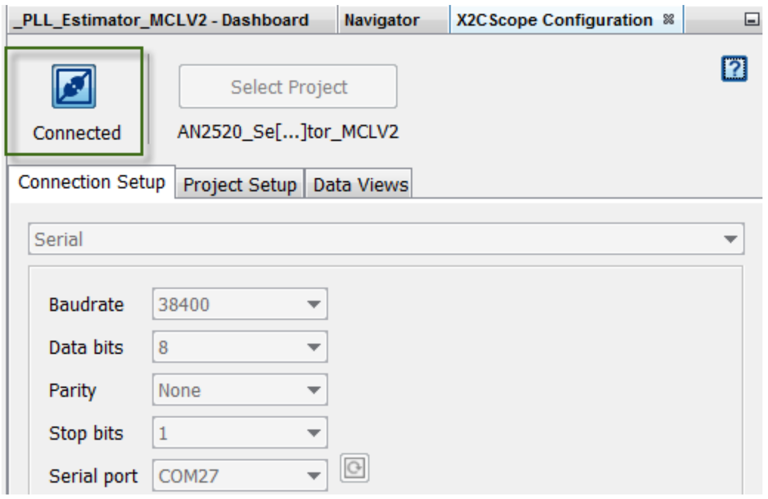


AN2520_Se[...]tor_MCLV2

Serial

Baudrate 38400
Data bits 8
Parity None
Stop bits 1
Serial port COM27

7. Connect to the target by clicking the *Connect/Disconnect* Button.

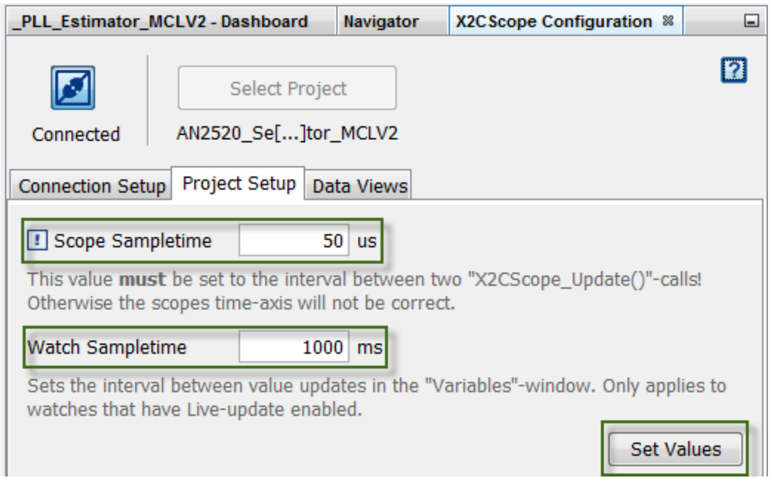


AN2520_Se[...]tor_MCLV2

Serial

Baudrate 38400
Data bits 8
Parity None
Stop bits 1
Serial port COM27

8. Under *Project Setup*, Set the Scope sample time to 50uS. In this demo, the X2CScope_Update function is called from the motor control ISR, which executes every 50uS. Set *Watch Sampletime* to a value at which you want to update the watch window variables (default is 1000mS). Click the *Set Values* button after setting the *Scope Sample time* and *Watch Sampletime.*



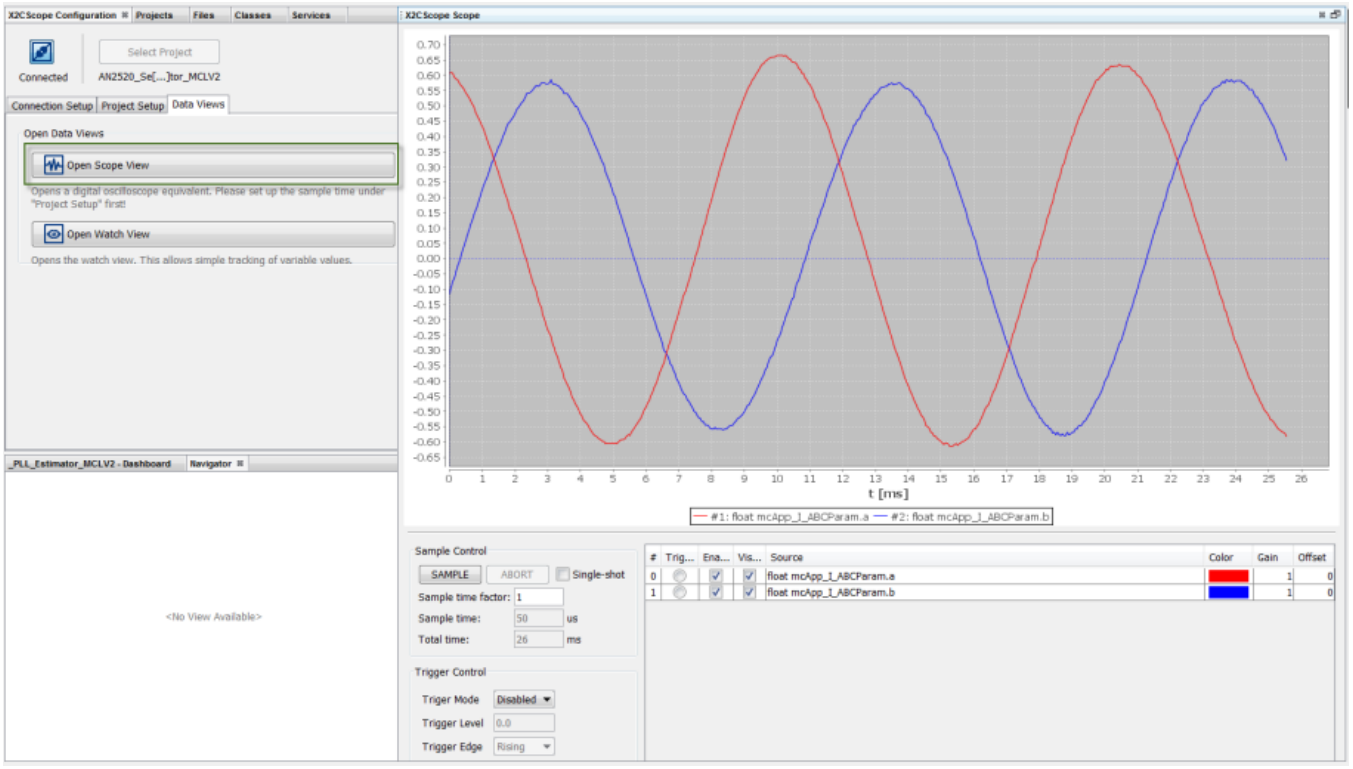
Scope Sampletime: 50 us

This value **must** be set to the interval between two "X2CScope_Update()"-calls! Otherwise the scopes time-axis will not be correct.

Watch Sampletime: 1000 ms

Sets the interval between value updates in the "Variables"-window. Only applies to watches that have Live-update enabled.

9. Under *Data Views*, select *Open Scope View* to plot any two global variables in run-time.



Opens a digital oscilloscope equivalent. Please set up the sample time under "Project Setup" first!

Opens the watch view. This allows simple tracking of variable values.

Sample Control

Sample time factor: 1

Sample time: 50 us

Total time: 26 ms

Trigger Control

Triger Mode: Disabled

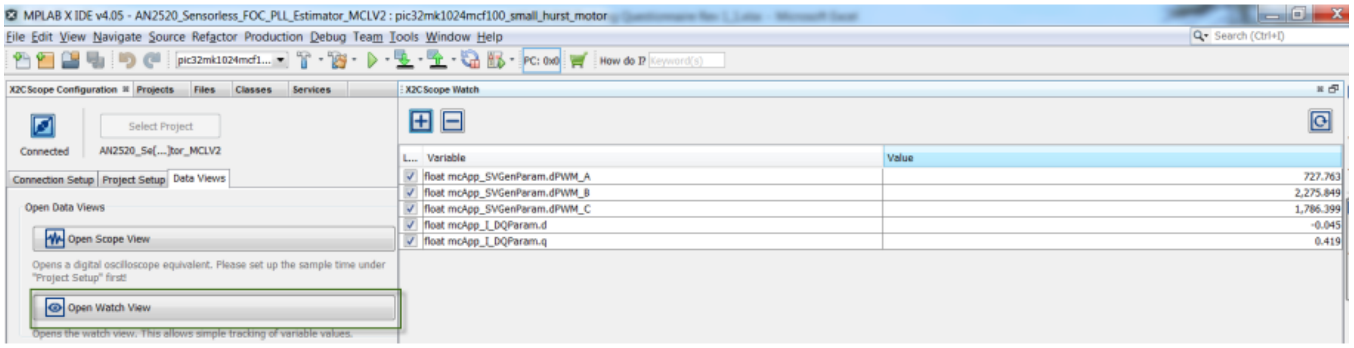
Trigger Level: 0.0

Trigger Edge: Rising

#	Trig...	Ena...	Vis...	Source	Color	Gain	Offset
0	○	✓	✓	float mcApp_I_ABCParam.a	🟥	1	0
1	○	✓	✓	float mcApp_I_ABCParam.b	🟦	1	0

Note:	The free version of X2C Scope allows plotting up to two global variables simultaneously. The Professional Version of X2C Scope allows the plotting of up to seven global variables simultaneously.

10. Under *Data Views*, select *Open Watch View* to read or write to any global variables in run time.



Opens a digital oscilloscope equivalent. Please set up the sample time under "Project Setup" first!

Opens the watch view. This allows simple tracking of variable values.

L...	Variable	Value
✓	float mcApp_SVGenParam.dPWM_A	727.763
✓	float mcApp_SVGenParam.dPWM_B	2,275.849
✓	float mcApp_SVGenParam.dPWM_C	1,786.399
✓	float mcApp_I_DQParam.d	-0.045
✓	float mcApp_I_DQParam.q	0.419

Motor Control Library Interface

a) Functions

	Name	Description
	mcLib_CalcPI	Calculates PI Compensator Output
	mcLib_CalcTimes	Space Vector to Duty Cycle Translation
	mcLib_ClarkeTransform	Clarke Transform
	mcLib_InitPI	Initialize PI Controller
	mcLib_InvParkTransform	Inverse Park Transform
	mcLib_ParkTransform	Park Transform
	mcLib_PLLEstimator	Estimates Rotor angle position
	mcLib_SinCosGen	Calculates Sine and Cosine values
	mcLib_SVPWMGen	Space Vector PWM Generation

b) Datatypes and Constants

	Name	Description
	mcParam_ABC	Structure containing component values for 3 Phase Stationary Reference Frame
	mcParam_AlphaBeta	Structure containing component values for 2 Phase Stationary Reference Frame
	mcParam_ControlRef	Structure containing variables used as control references for Current PI and Speed PI
	mcParam_DQ	Structure containing component values for 2 Phase Rotating Reference Frame
	mcParam_FOC	Structure containing variables used in Field Oriented Control
	mcParam_PIController	Structure containing variables used by PI Compensator
	mcParam_PLLEstimator	This is type mcParam_PLLEstimator.
	mcParam_SinCos	Structure containing variables used for Sine & Cosine calculation
	mcParam_SVPWM	Structure containing variables used in calculating Space Vector PWM Duty cycles

Description

a) Functions

mcLib_CalcPI Function

Calculates PI Compensator Output

File

mc_Lib.h

C

```
void mcLib_CalcPI(mcParam_PIController * pParam);
```

Returns

None.

Description

This function calculates the PI Compensator Output.

Remarks

None.

Preconditions

None.

Example

```
mcParam_PIController mcApp_PIParam;


//Set the PI Compensator Coefficients
mcApp_PIParam.qKp = 0.5;  // Proportional Coefficient of the PI Compensator
mcApp_PIParam.qKi = 0.2;  // Integral Coefficient of the PI Compensator
mcApp_PIParam.qKc = 0.3;  // Anti-windup Coefficient of the PI Compensator

// Set the PI Compensator Saturation Limits
mcApp_PIParam.qOutMax = 1;// Max output limit of the PI Compensator
mcApp_PIParam.qOutMin = -1;// Min output limit of the PI Compensator

// Update the reference and feedback value inputs to PI Compensator
mcApp_PIParam.qInRef = <Reference Value>;
mcApp_PIParam.qInMeas = <Feedback Value>;


// Calculate PI Compensator output using mcLib_CalcPI. The output of the
// PI Compensator is in mcApp_PIParam.qOut
 mcLib_CalcPI(&mcApp_PIParam);
```

Parameters

Parameters	Description
*pParam	Structure pointer pointing to the mcParam_PIController type structure containing PI Compensator Parameters.

Function

void mcLib_CalcPI(mcParam_PIController *pParm)

mcLib_CalcTimes Function

Space Vector to Duty Cycle Translation

File

mc_Lib.h

C

```
void mcLib_CalcTimes(mcParam_SVPWM * svParam);
```

Returns

- Following members of the mcParam_SVPWM type structure contain the output
- of this function float Ta; // Ta = To/2 + T1 + T2 float Tb; // Tb = To/2 + T2 float Tc; // Tc = To/2

None.

Description

This function translates the space vector durations to normalized duty cycle values for SVPWM

Remarks

None.

Preconditions

None.

Example

```
mcParam_SVPWM                    mcApp_SVGenParam;


//Set PWM Period in PWM Timer counts
mcApp_SVGenParam.PWMPeriod = 3000; // PWM Period in PWM Timer Counts

//Set the normalized lengths of Vector T1 and T2
mcApp_SVGenParam.T1 = 0.8; // Normalized length of vector T1
mcApp_SVGenParam.T2 = 0.2; // Normalized length of vector T2

// Calculate normalized duty cycles for SVPWM using mcLib_CalcTimes. The
// result of the transform is in mcApp_SVGenParam.Ta, mcApp_SVGenParam.Tb
// and mcApp_SVGenParam.Tc
 mcLib_CalcTimes(&mcApp_SVGenParam);
```

Parameters

Parameters	Description
*svParam	Structure pointer pointing to the mcParam_SVPWM type structure containing SVPWM parameters

Function

void mcLib_CalcTimes(mcParam_SVPWM *svParam);

mcLib_ClarkeTransform Function

Clarke Transform

File

mc_Lib.h

C

```
void mcLib_ClarkeTransform(mcParam_ABC * abcParam, mcParam_AlphaBeta * alphabetaParam);
```

Returns

None.

Description

This function calculates Clarke Transform

Remarks

None.

Preconditions

None.

Example

```
mcParam_ABC mcApp_I_ABCParam;
mcParam_AlphaBeta mcApp_I_AlphaBetaParam;

//Update Ia and Ib current values
mcApp_I_ABCParam.a = 1.2; // Phase A current in Amperes
mcApp_I_ABCParam.b = 0.8; // Phase B current in Amperes


// Calculate Clarke transform using mcLib_ClarkeTransform. The result
// of the transform is in mcApp_I_AlphaBetaParam.alpha and mcApp_I_AlphaBetaParam.beta
mcLib_ClarkeTransform(&mcApp_I_ABCParam, &mcApp_I_AlphaBetaParam);
```

Parameters

Parameters	Description
*abcParam	Structure pointer pointing to the mcParam_ABC type structure containing A,B,C axis components (3 Phase Stationary Frame)
*alphabetaParam	Structure pointer pointing to the mcParam_AlphaBeta type structure containing Alpha-Beta axis

Function

void mcLib_ClarkeTransform(mcParam_ABC *abcParam, mcParam_AlphaBeta *alphabetaParam)

mcLib_InitPI Function

Initialize PI Controller

File

mc_Lib.h

C

```
void mcLib_InitPI(mcParam_PIController * pParam);
```

Returns

None.

Description

This function initializes the PI Controller i.e. clears the integral sum and PI output.

Remarks

None.

Preconditions

None.

Example

```
mcParam_PIController mcApp_PIParam;


// Initialize PI Compensator output using mcLib_InitPI.
 mcLib_InitPI(&mcApp_PIParam);
```

Parameters

Parameters	Description
*pParam	Structure pointer pointing to the mcParam_PIController type structure containing PI Compensator Parameters

Function

void mcLib_InitPI(mcParam_PIController *pParm)

mcLib_InvParkTransform Function

Inverse Park Transform

File

mc_Lib.h

C

```
void mcLib_InvParkTransform(mcParam_DQ * dqParam, mcParam_SinCos * scParam, mcParam_AlphaBeta *
alphabetaParam);
```

Returns

None.

Description

This function calculates inverse Park Transform

Remarks

None.

Preconditions

None.

Example

```
mcParam_DQ                                  mcApp_V_DQParam;
mcParam_SinCos                              mcApp_SincosParam;
mcParam_AlphaBeta                           mcApp_V_AlphaBetaParam;


// Angle input = 1.571 radians (90 degree))
mcApp_SincosParam.Angle = (float) 1.571;

//Calculate Sine and Cosine using mcLib_SinCosGen
mcLib_SinCosGen(&mcApp_SincosParam);



// Update normalized D axis and Q axis values.
mcApp_V_DQParam.d = 0;
mcApp_V_DQParam.q = 0.8;


// Calculate inverse Park transform using mcLib_InvParkTransform. The result
// of the transform is in mcApp_V_AlphaBetaParam.alpha and
// mcApp_V_AlphaBetaParam.beta
mcLib_InvParkTransform(&mcApp_V_DQParam,&mcApp_SincosParam,
                       &mcApp_V_AlphaBetaParam);
```

Parameters

Parameters	Description
*dqParam	Structure pointer pointing to the mcParam_DQ type structure containing D-Q axis components
*scParam	Structure pointer pointing to the mcParam_SinCos type structure containing parameters related to Sine and Cosine Calculations.
*alphabetaParam	Structure pointer pointing to the mcParam_AlphaBeta type structure containing Alpha-Beta axis components(2 Phase Stationary Frame).

Function

void mcLib_InvParkTransform(mcParam_DQ *dqParam, mcParam_SinCos *scParam,
mcParam_AlphaBeta *alphabetaParam)

mcLib_ParkTransform Function

Park Transform

File

mc_Lib.h

C

```
void mcLib_ParkTransform(mcParam_AlphaBeta * alphabetaParam, mcParam_SinCos * scParam,
mcParam_DQ * dqParam);
```

Returns

None.

Description

This function calculates Park Transform.

Remarks

None.

Preconditions

None.

Example

```
mcParam_AlphaBeta mcApp_I_AlphaBetaParam;
mcParam_SinCos mcApp_SincosParam;
mcParam_DQ mcApp_I_DQParam;

// Angle input = 1.571 radians (90 degree)
mcApp_SincosParam.Angle = (float) 1.571;

//Calculate Sine and Cosine using mcLib_SinCosGen
mcLib_SinCosGen(&mcApp_SincosParam);


// Update Alpha and Beta axis currents
mcApp_I_AlphaBeta.alpha = 1.2;
mcApp_I_AlphaBeta.beta = 0.8;


// Calculate Park transform using mcLib_ParkTransform. The result
// of the transform is in mcApp_I_DQParam.d and mcApp_I_DQParam.q
 mcLib_ParkTransform(&mcApp_I_AlphaBetaParam,  &mcApp_SincosParam, &mcApp_I_DQParam);
```

Parameters

Parameters	Description
*alphabetaParam	Structure pointer pointing to the mcParam_AlphaBeta type structure containing Alpha-Beta axis components(2 Phase Stationary Frame)
*scParam	Structure pointer pointing to the mcParam_SinCos type structure containing parameters related to Sine and Cosine Calculations.
*dqParam	Structure pointer pointing to the mcParam_DQ type structure containing D-Q axis components(2 Phase Rotating Frame).

Function

void mcLib_ParkTransform(mcParam_AlphaBeta *alphabetaParam ,
mcParam_SinCos *scParam, mcParam_DQ *dqParam)

mcLib_PLLEstimator Function

Estimates Rotor angle position

File

mc_Lib.h

C

```
void mcLib_PLLEstimator(mcParam_PLLEstimator * pllestimatorParam, mcParam_SinCos * scParam,
mcParam_FOC * focParam, mcParam_ControlRef * ctrlParam, mcParam_AlphaBeta * I_alphabetaParam,
mcParam_AlphaBeta * V_alphabetaParam);
```

Returns

None.

Description

This function estimates rotor angle position based on back emf measurement using PLL type estimator

Remarks

None.

Preconditions

None.

Example

```
 mcParam_SinCos mcApp_SincosParam;
 mcParam_PLLEstimator mcApp_EstimParam;

//Initialize Motor parameter values required for PLL Estimator
mcApp_EstimParam.qLsDt = <Motor Phase Inductance/PWM Period>;
mcApp_EstimParam.qRs = <Motor Phase Resistance>;
mcApp_EstimParam.qKFi = <BACK EMF Constant in V-sec/rad>;
mcApp_EstimParam.qInvKFi_Below_Nominal_Speed = <1/BACK EMF Constant in V-sec/rad>;
mcApp_EstimParam.qLs_DIV_2_PI = <Motor Phase Inductance / (2*PI)>;
mcApp_EstimParam.qNominal_Speed = <Motor Nominal Speed>;
mcApp_EstimParam.qKfilterEsdq = <DQ Filter Coefficient>;
mcApp_EstimParam.qVelEstimFilterK =<Speed Filter Coefficient>;
mcApp_EstimParam.qDeltaT = <PWM Period>;
mcApp_EstimParam.RhoOffset = <OFFSET in Radians>;


// Angle input = 1.571 radians (90 degree)
mcApp_SincosParam.Angle = (float) 1.571;

// Read Phase A and Phase B current values
mcApp_focParam.Ia = <Phase A Current>;
mcApp_focParam.Ib = <Phase B Current>;

// Calculate Clarke Transform
mcLib_ClarkeTransform(&mcApp_focParam);

// Calculate Park Transform
mcLib_ParkTransform(&mcApp_focParam);

// Estimate rotor position using mcLib_PLLEstimator. Estimated rotor angle and velocity are in
// mcApp_ControlParam.qRho and mcApp_ControlParam.qVelEstim respectively
mcLib_PLLEstimator(&mcApp_EstimParam, &mcApp_SincosParam, &mcApp_focParam, &mcApp_ControlParam);
```

Parameters

Parameters	Description
*pllestimatorParam	Structure pointer pointing to the mcParam_PLLEstimator type structure containing PLL Estimator parameters

*scParam	Structure pointer pointing to the mcParam_SinCos type structure containing parameters related to Sine and Cosine Calculations.
*focParam	Structure pointer pointing to the mcParam_FOC type structure containing FOC parameters
*ctrlParam	Structure pointer pointing to the mcParam_ControlRef type structure containing reference values for D axis current, Q axis current and rotor electrical speed.

Function

void mcLib_PLLEstimator(mcParam_PLLEstimator *pllestimatorParam, mcParam_SinCos *scParam, mcParam_FOC *focParam, mcParam_ControlRef *ctrlParam)

mcLib_SinCosGen Function

Calculates Sine and Cosine values

File

mc_Lib.h

C

```
void mcLib_SinCosGen(mcParam_SinCos * scParam);
```

Returns

None.

Description

This function calculates Sine and Cosine values of a given angle (specified in radians)

Remarks

None.

Preconditions

None.

Example

```
 mcParam_SinCos mcApp_SincosParam;

// Angle input = 1.571 radians (90 degree)
mcApp_SincosParam.Angle = (float) 1.571;

//Calculate Sine and Cosine using mcLib_SinCosGen. The calculated Sine and Cosine
// values are in mcApp_SincosParam.Sin and mcApp_SincosParam.Cos respectively.
mcLib_SinCosGen(&mcApp_SincosParam);
```

Parameters

Parameters	Description
*scParam	Structure pointer pointing to the mcParam_SinCos type structure containing parameters related to Sine and Cosine Calculations.

Function

void mcLib_SinCosGen(mcParam_SinCos *scParam)

mcLib_SVPWMGen Function

Space Vector PWM Generation

File

mc_Lib.h

C

```
void mcLib_SVPWMGen(mcParam_AlphaBeta * alphabetaParam, mcParam_SVPWM * svParam);
```

Returns

None.

Description

This function calculates Duty cycles for Space Vector PWM Signals

Remarks

None.

Preconditions

None.

Example

```
mcParam_AlphaBeta                    mcApp_V_AlphaBetaParam;
mcParam_SVPWM                        mcApp_SVGenParam;


//Update normalized Valpha and Vbeta values
mcApp_V_AlphaBetaParam.alpha = 0.1; // Normalized Alpha axis voltage
mcApp_V_AlphaBetaParam.beta = 0.8; // Normalized Beta axis voltage

//Set PWM Period in PWM Timer counts
mcApp_SVGenParam.PWMPeriod = 3000; // PWM Period in PWM Timer Counts


// Calculate duty cycles for SVPWM using mcLib_SVPWMGen. The result
// of the transform is in mcApp_SVGenParam.dPWM_A, mcApp_SVGenParam.dPWM_B
// and mcApp_SVGenParam.dPWM_C
 mcLib_SVPWMGen(&mcApp_focParam, &mcApp_SVGenParam);
```

Parameters

Parameters	Description
*alphabetaParam	Structure pointer pointing to the mcParam_AlphaBeta type structure containing alpha-beta axis normalized stator voltage values
*svParam	Structure pointer pointing to the mcParam_SVPWM type structure containing SVPWM parameters.

Function

void mcLib_SVPWMGen(mcParam_AlphaBeta *alphabetaParam, mcParam_SVPWM *svParam)

b) Datatypes and Constants

mcParam_ABC Structure

File

mc_Lib.h

C

```
typedef struct {
  float a;
  float b;
  float c;
} mcParam_ABC;
```

Members

Members	Description
float a;	A axis component in 3 Phase Stationary Frame
float b;	B axis component in 3 Phase Stationary Frame
float c;	C axis component in 3 Phase Stationary Frame

Description

Structure containing component values for 3 Phase Stationary Reference Frame

mcParam_AlphaBeta Structure

File

mc_Lib.h

C

```
typedef struct {
  float alpha;
  float beta;
} mcParam_AlphaBeta;
```

Members

Members	Description
float alpha;	Alpha axis component in 2 Phase Stationary Frame
float beta;	Beta axis component in 2 Phase Stationary Frame

Description

Structure containing component values for 2 Phase Stationary Reference Frame

mcParam_ControlRef Structure

File

mc_Lib.h

C

```
typedef struct {
  float VelInput;
  float VelRef;
  float IdRef;
  float IqRef;
  float Diff;
  float IqRefmax;
} mcParam_ControlRef;
```

Members

Members	Description
float VelInput;	Speed Input. Speed Input passed through a rate limiter gives Speed Reference
float VelRef;	Speed Reference.
float IdRef;	D axis Current (Flux) reference value
float IqRef;	Q axis Current (Torque) reference value

float Diff;	Difference between Speed Input and Speed Reference
float IqRefmax;	Maximum Q axis current

Description

Structure containing variables used as control references for Current PI and Speed PI

mcParam_DQ Structure

File

mc_Lib.h

C

```
typedef struct {
  float d;
  float q;
} mcParam_DQ;
```

Members

Members	Description
float d;	D axis component in 2 Phase Rotating Frame
float q;	Q axis component in 2 Phase Rotating Frame

Description

Structure containing component values for 2 Phase Rotating Reference Frame

mcParam_FOC Structure

File

mc_Lib.h

C

```
typedef struct {
  float OpenLoopAngle;
  float DCBusVoltage;
  float MaxPhaseVoltage;
  float VdqNorm;
  float MaxVoltageCircleSquared;
  float VdSquaredDenorm;
  float VqSquaredDenorm;
  float VqRefVoltage;
} mcParam_FOC;
```

Members

Members	Description
float OpenLoopAngle;	Rotor Angle in Radians
float DCBusVoltage;	Measured DC Bus voltage in volts
float MaxPhaseVoltage;	Maximum Phase to Neutral Voltage
float VdqNorm;	Normalized vector sum of D and Q axis voltage
float MaxVoltageCircleSquared;	Square of Maximum Phase Voltage
float VdSquaredDenorm;	Square of D axis Voltage in Volts
float VqSquaredDenorm;	Square of Q axis Voltage in Volts
float VqRefVoltage;	Estimated Q axis voltage during Flux Weakening in Volts

Description

Structure containing variables used in Field Oriented Control

mcParam_PIController Structure

File

mc_Lib.h

C

```
typedef struct {
  float qdSum;
  float qKp;
  float qKi;
  float qKc;
  float qOutMax;
  float qOutMin;
  float qInRef;
  float qInMeas;
  float qOut;
  float qErr;
} mcParam_PIController;
```

Members

Members	Description
float qdSum;	Integrator Output of the PI Compensator
float qKp;	Proportional Coefficient of the PI Compensator
float qKi;	Integral Coefficient of the PI Compensator
float qKc;	Anti-windup Coefficient of the PI Compensator
float qOutMax;	Max output limit of the PI Compensator
float qOutMin;	Min output limit of the PI Compensator
float qInRef;	Reference input of the PI Compensator
float qInMeas;	Feedback input of the PI Compensator
float qOut;	Proportional + Integral Output of the PI Compensator
float qErr;	Error input of the PI Compensator

Description

Structure containing variables used by PI Compensator

mcParam_PLLEstimator Structure

File

mc_Lib.h

C

```
typedef struct {
  float qDeltaT;
  float qRho;
  float qOmegaMr;
  float qLastIalpha;
  float qLastIbeta;
  float qDIalpha;
  float qDIbeta;
  float qEsa;
  float qEsb;
  float qEsd;
```

```
    float qEsq;
    float qVIndalpha;
    float qVIndbeta;
    float qEsdf;
    float qEsqf;
    float qKfilterEsdq;
    float qVelEstim;
    float qVelEstimFilterK;
    float qLastValpha;
    float qLastVbeta;
    float RhoOffset;
    float qRs;
    float qLsDt;
    float qInvKFi;
    float qKFi;
    float qInvKFi_Below_Nominal_Speed;
    float qLs_DIV_2_PI;
    float qNominal_Speed;
    float qDecimate_Nominal_Speed;
  } mcParam_PLLEstimator;
```

Members

Members	Description
float qDeltaT;	Integration Interval
float qRho;	Estimated Rotor Angle
float qOmegaMr;	primary speed estimation
float qLastIalpha;	last value for Ialpha
float qLastIbeta;	last value for Ibeta
float qDIalpha;	difference Ialpha
float qDIbeta;	difference Ibeta
float qEsa;	BEMF alpha
float qEsb;	BEMF beta
float qEsd;	BEMF d
float qEsq;	BEMF q
float qVIndalpha;	dI*Ls/dt alpha
float qVIndbeta;	dI*Ls/dt beta
float qEsdf;	BEMF d filtered
float qEsqf;	BEMF q filtered
float qKfilterEsdq;	filter constant for d-q BEMF
float qVelEstim;	Estimated speed
float qVelEstimFilterK;	Filter Konstant for Estimated speed
float qLastValpha;	Value from last control step Ialpha
float qLastVbeta;	Value from last control step Ibeta
float RhoOffset;	estimated rotor angle init offset in radians
float qRs;	Rs value - stator resistance in ohms
float qLsDt;	Ls/dt value - stator inductand / dt - variable with speed
float qInvKFi;	InvKfi constant value (InvKfi = Omega/BEMF)
float qKFi;	Backemf Constant in V-sec/rad
float qInvKFi_Below_Nominal_Speed;	1/(Backemf Constant in V-sec/rad) when electrical speed < Nominal electrical Speed of the motor
float qLs_DIV_2_PI;	Phase Inductance Ls / 2*PI
float qNominal_Speed;	Nominal Electrical Speed of the motor radians/sec
float qDecimate_Nominal_Speed;	Nominal Electrical Speed/10 of the motor

Description

This is type mcParam_PLLEstimator.

mcParam_SinCos Structure

File

mc_Lib.h

C

```
typedef struct {
  float Angle;
  float Sin;
  float Cos;
} mcParam_SinCos;
```

Members

Members	Description
float Angle;	Angle in radians whose sine/cosine needs to be calculated
float Sin;	Sine(Angle)
float Cos;	Cosine(Angle)

Description

Structure containing variables used for Sine & Cosine calculation

mcParam_SVPWM Structure

File

mc_Lib.h

C

```
typedef struct {
  float PWMPeriod;
  float Vr1;
  float Vr2;
  float Vr3;
  float T1;
  float T2;
  float Ta;
  float Tb;
  float Tc;
  float dPWM_A;
  float dPWM_B;
  float dPWM_C;
} mcParam_SVPWM;
```

Members

Members	Description
float PWMPeriod;	PWM Period in PWM Timer Counts
float Vr1;	Normalized Phase A voltage obtained using modified Clarke transform
float Vr2;	Normalized Phase B voltage obtained using modified Clarke transform
float Vr3;	Normalized Phase C voltage obtained using modified Clarke transform
float T1;	Length of Vector T1
float T2;	Length of Vector T2
float Ta;	Ta = To/2 + T1 + T2
float Tb;	Tb = To/2 + T2
float Tc;	Tc = To/2

float dPWM_A;	Phase A Duty Cycle
float dPWM_B;	Phase B Duty Cycle
float dPWM_C;	Phase C Duty Cycle

Description

Structure containing variables used in calculating Space Vector PWM Duty cycles

Symbol Reference

Macros

The following table lists macros in this documentation.

Macros

	Name	Description
	ANGLE_2PI	Defines value for 2*PI
	ANGLE_STEP	Defines the angle resolution in the sine/cosine look up table
	ONE_BY_SQRT3	Defines value for 1/sqrt(3)
	SQRT3_BY2	Defines value for sqrt(3)/2
	TABLE_SIZE	Defines the size of sine/cosine look up table

ANGLE_2PI Macro

File

mc_Lib.h

C

```
#define ANGLE_2PI (2*M_PI)                  // Defines value for 2*PI
```

Description

Defines value for 2*PI

ANGLE_STEP Macro

File

mc_Lib.h

C

```
#define ANGLE_STEP (float)((float)ANGLE_2PI/(float)TABLE_SIZE) //Defines the angle resolution
in the sine/cosine look up table
```

Description

Defines the angle resolution in the sine/cosine look up table

ONE_BY_SQRT3 Macro

File

mc_Lib.h

C

```
#define ONE_BY_SQRT3 (float)0.5773502691    // Defines value for 1/sqrt(3)
```

Description

Defines value for 1/sqrt(3)

SQRT3_BY2 Macro

File

mc_Lib.h

C

```
#define SQRT3_BY2 (float)0.866025403788  // Defines value for sqrt(3)/2
```

Description

Defines value for sqrt(3)/2

TABLE_SIZE Macro

File

mc_Lib.h

C

```
#define TABLE_SIZE 256                       // Defines the size of sine/cosine look up table
```

Description

Defines the size of sine/cosine look up table

Files

The following table lists files in this documentation.

Files

Name	Description
mc_Lib.h	This is file mc_Lib.h.

mc_Lib.h

Functions

	Name	Description
◆	mcLib_CalcPI	Calculates PI Compensator Output
◆	mcLib_CalcTimes	Space Vector to Duty Cycle Translation
◆	mcLib_ClarkeTransform	Clarke Transform
◆	mcLib_InitPI	Initialize PI Controller
◆	mcLib_InvParkTransform	Inverse Park Transform
◆	mcLib_ParkTransform	Park Transform
◆	mcLib_PLLEstimator	Estimates Rotor angle position
◆	mcLib_SinCosGen	Calculates Sine and Cosine values
◆	mcLib_SVPWMGen	Space Vector PWM Generation

Macros

	Name	Description
	ANGLE_2PI	Defines value for 2*PI
	ANGLE_STEP	Defines the angle resolution in the sine/cosine look up table
	ONE_BY_SQRT3	Defines value for 1/sqrt(3)

	SQRT3_BY2	Defines value for sqrt(3)/2
	TABLE_SIZE	Defines the size of sine/cosine look up table

Structures

	Name	Description
	mcParam_ABC	Structure containing component values for 3 Phase Stationary Reference Frame
	mcParam_AlphaBeta	Structure containing component values for 2 Phase Stationary Reference Frame
	mcParam_ControlRef	Structure containing variables used as control references for Current PI and Speed PI
	mcParam_DQ	Structure containing component values for 2 Phase Rotating Reference Frame
	mcParam_FOC	Structure containing variables used in Field Oriented Control
	mcParam_PIController	Structure containing variables used by PI Compensator
	mcParam_PLLEstimator	This is type mcParam_PLLEstimator.
	mcParam_SinCos	Structure containing variables used for Sine & Cosine calculation
	mcParam_SVPWM	Structure containing variables used in calculating Space Vector PWM Duty cycles

Description

This is file mc_Lib.h.

Index