Microchip 32-bit Motor Control

Help for Microchip 32-bit Motor Control

© 2013 - 2019 Microchip Technology Inc. All rights reserved.

Motor Control Overview

This help documentation provides a comprehensive source of information on how to use Motor Control Applications. This contains information about the MHC components used to control the motor. This document talks about different motor control algorithm applications and how to run and observe the graphs in X2CScope.

X2CScope Plugin

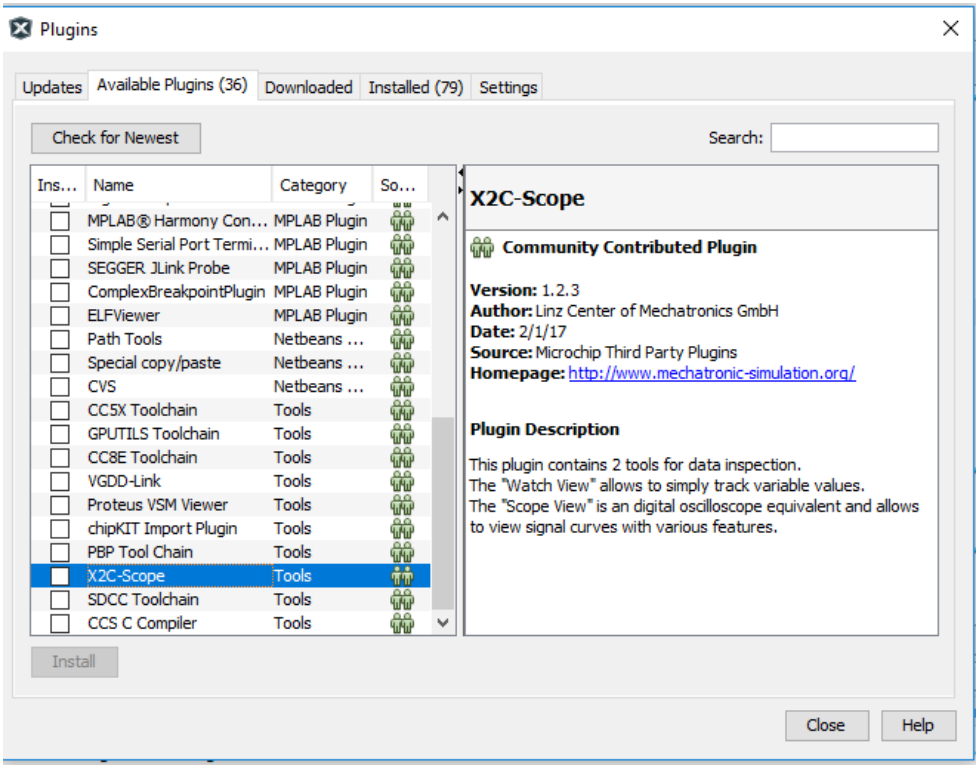
This topic describe how to use the X2C-Scope MPLAB X IDE plug-in.

Description

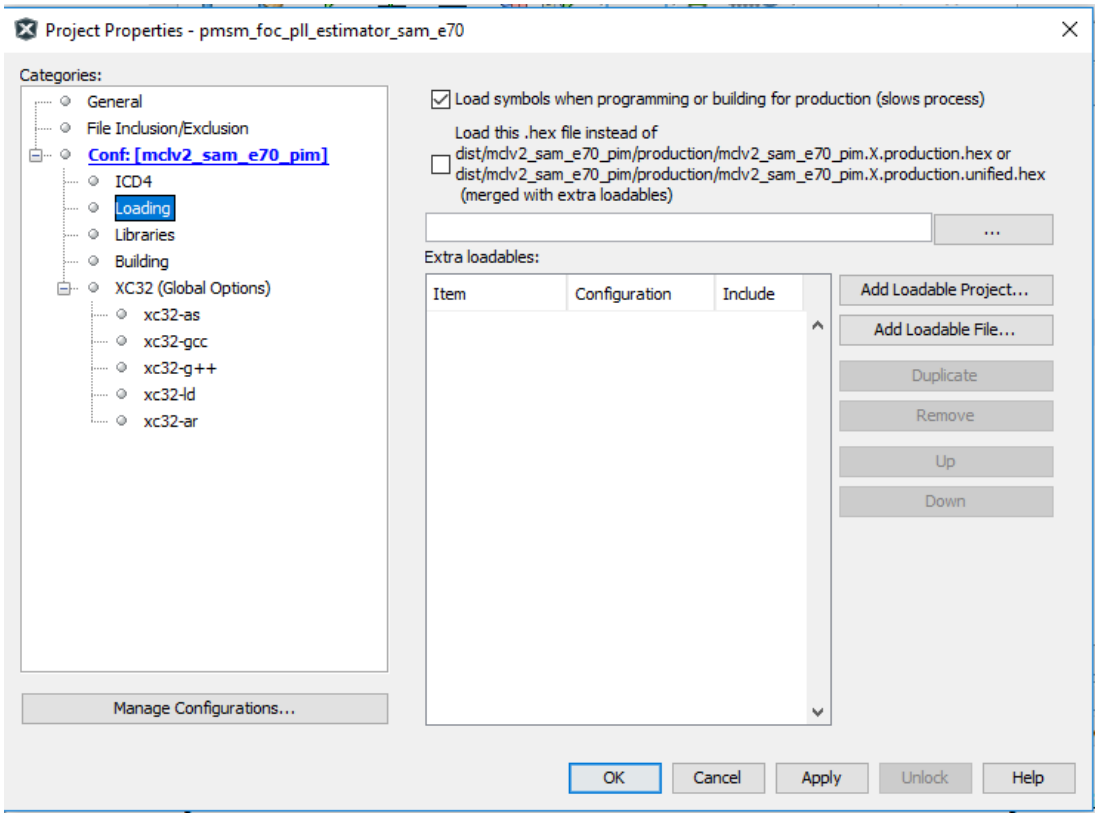
The MPLAB X IDE enables use of the X2C-Scope plug-in to read, write, and plot global variables in real time. The X2C-Scope communicates with the target using UART. The baud rate of the UART communication can be modified using the MHC.

Installation

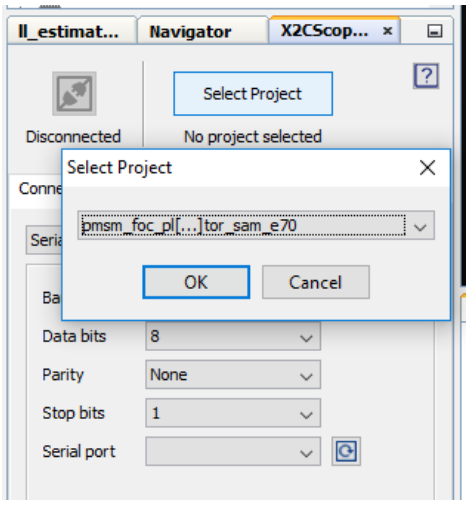
1. If not already installed, in MPLAB X IDE, select Tools > Plugins > Available Plugins > Select X2C-Scope> Install.
2. Restart MPLAB X IDE to complete the plug-in installation.



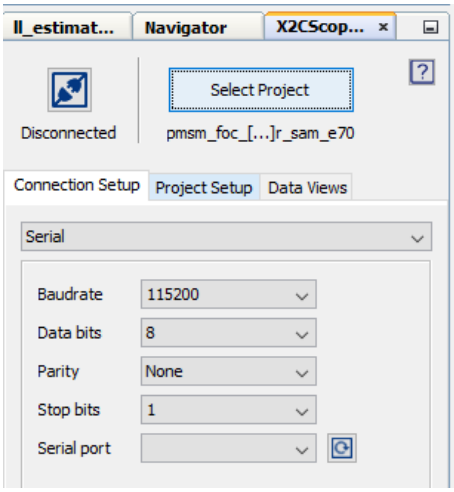
3. Open the X2CScope by selecting Tools -> Embedded -> X2CScope.
4. Ensure the symbols are loaded during the project build by selecting Project Properties -> Selected Configuration -> Loading, and ensure that the check box for Load Symbols when programming or building for production (slows process) is selected. If this check box was not selected, rebuild the project and reprogram the device.



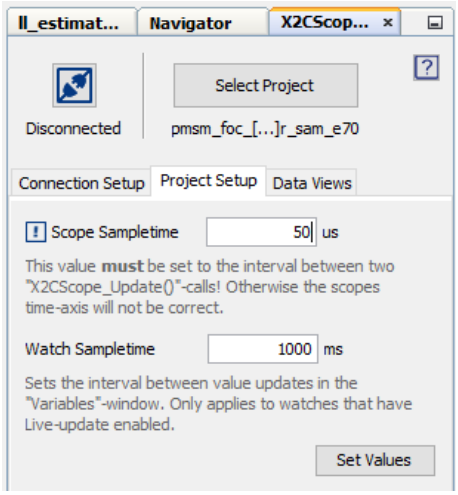
5. In the X2CScope Configuration tab, select the project.



6. In Connection Setup, set the Baud Rate to '115200', Data bits to '8', Parity to 'None', Stop Bits to '1', and then select the associated COM port.



7. In Project Setup, Set the Scope Sampletime to '50 us.' This is equal to PWM frequency. Set the Watch Sample time to a value at which you want to update the watch window variables (the default is 1000 ms). Click Set Values after setting the Scope Sampletime and Watch Sampletime.



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

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

Applications Help

This section provides help for the motor control applications that are available in the MPLAB Harmony 3 - Motor Control (MC).

The following applications are provided to demonstrate the usage of the motor control peripherals to run the motor control algorithm.

PMSM FOC Using PLL Estimator

This Section provides help for the PMSM Field Oriented Control(FOC) using PLL Estimator.

pmsm_foc_pll_estimator_sam_e70

This motor control example project shows how to control Permanent Magnet Synchronous Motor (PMSM) with PLL Estimator based Field Oriented Control (FOC) on SAME70 Micro-controller.

Description

This example shows how to configure motor control peripherals ADC, PWM for the control operation. The control strategy is the sensorless FOC in which rotor position and speed is determined by the PLL Estimator.

The following sections describe the basics of the PLL Estimator based sensor-less motor control algorithm and the steps to build, configure and run the PMSM motor for both low voltage and high voltage PMSM motor drive.

Building The Application

This section provides information on how to build an application using the MPLAB X IDE.

Description

The parent folder for all the MPLAB X projects for this application is given below:

Application Path	motor_control\apps\pmsm_foc_pll_estimator_sam_e70

To build the application, refer the following table and open the appropriate project file in the MPLAB X IDE.

Project Name	Description
mchv3_sam_e70_pim.X	MCHV3 Development Board (High Voltage)
mclv2_sam_e70_pim.X	MCLV2 Development Board (Low Voltage)

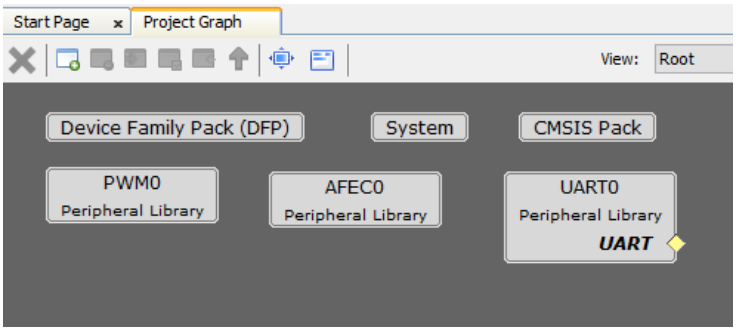
MPLAB Harmony Configurations

This topic provides information on the MHC configurations.

Description

The following project graph shows the MPLAB Harmony components that are included in this application:

mchv3_sam_e70_pim.X and mchv2_sam_e70_pim.X



Components Used

1. AFEC Peripheral
   - The AFEC (ADC) is used to measure analog quantities. Four channels are used to measure the Phase Current A, the Phase Current B, the DC Bus Voltage and the Potentiometer. Conversion is triggered at the PWM (zero match + offset of the switch delay)
2. PWM Peripheral
   - This peripheral is used to generated three phase synchronous PWM waveforms. Fault functionality is also enabled to switch

off the output waveforms asynchronously

3. UART Peripheral
   - The UART is used for X2CScope communication to observe graphs and variable values run time

Control Algorithm

This section describes the control strategy and explains the control loop flow.

Description

The project implements a PLL based sensor-less Field Oriented Control (FOC) algorithm on SAME70 32-bit micro-controller to regulate the speed of the PMSM motor. For details of PLL based sensor-less FOC, refer Application Note AN2520. The following section describes briefly about the PLL based sensor-less FOC algorithm, software design and implementation.

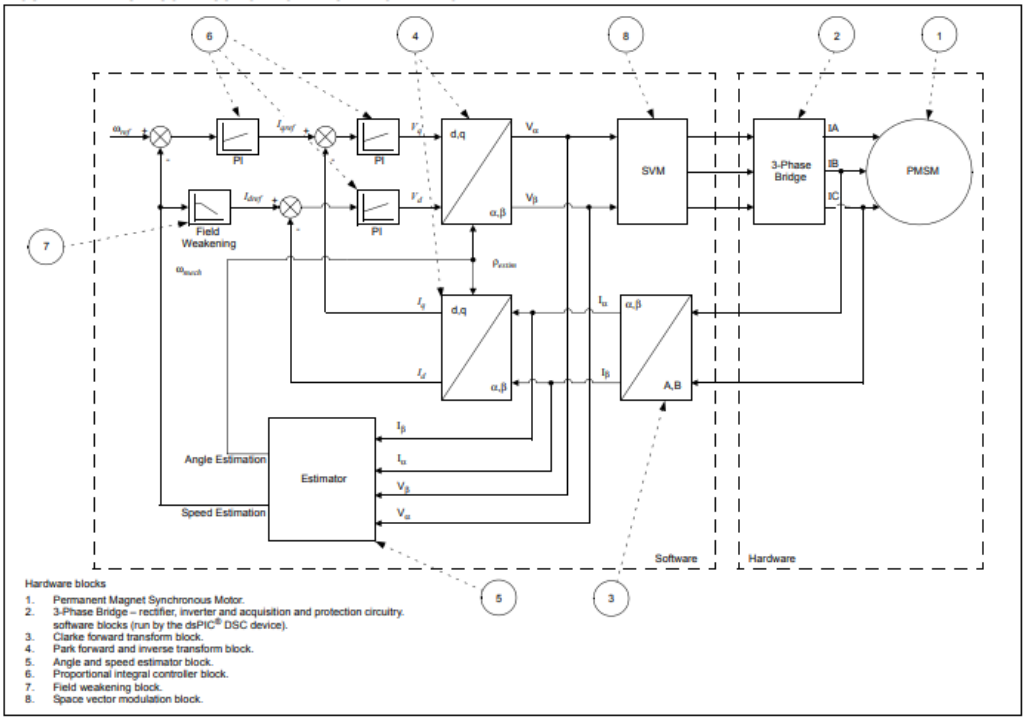
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from stationary reference frame to torque and flux producing currents components in rotating reference frame using mathematical transformations. The Field Oriented Control is done as follows-

1. Measure the motor phase currents.
2. Transform them into the two phase system (a, b) using Clarke transformation.
3. Calculate the rotor position angle.
4. Transform stator currents into the d,q-coordinate system using Park transformation.
5. The stator current torque (i_q) and flux (i_d) producing components are controlled separately by the controllers.
6. The output stator voltage space vector is transformed back from the d,q-coordinate system into the two phase system fixed with the stator by inverse Park transformation.
7. Using the space vector modulation, the output three-phase voltage is generated.

The phase currents are measured using two shunt resistors connected to Phase U and Phase V inverter legs respectively. The W phase current is determined based on Kirchoff's current law.

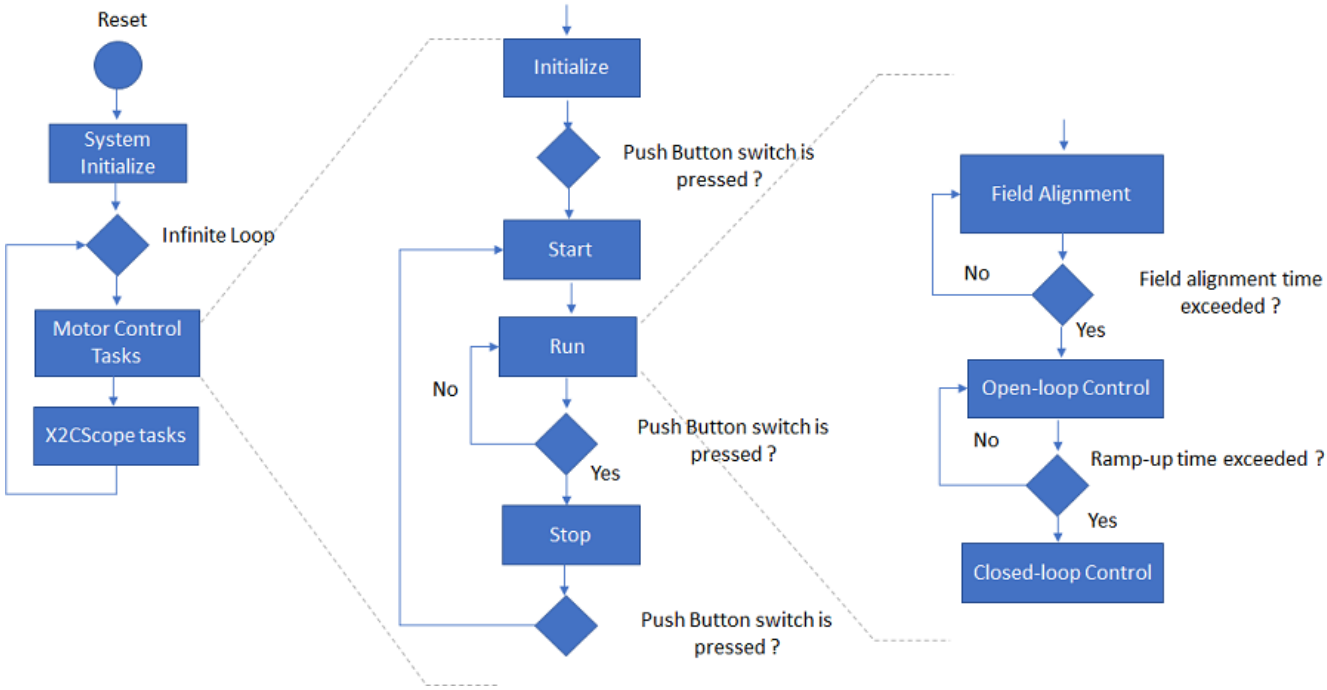
The electrical rotor position is calculated using the PLL estimator using **MCLIB_PLLEstimator** software block. The working principle of the PLL estimator can be found in Application Note AN2520. The main disadvantage of PLL estimator is its inability to estimate the rotor angle at lower rotor speeds because of very low value of back EMF. Hence, the FOC algorithm also integrates the ramp-up profile for motor start. The reference speed is incremented linearly using a open loop phase voltage control until the required minimum reference speed for the PLL estimator is reached.

The following block diagram shows the software realization of the FOC algorithm.



Software Design:

The following figure shows the various state machines of the the motor control software.



In the software, the PMSM speed control task is realized by a state machine as shown in the above figure. The following sections briefly describes the various states in the PMSM speed control task:

1. Initialize:

In this state, following tasks are performed:

- Initialization/ configuration of NVIC, AFEC, PWM motor control peripherals for generation of periodic ADC triggers and ADC

conversion interrupt

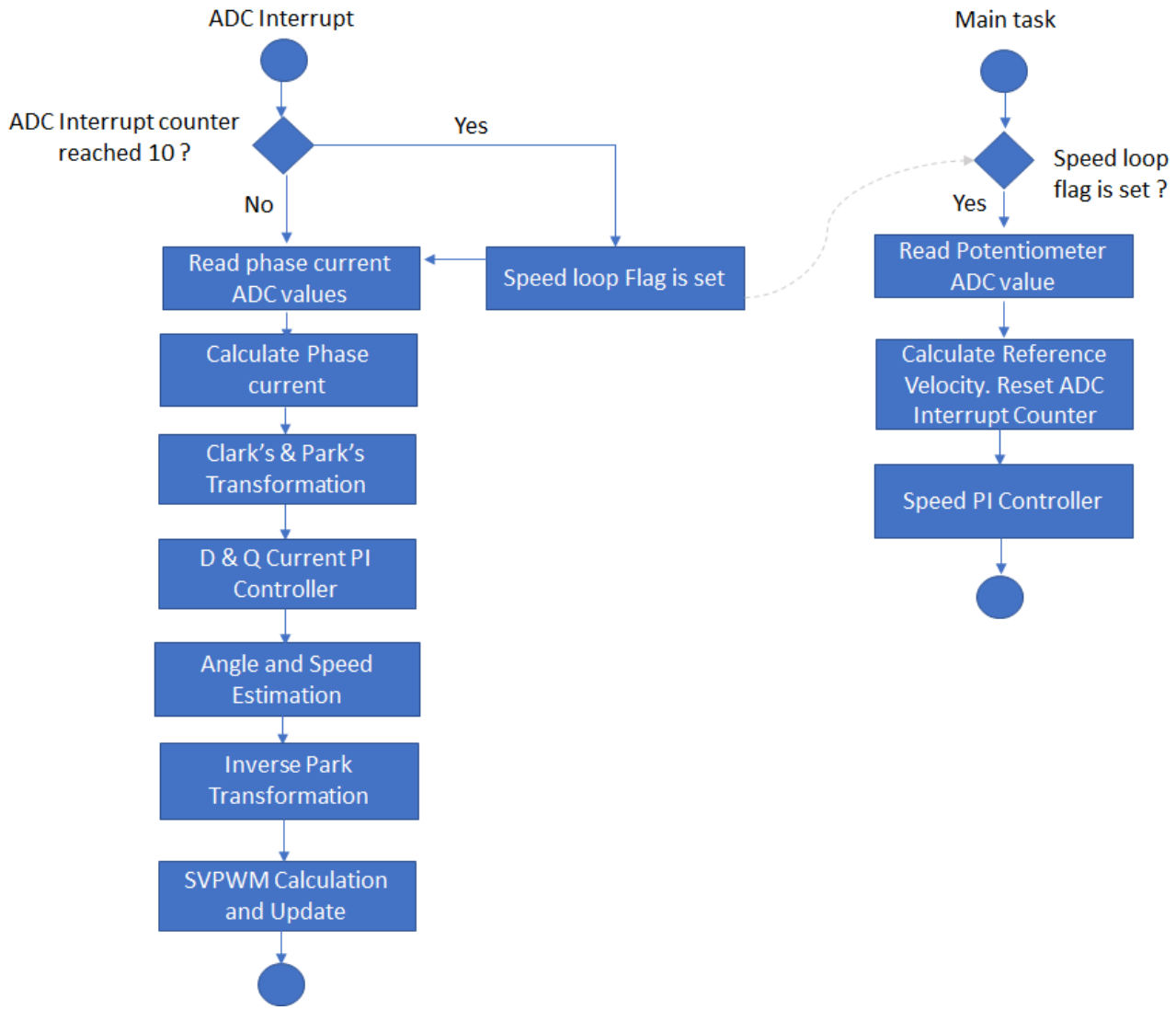
- Current Offset measurement and calibration
- Initialize PI controller parameters for speed and current control loops

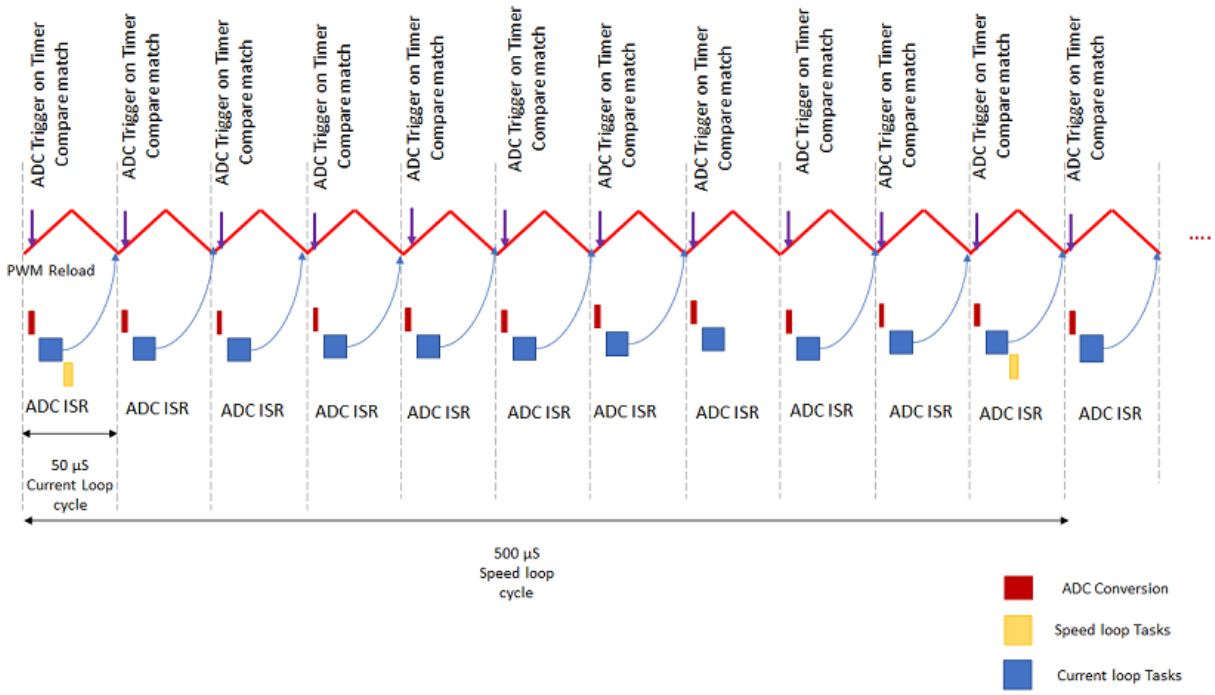
2. Start:

In this state, the motor control state variables are reset and periodic ADC conversion interrupt is enabled. Control waits for the switch press.

3. Run:

In this state, the motor starts spinning. The below flow chart and the timing diagram shows the tasks performed in run state:





In run state, two threads are executed- Main task thread and ADC Interrupt task thread. The current control is carried out in the ADC interrupt task thread, while the speed control is carried out in the main task thread. Hence, the ADC interrupt cycle indicates the current control frequency.

As pointed out earlier, the PLL estimator is unable to estimate the rotor position information at the start-up phase when the speed is very low. This necessitates the motor control algorithm to integrate an open-loop control mechanism to linearly ramp the motor speed to the required minimum speed before switching over to close open loop control. Therefore, the motor control state can further be classified into three sub-states - Field Alignment, Open-loop Control and Close loop Control.

- **Field Alignment:**

In this mode, a limited value of DC current is applied to the U phase of PMSM motor in order to align the rotor magnetic field with the U-phase of the motor. The time for which DC current has to be applied to achieve the field alignment depends on the time constant of the PMSM motor drive.

- **Open Loop Control:**

In this mode, the speed of the PMSM motor is gradually ramped up using an open loop control. The speed is ramped up linearly to a minimum value required for the sensor-less PLL estimator to estimate the speed of the PMSM motor with required accuracy.

- **Close Loop Control:**

In this mode, the speed can be regulated using closed loop PLL based sensor-less FOC algorithm.

4. Stop:

In this state, the PWM channels are disabled thereby stopping the motor**.** The periodic ADC trigger and conversion interrupt is disabled.

Software Configuration:

The FOC algorithm is used across different application fields. In order to get the optimal control of the PMSM motor, both the motor specific parameters and application parameters has to be updated in the software . The following section describes how to update both motor and application specific parameters in the the project.

1. **Setting motor control PWM frequency:**

The PWM controller frequency (in Hz) can be configured by setting following macro in "*userparams.h"* file.

This frequency should be same as the frequency configured in PWM peripheral in the MHC.

Macro	Description
PWM_FREQUENCY	Current controller and PWM frequency in Hz.

2. **Setting motor specific -parameter:**

Set the motor following motor parameters in "*userparams.h"* file.

Macro	Description	Units
MOTOR_PER_PHASE_RESISTANCE	Motor per phase resistance	Ohm
MOTOR_PER_PHASE_INDUCTANCE	Motor per phase inductance	H
MOTOR_BEMF_CONST_V_PEAK_LL_KRPM_MECH	Back EMF constant	Vpk(L-L)/KRPM
NUM_POLE_PAIRS	Number of pole pairs	
RATED_SPEED_RPM	Rated mechanical speed of the motor	RPM
MAX_SPEED_RPM	Maximum mechanical speed of the motor	RPM

3. Setting PI Controller parameters:

Depending on the type of motor used, and the corresponding application PI controller parameters should to be updated in*"userparams.h"* file.

Parameters for speed Control loop:

Macro	Description
SPEEDCNTR_PTERM	Proportional gain of speed control loop
SPEEDCNTR_ITERM	Integral gain of speed control loop
SPEEDCNTR_CTERM	Anti-windup term of speed control loop
SPEEDCNTR_OUTMAX	Maximum controller output of speed control loop

Parameters for I_d current loop:

Macro	Description
D_CURRCNTR_PTERM	Proportional gain of Id current control loop
D_CURRCNTR_ITERM	Integral gain of Id current control loop
D_CURRCNTR_CTERM	Anti-windup term of Id current control loop
D_CURRCNTR_OUTMAX	Maximum controller output of Id current control loop

Parameters for I_q current loop:

Macro	Description
Q_CURRCNTR_PTERM	Proportional gain of Iq current control loop
Q_CURRCNTR_ITERM	Integral gain of Iq current control loop
Q_CURRCNTR_CTERM	Anti-windup term of Iq current control loop
Q_CURRCNTR_OUTMAX	Maximum controller output of Iq current control loop

4. Enabling Field weakening feature:

Field weakening feature enables the PMSM motor to rotate beyond its rated speed. This feature can be enabled by setting following macros in "u*serparams.h"* file.

Macro	Description
FIELD_WEAKENING	Field weakening enable switch (0 - Disable, 1 - Enable)
MAX_FW_NEGATIVE_ID_REF	Maximum negative D-axis reference current if field weakening is enabled

5. Debugging Features:

Open loop functioning is useful for debugging when using new motor. This feature can be enabled by setting following macros in "u*serparams.h"* file.

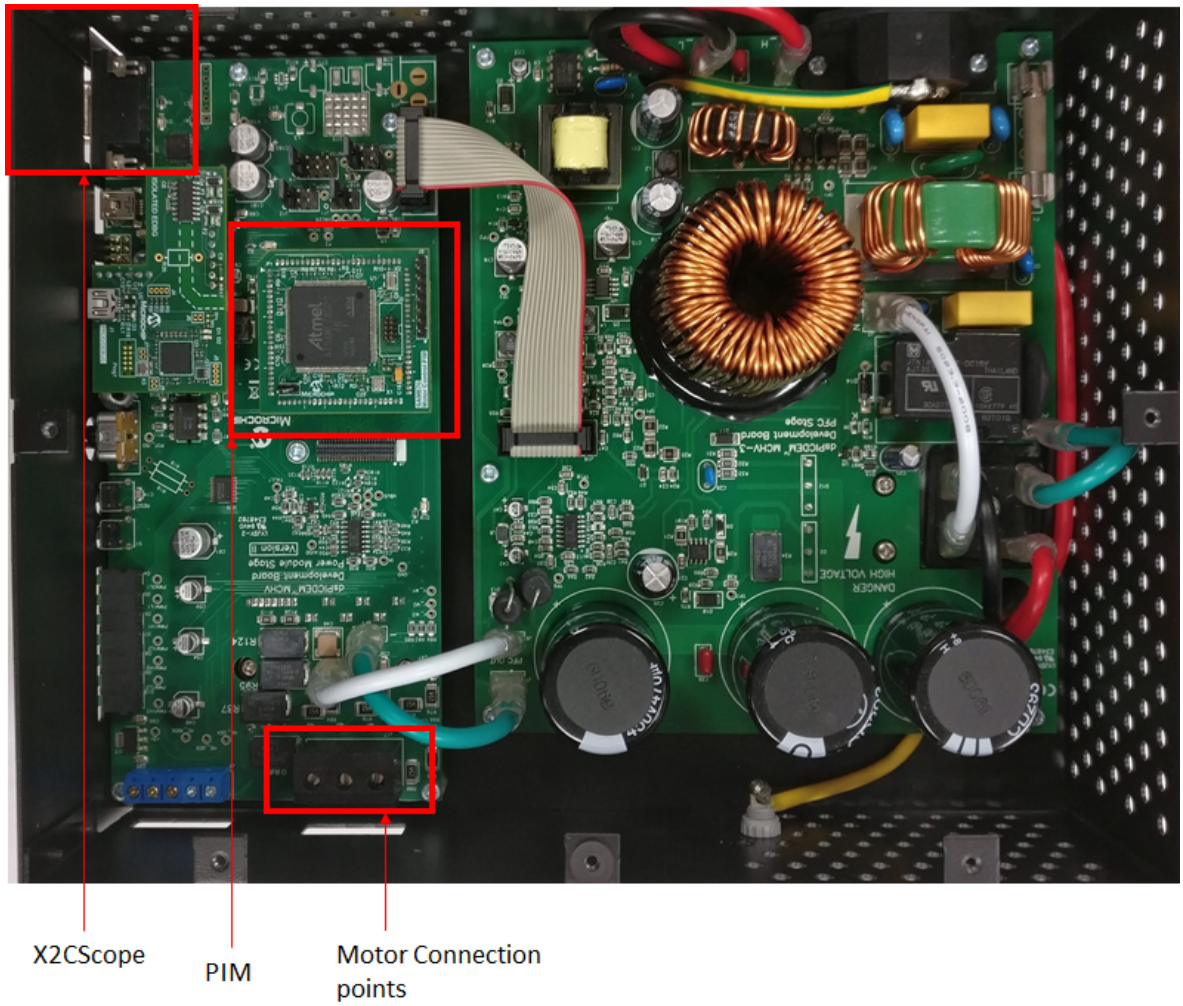
Macro	Description
OPEN_LOOP_FUNCTIONING	Open loop enable switch (0 - Disable, 1 - Enable)
Q_CURRENT_REF_OPENLOOP	Startup current in open loop
TORQUE_MODE	This a special mode where the speed control loop can be bypassed. The input of potentiometer is scaled and is fed directly to the q-axis PI controller. Torque mode enable switch (0 - Disable, 1 - Enable)

Hardware Setup

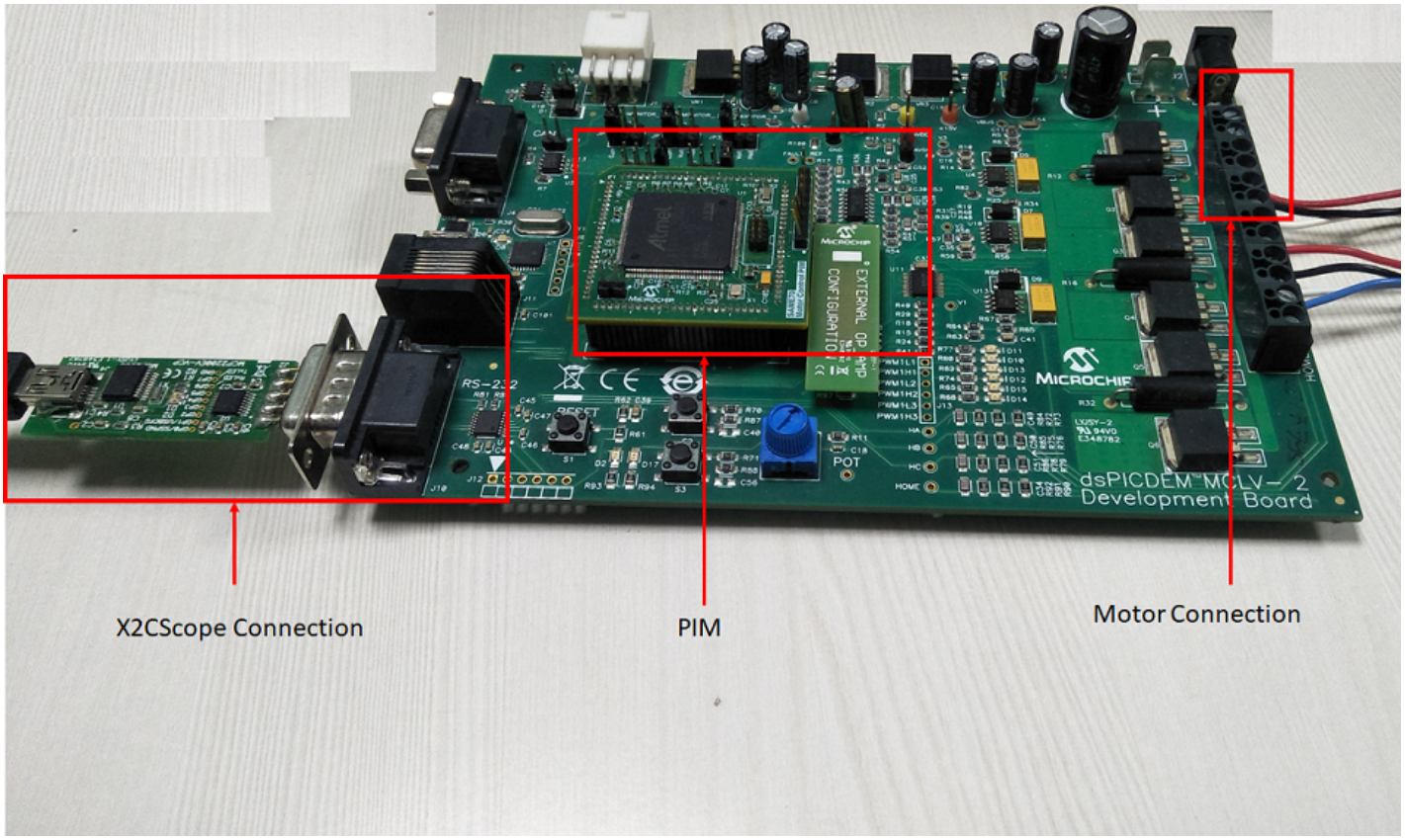
This section describes how to configure the supported hardware.

Description

1. **Project mchv3_sam_e70_pim.X.**
   - **Hardware Used**
     - MCHV3 Development Board (High Voltage)
     - Isolated Embedded Debugger Interface
     - ATSAME70 PIM
     - Leadshine EL5-M0400-1-24 Motor
   - **Hardware Setup**
     1. Mount the SAME70 144-pin Motor Control PIM on the dsPICDEM MCHV-3 Development Board, as shown in the following figure.
     2. Connect the three-phase wires of the PMSM motor to M1, M2 and M3 ports.
     3. Connect the AC mains using J1.
     4. This application supports communication with the X2C-Scope plug-in to monitor and 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. Refer to the jumper positions marked on the development board chassis to select between the RS-232 serial port (J8) and the mini-USB port (J6). Refer to Using the X2C-Scope Plug-in for details on how to use the X2C-Scope in this demonstration.



2. **Project mclv2_sam_e70_pim.X.**
   - **Hardware Used**
     - MCLV2 Development Board (Low Voltage)
     - ATSAME70 PIM
     - Long Hurst DMA0204024B101 Motor
   - **Hardware Setup**
     1. Mount the SAME70 144-pin Motor Control PIM on the dsPICDEM MCLV-2 Development Board, as shown in the following figure.
     2. Connect the External Op amp Configuration Matrix Board to header J14.
     3. Connect the three-phase wires coming from P1 header of the long Hurst motor (i.e., Red, White and Black to M1, M2 and M3 ports,respectively). Leave the green wire from the motor unconnected.
     4. Apply 24V DC at BP1-BP2/J2.
     5. This application 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). To use the RS-232 serial port, connect JP4 and JP5 to the UART position.
     6. Refer to Using the X2C Scope for details on how to use X2C Scope in this demonstration.



Running The Application

This section provides information on how to run the application using MPLAB X IDE.

Description

1. Build and Program the application using MPLAB X IDE.
2. Connect to the RS232 port and configure the serial settings of X2CScope as below:
   - Baud : 115200
   - Data : 8 Bits
   - Parity : None
   - Stop : 1 Bit
3. Observe the output as described in the following table:

Project Name	Output
mchv3_sam_e70_pim.X	1. Press the push button to start the motor. 2. Vary Potentiometer to change the speed of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the push button to stop the motor.
mclv2_sam_e70_pim.X	1. Press the Switch S2 to start the motor. 2. Vary Potentiometer to change the speed of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the Switch S2 to stop the motor.

PMSM FOC Using Quadrature Encoder

This Section provides help for the PMSM Field Oriented Control(FOC) using the Quadrature Encoder.

pmsm_foc_encoder_sam_e70

This motor control example project shows how to control Permanent Magnet Synchronous Motor (PMSM) using an Encoder based Field Oriented Control (FOC) on SAME70 Micro-controller.

Description

The permanent magnet synchronous motors (PMSM) is widely used in various industries due to its high power density, smaller size and high efficiency. The Field oriented control is one of the most popular control mechanisms for PMSM motor for applications which requires high dynamic performance.

This example shows how to configure motor control peripherals ADC, PWM for the control operation. The control strategy is the sensored FOC in which rotor position and speed is determined by the Quadrature Encoder.

The following sections describe the basics of the sensored FOC motor control algorithm and the steps to build, configure and run the PMSM motor for both low voltage PMSM motor drive.

Building The Application

This section provides information on how to build an application using MPLAB X IDE.

Description

The parent folder for all the MPLAB X projects for this application is given below:

Application Path	motor_control\apps\pmsm_foc_encoder_sam_e70

To build the application, refer the following table and open the appropriate project file in MPLAB X IDE.

Project Name	Description
mclv2_sam_e70_pim.X	MCLV2 Development Board (Low Voltage)

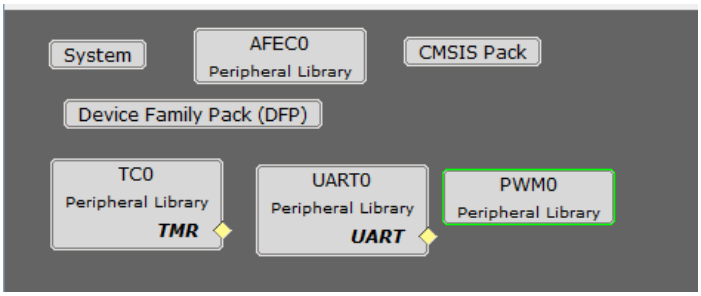
MPLAB Harmony Configurations

This topic provides information on the MHC configurations.

Description

The following project graph shows the MPLAB Harmony components that are included in this application:

mclv2_sam_e70_pim.X



Components Used

1. AFEC Peripheral
   - The AFEC (ADC) is used to measure analog quantities. Four channels are used to measure the Phase Current A, the Phase Current B, the DC Bus Voltage and the Potentiometer. Conversion is triggered at the PWM (zero match + offset of the switch delay)
2. PWM Peripheral
   - This peripheral is used to generated three phase synchronous PWM waveforms. Fault functionality is also enabled to switch off the output waveforms asynchronously
3. TC Peripheral
   - This peripheral is used in quadrature encoder mode for position measurement.
4. UART Peripheral
   - The UART is used for X2CScope communication to observe graphs and variable values run time

Control Algorithm

This section describes the control strategy and explains the control loop flow.

Description

The project implements an Encoder based sensored Field Oriented Control (FOC) algorithm on SAME70 32-bit micro-controller to regulate the speed of the PMSM motor. For details refer application note AN2757. The following section describes briefly about the FOC algorithm, and corresponding software design and implementation on MCLV2 development board.

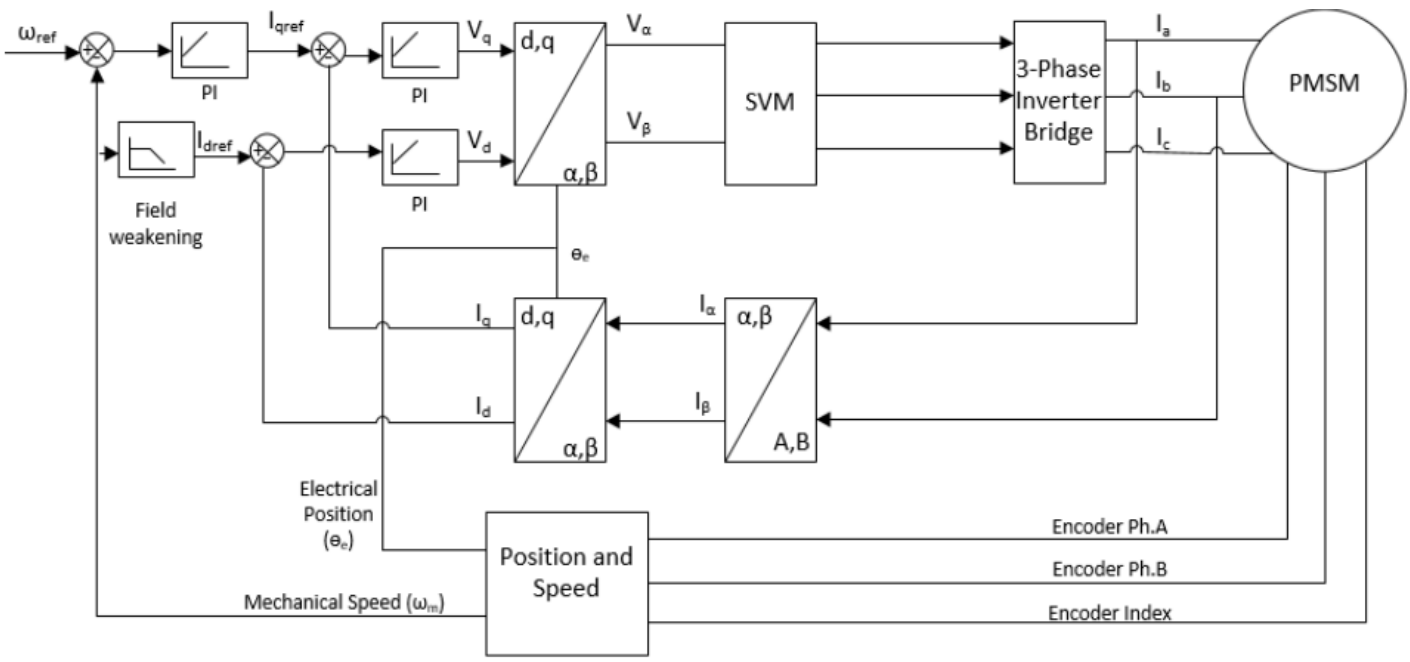
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from stationary reference frame to torque and flux producing currents components in rotating reference frame using mathematical transformations. The Field Oriented Control is achieved by performing following steps:

1. Measure the motor phase currents.
2. Transform them into the two phase system (a, b) using Clarke transformation.
3. Calculate the rotor position angle.
4. Transform stator currents into the d,q-coordinate system using Park transformation.
5. The stator current torque (i_q) and flux (i_d) producing components are controlled separately by the controllers.
6. The output stator voltage space vector is transformed back from the d,q-coordinate system into the two phase system fixed with the stator by inverse Park transformation.
7. Using the space vector modulation, the output three-phase voltage is generated.

The phase currents are measured using two shunt resistors connected to Phase U and Phase V inverter legs respectively. The W phase current is determined based on Kirchoff's current law.

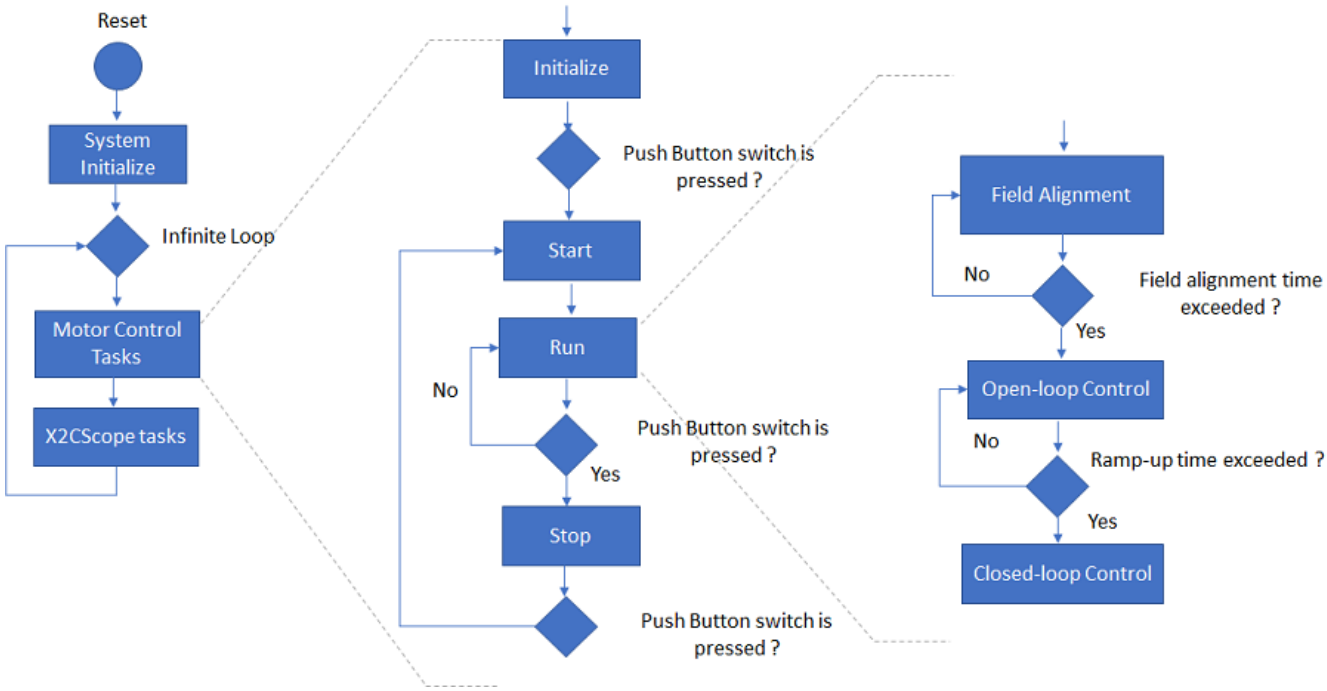
The rotor position angle is determined from Quadrature encoder. The details of the quadrature encoder interface can be referred from device datasheet.

The following block diagram shows the software realization of the FOC algorithm.



Software Design:

The following figure shows the various state machine of the the motor control software.



Brief description of the tasks undertaken in various motor control states are as follows-

Initialize:

In this state, following tasks are performed-

• Initialization/ configuration of NVIC, AFEC, PWM etc. motor control micro-controller peripherals for generation of periodic ADC triggers and ADC conversion interrupt.

• Current Offset measurement and calibration.

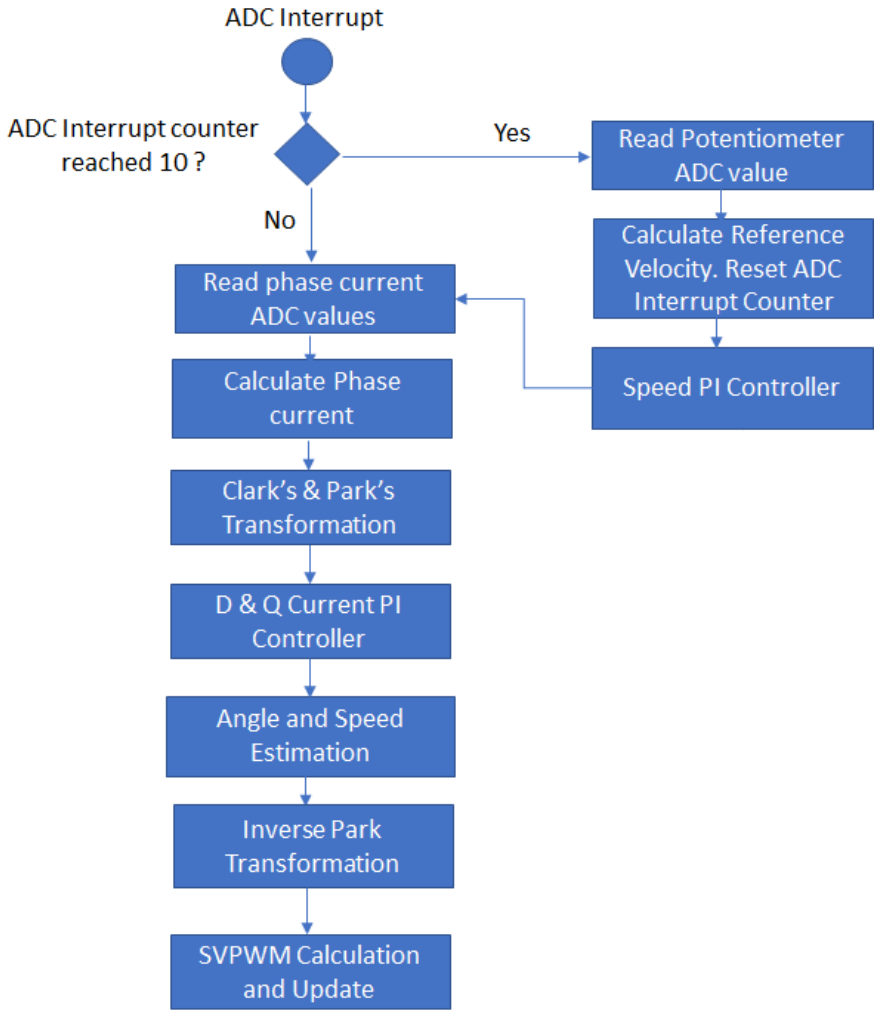
• Initialize PI controller parameters for speed and current control loops.

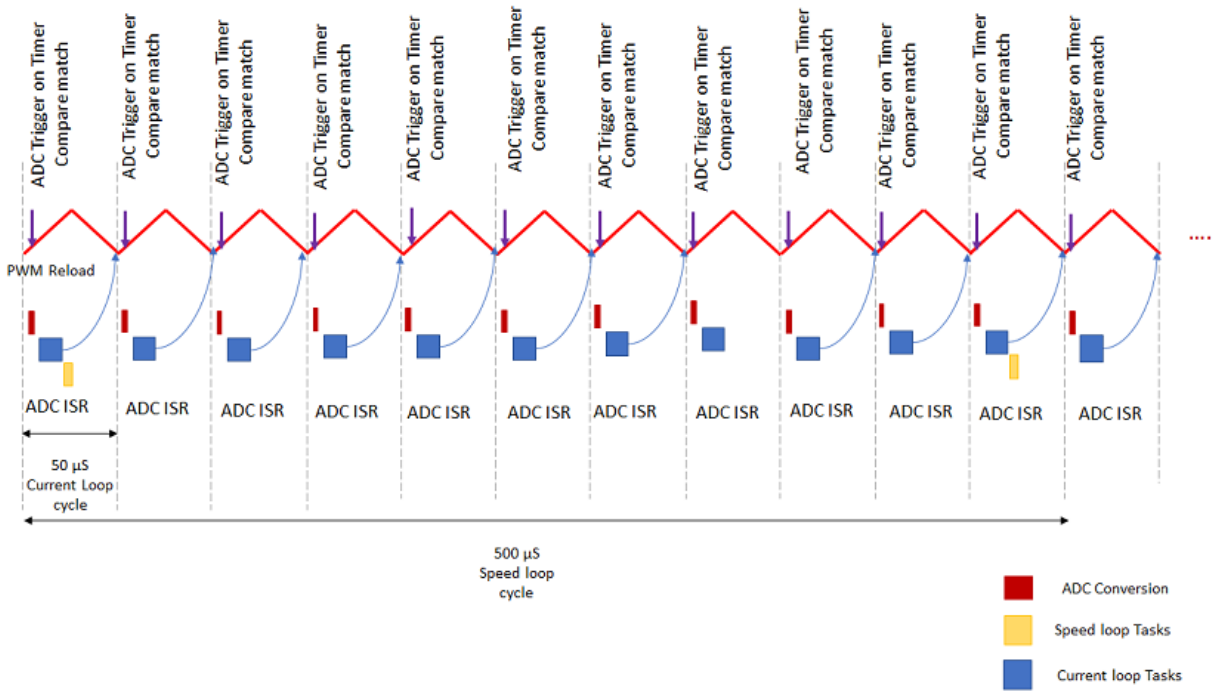
Start:

In this state, the motor control state variables reset and periodic ADC conversion interrupt is enabled.

Run:

In this state, the motor starts spinning. The below flow chart shows the tasks performed in run state-





In run state, two threads are executed- Main task thread and ADC Interrupt task thread. The current control is carried out in the ADC interrupt task thread, while the speed control is carried out in the main task thread. Hence the ADC interrupt cycle indicates the current control frequency. To maximum torque per ampere in the motor drive, a field alignment mode is integrated with the FOC algorithm. Brief description of field alignment and Close control is as follows-

• Field Alignment:

The quadrature encoder is an incremental encoder. Hence it is important to know the initial position of the rotor to maintain orthogonal rotor and stator flux. To achieve this, q-axis is excited with an small open loop current while d-axis is excited with zero current. After the initial alignment, electrical angle is shifted by 90 degrees to obtain maximum torque.

• Close Loop Control:

After the alignment, the close loop control by using FOC algorithm is done.

Stop:

In this state, the PWM channels are disabled thereby stopping the motor. The periodic ADC trigger and conversion interrupt is disabled.

Software Configuration:

The FOC algorithm is used for wide range of PMSM motors used across different application fields. In order to get the optimal control of the PMSM motor, both the motor specific parameters and application parameters has to be updated in the software. The following section describe how to update both motor and application specific parameters in the the project.

1. Setting motor control PWM frequency:

The PWM controller frequency (in Hz) can be configured by setting following macro in "userparams.h" file.

This frequency should be same as the frequency configured in PWM peripheral in the MHC.

Macro	Description	Unit
PWM_FREQUENCY	Current controller and PWM frequency	Hertz

2. Setting motor specific -parameter:

Set the motor following motor parameters in "userparams.h" file.

Macro	Description	Unit
MOTOR_PER_PHASE_RESISTANCE	Motor per phase resistance	Ohm
MOTOR_PER_PHASE_INDUCTANCE	Motor per phase inductance	Henry
MOTOR_BEMF_CONST_V_PEAK_LL_KRPM_MECH	Back EMF constant	Vpk(L-L)/KRPM
NUM_POLE_PAIRS	Number of pole pairs	
RATED_SPEED_RPM	Rated mechanical speed of the motor	rpm
MAX_SPEED_RPM	Maximum mechanical speed of the motor	rpm
ENCODER_PULSES_PER_REV	Number of encoder pulses per mechanical revolution	

3. Setting PI Controller parameters:

Depending on the type of motor used, and the corresponding application PI controller parameters has to be updated in "userparams.h" file.

Speed Control loop PI controller parameters:

Macro	Description
SPEEDCNTR_PTERM	Proportional gain of speed control loop
SPEEDCNTR_ITERM	Integral gain of speed control loop
SPEEDCNTR_CTERM	Anti-windup term of speed control loop
SPEEDCNTR_OUTMAX	Maximum controller output of speed control loop

I_d current Control loop PI controller parameters:

Macro	Description
D_CURRCNTR_PTERM	Proportional gain of I_d current control loop
D_CURRCNTR_ITERM	Integral gain of I_d current control loop
D_CURRCNTR_CTERM	Anti-windup term of I_d current control loop
D_CURRCNTR_OUTMAX	Maximum controller output of I_d current control loop

I_q current Control loop PI controller parameters:

Macro	Description
Q_CURRCNTR_PTERM	Proportional gain of I_q current control loop
Q_CURRCNTR_ITERM	Integral gain of I_q current control loop
Q_CURRCNTR_CTERM	Anti-windup term of I_q current control loop
Q_CURRCNTR_OUTMAX	Maximum controller output of I_q current control loop

6. Debugging Features:

The software also integrates features to enable user to configure/ tune their motor drive with ease. Some of them are listed as follows-

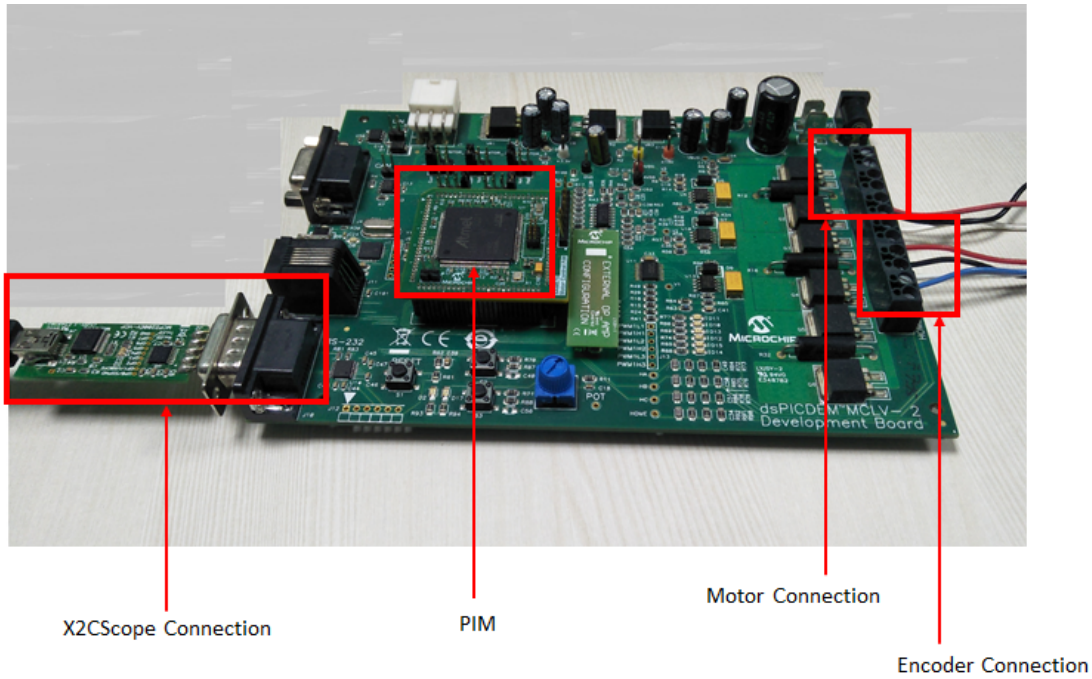
Macro	Description
TORQUE_MODE	This a special mode where the speed control loop can be bypassed. The input of potentiometer is scaled and is fed directly to the q-axis PI controller. Torque mode enable switch (0 - Disable, 1 - Enable)

Hardware Setup

This section describes how to configure the supported hardware.

Description

1. **Project mclv2_sam_e70_pim.X.**
   - **Hardware Used**
     - MCLV2 Development Board (Low Voltage)
     - ATSAME70 PIM
     - Long Hurst DMA0204024B101 Motor
   - **Hardware Setup**
     1. Mount the SAME70 144-pin Motor Control PIM on the dsPICDEM MCLV-2 Development Board, as shown in the following figure.
     2. Connect the External Op amp Configuration Matrix Board to header J14.
     3. Connect the three-phase wires coming from P1 header of the long Hurst motor (i.e., Red, White and Black to M1, M2 and M3 ports,respectively). Leave the green wire from the motor unconnected.
     4. Connect the quadrature encoder wires of the long hurst motor.
     5. Apply 24V DC at BP1-BP2/J2.
     6. This application 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). To use the RS-232 serial port, connect JP4 and JP5 to the UART position.
     7. Refer to Using the X2C Scope for details on how to use X2C Scope in this demonstration.



Running The Application

This section provides information on how to run the application using MPLAB X IDE.

Description

1. Build and Program the application using MPLAB X IDE.
2. Connect to the RS232 port and configure the serial settings of X2CScope as below:

- Baud : 115200
- Data : 8 Bits
- Parity : None
- Stop : 1 Bit

3. Press the switch S2 to start the motor.
4. Vary Potentiometer to change the speed of the motor.
5. Press the switch S2 to stop the motor.

PMSM FOC Using Reduced Order Luenberger Observer

This Section provides help for the Sensorless Field Oriented Control(FOC) of PMSM using Reduced Order Luenberger Observer (ROLO).

pmsm_foc_rolo_sam_c21

This application implements the sensorless field oriented control (FOC) of a permanent magnet synchronous motor (PMSM). It estimates rotor position from measured phase currents and a reduced order luenberger observer (ROLO). This algorithm is implemented on a SAMC21J18A MCU.

Building The Application

This section provides information on how to build an application using MPLAB X IDE.

Description

The parent folder for all the MPLAB X projects for this application is given below:

Application Path	motor_control\apps

To build the application, refer to the following table and open the appropriate project file in MPLAB X IDE.

Project Name	Description
mchv3_sam_c21_pim.X	pmsm_foc_rolo_sam_c21 is setup to work with ATSAMC21 Motor Control Plug In Module + dsPICDEM™ MCHV-3 Development Board (High Voltage)
mclv2_sam_c21_pim.X	pmsm_foc_rolo_sam_c21 is setup to work with ATSAMC21 Motor Control Plug In Module + dsPICDEM™ MCLV-2 Development Board (Low Voltage)

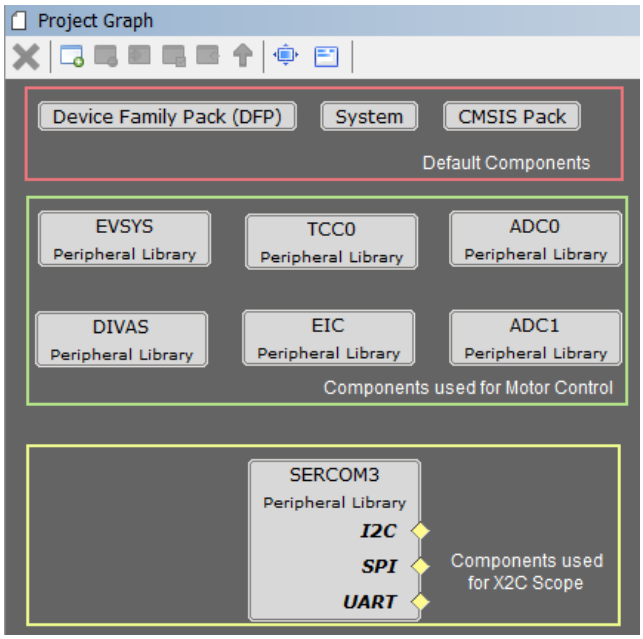
MPLAB Harmony Configurations

This topic provides information on the MHC configurations.

Description

The following project graph shows the MPLAB Harmony components that are included in this application:

pmsm_foc_rolo_sam_c21



Components Used

TCC0:

- Configured to generate 3 pairs of complimentary PWM signals at frequency of 10 kHz in "Dual Slope PWM with interrupt/event when counter = ZERO" a.k.a. "Center Aligned Mode".
- Event output is enabled which is generated when the counter reaches ZERO.
- Dead-time is enabled and set to 1 µs.
- Non-recoverable Fault is enabled on EV0. When an event is detected on EV0, all PWM channels are held low.

ADC0-ADC1:

- ADC0 and ADC1 are setup to operate in Master - Slave mode with ADC0 acting as a Master.
- Both ADCs convert single ended inputs. Phase U current is sampled and converted by ADC0 and Phase V current is sampled and converted by ADC1.
- Both ADCs are hardware triggered simultaneously by an event generated from TCC0 at the end of each PWM cycle.
- Conversion Ready interrupt is generated by ADC0. Since both ADCs are triggered simultaneously and have same resolution and sampling time, both ADCs complete conversion at the same time.

EIC:

- External Interrupt Controller detects hardware over-current fault input and generates a non-recoverable fault event for TCC0, thereby shutting down the PWM in the event of an over-current fault.

EVSYS:

- Event System acts as a traffic cop between event generator and event users.
- Event generated by the TCC0 when the counter reaches ZERO, is used by the ADC0 as a hardware trigger source via Event System.
- Event generated by the EIC upon over-current fault, is used by the TCC0 as a non-recoverable fault event via Event System.

DIVAS:

- This demo uses "Divide and Square Root Accelerator" to perform 32-bit signed and unsigned division and 32-bit unsigned square root operations.
- DIVAS is set to overload "Divide - / " operator i.e. c = a / b; would use DIVAS accelerator for division without using a special API. However, square root operation would still require a special API. Refer to the DIVAS help for more details.

SERCOM3:

- SERCOM3 is configured in USART mode and is set to operate at 115200 bps.
- This USART channel is used by the X2CScope plugin to plot or watch global variables in run-time. Refer to X2C Scope Plugin section for more details on how to install and use X2CScope.

Control Algorithm

This section describes the control strategy and explains the flow chart of the control loop implemented in this application.

Description

This application implements sensorless field oriented control (FOC) of permanent magnet synchronous motor (PMSM). It estimates rotor position from measured phase currents and a reduced order luenberger observer (ROLO). The algorithm is implemented on the SAMC21J18A MCU. For details of ROLO, refer to Application Note AN2590. The following section briefly describes the ROLO based sensorless FOC algorithm, software design and implementation.

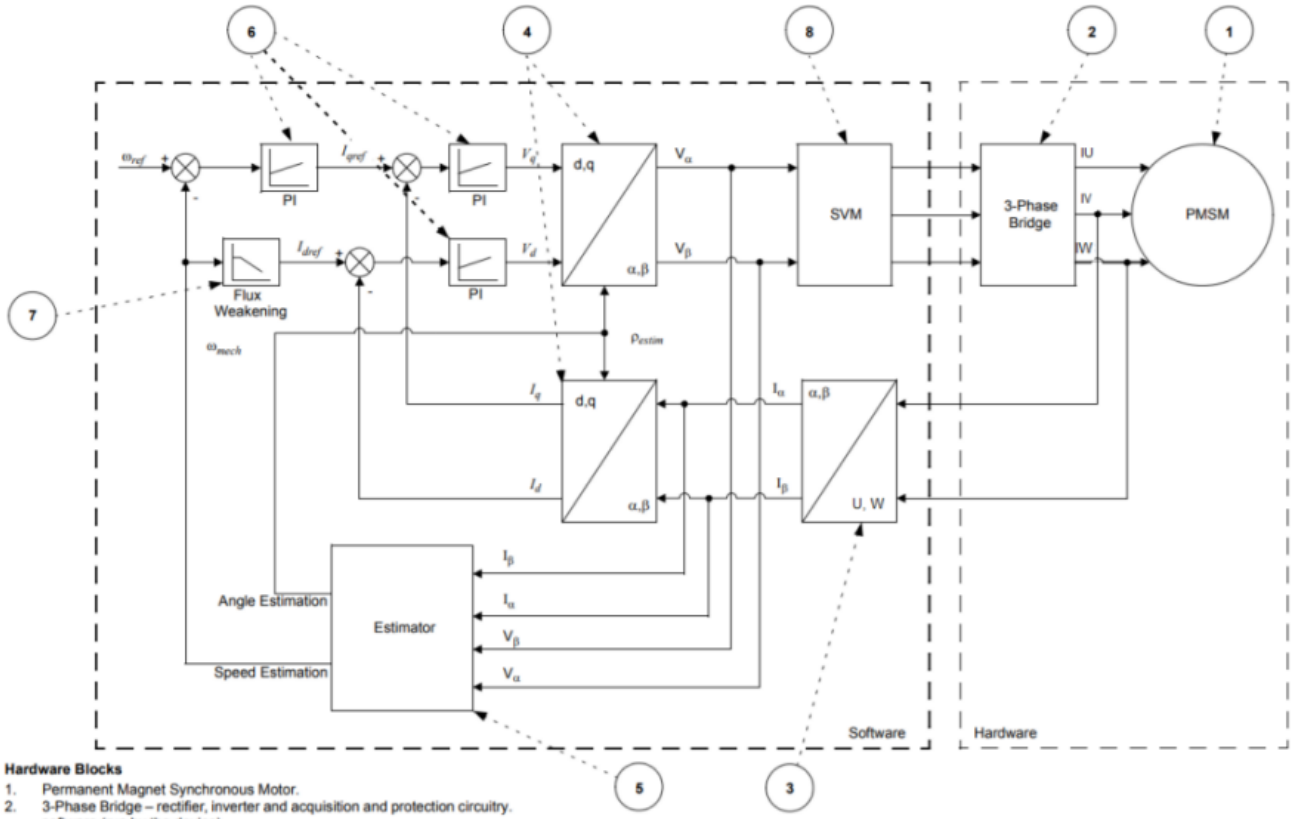
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from stationary reference frame to torque and flux producing currents components in rotating reference frame using mathematical transformations. The Field Oriented Control is implemented as follows:

1. Measure the motor phase currents.
2. Transform the three phase currents from three phase stationary reference frame (u,v,w) into two phase stationary reference frame (alpha, beta) using the Clarke transformation.
3. Estimate (sensorless) or measure (sensored) the rotor position angle.
4. Transform the stator currents from a two phase stationary reference frame (alpha, beta) into a two phase rotating reference frame (d,q) using the Park transformation. The rotating reference frame rotates synchronously with the rotor axis and therefore requires rotor angle information.
5. Clarke and Park transformation allows the transformation of three phase AC stator currents into an equivalent two phase DC currents which decouples flux (i_d) and torque (i_q) producing components of the stator currents.
6. The stator current torque (i_q) and flux (i_d) producing components are regulated independently by two independent PI controllers. These PI controllers generate an output voltage reference in d,q reference frame which is transformed into three phase stationary frame AC voltages using the inverse park and inverse clarke transformations.
7. In order to increase the DC bus utilization, the three phase sinusoidal AC output voltages are translated into space vector modulated voltages and applied across the stator windings using a Three Phase Half-Bridge Inverter.

The phase currents are measured using two shunt resistors connected to Phase U and Phase V inverter legs respectively. The W phase current is determined based on Kirchoff's current law.

The electrical rotor position is derived from back EMF voltage estimated by Reduced Order Luenberger Observer which is implemented in ***position_and_speed_estimation*** function. Back EMF voltage is a function of rotor speed. At lower rotor speeds, the estimated back EMF has poor signal to noise ratio and as a result does not provide accurate rotor angle estimation. Therefore, the FOC algorithm also integrates an open loop ramp-up profile for motor startup. The reference speed is incremented linearly until the required minimum reference speed is achieved such that the estimated back EMF has a high enough signal to noise ratio for accurate rotor angle estimation, at which point the algorithm uses estimated rotor angle for further control.

The following block diagram shows the software realization of the FOC algorithm.

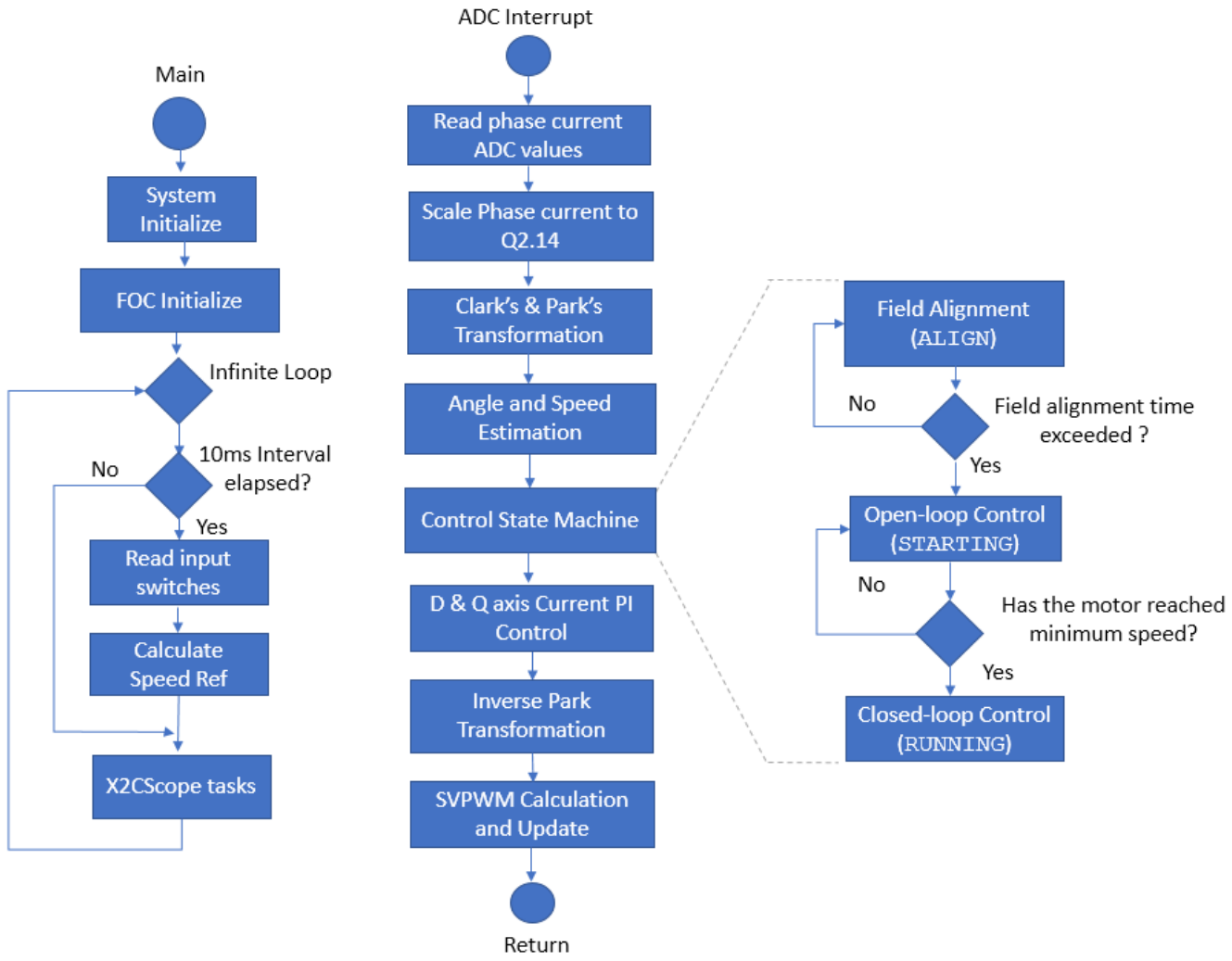


Hardware Blocks

1. Permanent Magnet Synchronous Motor.
2. 3-Phase Bridge – rectifier, inverter and acquisition and protection circuitry. software (run by the device).
3. Clarke forward transform.
4. Park forward and inverse transform.
5. Angle and speed estimator.
6. Proportional integral controller.
7. Flux weakening.
8. Space vector modulation.

Software Design:

The following figure shows the flow chart of the implemented algorithm:



Field Alignment - `ALIGN`:

In this mode, a pre-defined value of the current is asserted across the Q axis of the rotor for a pre-defined length of time, in order to align the rotor to a known angle of 90 electrical degrees. The magnitude of the current and the length of the time for which it is applied depends upon the electrical and mechanical time constant of the PMSM motor drive. Electrical time constant of the motor is a function of R and L values of the motor windings, whereas the mechanical time constant of the motor drive is primarily a function of the static load on the motor shaft.

Open Loop Control - `STARTING`:

In this mode, the speed of the PMSM motor is gradually ramped up using an open loop control. During this mode, the rotor angle is derived from the asserted open loop speed reference. This derived rotor angle would be lagging from the actual rotor angle. The speed is ramped up linearly to a minimum value required for the ROLO-based estimator to estimate the alpha and beta axis back EMF of the PMSM motor with sufficient accuracy. Rotor angle information is extracted from arctan($BEMF_{beta}$/ $BEMF_{alpha}$).

Close Loop Control - `RUNNING`:

In this mode, the estimated rotor angle obtained from ROLO-based estimator is used to perform field oriented control of the PMSM motor.

Timing Diagram:

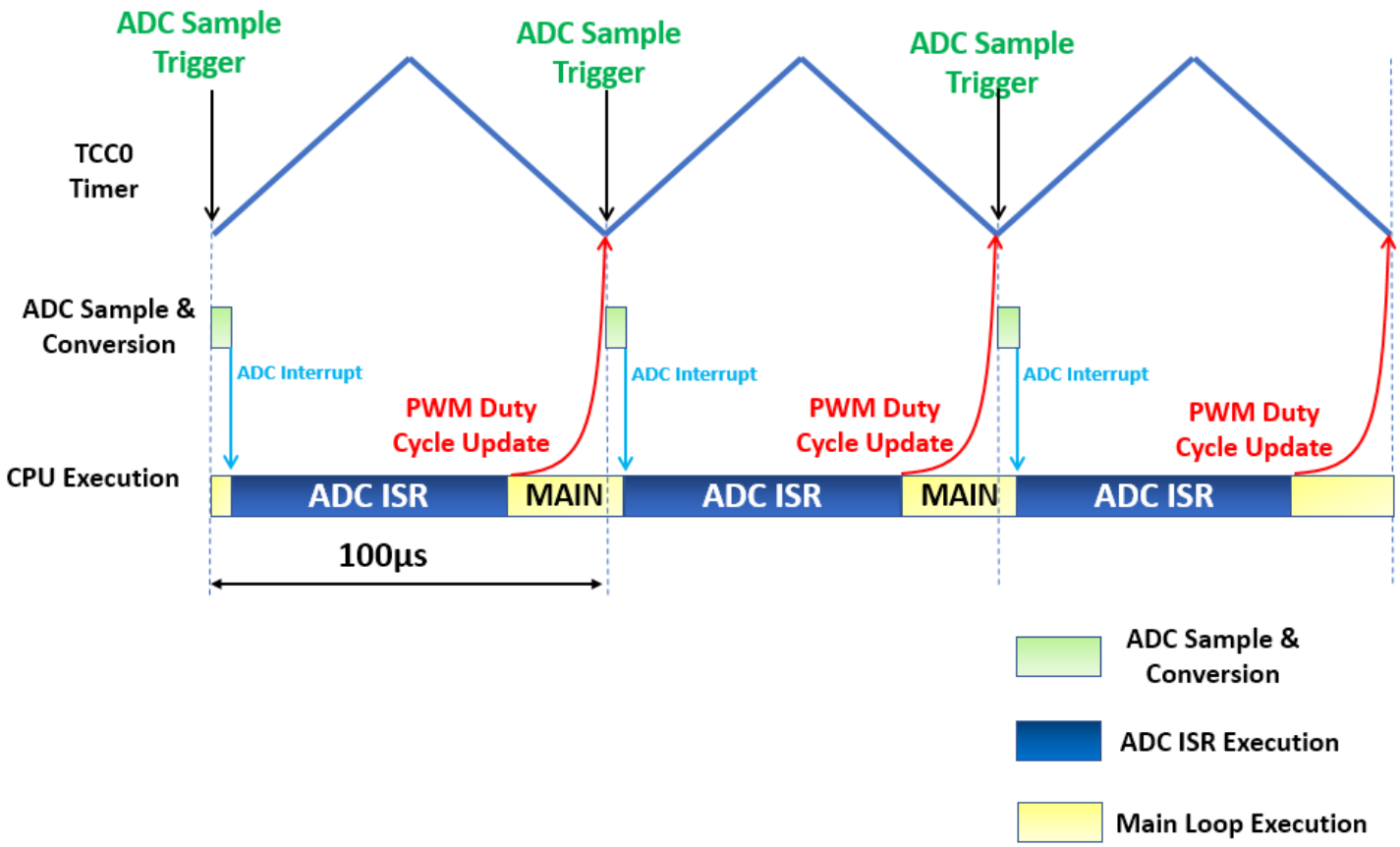
The three critical blocks of the MCU which are involved motor control operations are:

- ADC - It measures motor phase currents at the start of the PWM cycle.
- CPU - Based on motor phase currents measurements, CPU executes the control algorithm to calculate the PWM duty cycle

needed to apply required voltage across motor windings.

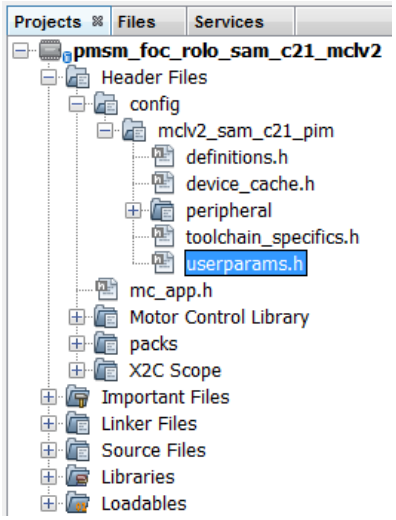
- TCC0 - It generates the PWM signals needed to apply required voltage across motor windings.

The following timing diagram, chronological role played by the ADC, CPU and TCC0 in the execution of FOC algorithm.



Software Configuration:

The FOC algorithm is used across different application fields. In order to get the optimal control of the PMSM motor, the motor specific parameters, board specific parameters and application parameter macros have to be updated in the software. The following section describes how to update both motor and application specific parameters in the the project. These parameter macros can be found in userparams.h header file which can be found under logical path: *Header Files > config > <config_name> >userparams.h* in MPLABX IDE..

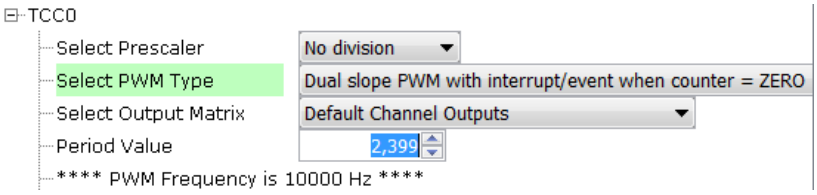


1. Setting motor control PWM frequency and dead-time:

PWM frequency is set by configuring the timer period of the TCC0 in terms of TCC0 clock counts. This frequency needs to be

communicated to other sections in the algorithm by defining the "Period Value" (set in MPLAB Harmony Configurator) + 1 as a macro in userparams.h.

For example, in order to achieve a PWM frequency of 10KHz in center aligned mode with peripheral clock frequency of 48MHz, the period value defined in MHC is 2399. Therefore, the macro defined in userparams.h should be "period value" + 1 = 2400.



Similarly, dead-time is configured in terms of TCC0 clock counts in the MHC. The same value needs to be defined as macro in `userparams.h`.

Macro	Description
PWM_HPER_TICKS	PWM Timer period value = "Period value" defined in MHC + 1
DEAD_TICKS	Dead-time defined in terms of PWM timer ticks = "Dead time value" defined in MHC

2. Setting motor specific parameters:

Set the following motor specific parameters in the `userparams.h` file.

Macro	Description	Units
MAX_FRE_HZ	Electrical Frequency at maximum motor speed	Hz
MIN_FRE_HZ	Electrical Frequency at minimum motor speed	Hz
POLAR_COUPLES	Number of Pole Pairs	-
R_STA	Motor per phase resistance	Ohm
L_SYN	Motor per phase inductance	H
MAX_CUR_AMP	Maximum motor current	A
START_CUR_AMP	Current reference during field alignment and open loop startup	A
ACC_RPM_S	Acceleration ramp rate	RPM/s
DEC_RPM_S	Deceleration ramp rate	RPM/s
STUP_ACCTIME_S	Open loop start up time	s
CUR_RISE_T	Current rise time during field alignment	s

3. Setting board specific parameters:

Set the following board specific parameters in the `userparams.h` file.

Macro	Description	Units
CUR_SGN_REV	Define this macro, if the terminal of the shunt resistor connected to the return path of the inverter is connected to the inverting terminal of the signal conditioning differential operational amplifier. Do Not define this macro, if the terminal of the shunt resistor connected to the return path of the inverter is connected to the non-inverting terminal of the signal conditioning differential operational amplifier.	-
AD_RBA	Current measurement resolution of the board. This value is a function of shunt resistor, gain and bias of the differential opamp, reference voltage and the resolution of the ADC. For example, a board with a shunt resistance of 25 mOhm, differential opamp with a gain of 15 and bias of 1.65V, 12-bit ADC with reference voltage 3.3V, is capable of measuring +/- 4.4A i.e. a total of 8.8 A before ADC saturates. For such a board, AD_RBA = 4095/8.8 = 465.34 bits/A.	bits/A
AD_RBV	Bus voltage measurement resolution of the board. This value is a function of DC bus voltage divider ratio, reference voltage and resolution of the ADC. For example, a board with a DC bus voltage divider ratio of 1:16, 12-bit ADC with reference voltage 3.3V, is capable of measuring up to 52.8 V before ADC saturates. For such a board, AD_RBV = 4095/52.8 = 77.55 bits/V.	bits/V

4. Setting PI Controller parameters:

Depending on the type of motor used, and the corresponding application PI controller parameters should be updated in the `userparams.h` file.

Speed PI controller gains:

Macro	Description
KP_AS_R	Proportional gain of speed controller
KI_A_R	Integral gain of speed controller

Current PI controller gains for D and Q axis currents:

Macro	Description
KP_V_A	Proportional gain for D and Q axis current controller.
KI_V_AS	Integral gain for D and Q axis current controller.

5. Debug & Optimization Modes :

This example provides compile time re-configuration options using #define macro directives to enable or disable different debug and optimization modes. These macro directives can be accessed in the `userparams.h` header file.

- Current PI Tuning - `CURPI_TUN`
  - This mode allows tuning of the Current PI controllers. This mode generates a step current reference and depending upon the actual current response, user can accordingly adjust current PI gains.
  - This mode can be enabled by defining the macro `CURPI_TUN` in `userparams.h` header file.
  - In this mode, the step current reference is asserted across the D axis.
  - Macro `CUR_STEP_AMP` defines the step size of the asserted D axis current reference in amperes and macro `CUR_STEP_TIM` defines the length of the step in seconds.
- Open Loop Mode - `OPEN_LOOP_FUNCTIONING`
  - This mode is used to debug the open loop startup of the motor. This mode could also be used in tuning of Current PI controllers.
  - In this mode, the motor operates in open loop rotor angle mode. In other words, the rotor angle used by the algorithm to assert current will be different than actual rotor angle and therefore, the asserted current may not be orthogonal to the rotor magnetic axis, resulting in lower torque generation.
  - This mode can be enabled by defining the macro `OPEN_LOOP_FUNCTIONING` in `userparams.h` header file.
  - The torque current reference in this mode is defined by the macro `START_CUR_AMP`.
  - Depending upon the load inertia and startup acceleration time, defined by the macro `STUP_ACCTIME_S`, if the generated torque is not sufficient, then it may result in stalling of the motor.
  - `START_CUR_AMP` and `STUP_ACCTIME_S` need to be modified to ensure that the motor starts up successfully in open loop mode without stalling. If the motor stalls and the PWM signals are shutdown, the algorithm enters into a critical fault mode. The MCU needs to be reset to recover from the critical fault.
- Torque Mode - `TORQUE_MODE`
  - In this mode, as the name suggests, the algorithm operates in torque mode. The torque reference is obtained from the potentiometer.
  - This mode can be enabled by defining the macro `TORQUE_MODE` in `userparams.h` header file.
  - The potentiometer min - max input is scaled to generate torque current reference in the range of `TORQUE_MODE_MIN_CUR_AMP` - `MAX_CUR_AMP`.
  - Depending upon the load, if the torque is too low then it could cause the motor to stall and algorithm to enter a critical fault mode. The MCU needs to be reset to recover from the critical fault.
- Executing from RAM - `RAM_EXECUTE`
  - In order to speed up execution, the frequently called motor control related functions can be placed and fetched from RAM during execution.
  - Fetching instructions from RAM is faster than fetching instructions from Flash, as fetches from RAM can occur at CPU clock speed, unlike the Flash, which requires wait states to be asserted.
  - This speed optimization feature can be enabled by defining the macro `RAM_EXECUTE` in `userparams.h` header file.
  - Enabling this mode would result in a larger data memory footprint.

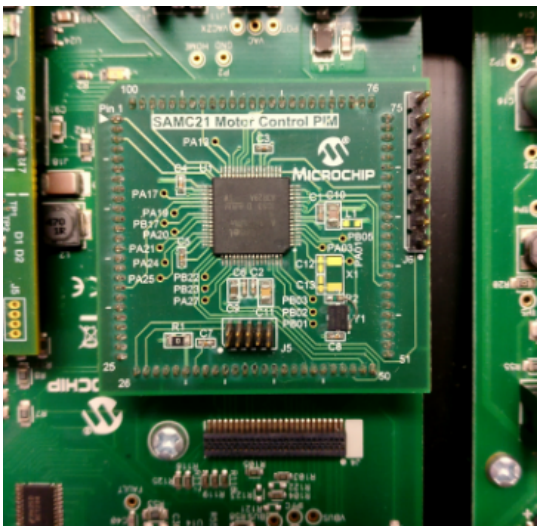
- Exercise caution while using this feature as the functions in RAM are vulnerable to get overwritten due to stack overruns or unguarded buffer overruns.

Hardware Setup

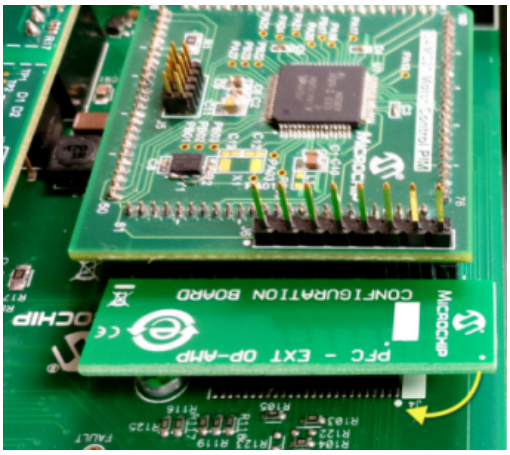
This section describes how to configure the supported hardware.

Description

1. **Project mchv3_sam_c21_pim.X.**
   - **Hardware Used**
     - dsPICDEMTM MCHV3 Development Board (High Voltage)
     - ATSAMC21 Motor Control Plug In Module
     - Isolated Embedded Debugger Interface for MCHV
     - Leadshine EL5-M0400-1-24 Motor
   - **Hardware Setup**
     1. Mount the ATSAMC21 Motor Control Plug In Module on U9 header.



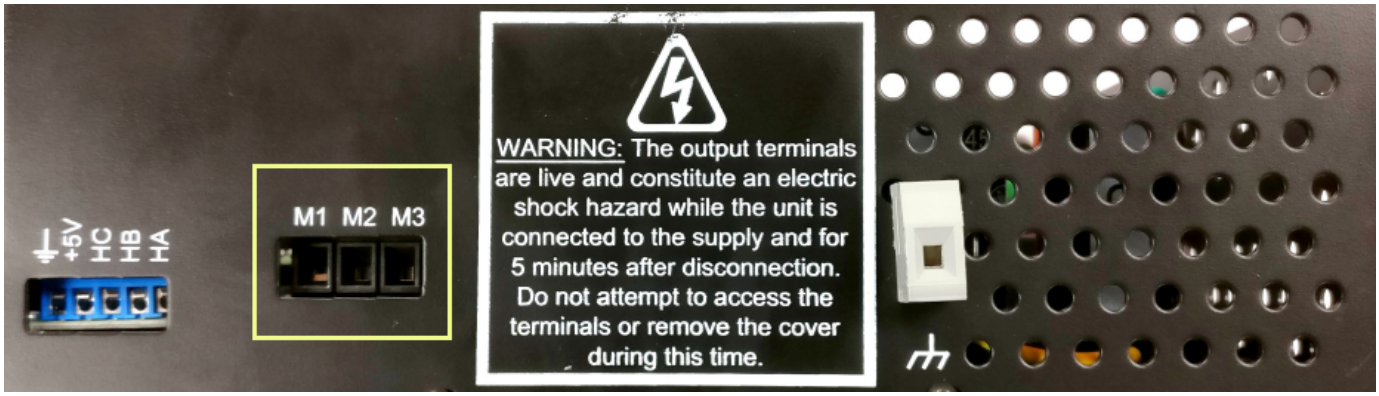
2. Place the "PFC - External Opamp Configuration" Matrix board at J4.



3. Motor Connections:

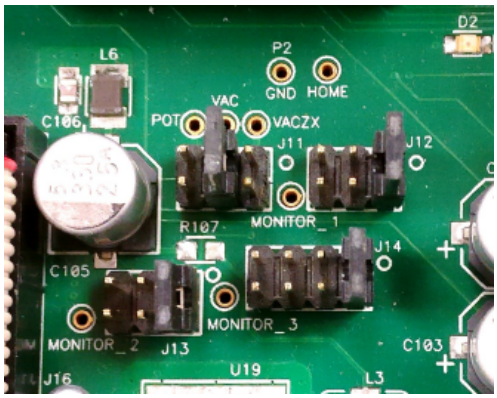
- Phase U - M1

- Phase V - M2
- Phase W - M3

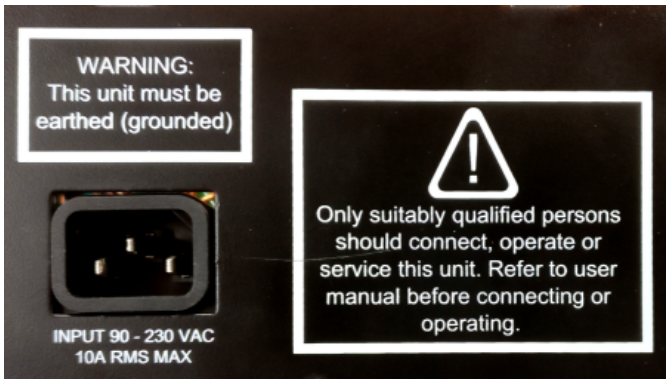


4. Jumper Settings:

- J11 - VAC (Short Pin 3 - 4)
- J12 - IA (Short Pin 1 - 2)
- J13 - IB (Short Pin 1 - 2)
- J14 - Fault_IP/IBUS (Short Pin 1 - 2)

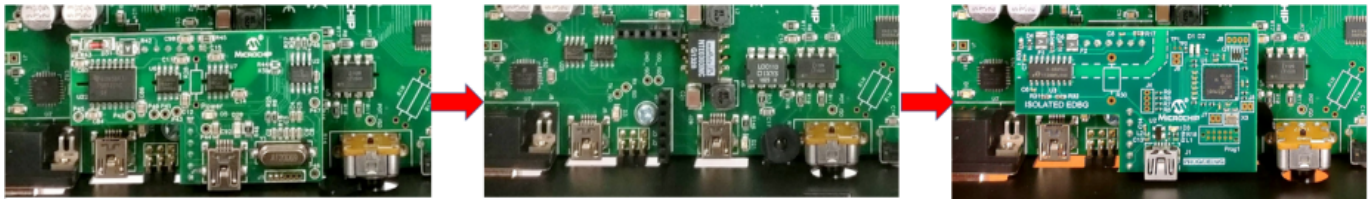


5. Power the board with (110V/220V) AC mains. For additional safety, it is recommended to use a current limited power supply while testing this software demonstration on a non-default hardware and motor.

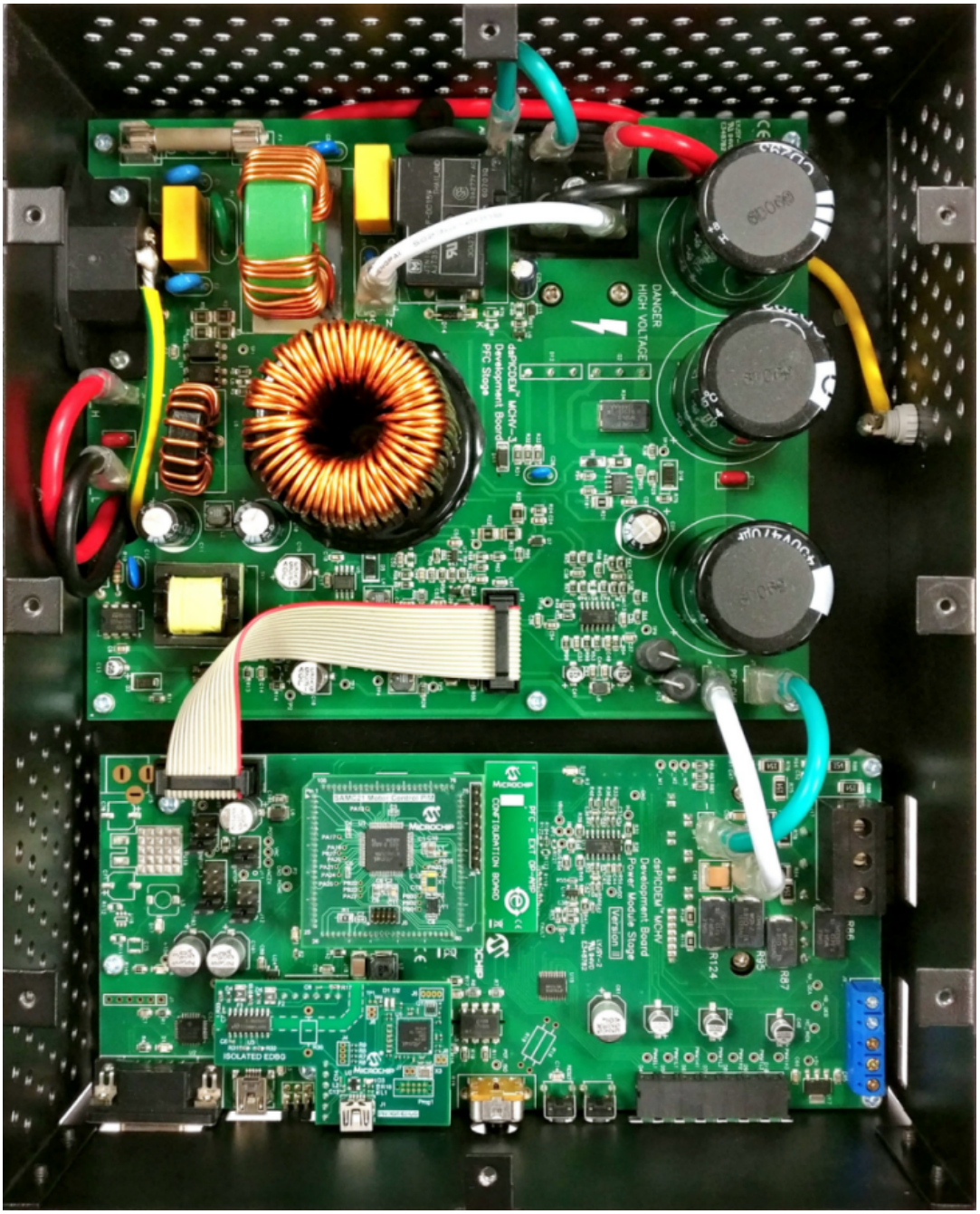


- **Installing Isolated Embedded Debugger**
  - Default programmer or debugger daughter card shipped with the MCHV3 board cannot program or debug SAM series MCU

and therefore, it needs to be replaced with an Isolated Embedded Debugger Interface for MCHV.



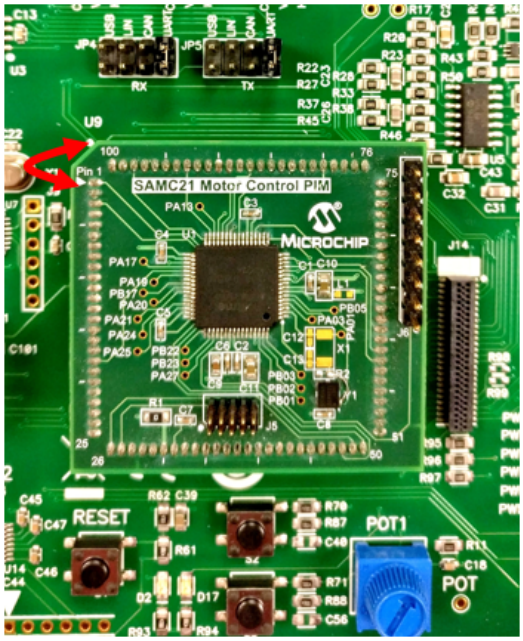
- Complete Setup



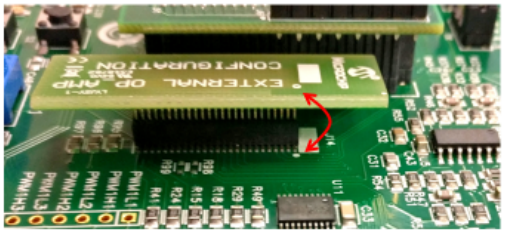
2. **Project mclv2_sam_c21_pim.X.**

- **Hardware Used**
  - dsPICDEMTM MCLV2 Development Board (Low Voltage)
  - ATSAMC21 Motor Control Plug In Module
  - Long Hurst DMA0204024B101 Motor
- **Hardware Setup**

  1. Mount the ATSAMC21 Motor Control Plug In Module on U9 header.

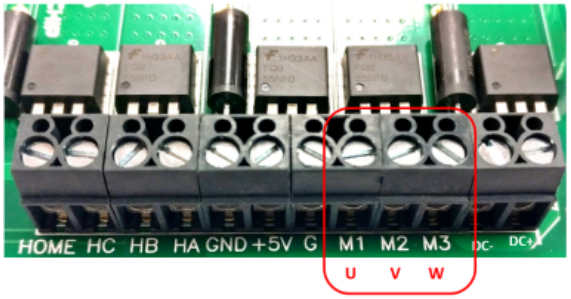


2. Place the "External Opamp Configuration" Matrix board at J14.



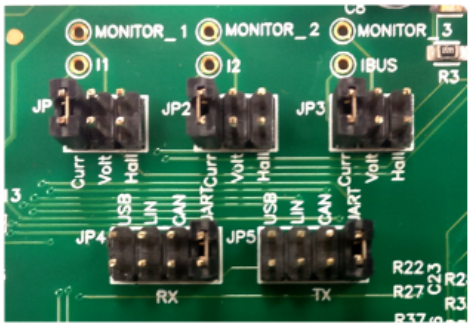
3. Motor Connections:

- Phase U - M1
- Phase V - M2
- Phase W - M3

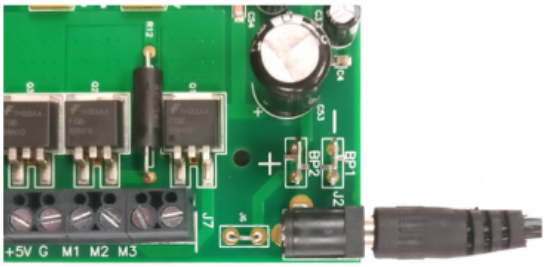


4. Jumper Settings:

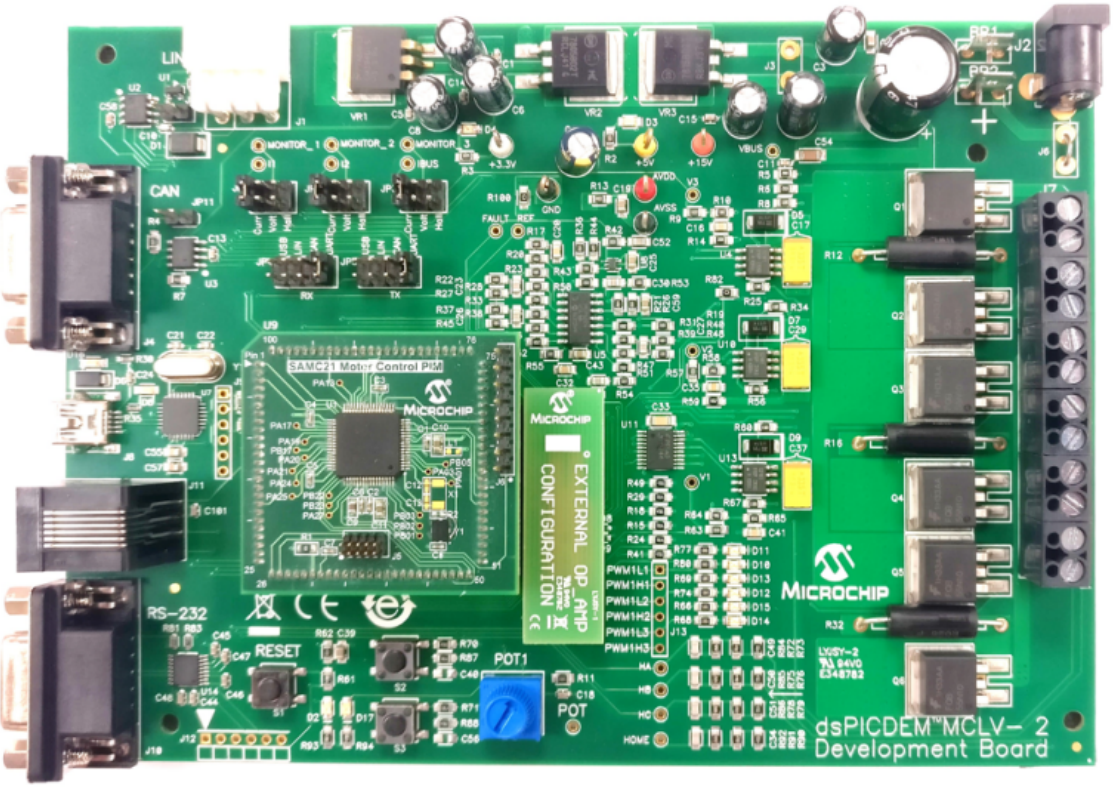
- JP1 - Curr, JP2 - Curr, JP3 - Curr
- In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



5. Power the board with a 24V DC supply using J2 or BP1-BP2. For additional safety, it is recommended to use a current limited power supply while testing this software demonstration on a non-default hardware and motor.



- **Complete Setup**



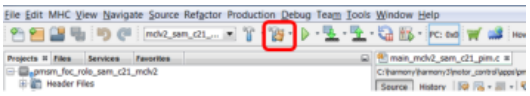
Running The Application

This section provides information on how to run the application using MPLAB X IDE.

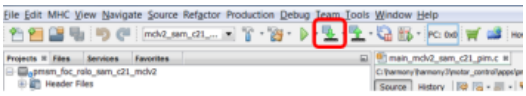
Description

Running the Application : pmsm_foc_rolo_sam_c21 - MCHV3

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_rolo_sam_c21\firmware\mchv3_sam_c21_pim.X
2. Clean and Build Project:



3. Program the project:



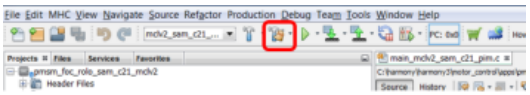
4. Press the switch "Push Button" to start/stop the motor.
5. Vary Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_rolo_wm_sam_c21 - MCHV3

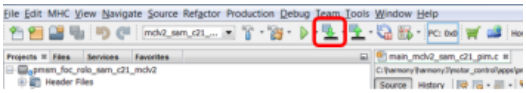
If LED D19 is ON, it indicates a non-recoverable fault caused by either over-current, phase loss or due to erroneous rotor position estimation. The MCU needs to be reset in order to recover from this fault mode.

Running the Application : pmsm_foc_rolo_sam_c21 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_rolo_sam_c21\firmware\mclv2_sam_c21_pim.X
2. Clean and Build Project:



3. Program the project:



4. Press the switch S2 to start/stop the motor.
5. Press the switch S3 to toggle motor direction.

LED D2 Status	Description
OFF	Motor spin direction is "positive"
ON	Motor spin direction is "negative"

NOTE: Direction toggle command is accepted only when the motor is stationary.

6. Vary the Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_rolo_sam_c21 - MCLV2

If LED D17 is ON, it indicates a non-recoverable fault caused by either over-current, phase loss or due to erroneous rotor position estimation. The MCU needs to be reset in order to recover from this fault mode.

Project Name	Output
`mchv3_sam_c21_pim.X`	1. Press the push button to start the motor. 2. Vary Potentiometer to change the speed of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the push button to stop the motor.
`mclv2_sam_c21_pim.X`	1. Press the Switch S2 to start the motor. 2. Vary Potentiometer to change the speed of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the Switch S2 to stop the motor.

PMSM FOC Using Reduced Order Luenberger Observer with Windmilling Capability

This Section provides help for the Sensorless Field Oriented Control(FOC) of PMSM using Reduced Order Luenberger Observer (ROLO) with Windmilling capability.

pmsm_foc_rolo_wm_sam_c21

This application implements sensorless field oriented control (FOC) of permanent magnet synchronous motor (PMSM). It estimates rotor position from measured phase currents and a reduced order luenberger observer (ROLO). This algorithm has the capability to achieve a "sensorless" estimation of rotor angle, speed and direction of a freewheeling or "Windmilling" PMSM motor and achieve a smooth on-the-fly start if the command direction is same as freewheeling direction. This algorithm is implemented on SAMC21J18A MCU.

Building The Application

This section provides information on how to build an application using MPLAB X IDE.

Description

The parent folder for all the MPLAB X projects for this application is given below:

Application Path	motor_control\apps

To build the application, refer to the following table and open the appropriate project file in MPLAB X IDE.

Project Name	Description
mchv3_sam_c21_pim.X	pmsm_foc_rolo_wm_sam_c21 is setup to work with ATSAMC21 Motor Control Plug In Module + dsPICDEM™ MCHV-3 Development Board (High Voltage)
mclv2_sam_c21_pim.X	pmsm_foc_rolo_wm_sam_c21 is setup to work with ATSAMC21 Motor Control Plug In Module + dsPICDEM™ MCLV-2 Development Board (Low Voltage)

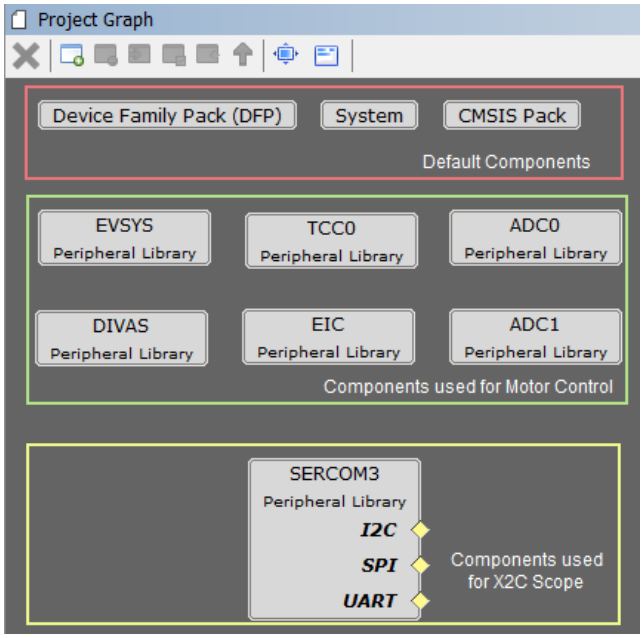
MPLAB Harmony Configurations

This topic provides information on the MHC configurations.

Description

The following project graph shows the MPLAB Harmony components that are included in this application:

pmsm_foc_rolo_wm_sam_c21



Components Used

TCC0:

- Configured to generate 3 pairs of complimentary PWM signals at frequency of 10 kHz in "Dual Slope PWM with interrupt/event when counter = ZERO" a.k.a. "Center Aligned Mode".
- Event output is enabled which is generated when the counter reaches ZERO.
- Dead-time is enabled and set to 1 µs.
- Non-recoverable Fault is enabled on EV0. When an event is detected on EV0, all PWM channels are held low.

ADC0-ADC1:

- ADC0 and ADC1 are setup to operate in Master - Slave mode with ADC0 acting as a Master.
- Both ADCs convert single ended inputs. Phase U current is sampled and converted by ADC0 and Phase V current is sampled and converted by ADC1.
- Both ADCs are hardware triggered simultaneously by an event generated from TCC0 at the end of each PWM cycle.
- Conversion Ready interrupt is generated by ADC0. Since both ADCs are triggered simultaneously and have same resolution and sampling time, both ADCs complete conversion at the same time.

EIC:

- External Interrupt Controller detects hardware over-current fault input and generates a non-recoverable fault event for TCC0, thereby shutting down the PWM in the event of an over-current fault.

EVSYS:

- Event System acts as a traffic cop between event generator and event users.
- Event generated by the TCC0 when the counter reaches ZERO, is used by the ADC0 as a hardware trigger source via Event System.
- Event generated by the EIC upon over-current fault, is used by the TCC0 as a non-recoverable fault event via Event System.

DIVAS:

- This demo uses "Divide and Square Root Accelerator" to perform 32-bit signed and unsigned division and 32-bit unsigned square root operations.
- DIVAS is set to overload "Divide - / " operator i.e. c = a / b; would use DIVAS accelerator for division without using a special API. However, square root operation would still require a special API. Refer to the DIVAS help for more details.

SERCOM3:

- SERCOM3 is configured in USART mode and is set to operate at 115200 bps.
- This USART channel is used by the X2CScope plugin to plot or watch global variables in run-time. Refer to X2C Scope Plugin section for more details on how to install and use X2CScope.

Control Algorithm

This section describes the control strategy and explains the flow chart of the control loop implemented in this application.

Description

This application implements sensorless field oriented control (FOC) of permanent magnet synchronous motor (PMSM) with "windmilling" capability. It estimates rotor position from measured phase currents and a reduced order luenberger observer (ROLO). The algorithm is implemented on the SAMC21J18A MCU. For details of ROLO, refer to Application Note AN2590. The following section briefly describes the ROLO based sensorless FOC algorithm, software design and implementation.

Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from stationary reference frame to torque and flux producing currents components in rotating reference frame using mathematical transformations. The Field Oriented Control is implemented as follows:

1. Measure the motor phase currents.
2. Transform the three phase currents from three phase stationary reference frame (u,v,w) into two phase stationary reference frame (alpha, beta) using the Clarke transformation.
3. Estimate (sensorless) or measure (sensored) the rotor position angle.
4. Transform the stator currents from a two phase stationary reference frame (alpha, beta) into a two phase rotating reference frame (d,q) using the Park transformation. The rotating reference frame rotates synchronously with the rotor axis and therefore requires rotor angle information.
5. Clarke and Park transformation allows the transformation of three phase AC stator currents into an equivalent two phase DC currents which decouples flux (i_d) and torque (i_q) producing components of the stator currents.
6. The stator current torque (i_q) and flux (i_d) producing components are regulated independently by two independent PI controllers. These PI controllers generate an output voltage reference in d,q reference frame which is transformed into three phase stationary frame AC voltages using the inverse park and inverse clarke transformations.
7. In order to increase the DC bus utilization, the three phase sinusoidal AC output voltages are translated into space vector modulated voltages and applied across the stator windings using a Three Phase Half-Bridge Inverter.

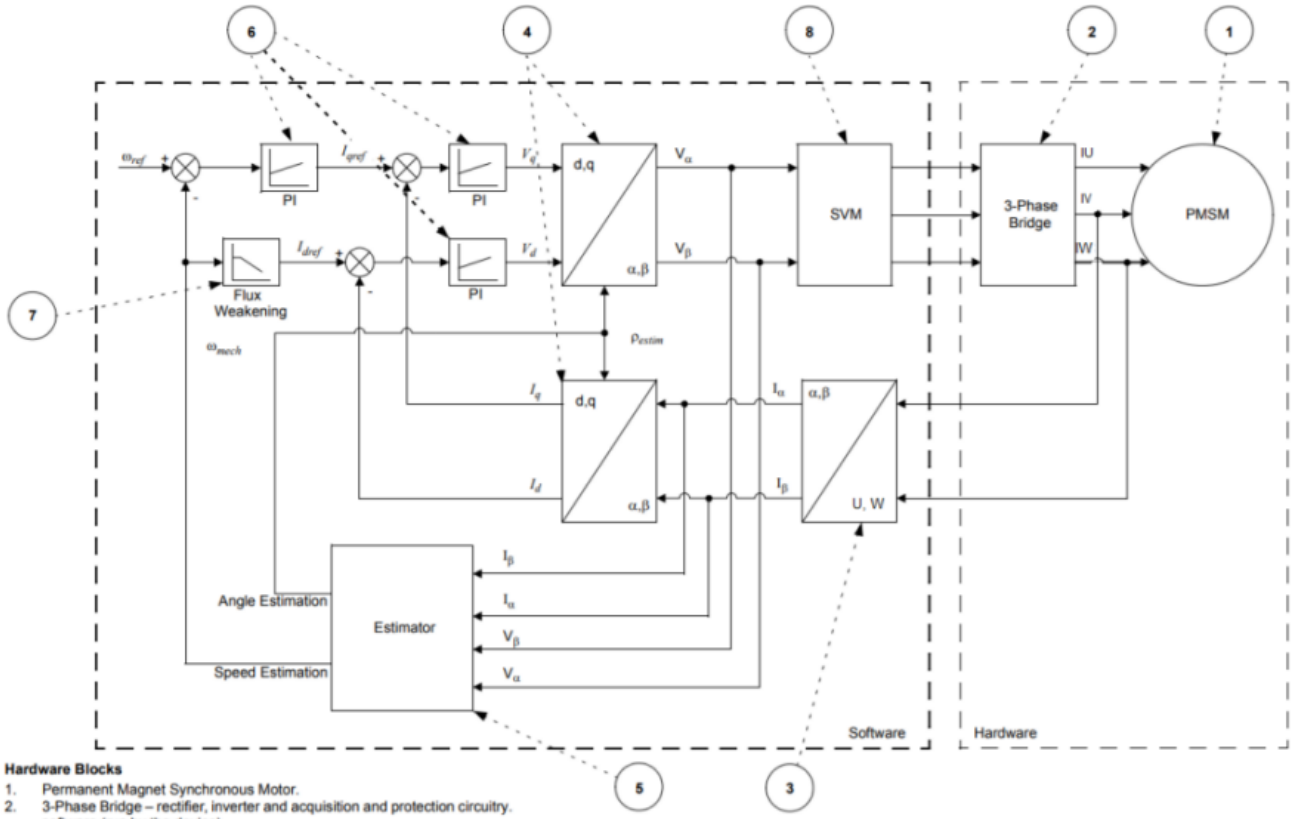
The phase currents are measured using two shunt resistors connected to Phase U and Phase V inverter legs respectively. The W phase current is determined based on Kirchoff's current law.

The electrical rotor position is derived from back EMF voltage estimated by Reduced Order Luenberger Observer. In this demonstration, the motor operation can be divided into two modes:

1. Windmilling mode - In this mode, the motor is not actively energized by the inverter but the motor shaft is "Windmilling" or freewheeling due to stored momentum from a prior active operation or external force, for example, ambient wind flow causing an un-energized fan motor to freewheel. In this mode, the rotor angle estimation is performed by ***bemf_position_and_speed_estimation*** function.
2. Normal Run mode - In this mode, the motor is actively energized by the inverter and operated in speed regulation mode. In this mode, the rotor angle estimation is performed by ***position_and_speed_estimation*** function.

Since, the rotor angle is derived from the estimated back EMF voltage, which is a function of rotor speed; at lower rotor speeds, the estimated back EMF has poor signal to noise ratio and as a result does not provide accurate rotor angle estimation. Therefore, the FOC algorithm in "Normal Run mode" integrates an open loop ramp-up profile for motor startup. The reference speed is incremented linearly until the required minimum reference speed is achieved such that the estimated back EMF has a high enough signal to noise ratio for accurate rotor angle estimation, at which point the algorithm uses estimated rotor angle for further control.

The following block diagram shows the software realization of the FOC algorithm.

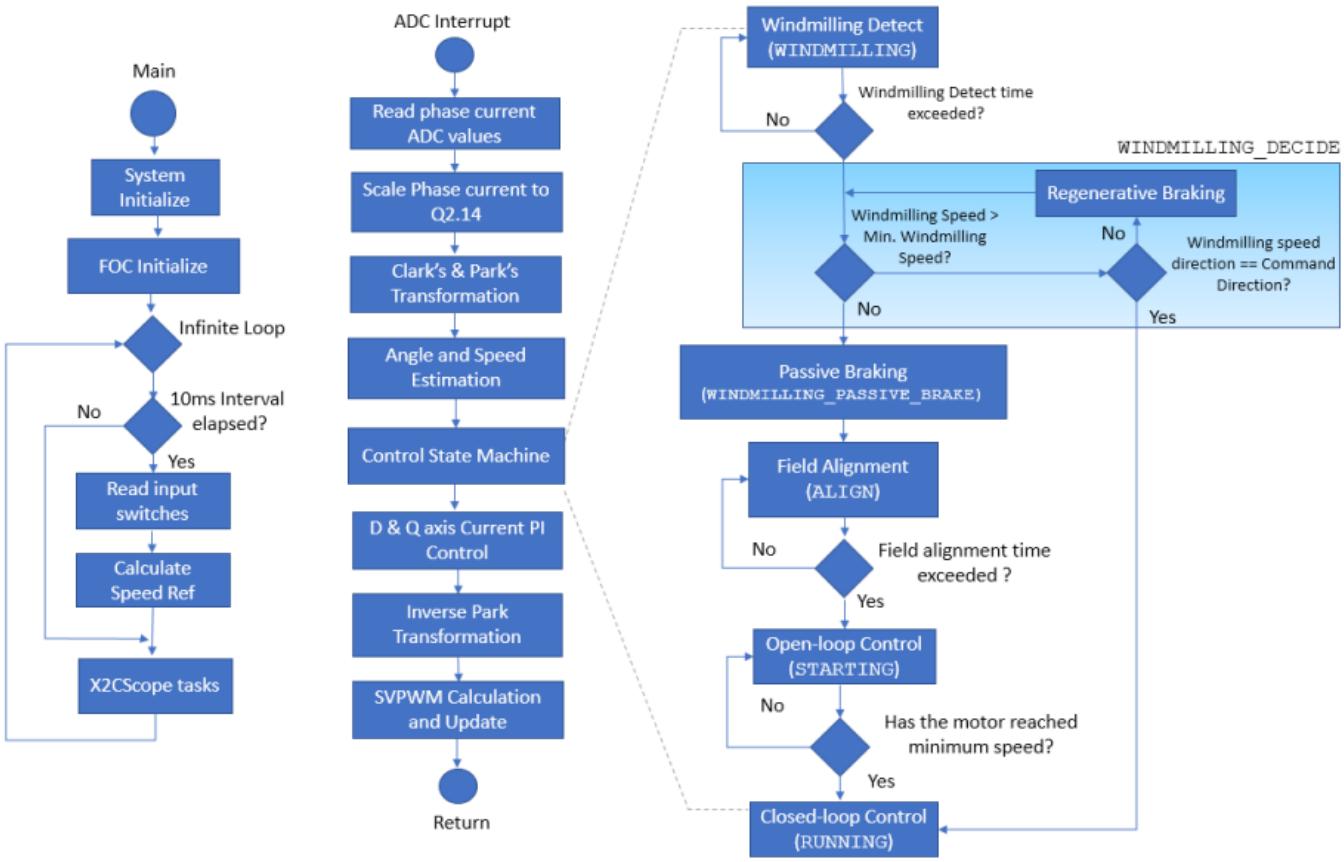


Hardware Blocks

1. Permanent Magnet Synchronous Motor.
2. 3-Phase Bridge – rectifier, inverter and acquisition and protection circuitry. software (run by the device).
3. Clarke forward transform.
4. Park forward and inverse transform.
5. Angle and speed estimator.
6. Proportional integral controller.
7. Flux weakening.
8. Space vector modulation.

Software Design:

The following figure shows the flow chart of the implemented algorithm:



Windmilling Detect - `WINDMILLING`:

In this state, the algorithm detects if the motor shaft is "Windmilling" or freewheeling due to stored momentum or external force and estimates the rotor angle, speed and direction of the windmilling motor. The length of duration of this state is defined by the macro `WINDMILLING_TIME_SEC`.

Windmilling Decide - `WINDMILLING_DECIDE`:

In this state, the algorithm makes determination of the further action based on information obtained from Windmilling Detect state.

- If the detected windmilling speed is greater than the "minimum windmilling speed" and if the direction of the windmilling motor is same as the "command" direction, the algorithm switches to "Closed Loop Control" state.
- If the detected windmilling speed is greater than the "minimum windmilling speed" but the direction of the windmilling motor is opposite to that of the "command" direction, the algorithm brakes the motor using active or regenerative braking. Once the speed of the windmilling motor drops below the minimum windmilling speed, the algorithm applies a passive brake by short circuiting all three phase windings to bring the motor to a halt. Once the motor is halted, the algorithm switches to "Field Alignment" state.
- If the detected windmilling speed is less than the "minimum windmilling speed", the algorithm applies a passive brake by short circuiting all three phase windings to bring the motor to a halt. Once the motor is halted, the algorithm switches to "Field Alignment" state.
- "Minimum Windmilling Speed" in hertz (Hz) is defined by the macro `MIN_WM_FRE_HZ.`
- "Maximum Regenerative Braking Current" in amperes (A) is defined by the macro `RGN_BRK_CUR_AMP`.
- Duration of the "Passive Braking" in seconds (s) is defined by the macro `PASSIVE_BRAKING_TIME_SEC.`

Field Alignment - `ALIGN`:

In this mode, a pre-defined value of the current is asserted across the Q axis of the rotor for a pre-defined length of time, in order to align the rotor to a known angle of 90 electrical degrees. The magnitude of the current and the length of the time for which it is applied depends upon the electrical and mechanical time constant of the PMSM motor drive. Electrical time constant of the motor is a function of R and L values of the motor windings, whereas the mechanical time constant of the motor drive is primarily a function of the static load on the motor shaft.

Open Loop Control - `STARTING`:

In this mode, the speed of the PMSM motor is gradually ramped up using an open loop control. During this mode, the rotor angle

is derived from the asserted open loop speed reference. This derived rotor angle would be lagging from the actual rotor angle. The speed is ramped up linearly to a minimum value required for the ROLO-based estimator to estimate the alpha and beta axis back EMF of the PMSM motor with sufficient accuracy. Rotor angle information is extracted from arctan($BEMF_{beta}$/ $BEMF_{alpha}$).

Close Loop Control - `RUNNING`:

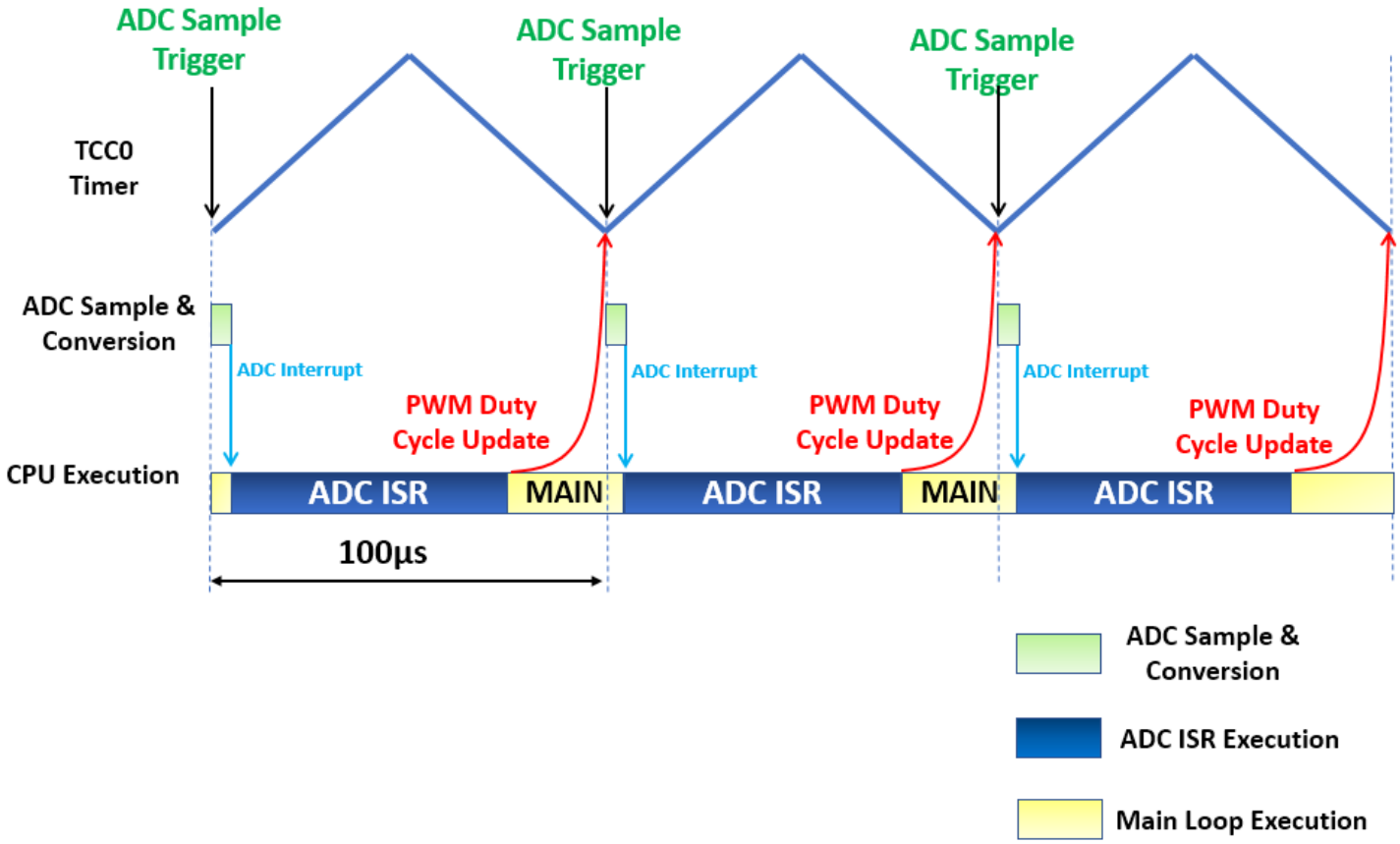
In this mode, the estimated rotor angle obtained from ROLO-based estimator is used to perform field oriented control of the PMSM motor.

Timing Diagram:

The three critical blocks of the MCU which are involved motor control operations are:

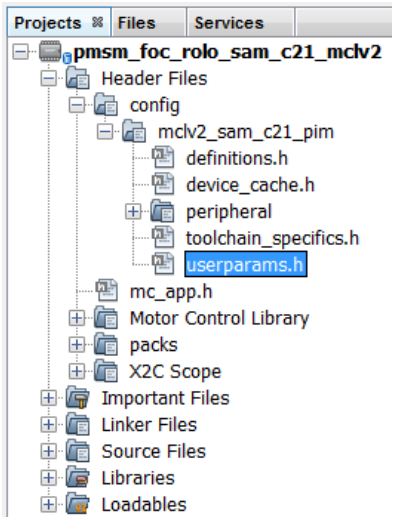
- ADC - It measures motor phase currents at the start of the PWM cycle.
- CPU - Based on motor phase currents measurements, CPU executes the control algorithm to calculate the PWM duty cycle needed to apply required voltage across motor windings.
- TCC0 - It Generates the PWM signals needed to apply required voltage across motor windings.

The following timing diagram, chronological role played by the ADC, CPU and TCC0 in the execution of FOC algorithm.



Software Configuration:

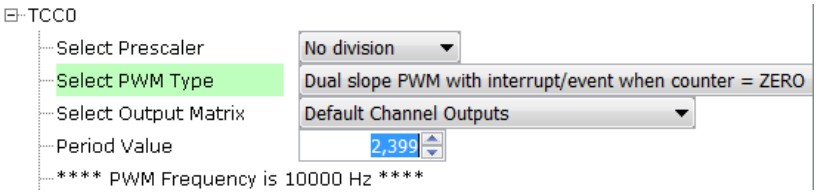
The FOC algorithm is used across different application fields. In order to get the optimal control of the PMSM motor, the motor specific parameters, board specific parameters and application parameter macros have to be updated in the software.. The following section describes how to update both motor and application specific parameters in the the project. These parameter macros can be found in userparams.h header file which can be found under logical path : *Header Files > config > <config_name> >userparams.h* in MPLABX IDE.



1. Setting motor control PWM frequency and dead-time:

PWM frequency is set by configuring the timer period of the TCC0 in terms of TCC0 clock counts. This frequency needs to be communicated to other sections in the algorithm by defining the "Period Value" (set in MPLAB Harmony Configurator) + 1 as a macro in `userparams.h`.

For example, in order to achieve a PWM frequency of 10KHz in center aligned mode with peripheral clock frequency of 48MHz, the period value defined in MHC is 2399. Therefore, the macro defined in `userparams.h` should be "period value" + 1 = 2400.



Similarly, dead-time is configured in terms of TCC0 clock counts in the MHC. The same value needs to be defined as macro in `userparams.h`.

Macro	Description
PWM_HPER_TICKS	PWM Timer period value = "Period value" defined in MHC + 1
DEAD_TICKS	Dead-time defined in terms of PWM timer ticks = "Dead time value" defined in MHC

2. Setting motor specific parameters:

Set the following motor specific parameters in the `userparams.h` file.

Macro	Description	Units
MAX_FRE_HZ	Electrical Frequency at maximum motor speed	Hz
MIN_FRE_HZ	Electrical Frequency at minimum motor speed	Hz
POLAR_COUPLES	Number of Pole Pairs	-
R_STA	Motor per phase resistance	Ohm
L_SYN	Motor per phase inductance	H
MAX_CUR_AMP	Maximum motor current	A
START_CUR_AMP	Current reference during field alignment and open loop startup	A
ACC_RPM_S	Acceleration ramp rate	RPM/s
DEC_RPM_S	Deceleration ramp rate	RPM/s
STUP_ACCTIME_S	Open loop start up time	seconds
CUR_RISE_T	Current rise time during field alignment	seconds

3. Setting Windmilling specific parameters

Macro	Description	Units
MIN_WM_FRE_HZ	Electrical frequency at minimum windmilling speed. Below this threshold, brake is applied to bring a windmilling motor to a complete halt.	Hz
WINDMILL_CUR_AMP	Initial Q axis current reference when the algorithm switches from Windmilling state to Closed Loop Control state.	A
RGN_BRK_CUR_AMP	Negative Q axis current reference applied during regenerative braking.	A
WINDMILLING_TIME_SEC	Time Duration of Windmilling state.	seconds
PASSIVE_BRAKING_TIME_SEC	Time Duration of Passive Braking state.	seconds

4. Setting board specific parameters:

Set the following board specific parameters in the `userparams.h` file.

Macro	Description	Units
CUR_SGN_REV	Define this macro, if the terminal of the shunt resistor connected to the return path of the inverter is connected to the inverting terminal of the signal conditioning differential operational amplifier. Do Not define this macro, if the terminal of the shunt resistor connected to the return path of the inverter is connected to the non-inverting terminal of the signal conditioning differential operational amplifier.	-
AD_RBA	Current measurement resolution of the board. This value is a function of shunt resistor, gain and bias of the differential opamp, reference voltage and the resolution of the ADC. For example, a board with a shunt resistance of 25 mOhm, differential opamp with a gain of 15 and bias of 1.65V, 12-bit ADC with reference voltage 3.3V, is capable of measuring +/- 4.4A i.e. a total of 8.8 A before ADC saturates. For such a board, AD_RBA = 4095/8.8 = 465.34 bits/A.	bits/A
AD_RBV	Bus voltage measurement resolution of the board. This value is a function of DC bus voltage divider ratio, reference voltage and resolution of the ADC. For example, a board with a DC bus voltage divider ratio of 1:16, 12-bit ADC with reference voltage 3.3V, is capable of measuring up to 52.8 V before ADC saturates. For such a board, AD_RBV = 4095/52.8 = 77.55 bits/V.	bits/V

5. Setting PI Controller parameters:

Depending on the type of motor used, and the corresponding application PI controller parameters should be updated in the `userparams.h` file.

Speed PI controller gains:

Macro	Description
KP_AS_R	Proportional gain of speed controller
KI_A_R	Integral gain of speed controller

Current PI controller gains for D and Q axis currents:

Macro	Description
KP_V_A	Proportional gain for D and Q axis current controller.
KI_V_AS	Integral gain for D and Q axis current controller.

6. Debug & Optimization Modes :

This example provides compile time re-configuration options using #define macro directives to enable or disable different debug and optimization modes. These macro directives can be accessed in the `userparams.h` header file.

- Current PI Tuning - `CURPI_TUN`
  - This mode allows tuning of the Current PI controllers. This mode generates a step current reference and depending upon the actual current response, user can accordingly adjust current PI gains.
  - This mode can be enabled by defining the macro `CURPI_TUN` in `userparams.h` header file.

  - In this mode, the step current reference is asserted across the D axis.
  - Macro `CUR_STEP_AMP` defines the step size of the asserted D axis current reference in amperes and macro `CUR_STEP_TIM` defines the length of the step in seconds.
- Open Loop Mode - `OPEN_LOOP_FUNCTIONING`
  - This mode is used to debug the open loop startup of the motor. This mode could also be used in tuning of Current PI controllers.
  - In this mode, the motor operates in open loop rotor angle mode. In other words, the rotor angle used by the algorithm to assert current will be different than actual rotor angle and therefore, the asserted current may not be orthogonal to the rotor magnetic axis, resulting in lower torque generation.
  - This mode can be enabled by defining the macro `OPEN_LOOP_FUNCTIONING` in `userparams.h` header file.
  - The torque current reference in this mode is defined by the macro `START_CUR_AMP`.
  - Depending upon the load inertia and startup acceleration time, defined by the macro `STUP_ACCTIME_S`, if the generated torque is not sufficient, then it may result in stalling of the motor.
  - `START_CUR_AMP` and `STUP_ACCTIME_S` need to be modified to ensure that the motor starts up successfully in open loop mode without stalling. If the motor stalls and the PWM signals are shutdown, the algorithm enters into a critical fault mode. The MCU needs to be reset to recover from the critical fault.
- Torque Mode - `TORQUE_MODE`
  - In this mode, as the name suggests, the algorithm operates in torque mode. The torque reference is obtained from the potentiometer.
  - This mode can be enabled by defining the macro `TORQUE_MODE` in `userparams.h` header file.
  - The potentiometer min - max input is scaled to generate torque current reference in the range of `TORQUE_MODE_MIN_CUR_AMP` - `MAX_CUR_AMP`.
  - Depending upon the load, if the torque is too low then it could cause the motor to stall and algorithm to enter a critical fault mode. The MCU needs to be reset to recover from the critical fault.
- Windmilling Enable/Disable - `WINDMILLING_ENABLE`
  - Undefining this macro, disables "Windmilling" capability of the algorithm.

- Active Vector / Null Vector Windmilling - `ACTIVE_VECTOR_WINDMILLING`
  - Defining this macro, configures the algorithm to operate in Active Vector Windmilling mode. This mode results in low braking torque but can have noisy angle tracking and may assert an unbalanced torque until ROLO based estimator converges and tracks the actual rotor angle.
  - Un-defining this macro, configures the algorithm to operate in Null Vector Windmilling mode. This mode results in high braking torque which increases with windmilling speed but has relatively less noisy angle tracking.
- Windmilling Calibration - `WINDMILLING_CALIBRATION`
  - Defining this macro, configures the algorithm to operating in windmilling mode perpetually. This mode is mainly used to debug angle and speed tracking in windmilling mode. In order to debug in this mode, the motor needs to operate as a generator i.e. driven by an external force which will acts as prime mover.

NOTE: Windmilling must be enabled, in order for `ACTIVE_VECTOR_WINDMILLING` and `WINDMILLING_CALIBRATION` macros to have any effect.

- Executing from RAM - `RAM_EXECUTE`
  - In order to speed up execution, the frequently called motor control related functions can be placed and fetched from RAM during execution.
  - Fetching instructions from RAM is faster than fetching instructions from Flash, as fetches from RAM can occur at CPU clock speed, unlike the Flash, which requires wait states to be asserted.
  - This speed optimization feature can be enabled by defining the macro `RAM_EXECUTE` in `userparams.h` header file.
  - Enabling this mode would result in a larger data memory footprint.
  - Exercise caution while using this feature as the functions in RAM are vulnerable to get overwritten due to stack overruns or unguarded buffer overruns.

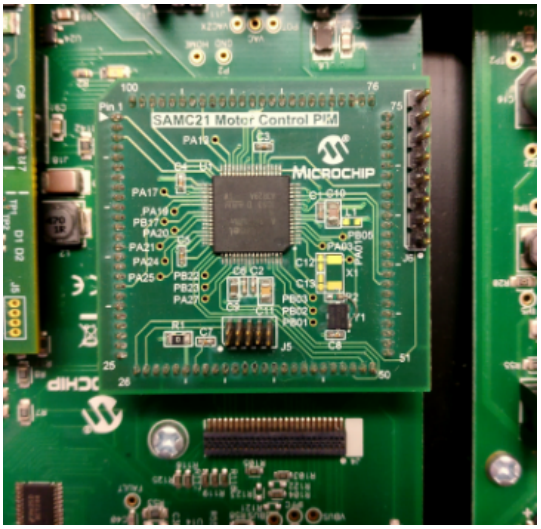
Hardware Setup

This section describes how to configure the supported hardware.

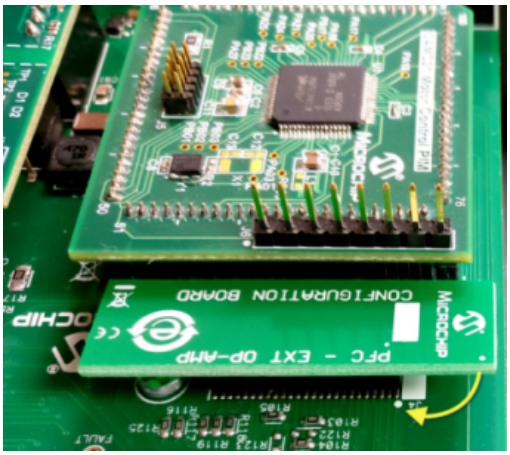
Description

1. **Project mchv3_sam_c21_pim.X.**

- **Hardware Used**
  - dsPICDEMTM MCHV3 Development Board (High Voltage)
  - ATSAMC21 Motor Control Plug In Module
  - Isolated Embedded Debugger Interface for MCHV
  - Leadshine EL5-M0400-1-24 Motor
- **Hardware Setup**
  1. Mount the ATSAMC21 Motor Control Plug In Module on U9 header.

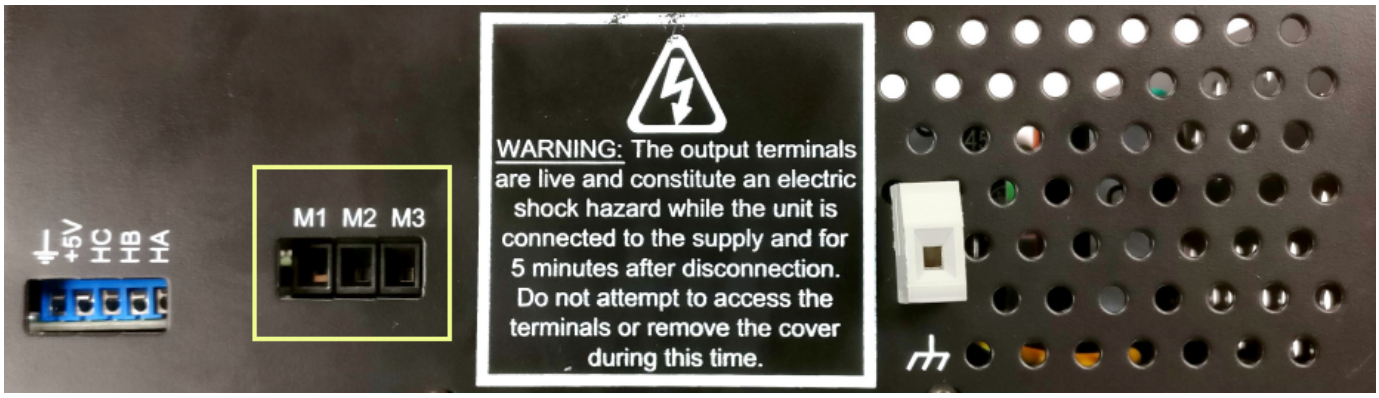


2. Place the "PFC - External Opamp Configuration" Matrix board at J4.



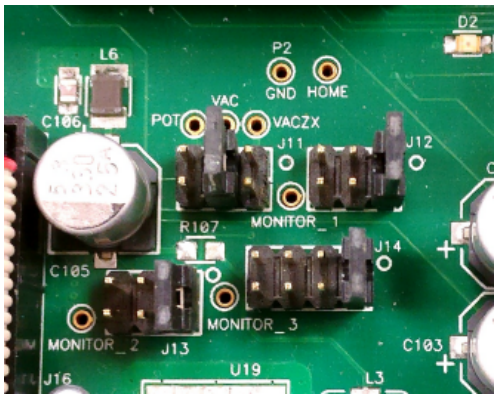
3. Motor Connections:

- Phase U - M1
- Phase V - M2
- Phase W - M3



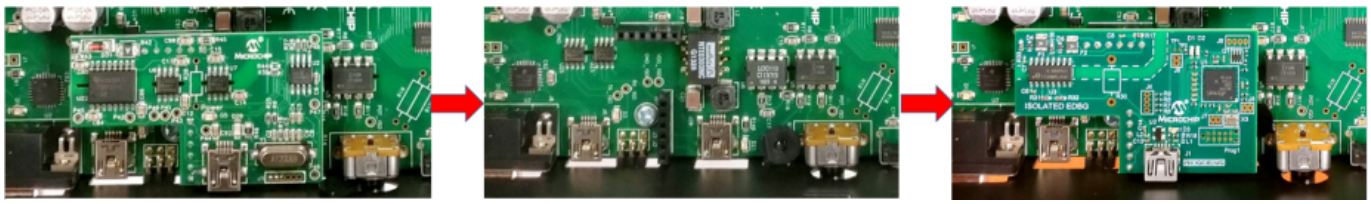
4. Jumper Settings:

- J11 - VAC (Short Pin 3 - 4)
- J12 - IA (Short Pin 1 - 2)
- J13 - IB (Short Pin 1 - 2)
- J14 - Fault_IP/IBUS (Short Pin 1 - 2)

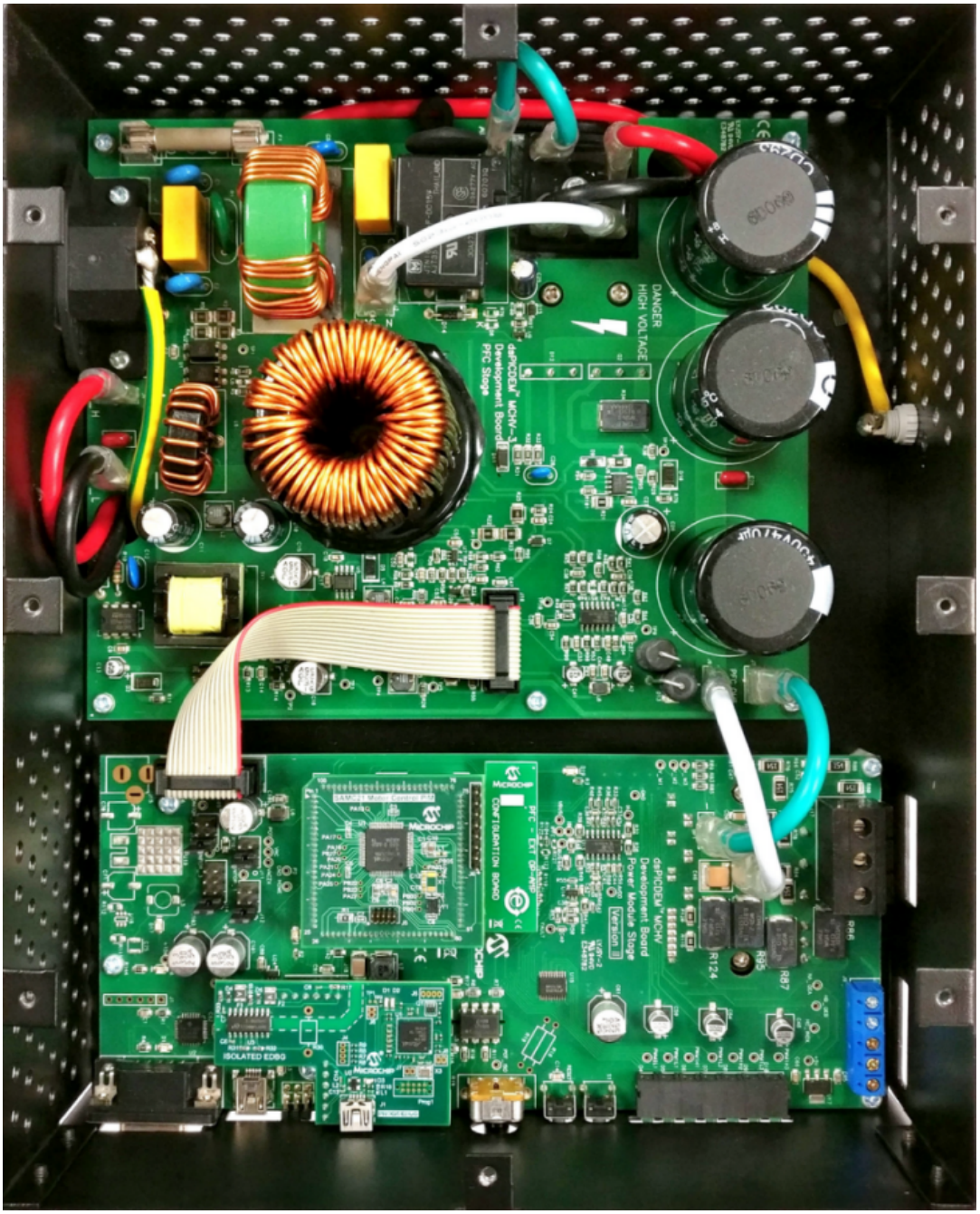


5. Power the board with (110V/220V) AC mains. For additional safety, it is recommended to use a current limited power supply while testing this demo on a non-default hardware and motor.

- Installing Isolated Embedded Debugger
  - Default programmer or debugger daughter card shipped with the MCHV3 board cannot program or debug SAM series MCU and therefore, it needs to be replaced with Isolated Embedded Debugger Interface for MCHV.

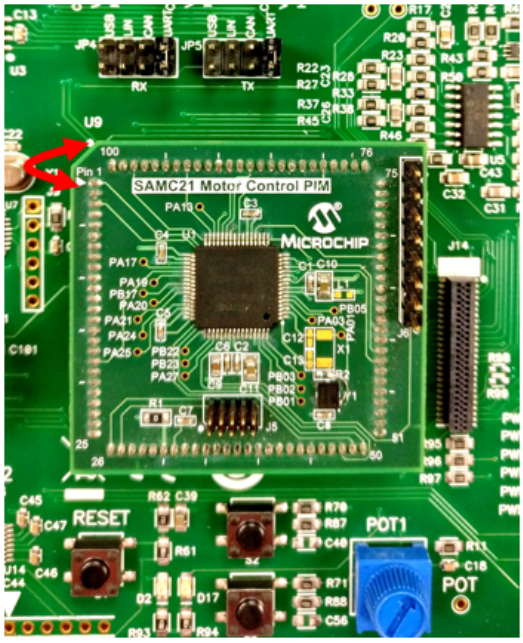


- Complete Setup

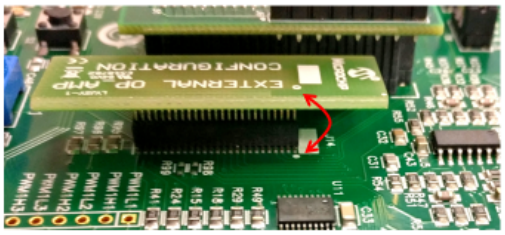


2. **Project mclv2_sam_c21_pim.X.**

- **Hardware Used**
  - dsPICDEMTM MCLV2 Development Board (Low Voltage)
  - ATSAMC21 Motor Control Plug In Module
  - Long Hurst DMA0204024B101 Motor
- **Hardware Setup**
  1. Mount the ATSAMC21 Motor Control Plug In Module on U9 header.

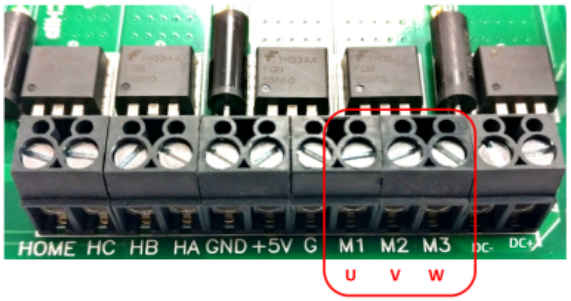


2. Place the "External Opamp Configuration" Matrix board at J14.



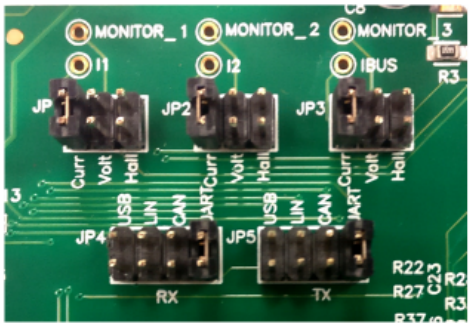
3. Motor Connections:

- Phase U - M1
- Phase V - M2
- Phase W - M3

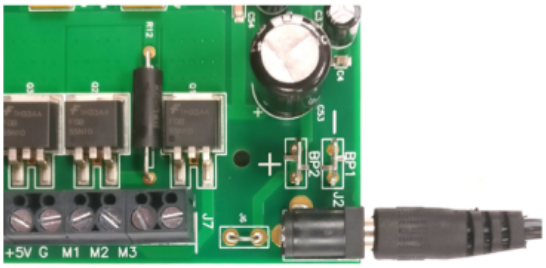


4. Jumper Settings:

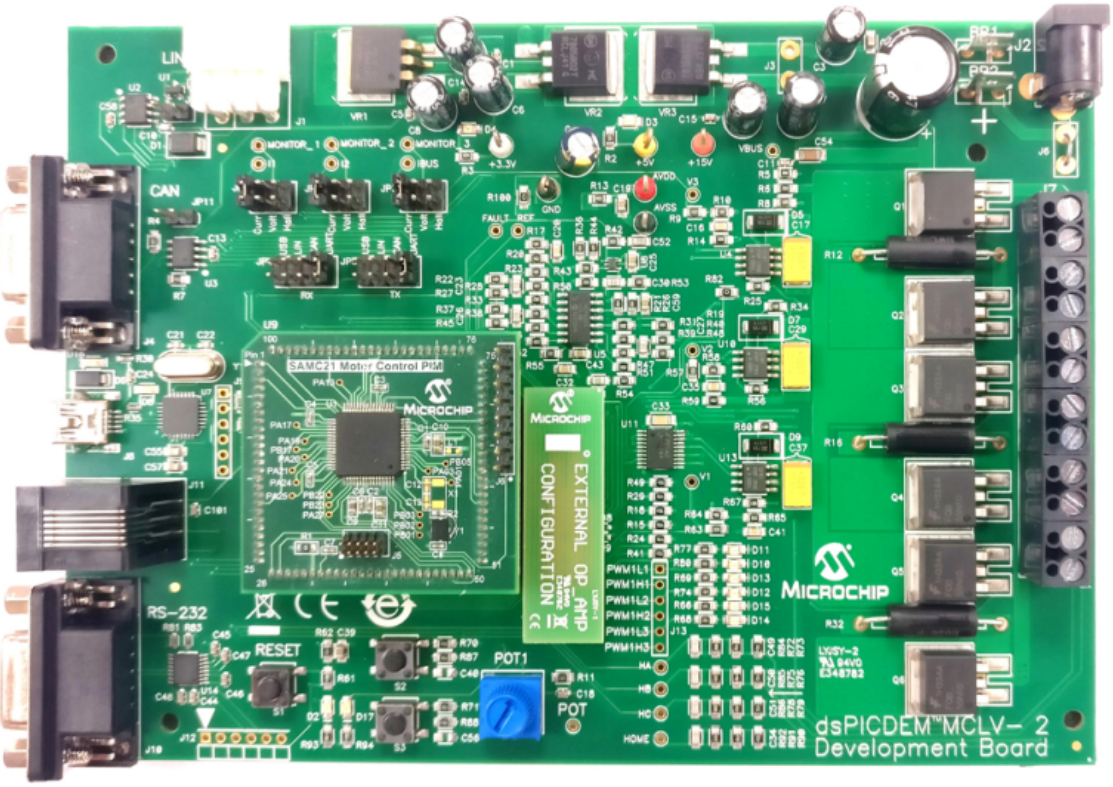
- JP1 - Curr, JP2 - Curr, JP3 - Curr
- In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



5. Power the board with a 24V DC supply using J2 or BP1-BP2. For additional safety, it is recommended to use a current limited power supply while testing this software demonstration on a non-default hardware and motor.



- **Complete Setup**



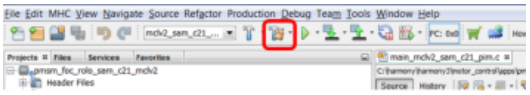
Running The Application

This section provides information on how to run the application using MPLAB X IDE.

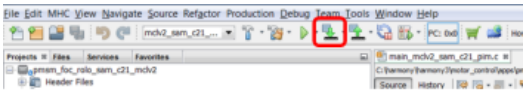
Description

Running the Application : pmsm_foc_rolo_wm_sam_c21 - MCHV3

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_rolo_sam_c21\firmware\mchv3_sam_c21_pim.X
2. Clean and Build Project:



3. Program the project:



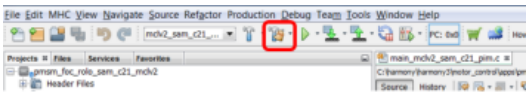
4. Press the switch "Push Button" to start or stop the motor.
5. Vary the Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_rolo_wm_sam_c21 - MCHV3

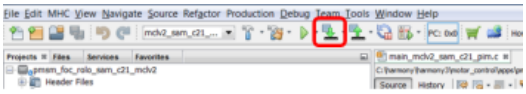
If LED D19 is ON, it indicates a non-recoverable fault caused by either over-current, phase loss or due to erroneous rotor position estimation. The MCU needs to be reset in order to recover from this fault mode.

Running the Application : pmsm_foc_rolo_wm_sam_c21 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_rolo_sam_c21\firmware\mclv2_sam_c21_pim.X
2. Clean and Build Project:



3. Program the project:



4. Press the switch S2 to start/stop the motor.
5. Press the switch S3 to toggle motor direction.

LED D2 Status	Description
OFF	Motor spin direction is "positive"
ON	Motor spin direction is "negative"

NOTE: Direction toggle command is accepted only when the motor is stationary.

6. Vary the Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_rolo_wm_sam_c21 - MCLV2

If LED D17 is ON, it indicates a non-recoverable fault caused by either over-current, phase loss or due to erroneous rotor position estimation. The MCU needs to be reset in order to recover from this fault mode.

Project Name	Output
mchv3_sam_c21_pim.X	1. Press the push button to start the motor. 2. Vary Potentiometer to change the speed of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the push button to stop the motor.

mclv2_sam_c21_pim.X	1. Press the Switch S2 to start the motor. 2. Vary Potentiometer to change the speed of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the Switch S2 to stop the motor.

Index

A

Applications Help 4

B

Building The Application 5, 14, 21, 35

C

Control Algorithm 6, 15, 23, 37

H

Hardware Setup 11, 20, 29, 43

M

Motor Control Overview 2

MPLAB Harmony Configurations 5, 14, 21, 35

P

PMSM FOC Using PLL Estimator 4

PMSM FOC Using Quadrature Encoder 14

PMSM FOC Using Reduced Order Luenberger Observer 21

PMSM FOC Using Reduced Order Luenberger Observer with Windmilling Capability 35

pmsm_foc_encoder_sam_e70 14

pmsm_foc_pll_estimator_sam_e70 4

pmsm_foc_rolo_sam_c21 21

pmsm_foc_rolo_wm_sam_c21 35

R

Running The Application 13, 20, 33, 48

X

X2CScope Plugin 2