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

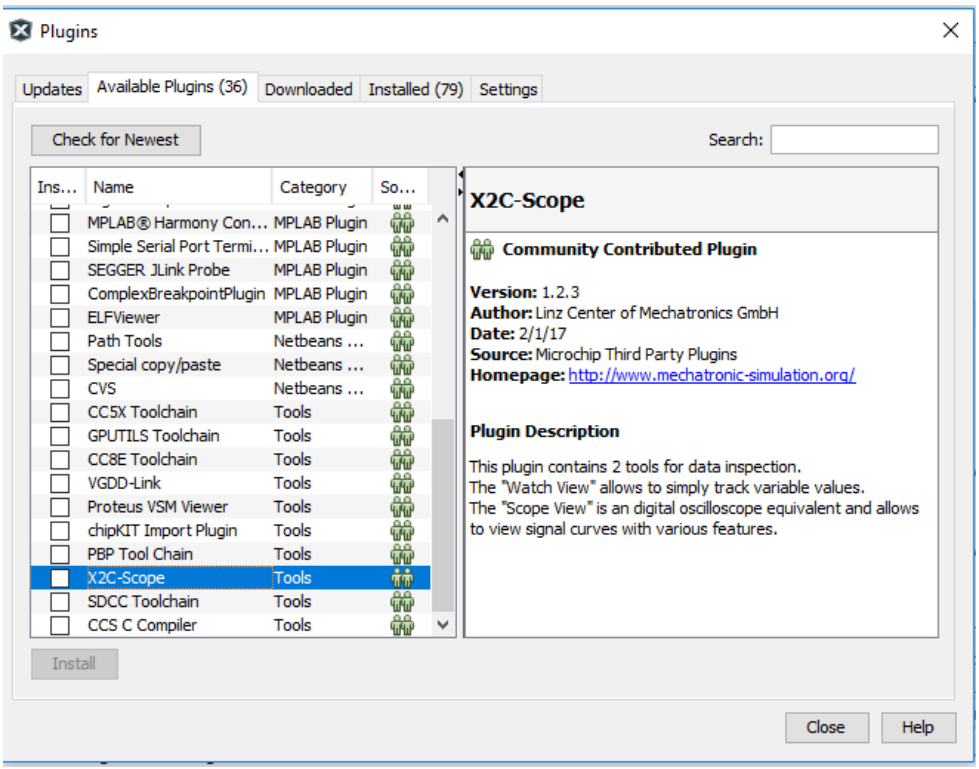
This topic describse how to use the X2C-Scope MPLAB X IDE plug-in.

Description

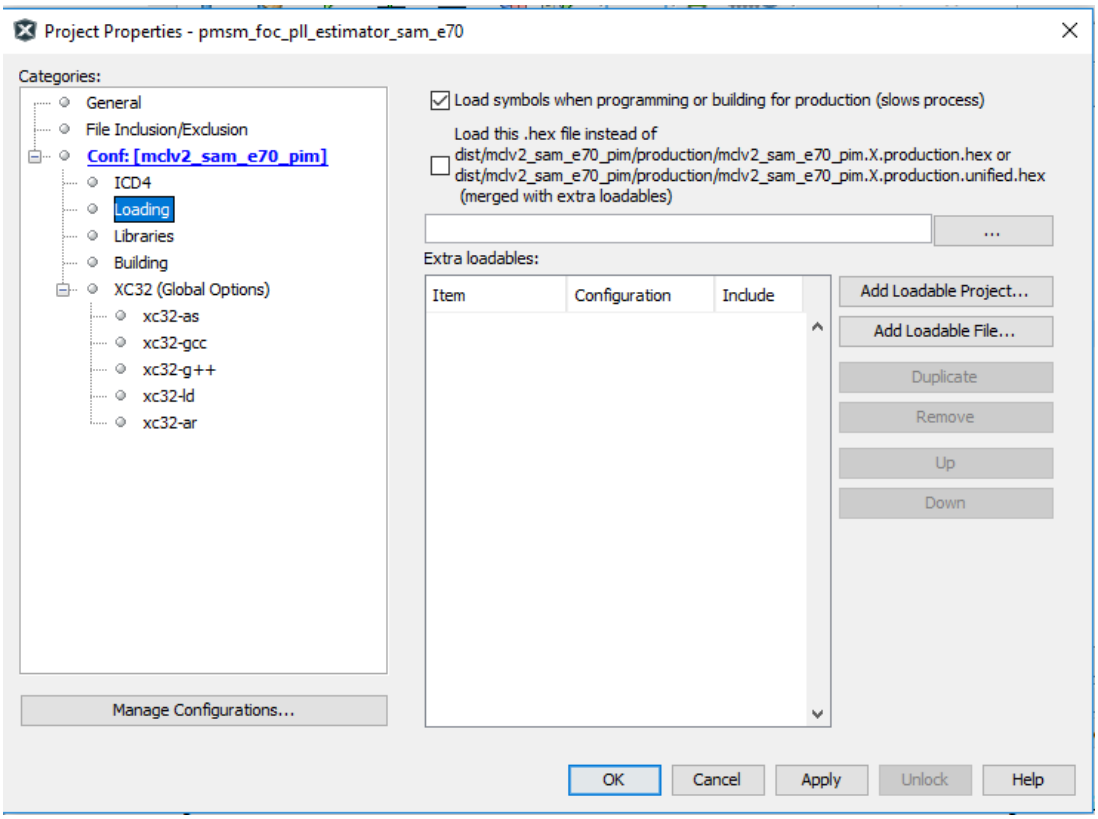
The MPLAB X IDE enables use of the X2C-Scope plug-in to read, write, and plot global variables in real time. The X2C-Scope communicates with the target using the UART. The baud rate of the UART communication can be modified using the MHC.

Installation

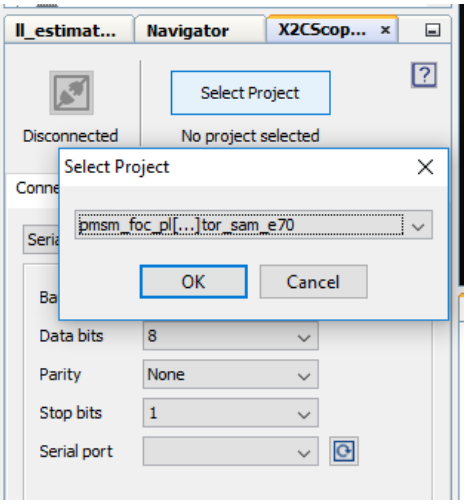
1. If not already installed, in MPLAB X IDE, select Tools > Plugins > Available Plugins > Select X2C-Scope> Install.
2. Restart MPLAB X IDE to complete the plug-in installation.



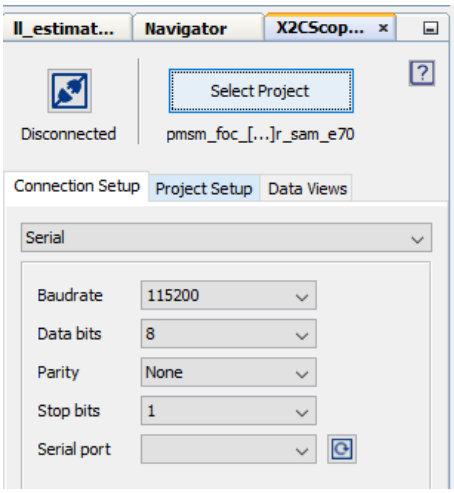
3. Open the X2CScope by selecting Tools > Embedded > X2CScope.
4. Ensure the symbols are loaded during the project build by selecting Project Properties > Selected Configuration > Loading, and ensure that the check box for Load Symbols when programming or building for production (slows process) is selected.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



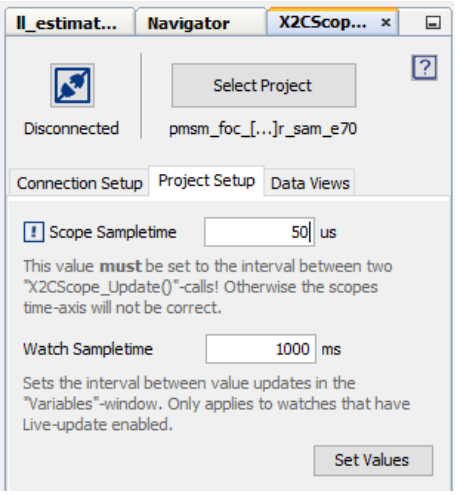
5. In the **X2CScope Configuration** tab, select the project.



6. In the **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.

ACIM VHz Open loop

This Section provides help for the ACIM V/Hz Control application.

acim_vhz_sam_c21

This motor control application demonstrates V/Hz open loop algorithm for AC Induction Motor (ACIM) using SAMC21 micro-controller.

Description

This application implements the V/Hz Open loop speed control of an AC Induction motor (ACIM). This application generates the three phase sinusoidal voltages based on the speed command using the V/Hz profile. The 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 the following table and open the appropriate project file in MPLAB X IDE.

Project Name	Description
mchv3_sam_c21_pim.X	acim_vhz_sam_c21 is setup to work with ATSAMC21 Motor Control Plug In Module + dsPICDEM™ MCHV-3 Development Board (High Voltage)

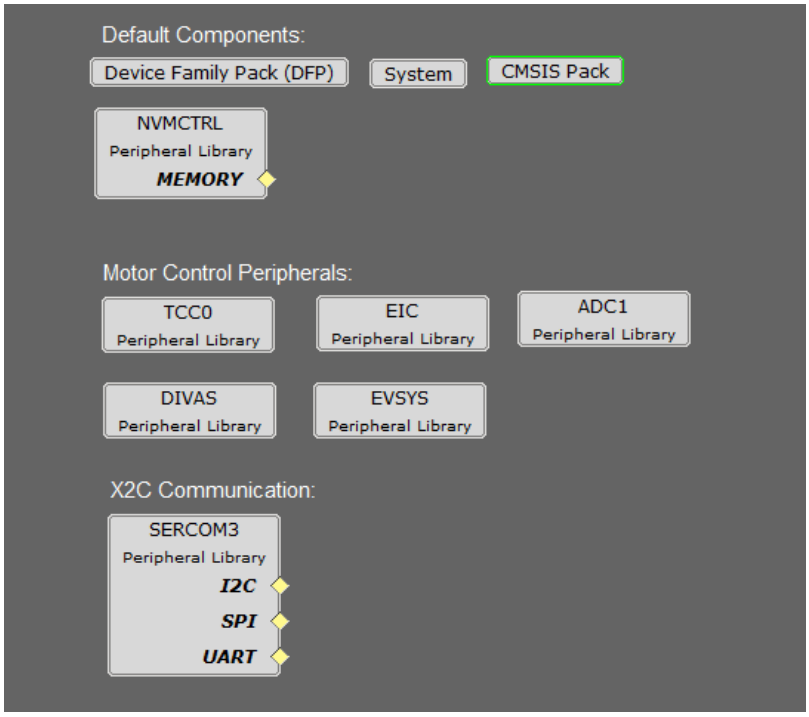
MPLAB Harmony Configurations

This topic provides information on the MHC configurations.

Description

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

acim_vhz_sam_c21



Components Used

TCC0:

- Configured to generate 3 pairs of complimentary PWM signals at a frequency of 20 kHz in "Dual Slope PWM with interrupt and event when the counter = ZERO".
- 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.

ADC1:

- ADC1 operates to read the Potentiometer Speed Reference.
- The ADC is hardware triggered by an event generated from the TCC0.
- Enable Ready interrupt is generated by ADC1.

EIC:

- EIC Channel 2: External Interrupt Controller detects the hardware over current fault input and generates a non-recoverable fault event for the TCC0, thereby shutting down the PWM in the event of an over-current fault.
- EIC Channel 11: External Interrupt Controller detects the Start and Stop push button command.

EVSYS:

- Event System acts as a moderator between the event generator and event users.
- Event generated by the TCC0 when the counter reaches ZERO, is used by the ADC1 as a hardware trigger source via the Event System.
- Event generated by the EIC upon over-current fault, is used by the TCC0 as a non-recoverable fault event via the Event System.

DIVAS:

- This project 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, the square root operation would still require a special API. Refer to the DIVAS help for more details.

SERCOM3:

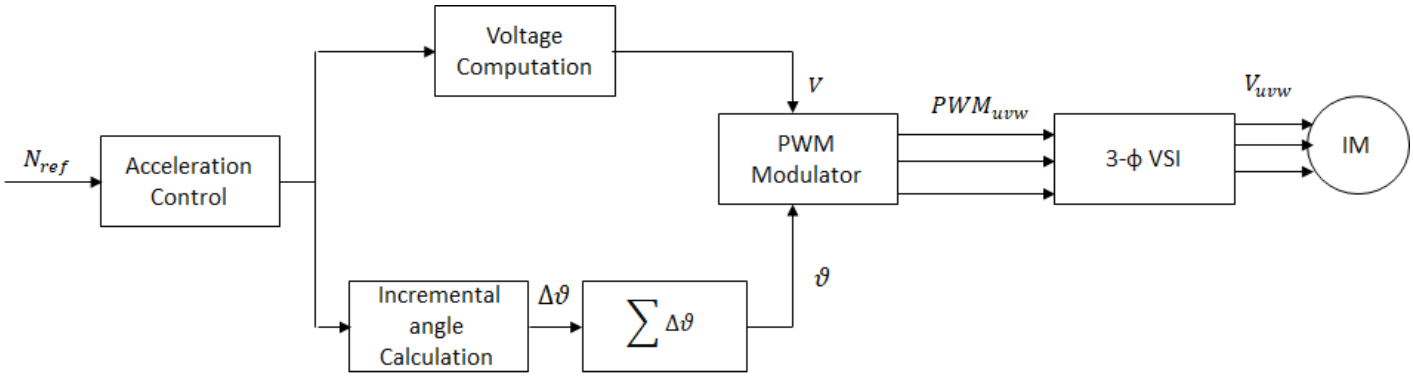
- Configured in USART mode and is set to operate at 115200 bps.
- This USART channel is used by the X2CScope plugin to plot or watch global variables in run-time.

Control Algorithm

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

Description

The project implements a V/Hz control algorithm on SAMC21 32-bit micro-controller to regulate the speed of the high voltage ACIM motor drive. The following section describes briefly the V/Hz control algorithm, software design, software configuration, and implementation details. The following block diagram shows a typical V/Hz control block diagram.



V/Hz control algorithm:

The V/Hz control is a scalar control motor control strategy where the applied ratio of applied AC voltage and frequency is kept constant. The V/Hz control is carried out in the following steps-

1. Read the commanded speed request.
2. Determination of angular displacement per speed loop cycle.
3. Determination of AC voltage magnitude based on the angular displacement per speed loop cycle.
4. Determination of PWM modulation values for voltage source inverter.
5. Application of calculated PWM values to three phases of the ACIM motor.

Determination of angular displacement per speed loop cycle:

The requested ACIM motor speed is mapped to the angular displacement by using the following relation:

$$\Delta\vartheta_{el} = \frac{360 * N_{ref}}{60} * \frac{Z}{2} * T_{SL}$$

$$or, \quad \Delta\vartheta_{el} = \frac{360 * N_{ref}}{60} * \frac{Z}{2} * \frac{1}{f_{SL}} \qquad (i)$$

$where,$

N_{ref} $is\ the\ requested\ speed\ in\ RPM$

T_{SL} $is\ the\ speed\ loop\ time\ period\ in\ S$

f_{SL} $is\ the\ speed\ loop\ frequency\ in\ hertz$

Z $is\ the\ number\ of\ poles$

For fixed point implementation, the input speed is scaled from (0 , N_{rated}) to (0 , 2^{14}) units.

$$N_{scaled} = \frac{2^{14} * N_{ref}}{N_{rated}} \qquad (ii)$$

$where,$

N_{rated} $is\ the\ rated\ speed\ in\ RPM$

From (i) and (ii), we get,

$$\Delta\vartheta_{el} = \frac{360 * N_{scaled} * {}^{N_{rated}}/_{2^{14}}}{60} * \frac{Z}{2} * \frac{1}{f_{SL}}$$

For fixed point implementation, the angle is scaled from (0 , 360 deg) to (0 , 2^{16}) units. Thus the final equation becomes:

$$\Delta\vartheta_{el_Scaled} = \frac{2^{16}}{360} * \frac{360 * N_{scaled} * {}^{N_{rated}}/_{2^{14}}}{60} * \frac{Z}{2} * \frac{1}{f_{SL}}$$

$$or, \quad \Delta\vartheta_{el_Scaled} = \frac{N_{scaled} * N_{rated} * Z}{30} * \frac{1}{f_{SL}}$$

$$or, \quad \Delta\vartheta_{el_Scaled} = \frac{N_{scaled} * N_{rated} * Z}{30000} * \frac{1}{f_{SL(kHz)}} \qquad (iii)$$

Determination of AC voltage magnitude for voltage source inverter:

The AC voltage magnitude is determined based on the requested RPM based on following relation:

$$V = VF_{constant} * N_{ref}$$

$$or, \quad V_{scaled} = VF_{constant_scaled} * N_{scaled} \qquad (iv)$$

$where,$

$VF_{constant}$ $is\ the\ ratio\ rated\ voltage\ magnitude\ to\ the\ rated\ RPM$

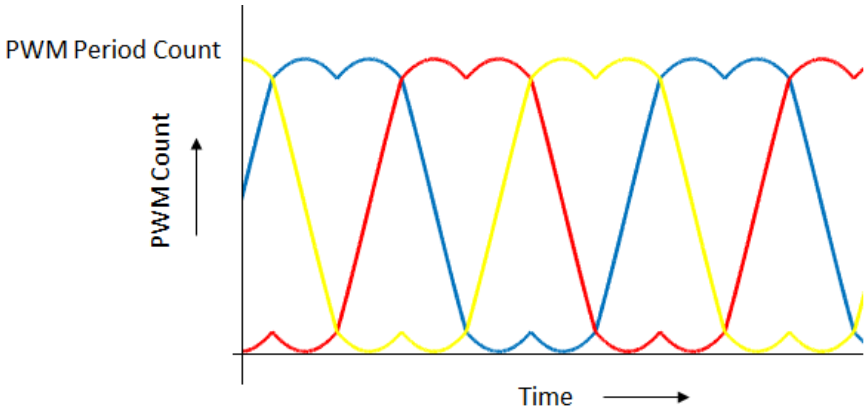
$V\ is\ the\ voltage\ in\ volts$

V_{scaled} $is\ the\ scaled\ voltage$

$VF_{constant_scaled}$ $is\ the\ ratio\ rated\ sacled\ voltage\ magnitude\ to\ the\ rated\ scaled\ RPM$

Determination of PWM modulation values for voltage source inverter:

The calculated voltage magnitude and phase is modulated using SVPWM approach to generate corresponding three phase gate control signals. This PWM modulation approach utilizes third order harmonic injection to increase the inverter DC bus utilization by about 15%. The following diagram shows the typical SVPWM waveform with third order harmonic injection -

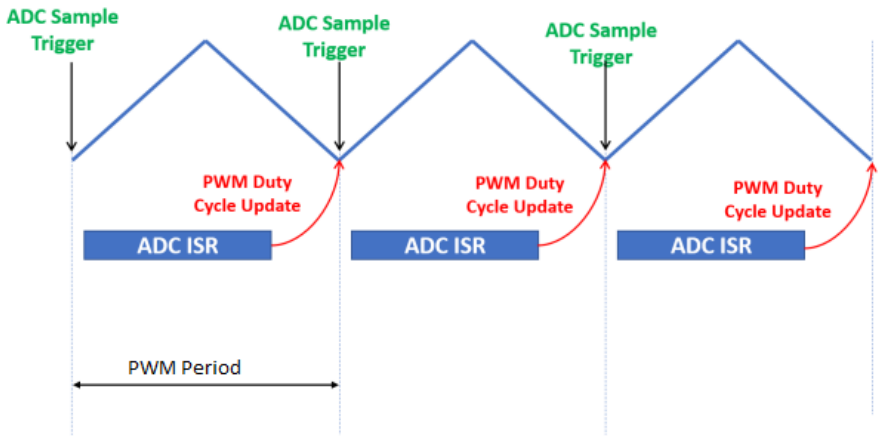


Software Design:

An interrupt based approach is used for the software implementation. There are three threads: main thread, ADC ISR thread, and a push button ISR thread.

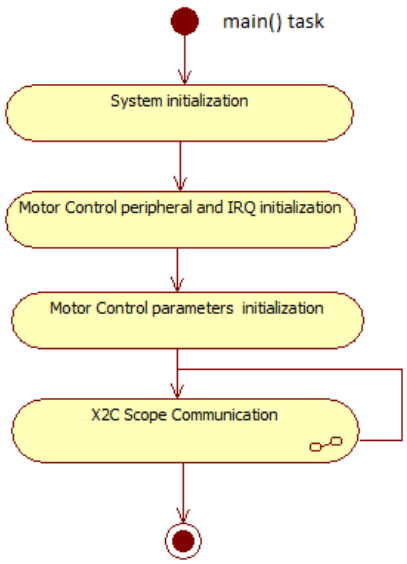
The main thread is used to carry out UART based data communication in polling mode.

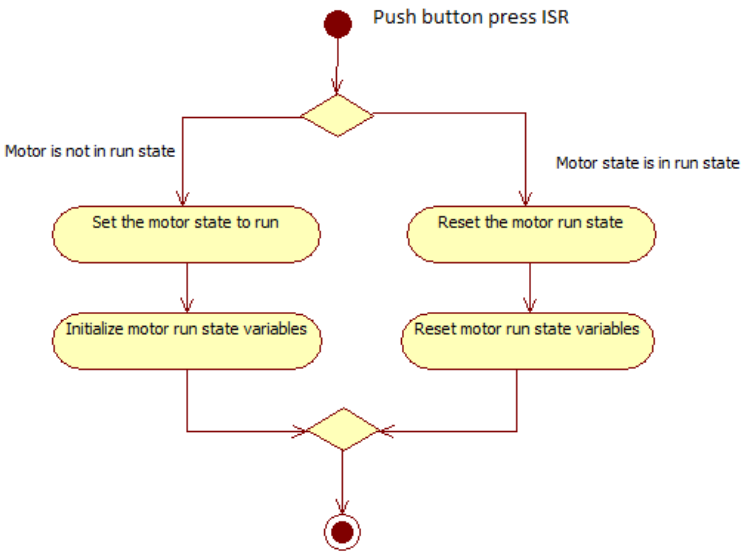
ADC ISR thread is used to carry out the actual V/Hz speed control algorithm. This thread is executed cyclically every PWM period as shown in the following figure:

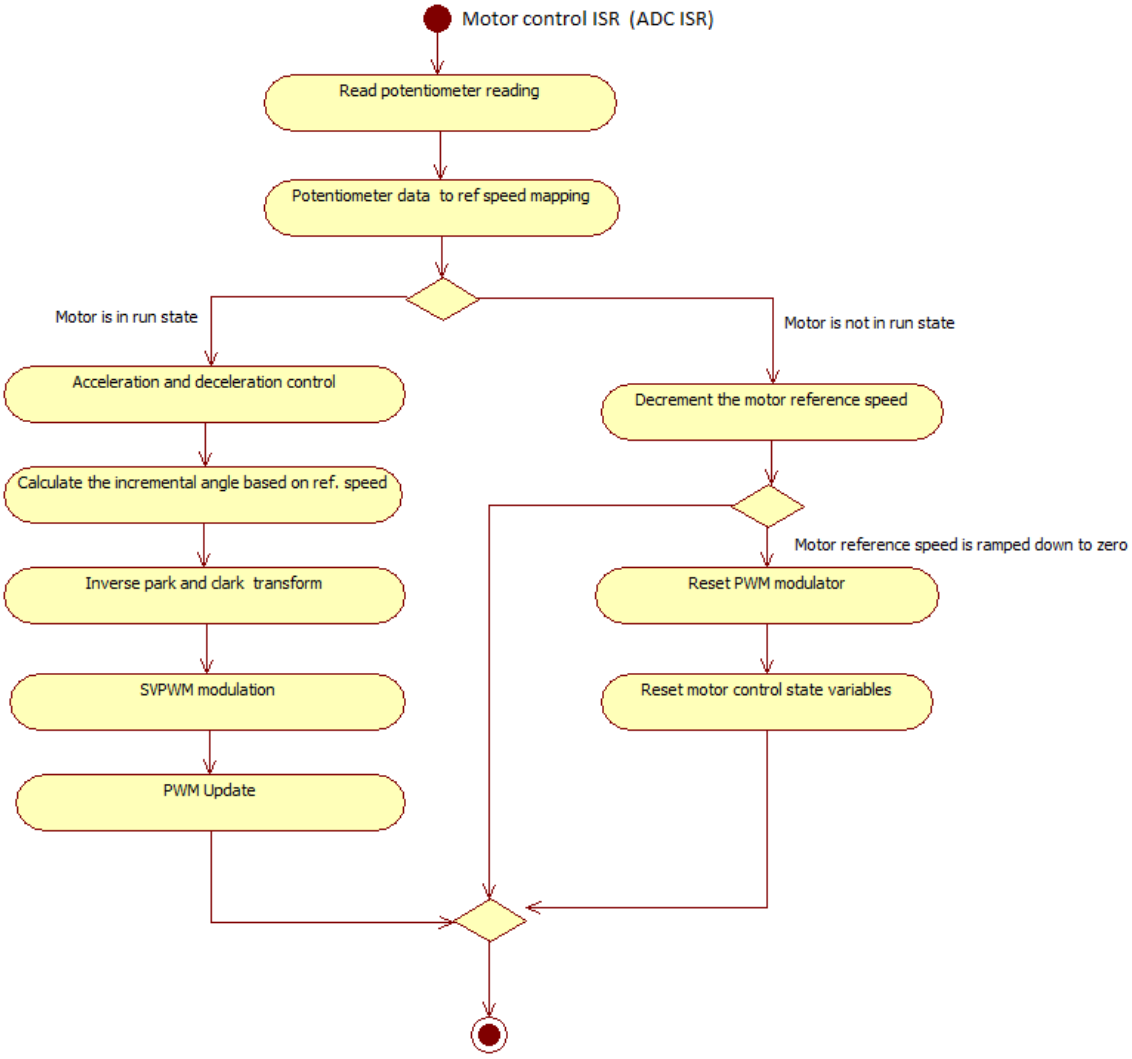


Push button ISR thread is used to carry out start and stop motor operations. This thread is non-cyclic and is executed once there is an external push button switch event.

The following figure shows the activity diagram of threads used in the software -

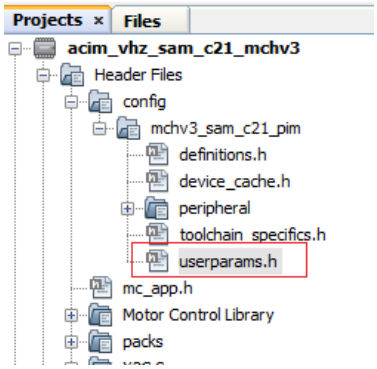

main() task
System initialization
Motor Control peripheral and IRQ initialization
Motor Control parameters initialization
X2C Scope Communication


Push button press ISR
Motor is not in run state
Motor state is in run state
Set the motor state to run
Reset the motor run state
Initialize motor run state variables
Reset motor run state variables


Motor control ISR (ADC ISR)
Read potentiometer reading
Potentiometer data to ref speed mapping
Motor is in run state
Motor is not in run state
Acceleration and deceleration control
Calculate the incremental angle based on ref. speed
Inverse park and clark transform
SVPWM modulation
PWM Update
Decrement the motor reference speed
Motor reference speed is ramped down to zero
Reset PWM modulator
Reset motor control state variables

Software Configuration:

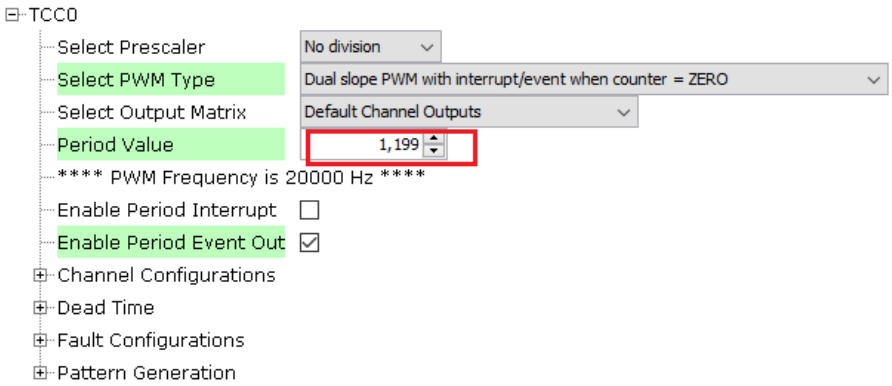
The V/Hz control algorithm can be used to control different ACIM motors. In order to get the correct control of the ACIM motor, the motor specific parameters and application parameters need to be updated in the software. The following section describes how to update both motor and application specific 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.*



1. Setting motor control PWM frequency:

PWM frequency is set by configuring the timer period of the TCC0 in terms of TCC0 clock counts. The period value +1 set using the MHC needs to be redefined as a macro in `userparams.h`.

For example, in order to achieve a PWM frequency of 20 KHz, the period value defined in the MHC is 1199. Therefore, the macro defined in `userparams.h` should be "period value" + 1 = 1200.



Macro	Description
PWM_HPER_TICKS	PWM Timer period value = "Period value" defined in MHC + 1

2. Setting motor specific parameters:

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

Macro	Description	Units
MOTOR_VOLTAGE	Motor Voltage [Volts]	Volts
MAX_MOTOR_SPEED	Maximum motor speed	RPM
NUMBER_OF_POLES	Number of poles	-
MAX_SPEED_SCALED	Max speed	Internal units
VF_CONSTANT	Volts per RPM constant	Internal units
VF_OFFSET	Voltage offset at starting zero frequency	Internal units
START_SPEED_DEFAULT	Default startup speed	RPM
ACC_RAMP	acceleration ramp count in internal unit	Internal units
DEC_RAMP	deceleration ramp count in internal unit	Internal units
SPEED_FILTER_COEFF	Speed filter coefficient range [1-16]	-

Software Implementation:

The SAMC21 microcontroller devices do not have a floating point unit. Therefore, all the physical real variables are mapped to internal units for fixed point data processing in the software. The following table shows the mapping of physical quantities to their corresponding internal units.

Physical Quantities	Physical Range	Conversion
Speed	0 to N_{rated}	Physical Value * 2^{14} / N_{rated}
Angle	0 to 360 deg	Physical Value * 2^{16} / 360
V/F constant	-	Physical Value * 12
Speed Filter coefficient	0-1	Physical Value * 16

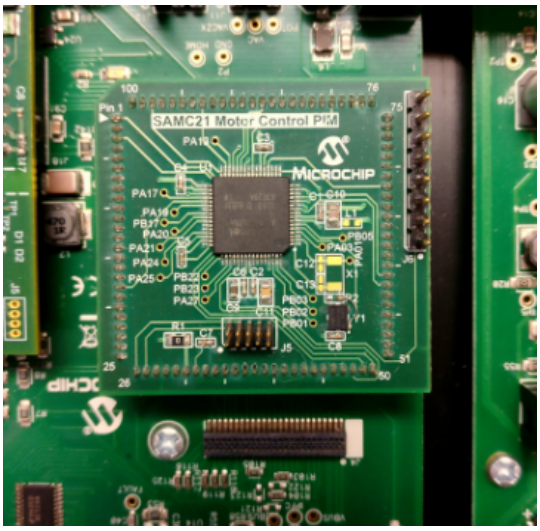
Hardware Setup

This section describes how to configure the supported hardware.

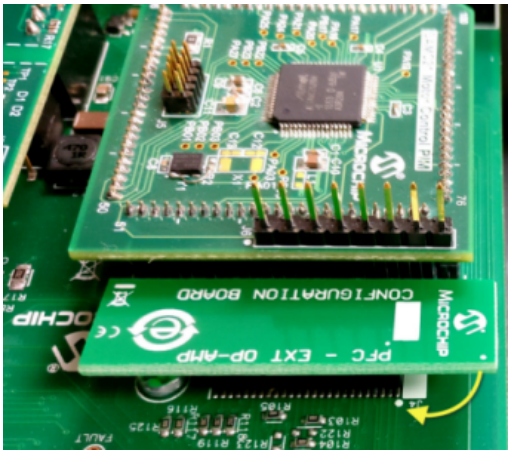
Description

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
     - 230 volts ACIM motor

- **Hardware Setup**
  1. Mount the ATSAMC21 Motor Control Plug In Module on U9 header.

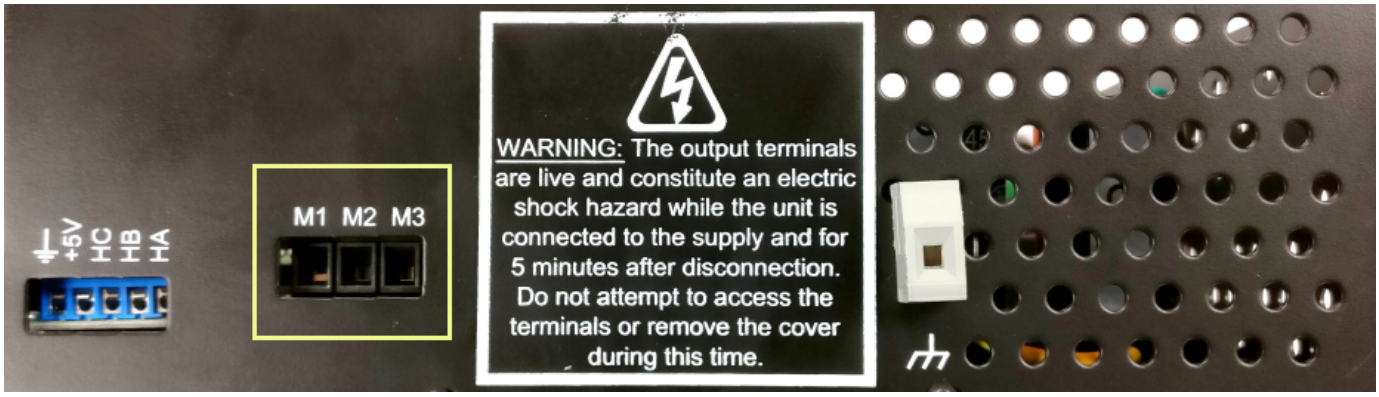


2. Place the "PFC - External Opamp Configuration" Matrix board at J4.



3. Motor Connections:

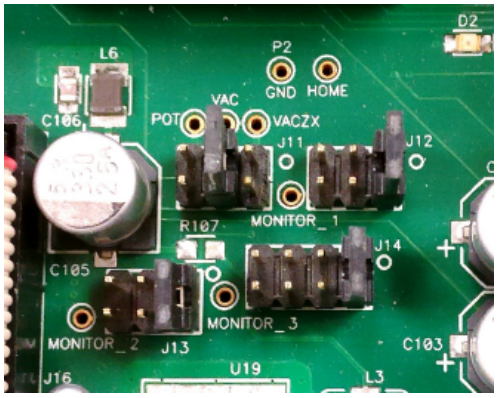
- Phase U - M1
- Phase V - M2
- Phase W - M3



4. Jumper Settings:

- J11 - VAC (Short Pin 3 - 4)
- J12 - IA (Short Pin 1 - 2)
- J13 - IB (Short Pin 1 - 2)

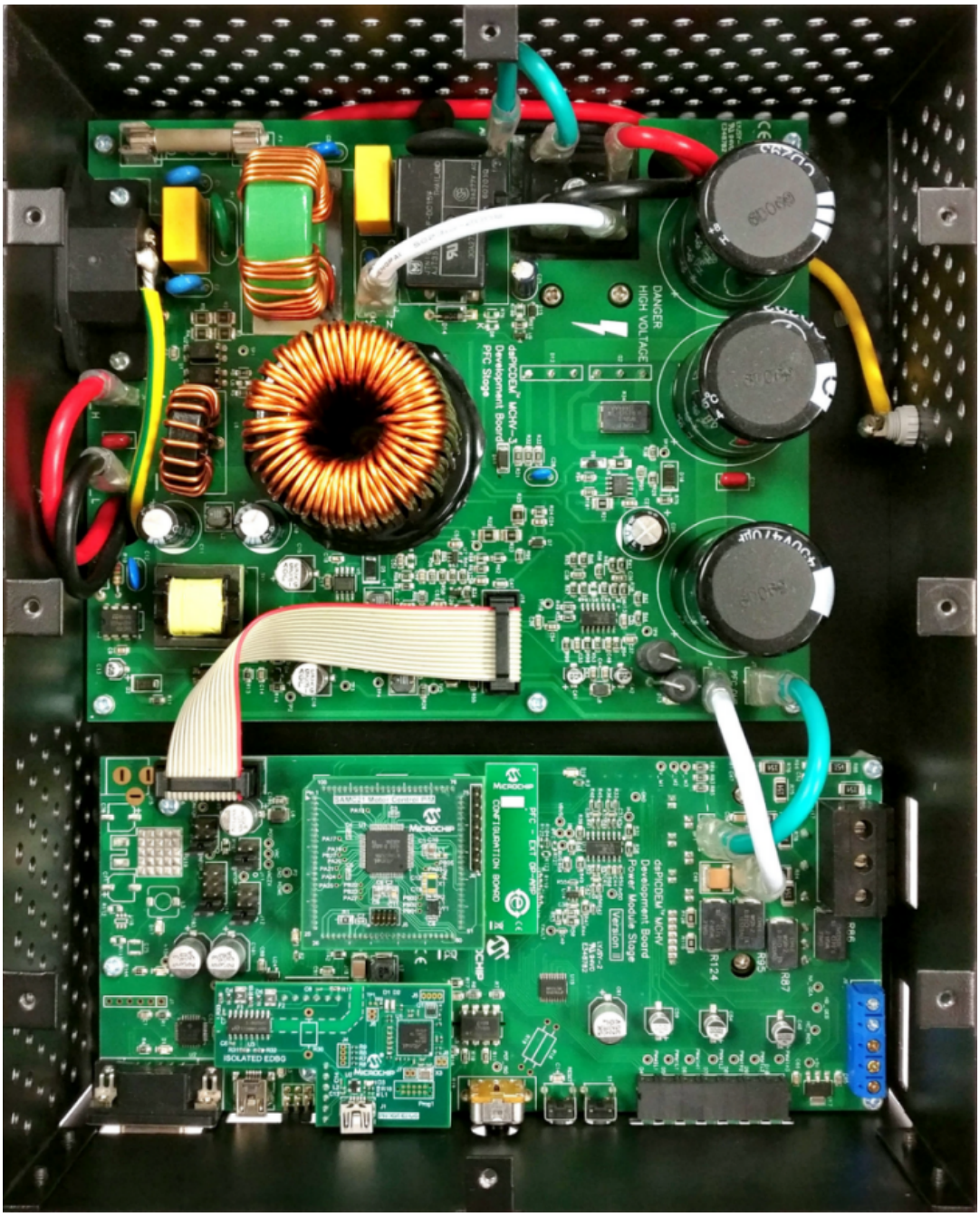
- J14 - Fault_IP/IBUS (Short Pin 1 - 2)



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



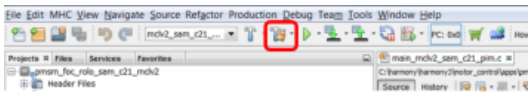
Running The Application

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

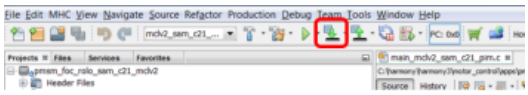
Description

Running the Application : acim_vhz_sam_c21_mchv3 - MCHV3

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\acim_vhz_sam_c21_mchv3\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.



4. Press the switch "Push Button" to start/stop the motor.
5. Vary Potentiometer knob from mid point to extreme left to change the speed of the motor in clock-wise direction.
6. Vary Potentiometer knob from mid point to extreme right to change the speed of the motor in clock-wise direction.

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.

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 the Permanent Magnet Synchronous Motor (PMSM) with PLL Estimator based Field Oriented Control (FOC) on a 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 to 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)

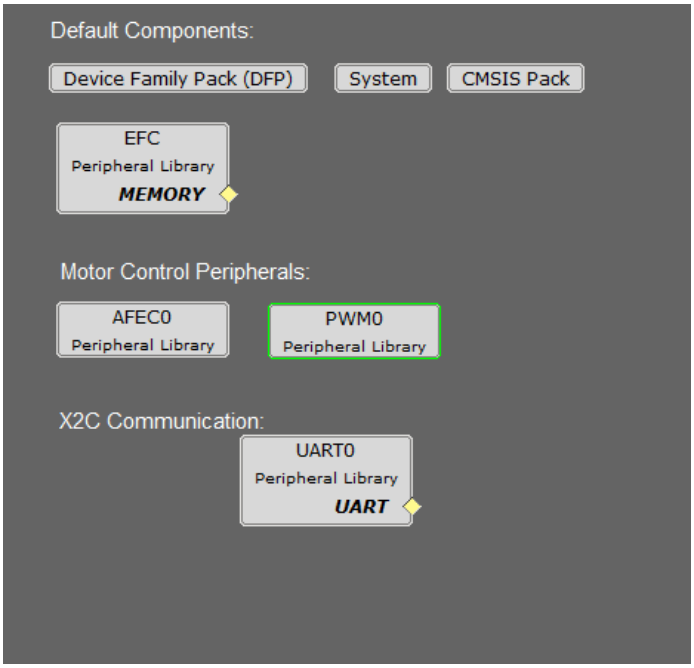
MPLAB Harmony Configurations

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 in 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 to Application Note AN2520. The following section describes briefly about the PLL based sensor-less FOC algorithm, software design and implementation.

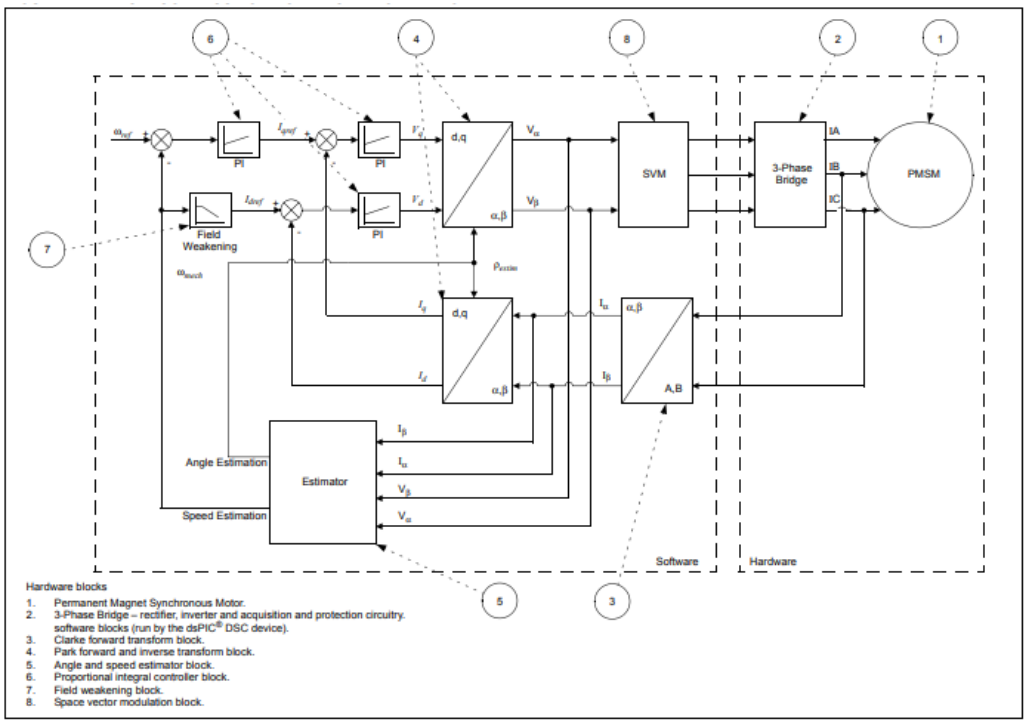
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from stationary reference frame to torque and flux producing currents components in rotating reference frame using mathematical 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 the Clarke transformation.
3. Calculate the rotor position angle.
4. Transform stator currents into the d,q-coordinate system using the 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 the Inverse Park transformation.
7. Using the space vector modulation, the three-phase output 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.

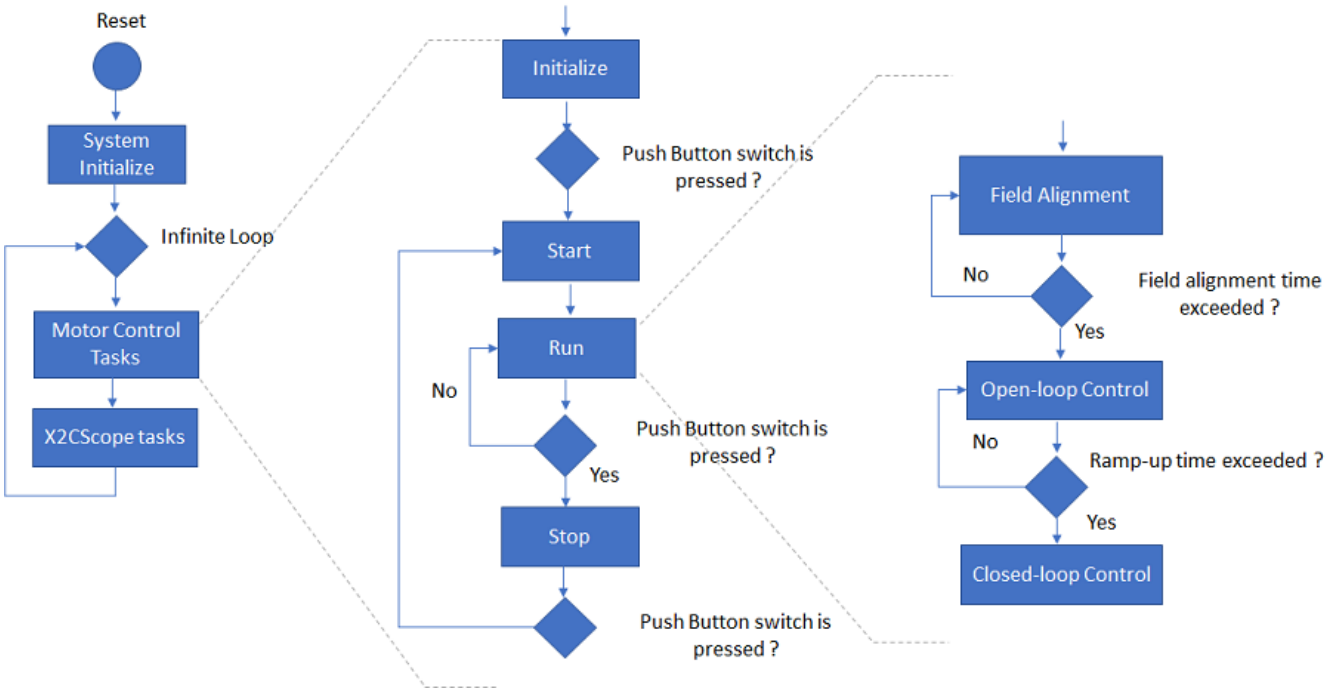
The electrical rotor position is calculated using the PLL estimator using **MCLIB_PLLEstimator** software block. The working principle of the PLL estimator can be found in Application Note AN2520. The main disadvantage of PLL estimator is its inability to estimate the rotor angle at lower rotor speeds because of very low value of back EMF. Therefore, 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 previous figure. The following sections briefly describes the various states in the PMSM speed control task:

1. Initialize:

In this state, following tasks are performed:

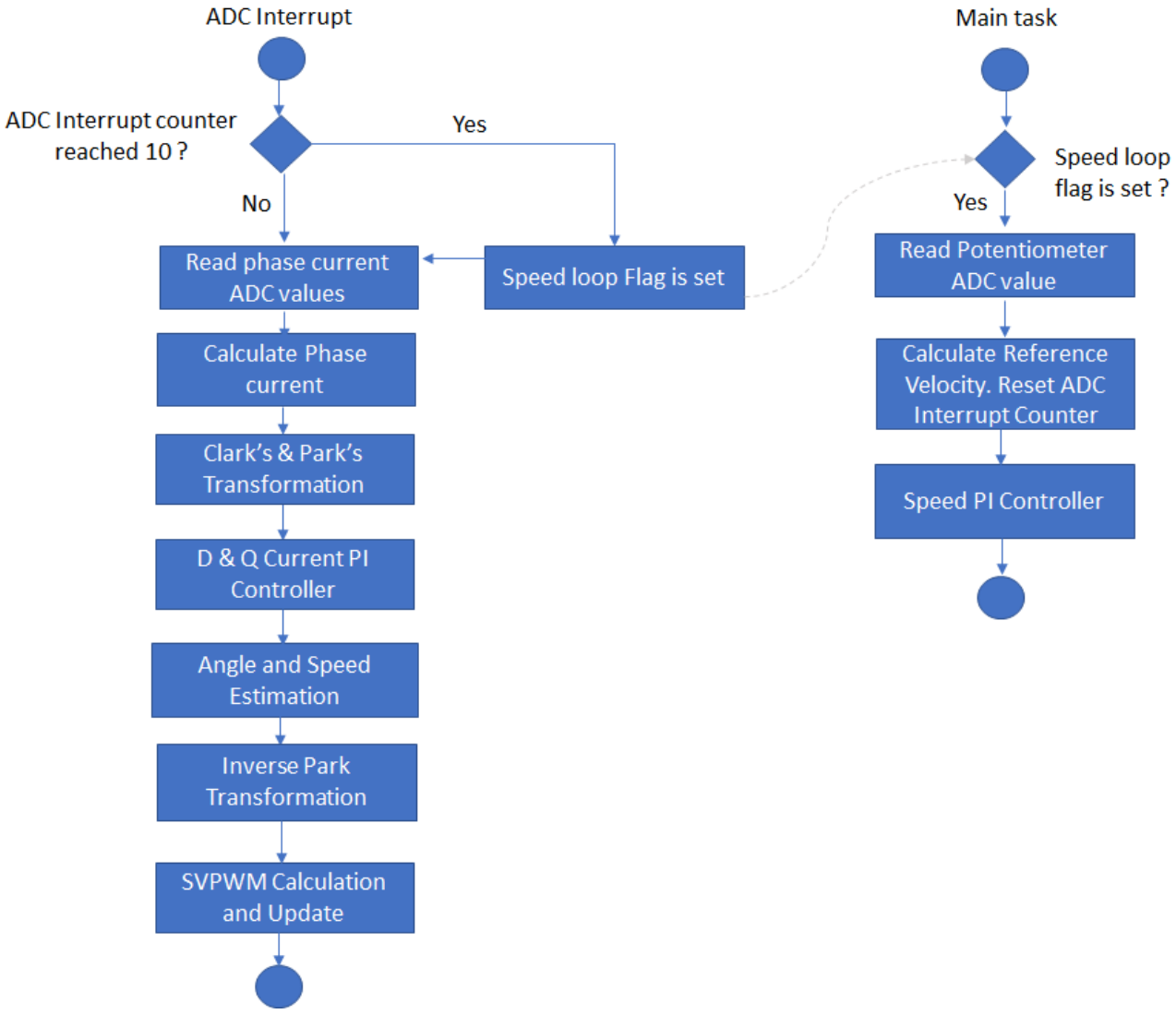
- Initialization and configuration of NVIC, AFEC, PWM motor control peripherals for generation of periodic ADC triggers and ADC conversion interrupt
- Current Offset measurement and calibration
- Initialize PI controller parameters for speed and current control loops

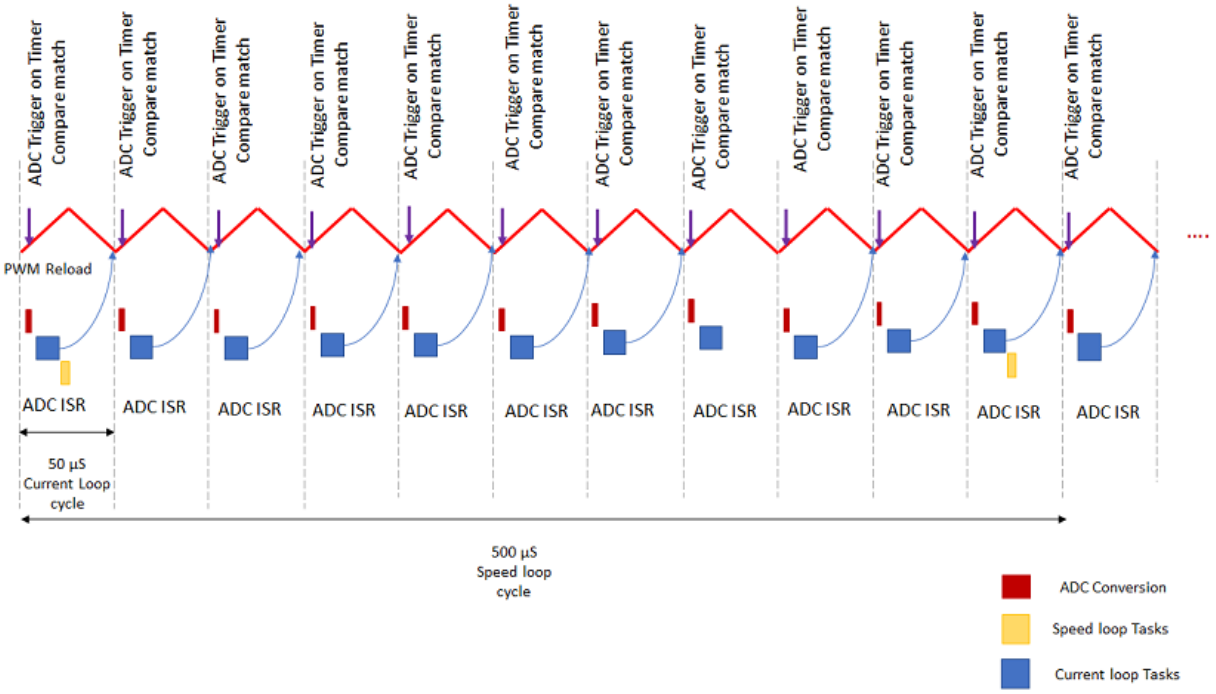
2. Start:

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 following flow chart and the timing diagram shows the tasks performed in run state:


ADC Interrupt
ADC Interrupt counter reached 10 ?
Yes
No
Read phase current ADC values
Speed loop Flag is set
Calculate Phase current
Clark's & Park's Transformation
D & Q Current PI Controller
Angle and Speed Estimation
Inverse Park Transformation
SVPWM Calculation and Update
Main task
Speed loop flag is set ?
Yes
Read Potentiometer ADC value
Calculate Reference Velocity. Reset ADC Interrupt Counter
Speed PI Controller



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. Therefore, 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 need 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 the following macro in `userparams.h` file.

This frequency should be same as the frequency configured in the 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 the following macros in `userparams.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 the following macros in `userparams.h` file.

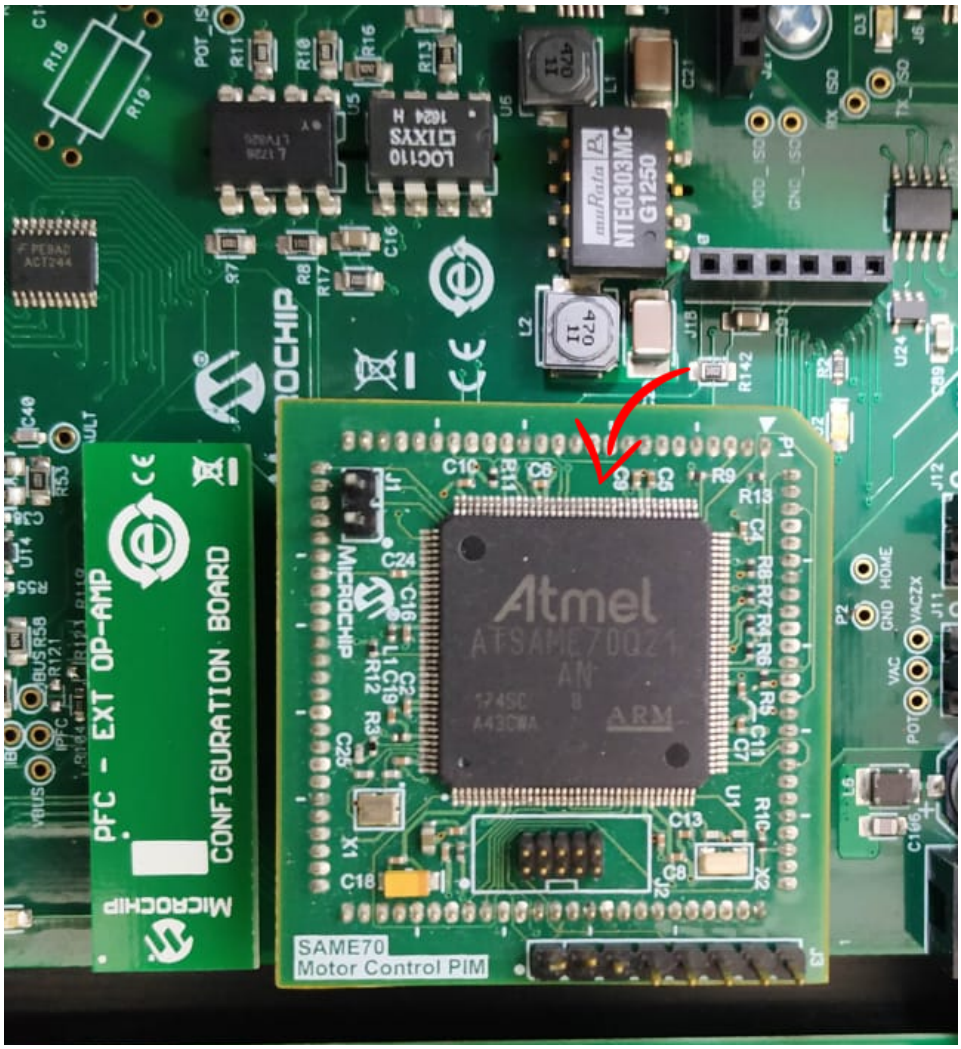
Macro	Description
OPEN_LOOP_FUNCTIONING	Open loop enable switch (0 - Disable, 1 - Enable)
Q_CURRENT_REF_OPENLOOP	Startup current in open loop
TORQUE_MODE	This a special mode where the speed control loop can be bypassed. The input of potentiometer is scaled and is fed directly to the q-axis PI controller.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

Hardware Setup

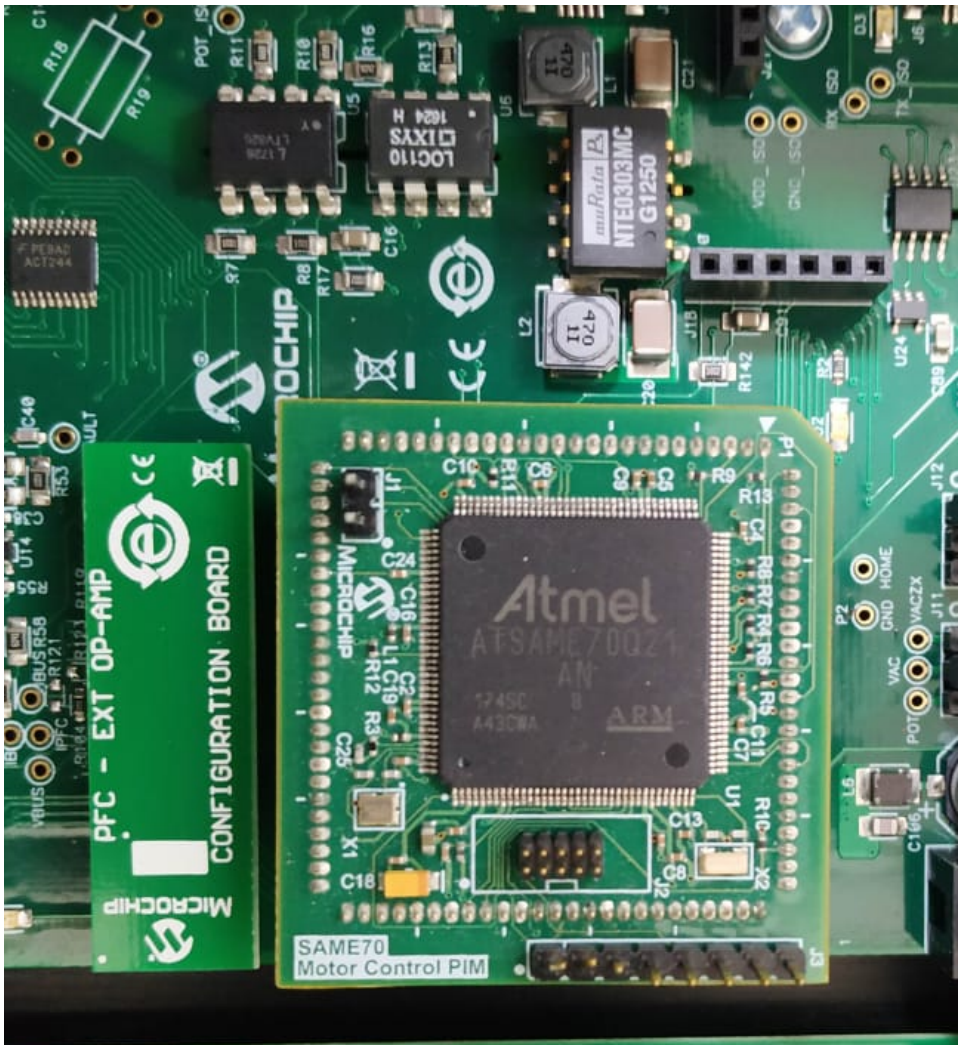
This section describes how to configure the supported hardware.

Description

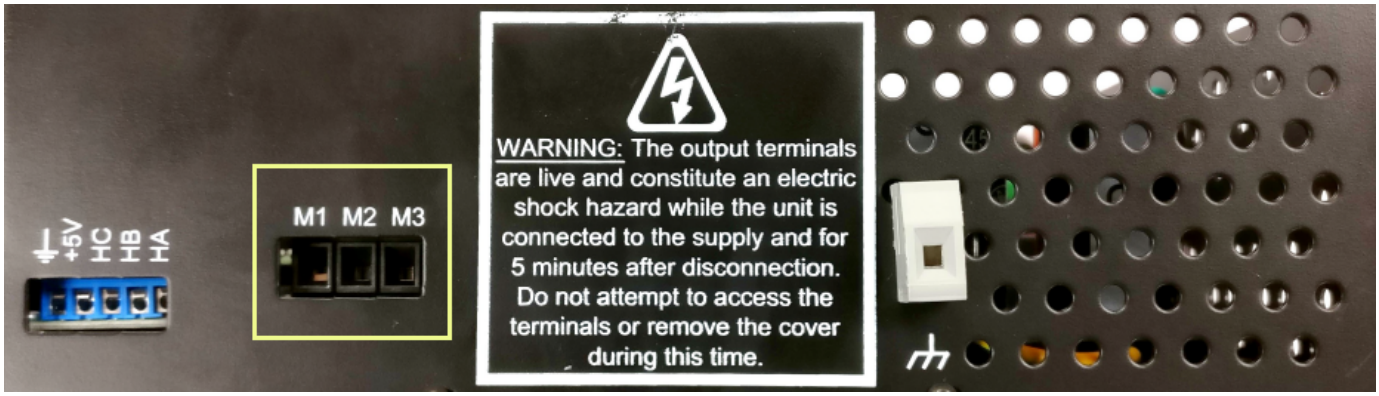
1. **Project mchv3_sam_e70_pim.X.**
   - **Hardware Used**
     - MCHV3 Development Board (High Voltage)
     - Isolated Embedded Debugger Interface
     - ATSAME70 PIM
     - Leadshine EL5-M0400-1-24 Motor
   - **Hardware Setup**
     1. Mount the ATSAME70 Motor Control Plug In Module on U9 header.



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

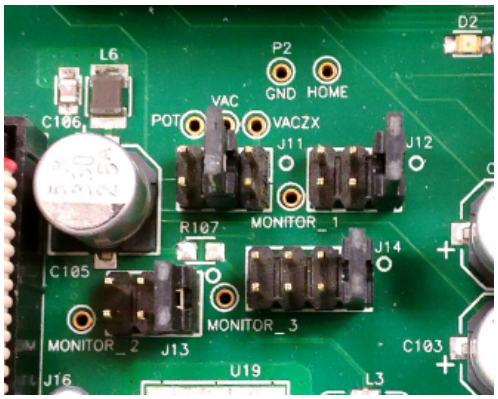


3. Motor Connections:
   - Phase U - M1
   - Phase V - M2
   - Phase W - M3



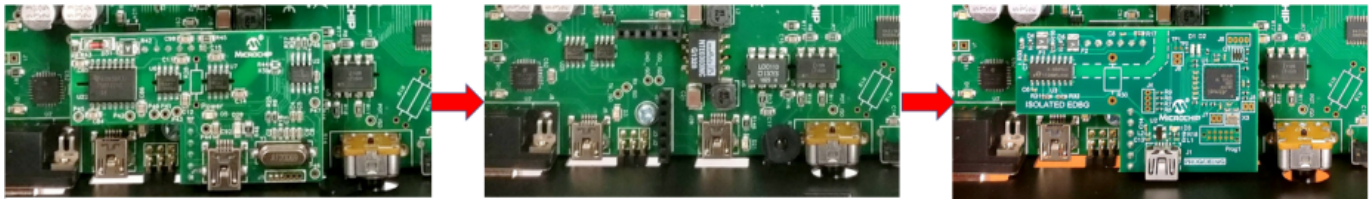
4. Jumper Settings:
   - J11 - VAC (Short Pin 3 - 4)

- J12 - IA (Short Pin 1 - 2)
- J13 - IB (Short Pin 1 - 2)
- J14 - Fault_IP/IBUS (Short Pin 1 - 2)

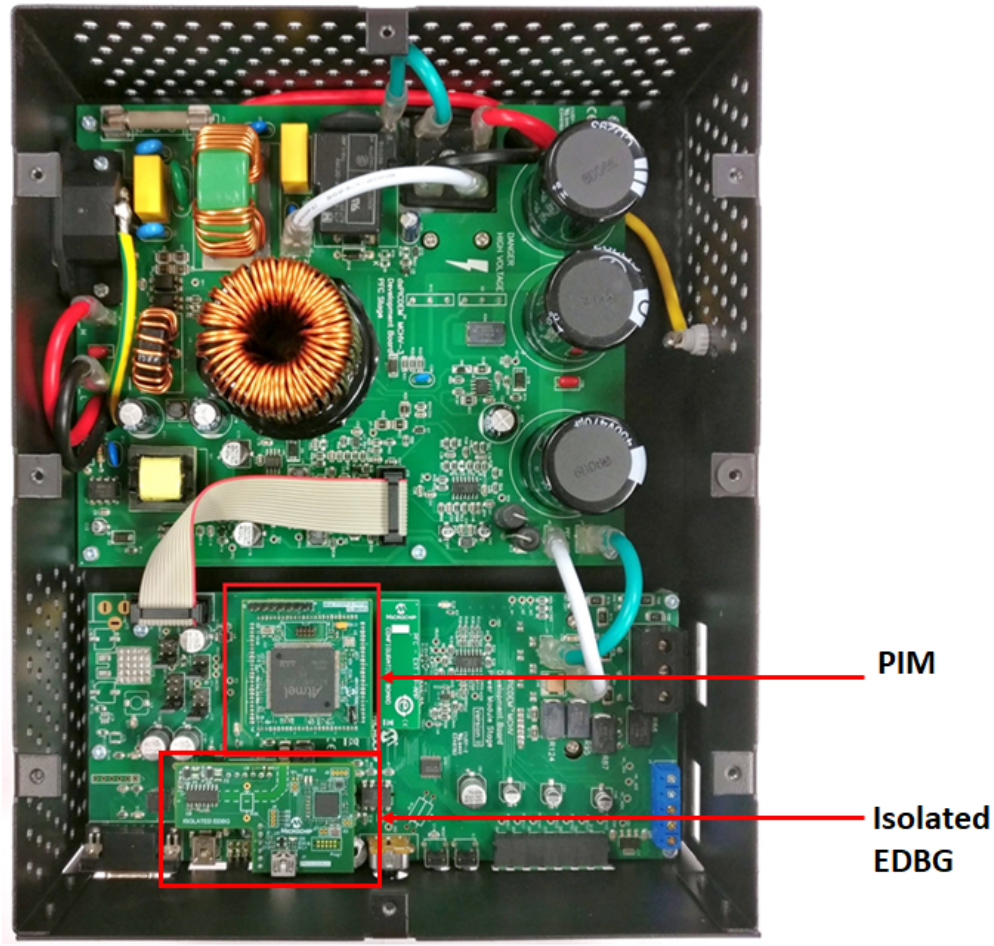


5. Power the board with (110V/220V) AC mains. For additional safety, it is recommended to use a current limited power supply while testing this software demonstration on a non-default hardware and motor.

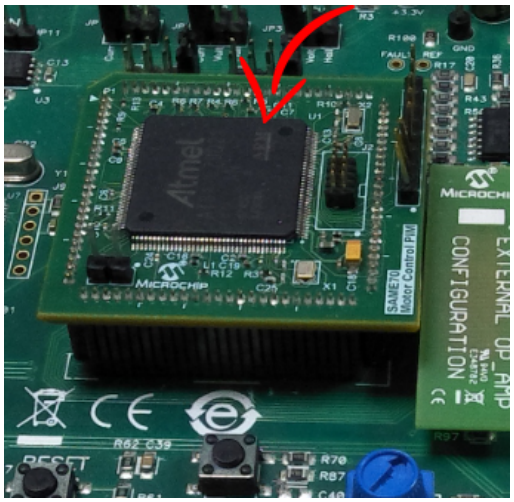
- **Installing Isolated Embedded Debugger**
  - Default programmer or debugger daughter card shipped with the MCHV3 board cannot program or debug SAM series MCU and therefore, it needs to be replaced with an Isolated Embedded Debugger Interface for MCHV.



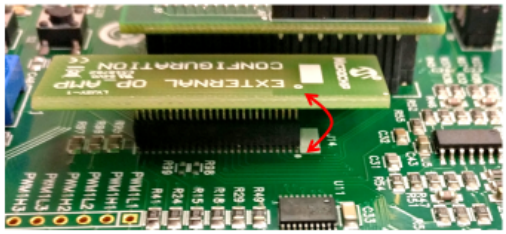
- **Complete Setup**



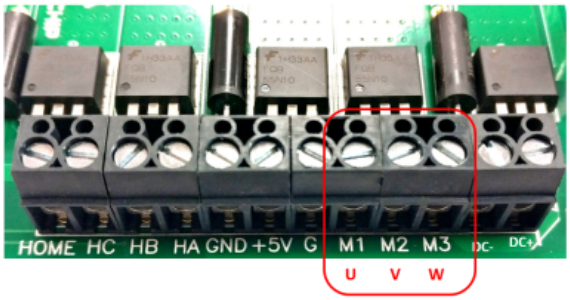
2. **Project mclv2_sam_e70_pim.X.**
   - **Hardware Used**
     - MCLV2 Development Board (Low Voltage)
     - ATSAME70 PIM
     - Long Hurst DMA0204024B101 Motor
- **Hardware Setup**
  1. Mount the ATSAME70 Motor Control Plug In Module on U9 header.



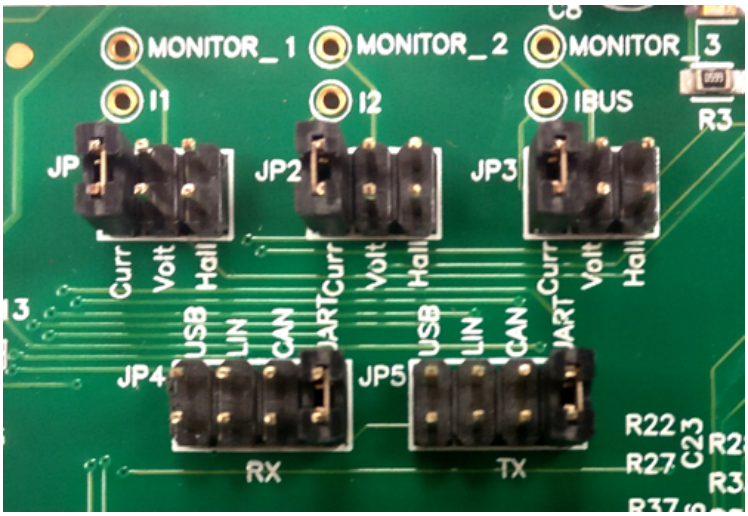
2. Place the "External Opamp Configuration" Matrix board at J14



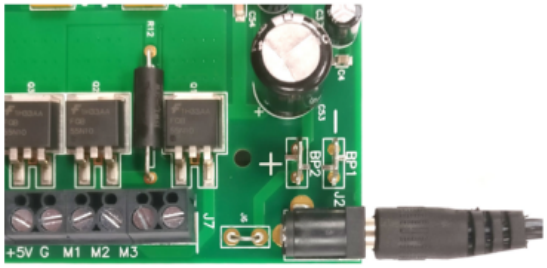
3. Motor Connections:
   - Phase U - M1
   - Phase V - M2
   - Phase W - M3



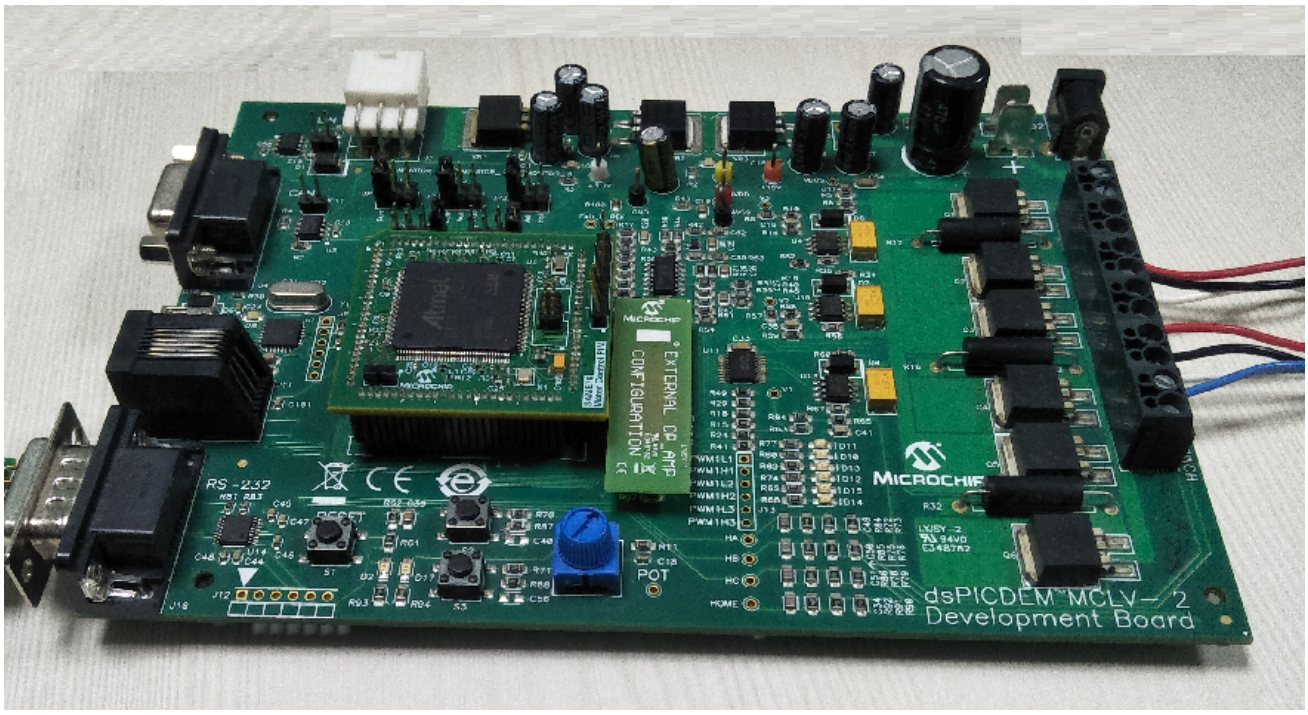
4. Jumper Settings:
   - JP1 - Curr, JP2 - Curr, JP3 - Curr
   - In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
   - In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**

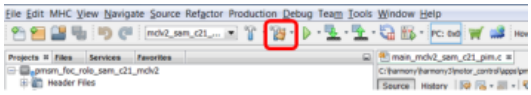


Running The Application

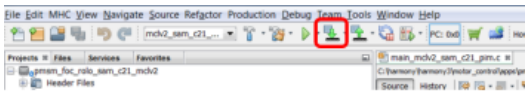
This section provides information on how to run the application using MPLAB X IDE.

Description

1. **Running the Application : pmsm_foc_pll_estimator_sam_e70_mchv3**
1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_pll_estimator_sam_e70\firmware\mchv3_sam_e70_pim.X
2. Clean and Build Project:



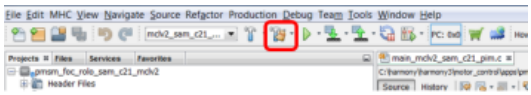
3. Program the project:



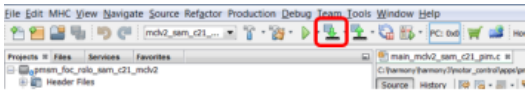
4. Press the switch "Push Button" to start/stop the motor.
5. Vary Potentiometer to change the speed of the motor.

Running the Application : pmsm_foc_pll_estimator_sam_e70 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_pll_estimator_sam_70\firmware\mclv2_sam_e70_pim.X
2. Clean and Build Project



3. Program the project



4. Press the switch S2 to start/stop the motor.

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_pll_estimator_sam_e54

This motor control example project shows how to control the Permanent Magnet Synchronous Motor (PMSM) with PLL

Estimator based Field Oriented Control (FOC) on a SAME54 Micro-controller.

Description

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 PLL based estimator. This algorithm is implemented on a SAME54P20A 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_e54_pim.X	pmsm_foc_pll_estimator_sam_e54_mchv3 is setup to work with ATSAME54 Motor Control Plug In Module + dsPICDEM™ MCHV-3 Development Board (High Voltage)
mclv2_sam_e54_pim.X	pmsm_foc_pll_estimator_sam_e54_mclv2 is setup to work with ATSAME54 Motor Control Plug In Module + dsPICDEM™ MCLV-2 Development Board (Low Voltage)

MPLAB Harmony Configurations

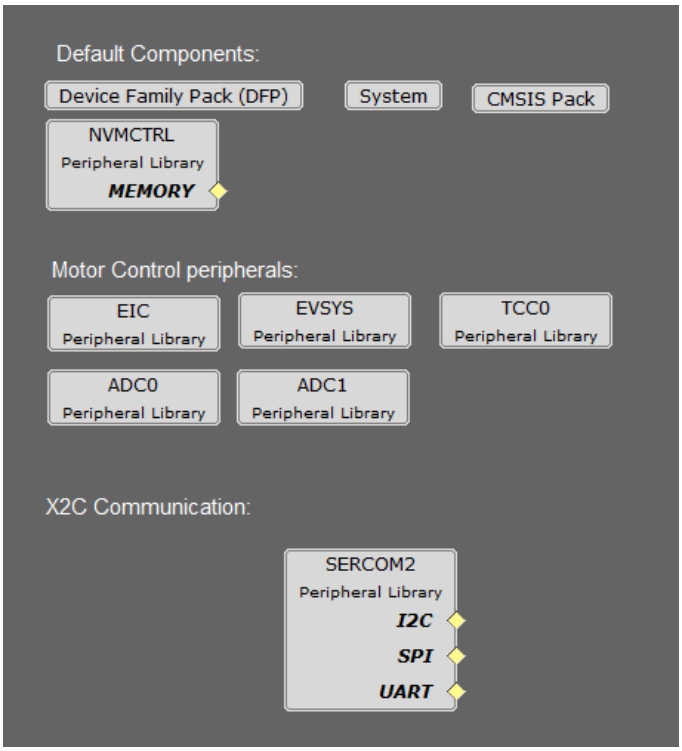
This topic provides information on the MHC configurations.

Description

Description

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

pmsm_foc_pll_estimator_sam_e54



Components Used

TCC0:

- Configured to generate three pairs of complimentary PWM signals at a frequency of 20 kHz in "Dual Slope PWM with interrupt/event when counter = TOP" a.k.a. "Center Aligned Mode".
- In "Dual Slop PWM" mode, the PWM duty cyle has an inverse relation with the compare value. For example, for a Period Count (PER) = 3000, Compare Count (CCx) = 2000 (66%); would result in a PWM with a 33% duty cycle. In order to achieve a direct relation between compare count and duty cycle, the PWMxH and PWMxL outputs are swapped.
- In dual shunt configuration, phase current measurements are performed at the end of the PWM cycle. Since the PWM outputs are swapped, the ADC conversion event is generated when the counter reaches TOP.
- Dead-time is enabled and set to
  - mchv3_sam_e54_pim.X - 2uS
  - mclv2_sam_e54_pim.X - 1uS
- 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 the same resolution and sampling time, both ADCs complete conversion at the same time.

EIC:

- External Interrupt Controller detects a 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 an intermediary between event generator and event users.
- Event generated by the TCC0 when the counter reaches TOP, is used by the ADC0 as a hardware trigger source via the 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.

SERCOM2:

- SERCOM2 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 the 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 PLL based estimator. The algorithm is implemented on the ATSAME54P20A MCU. For more details about PLL Estimator, refer to Application Note AN2520. The following section briefly describes the PLL Estimator based sensorless FOC algorithm, software design and implementation.

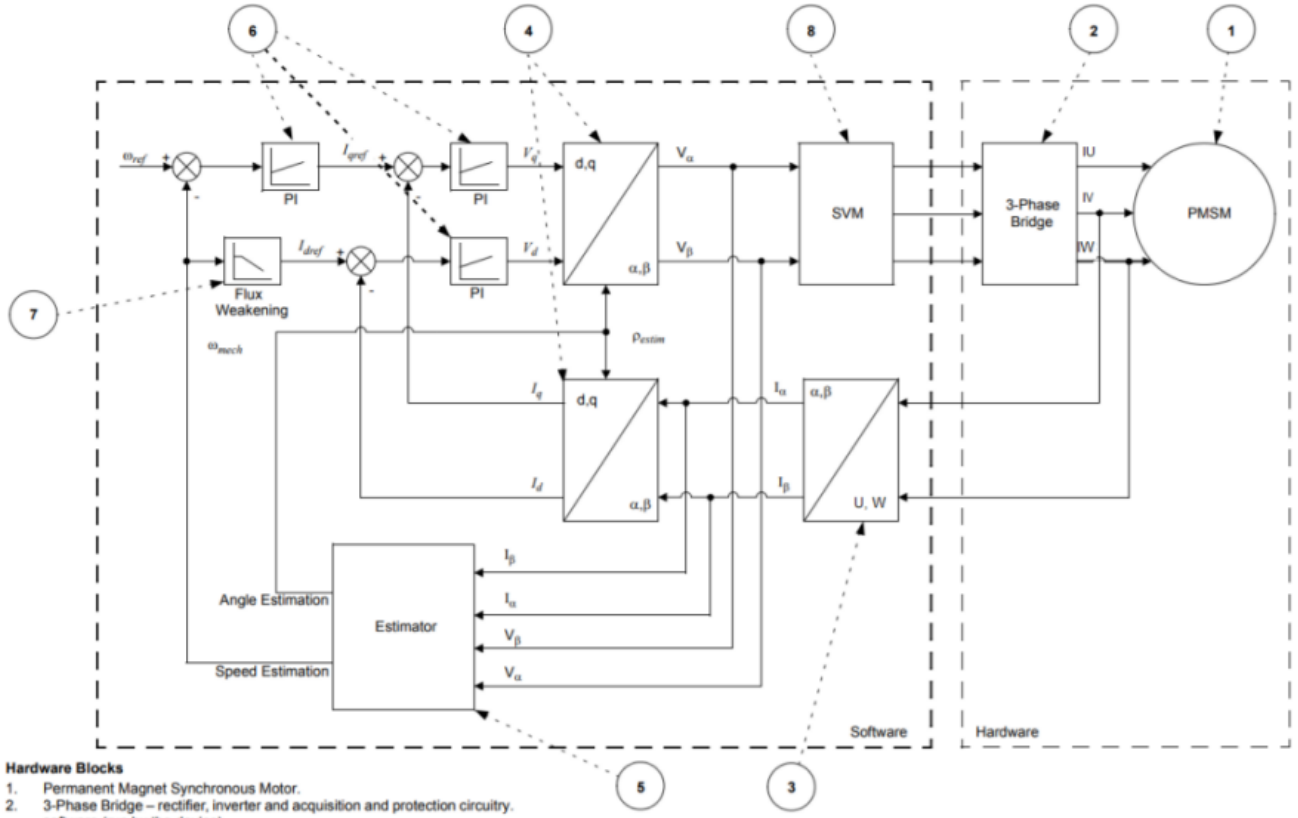
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from stationary reference frame to torque and flux producing current components in rotating reference frame using mathematical 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 the 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 PLL Estimator which is implemented in **mcLib_PLLEstimator** 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.

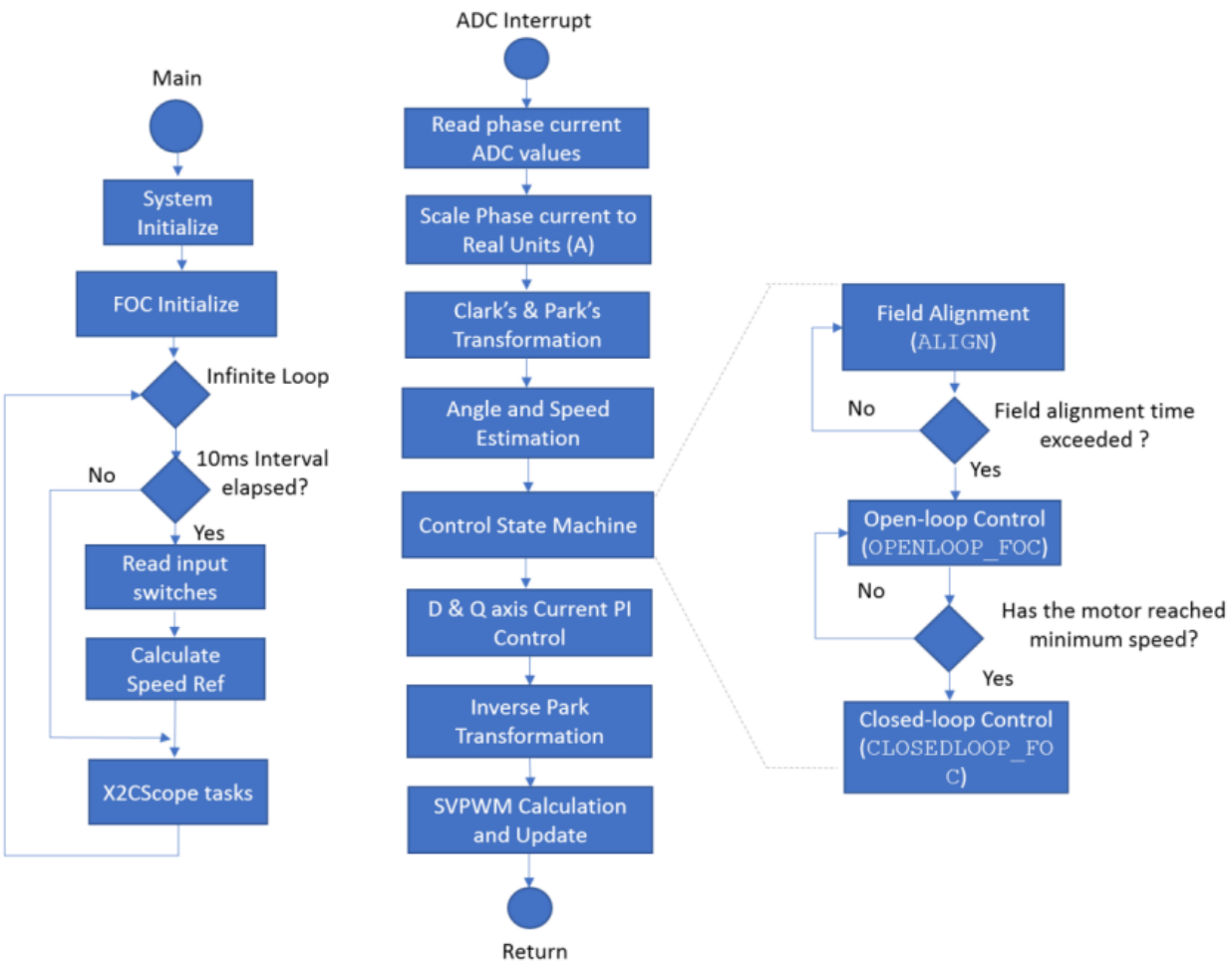


Hardware Blocks

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 - `OPENLOOP_FOC`:

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 PLL estimator to estimate the electrical speed of the PMSM motor with sufficient accuracy. Rotor angle information is obtained by integrating the estimated electrical speed of the motor.

Close Loop Control - `CLOSEDLOOP_FOC`:

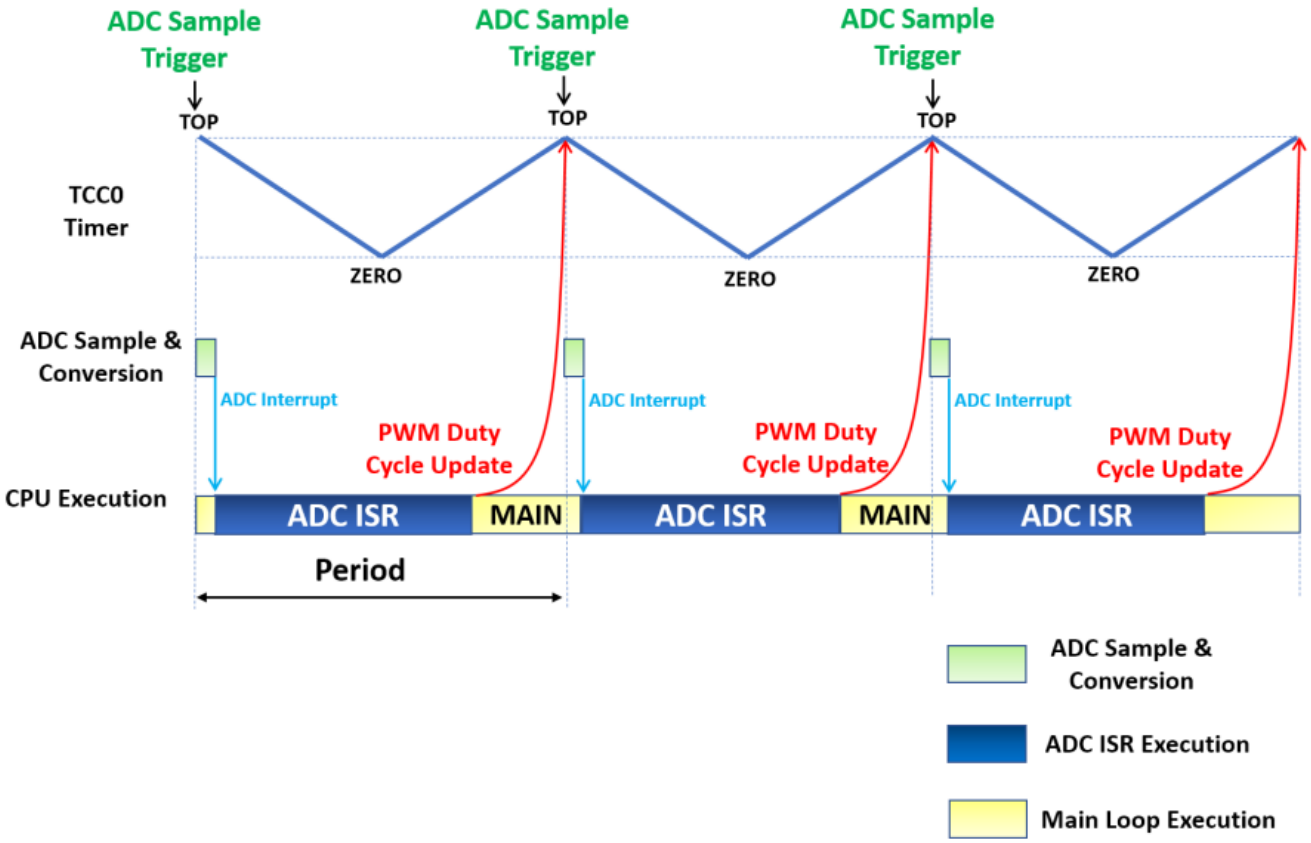
In this mode, the estimated rotor angle obtained from PLL-based estimator is used to perform field oriented control of the PMSM motor.

Timing Diagram:

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

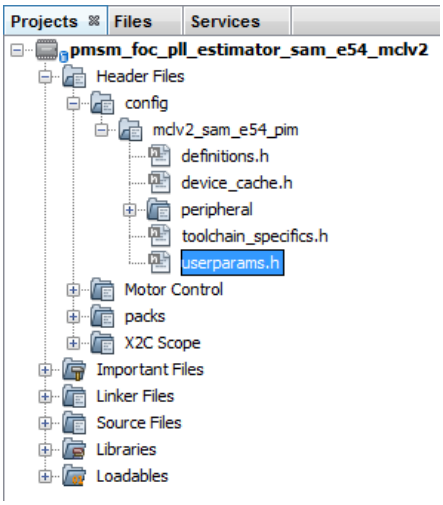
- ADC - It measures motor phase currents at the start of the PWM cycle.
- CPU - Based on motor phase currents measurements, the CPU executes the control algorithm to calculate the PWM duty cycle needed to apply required voltage across motor windings.
- TCC0 - It generates the PWM signals needed to apply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

The following timing diagram, shows the 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 PWM Frequency and delay loop interval:

Set the following PWM frequency and delay loop interval specific parameters in the `userparams.h` file.

Macro	Description
PWM_CLK	TCC0 peripheral clock in Hz
PWM_FREQ	PWM Frequency in Hz
DELAY_MS	Delay interval for reading switch input and calculating speed reference in milliseconds
SW_DEBOUNCE_DLY_MS	Switch debounce delay in milliseconds

2. Setting motor specific parameters:

Set the following motor specific parameters in the `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_BACK_EMF_CONSTANT_Vpeak_Line_Line_KRPM_MECH	Motor Back EMF Constant	Vpeak(L-L)/KRPM
NOPOLESPAIRS	Number of Pole Pairs	-
STAR_CONNECTED_MOTOR	1 : Motor is Star Connected, 0 : Motor is Delta Connected	-
NOMINAL_SPEED_RPM	Nominal Speed of the motor (Base Speed)	RPM
FW_SPEED_RPM	Max Speed of the motor in Flux Weakening mode	RPM
ALIGN_TIME_IN_SEC	Motor Alignment duration	seconds
OPENLOOP_RAMP_TIME_IN_SEC	Ramp time to reach from 0 to Open Loop Speed (Open Loop FOC duration)	seconds
OPENLOOP_END_SPEED_RPM	Motor speed at which the algorithm switches from open loop FOC to closed loop FOC operaton. In other words, maximum open loop speed	RPM
CLOSEDLOOP_RAMP_RATE_RPM_SEC	Speed ramp rate in closed loop FOC mode	RPM/sec

ALIGN_Q_CURRENT_REF	Q axis current reference during "alignment" mode	A
OPENLOOP_Q_CURRENT_REF	Q axis current reference during "open loop FOC" mode	A
RHO_OFFSET_ELEC_DEG	Angle Offset between actual rotor angle and estimated angle during open loop mode. This offset helps in smooth transition from Open Loop to Closed Loop	electrical degrees
MAX_FW_NEGATIVE_ID_REF	Maximum negative D axis reference current during Flux Weakening	A
TORQUE_MODE_MAX_CUR	Maximum Torque Mode Current Reference	A
MAX_MOTOR_CURRENT	Maximum Motor Current	A

3. Setting board specific parameters:

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

Macro	Description	Units
MAX_BOARD_CURRENT	Maximum measurable current by ADC without saturating	A
DCBUS_SENSE_TOP_RESISTOR	DC Bus voltage Divider - Top Side Resistor	Kohm
DCBUS_SENSE_BOTTOM_RESISTOR	DC Bus voltage Divider - Bottom Side Resistor	Kohm

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
SPEEDCNTR_PTERM	Proportional gain of speed controller
SPEEDCNTR_ITERM	Integral gain of speed controller
SPEEDCNTR_CTERM	Anit-windup gain of speed controller
SPEEDCNTR_OUTMAX	Speed Loop PI Controller Maximum Output - Max Q axis Current Reference in A

Current PI controller gains for D axis:

Macro	Description
D_CURRCNTR_PTERM	Proportional gain for D axis current controller
D_CURRCNTR_ITERM	Integral gain for D axis current controller
D_CURRCNTR_CTERM	Anti-windup gain for D axis current controller
D_CURRCNTR_OUTMAX	D axis PI Controller Maximum Output - Max D axis Voltage (Normalized)

Current PI controller gains for Q axis:

Macro	Description
Q_CURRCNTR_PTERM	Proportional gain for Q axis current controller
Q_CURRCNTR_ITERM	Integral gain for Q axis current controller
Q_CURRCNTR_CTERM	Anti-windup gain for Q axis current controller

Q_CURRCNTR_OUTMAX	Q axis PI Controller Maximum Output - Max Q axis Voltage (Normalized)

5. PLL Estimator Parameters:

Set the following PLL Estimator specific parameters in the `userparams.h` file.

Macro	Description
KFILTER_ESDQ	D, Q axis Back EMF Low Pass Filter Coefficient - Cut off frequency of the filter is proportional to the filter coefficient
KFILTER_VELESTIM	PLL Estimator Speed Filter Coefficient - Cut off frequency of the filter is proportional to the filter coefficient
KFILTER_IDREF	D axis Current Reference Filter in Flux Weakening mode - Cut off frequency of the filter is proportional to the filter coefficient

6. Operating Modes :

This example provides compile time re-configuration options using #define macro directives to enable or disable different operating 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 `OPENLOOP_Q_CURRENT_REF`.
  - Depending upon the load inertia and startup acceleration time (`OPENLOOP_RAMP_TIME_IN_SEC`), if the generated torque is not sufficient, then it may result in stalling of the motor.
  - `OPENLOOP_Q_CURRENT_REF` and `OPENLOOP_RAMP_TIME_IN_SEC` need to be modified to ensure that the motor starts up successfully in open loop mode without stalling.
- 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 0 - `TORQUE_MODE_MAX_CUR`.
  - Depending upon the load, if the torque is too low then it could cause the motor to stall.
- Flux Weakening - `ENABLE_FLUX_WEAKENING`
  - Defining this macro, allows the motor to enter "flux weakening" mode thereby allowing the motor to exceed its base speed
  - Maximum speed in flux weakening mode is set by `FW_SPEED_RPM`
  - Maximum D axis current reference in flux weakening mode is set by `MAX_FW_NEGATIVE_ID_REF`
  - Note: In flux weakening mode, the motor cannot provide constant torque which could cause the motor to stall. Additionally, operating motor in flux weakening mode for prolonged duration may result in demagnetization of the permanent magnets of the PMSM.

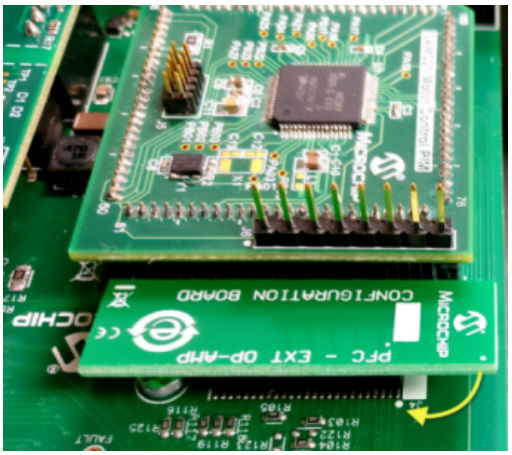
Hardware Setup

This section describes how to configure the supported hardware.

Description

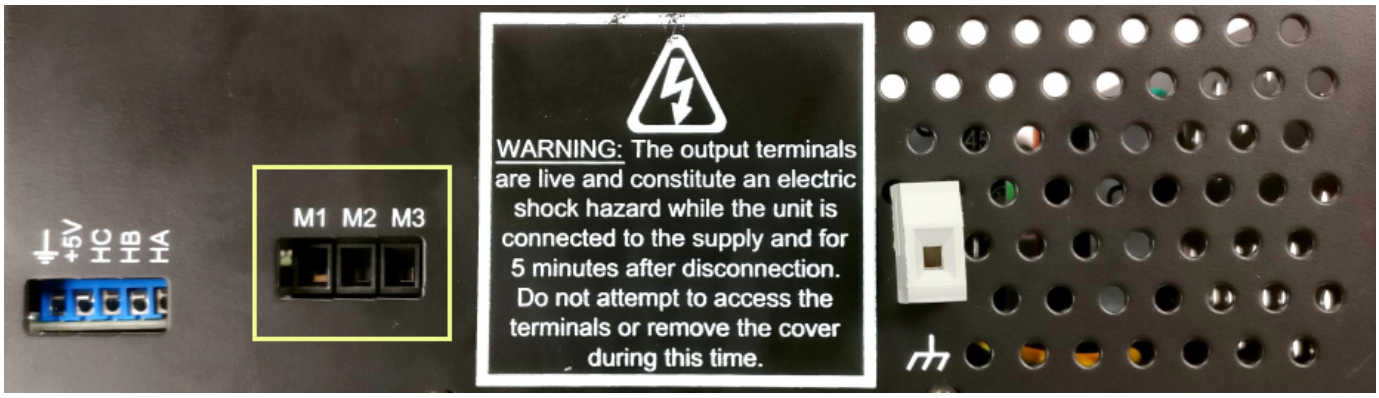
1. **Project mchv3_sam_e54_pim.X.**
   - **Hardware Used**
     - dsPICDEMTM MCHV3 Development Board (High Voltage)
     - ATSAME54 Motor Control Plug In Module
     - Isolated Embedded Debugger Interface for MCHV
     - Leadshine EL5-M0400-1-24 Motor
   - **Hardware Setup**
     1. Mount the ATSAME54 Motor Control Plug In Module on U9 header.

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

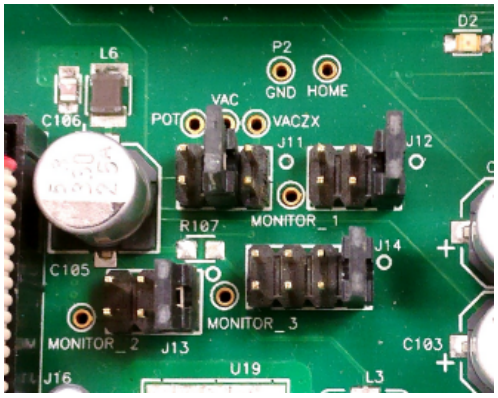


3. Motor Connections:

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



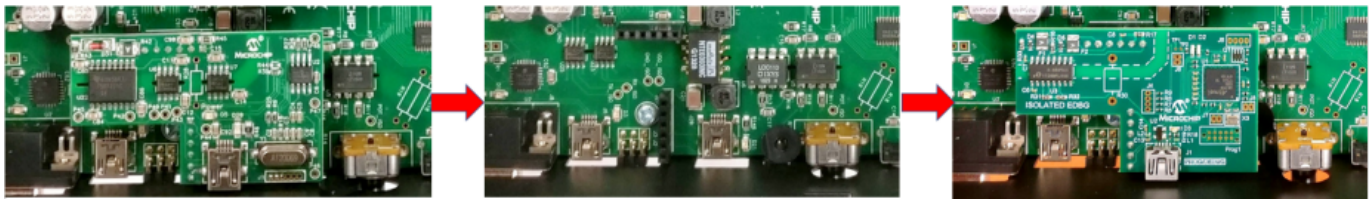
4. Jumper Settings:
   - J11 - VAC (Short Pin 3 - 4)
   - J12 - IA (Short Pin 1 - 2)
   - J13 - IB (Short Pin 1 - 2)
   - J14 - Fault_IP/IBUS (Short Pin 1 - 2)



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

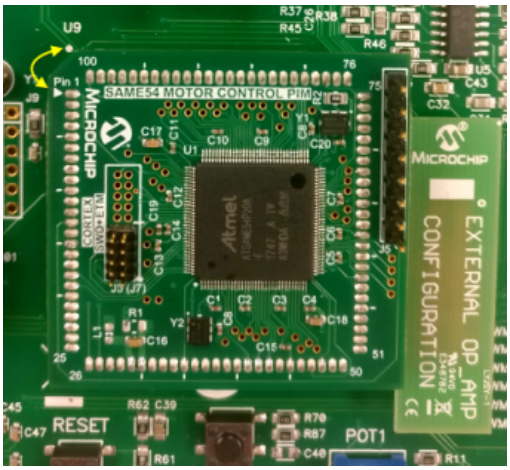
MCU and therefore, it needs to be replaced with an Isolated Embedded Debugger Interface for MCHV.



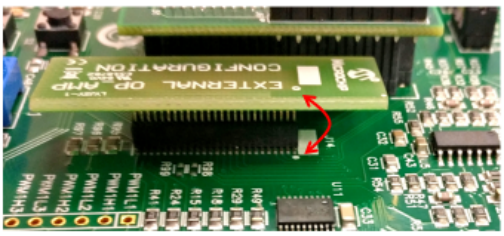
- **Complete Setup**

2. **Project mclv2_sam_e54_pim.X.**

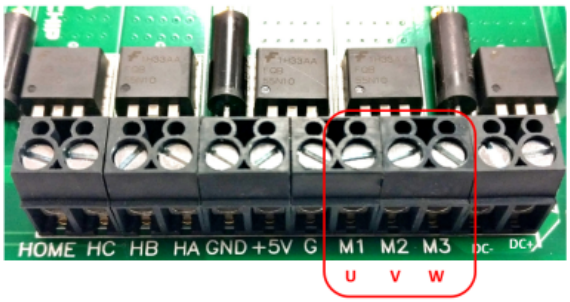
- **Hardware Used**
  - dsPICDEMTM MCLV2 Development Board (Low Voltage)
  - ATSAME54 Motor Control Plug In Module
  - Long Hurst DMA0204024B101 Motor
- **Hardware Setup**
  1. Mount the ATSAME54 Motor Control Plug In Module on U9 header.



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

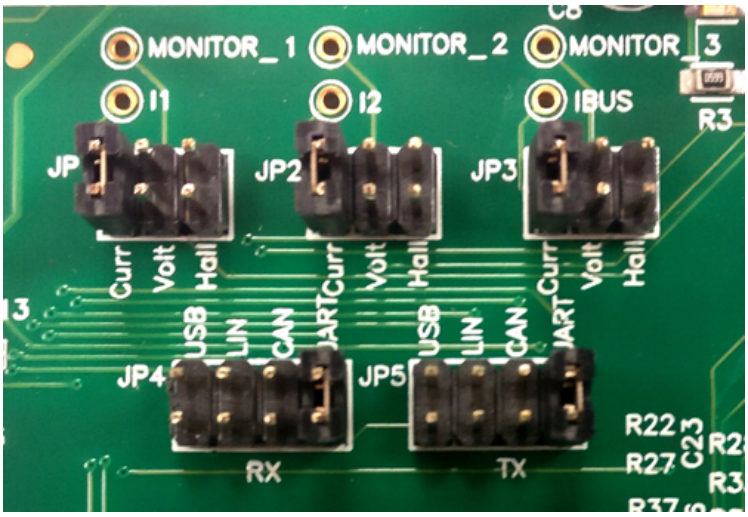


3. Motor Connections:
   - Phase U - M1
   - Phase V - M2
   - Phase W - M3

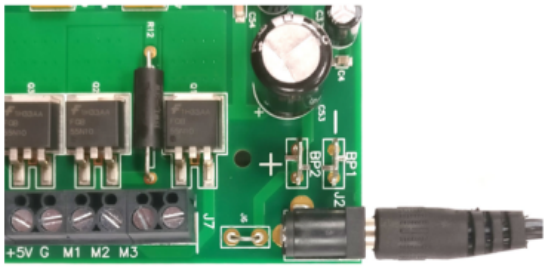


4. Jumper Settings:

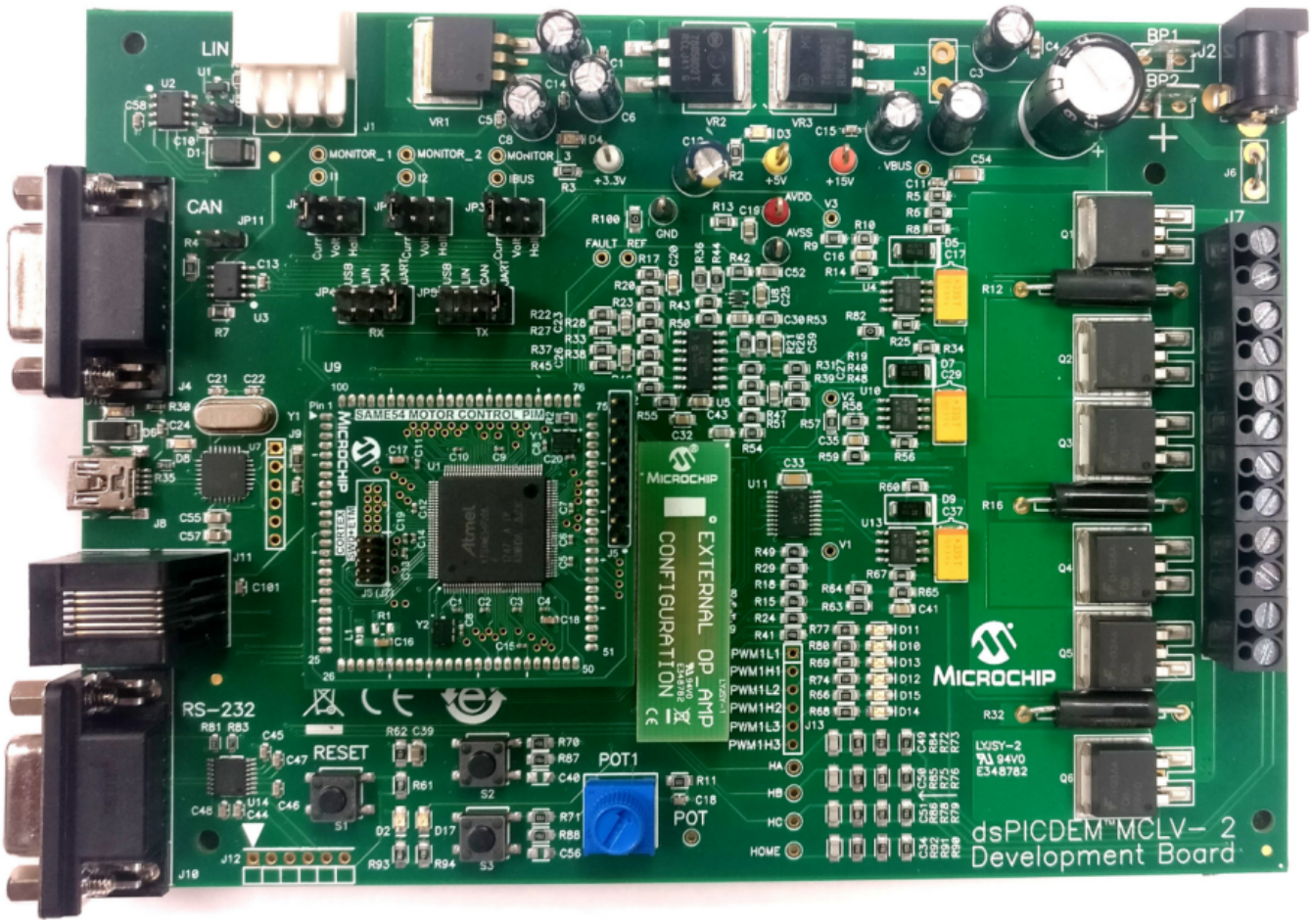
- JP1 - Curr, JP2 - Curr, JP3 - Curr
- In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**



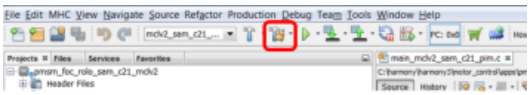
Running The Application

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

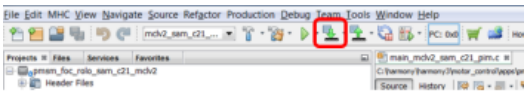
Description

Running the Application : pmsm_foc_pll_estimator_sam_e54_mchv3

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_pll_estimator_sam_e54\firmware\mchv3_sam_e54_pim.X
2. Clean and Build Project:



3. Program the project:



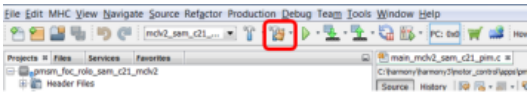
4. Press the switch "Push Button" to start/stop the motor.
5. Vary Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_pll_estimator_sam_e54 - MCHV3

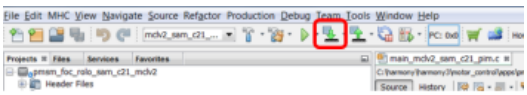
If LED D19 is ON, it indicates a non-recoverable fault caused by over-current condition. The MCU needs to be reset in order to recover from this fault mode.

Running the Application : pmsm_foc_pll_estimator_sam_e54 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_pll_estimator_sam_e54\firmware\mclv2_sam_e54_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.

Fault Indication and Recovery : pmsm_foc_pll_estimator_sam_e54 - MCLV2

If LED D17 is ON, it indicates a non-recoverable fault caused by over-current condition. The MCU needs to be reset in order to recover from this fault mode.

Project Name	Output
`mchv3_sam_e54_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_e54_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 the Permanent Magnet Synchronous Motor (PMSM) using an Encoder based Field Oriented Control (FOC) on a 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 the PMSM motor for applications which requires high dynamic performance.

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

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 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 to 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)

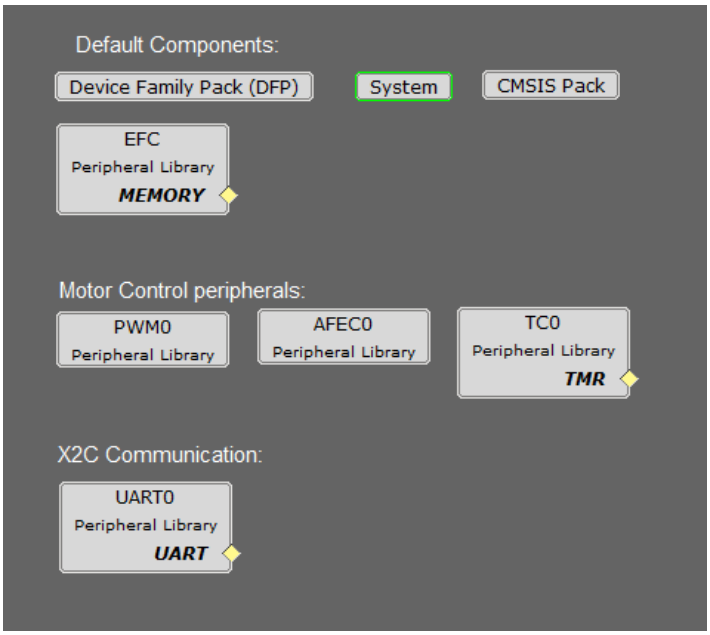
MPLAB Harmony Configurations

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 in run time

Control Algorithm

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

Description

The project implements an encoder based Field Oriented Control (FOC) algorithm on the SAME70 32-bit micro-controller to regulate the speed of the PMSM motor. For details of encoder based FOC, refer to Application Note AN2757. The following section describes briefly about the 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 a stationary reference frame to torque and flux producing currents components in a rotating reference frame using mathematical transformations. The Field Oriented Control is done as follows:

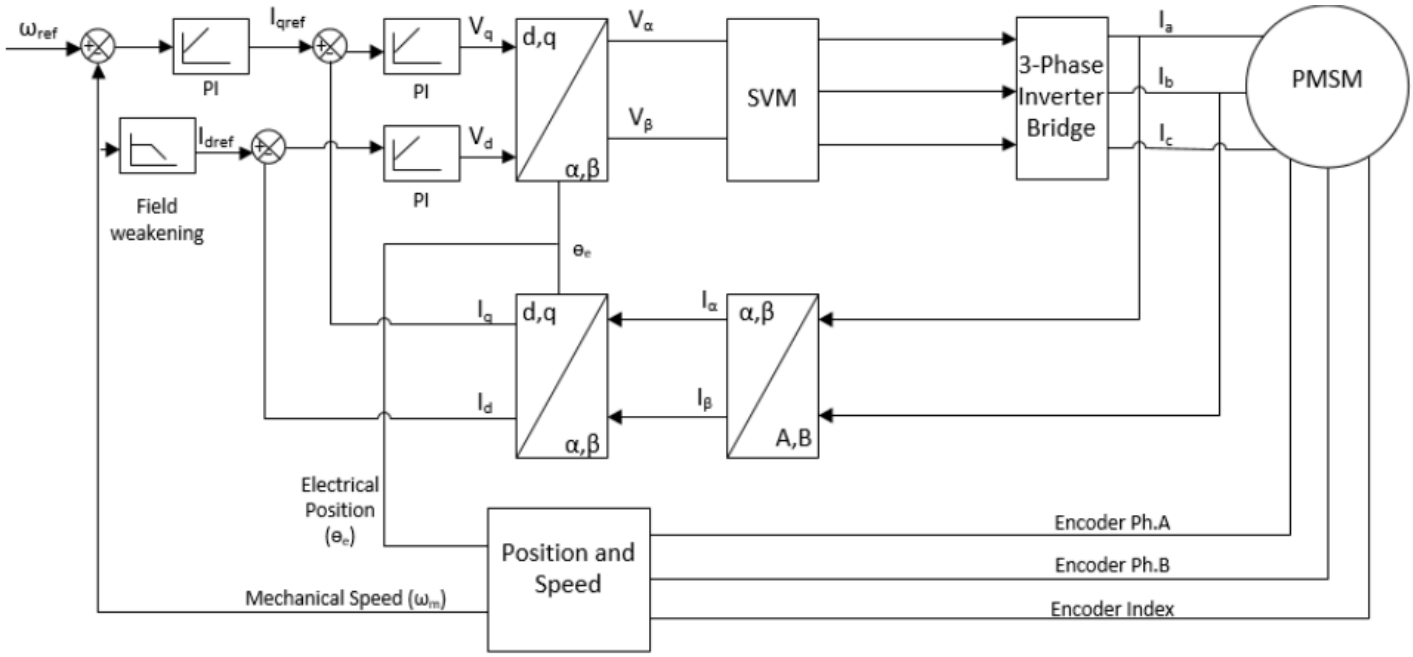
1. Measure the motor phase currents.
2. Transform them into the two phase system (a, b) using the Clarke transformation.
3. Calculate the rotor position angle.
4. Transform stator currents into the d,q-coordinate system using the Park transformation.

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 the Inverse Park transformation.
7. Using the space vector modulation, the three-phase output 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.

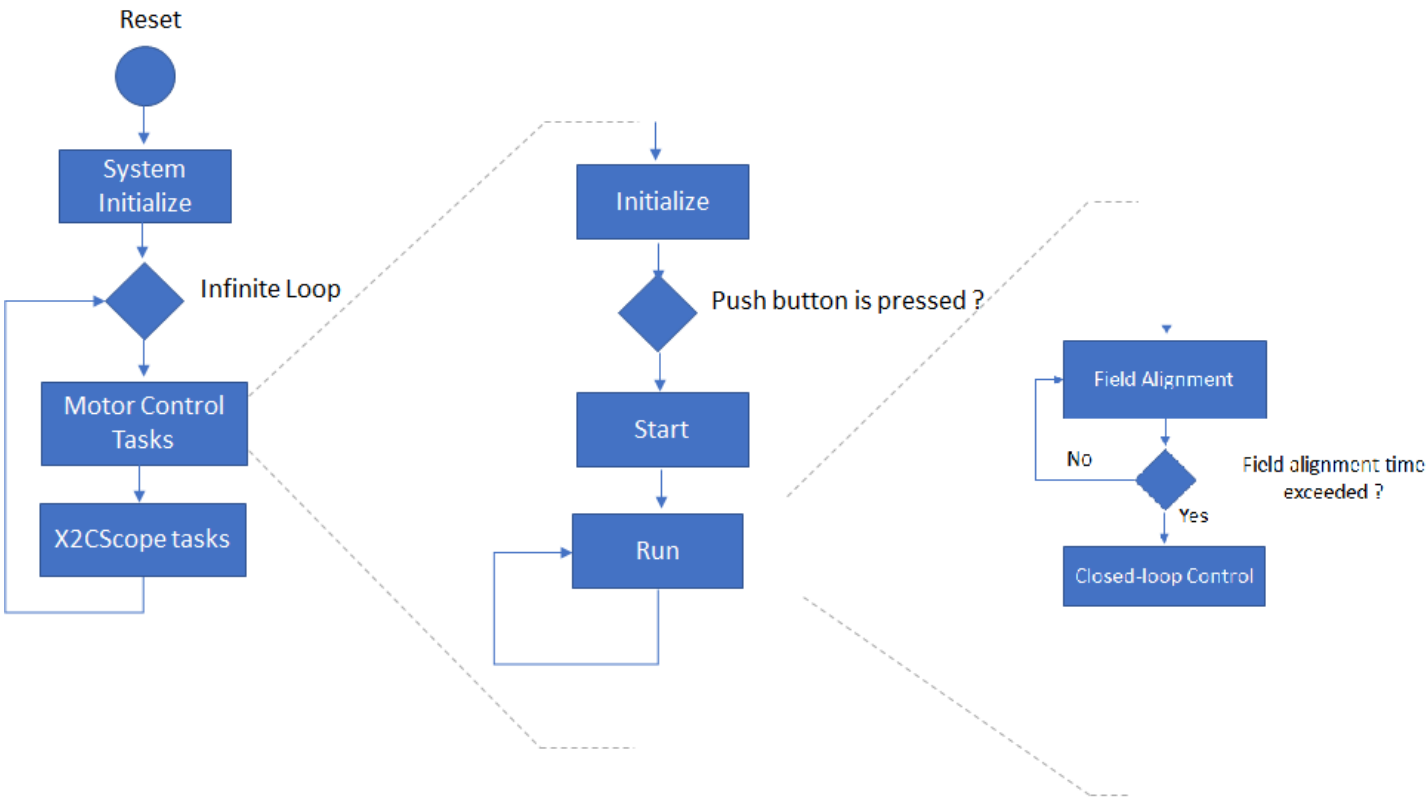
The rotor position angle is determined from the Quadrature encoder.

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 previous figure. The following sections briefly describes the various states in the PMSM speed control task:

1. Initialize:

In this state, following tasks are performed:

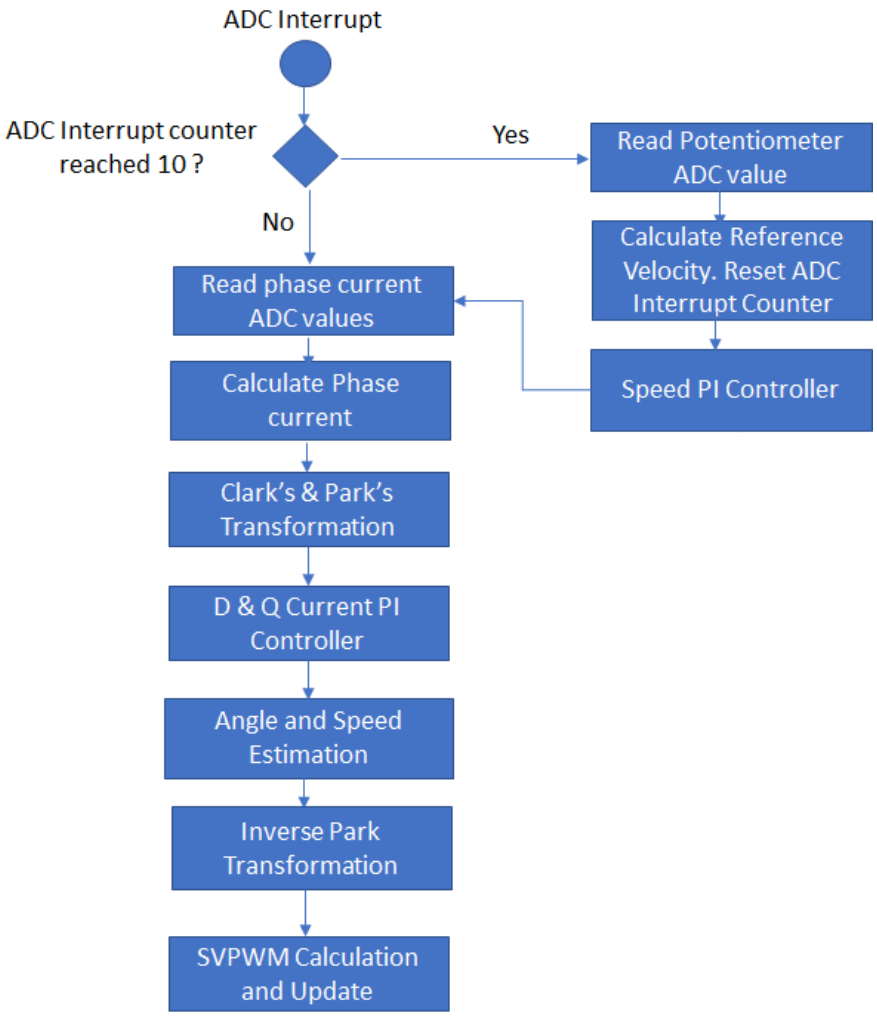
- Initialization andconfiguration of NVIC, AFEC, PWM, TC QDEC motor control peripherals for generation of periodic ADC triggers and ADC conversion interrupt
- Current Offset measurement and calibration
- Initialize PI controller parameters for speed and current control loops

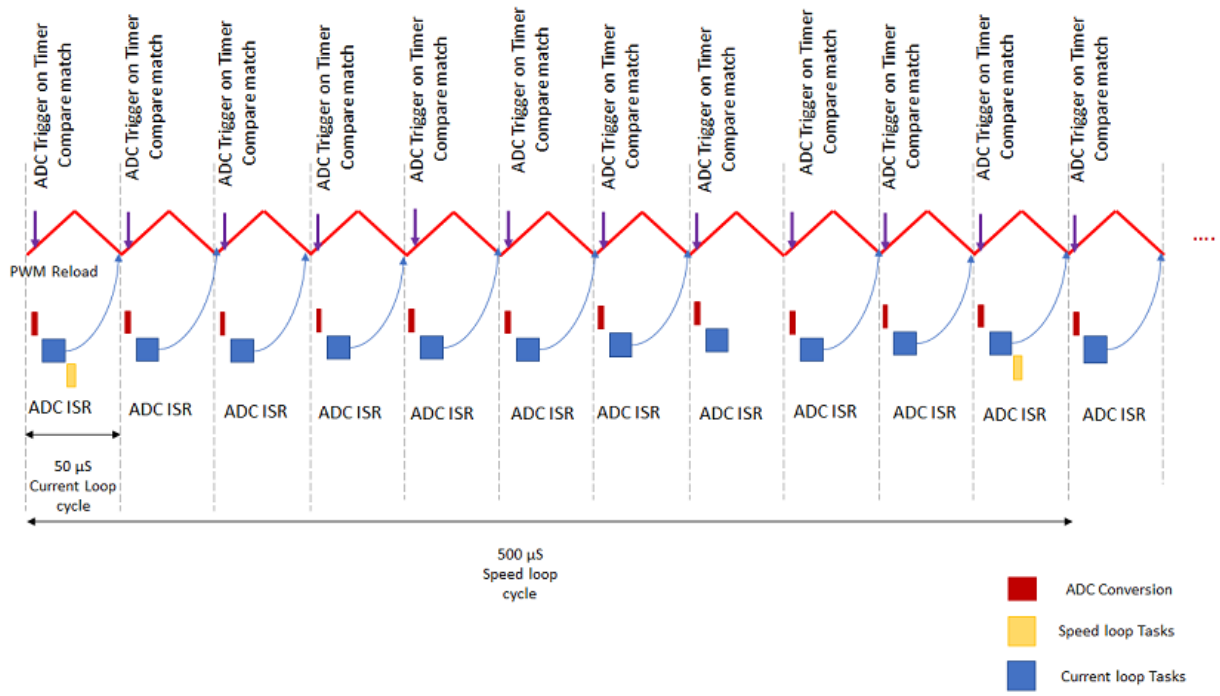
2. Start:

In this state, the motor control state variables are reset and the 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:


ADC Interrupt
ADC Interrupt counter reached 10 ?
Yes
No
Read Potentiometer ADC value
Calculate Reference Velocity. Reset ADC Interrupt Counter
Speed PI Controller
Read phase current ADC values
Calculate Phase current
Clark's & Park's Transformation
D & Q Current PI Controller
Angle and Speed Estimation
Inverse Park Transformation
SVPWM Calculation and Update


ADC Trigger on Timer Compare match
PWM Reload
ADC ISR
50 µS Current Loop cycle
500 µS Speed loop cycle
ADC Conversion
Speed loop Tasks
Current loop Tasks

In run state, two threads are executed; the 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. Therefore, the ADC interrupt cycle indicates the current control frequency.

- **Field Alignment:**

The quadrature encoder is an incremental encoder. It is important to know the initial position of the rotor to maintain orthogonal rotor and stator flux. To achieve this, the q-axis is excited with a small open loop current while the d-axis is excited with zero current.

In order to avoid pseudo-alignment, i.e. rotor alignment at 180 degrees out of phase, dual step alignment approach is used. The rotor is aligned to an intermediate angle before aligning it to the zero position.

After the initial alignment, an electrical angle is shifted by 90 degrees to obtain the maximum torque.

- **Close Loop Control:**

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

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 need 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 the following macro in `userparams.h` file.

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

Macro	Description
PWM_FREQUENCY	Current controller and PWM frequency in Hz.

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

Set the 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 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. Debugging Features:

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

Macro	Description
TORQUE_MODE	This is 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).

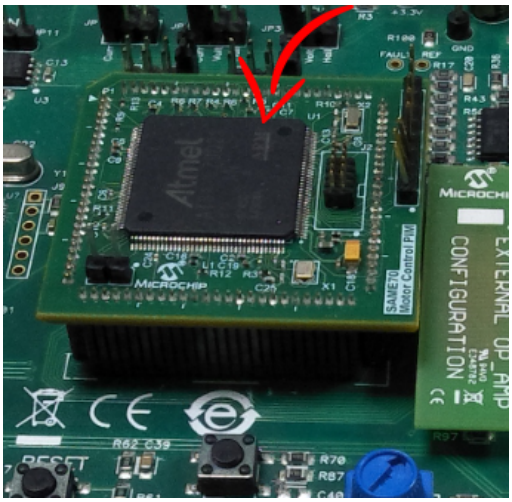
Hardware Setup

This section describes how to configure the supported hardware.

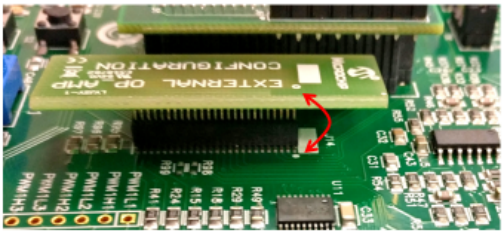
Description

1. **Project mclv2_sam_e70_pim.X.**
   - **Hardware Used**
     - MCLV2 Development Board (Low Voltage)
     - ATSAME70 PIM
     - Long Hurst DMA0204024B101 Motor
   - **Hardware Setup**
- **Hardware Setup**

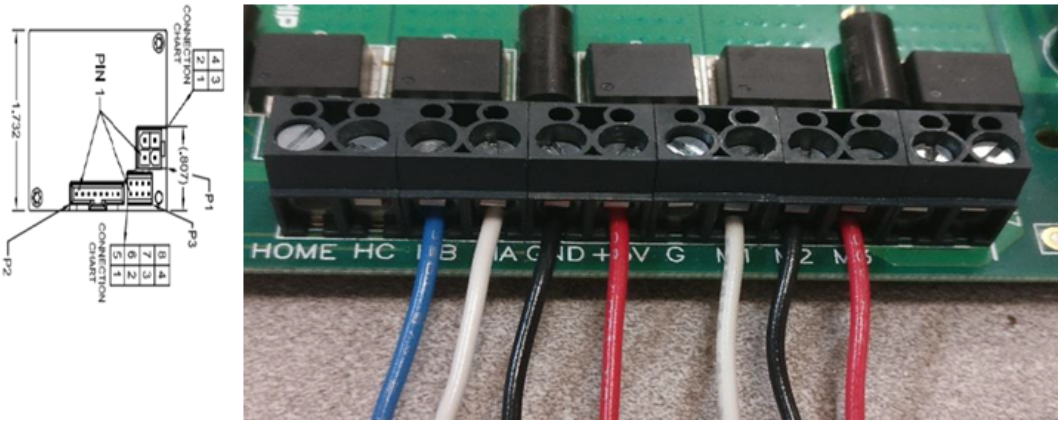
1. Mount the ATSAME70 Motor Control Plug In Module on U9 header.



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

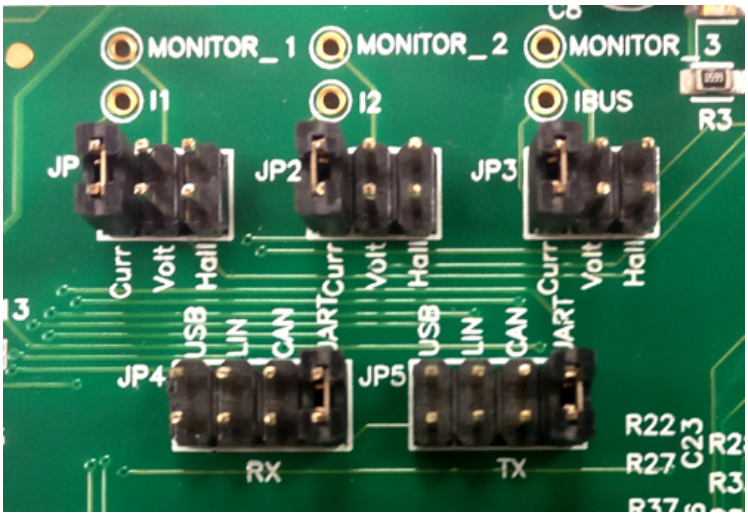


3. Connect motor and encoder terminals.

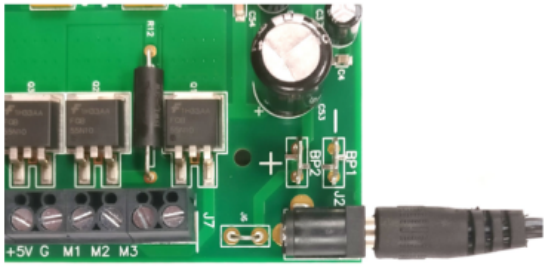


4. Jumper Settings:

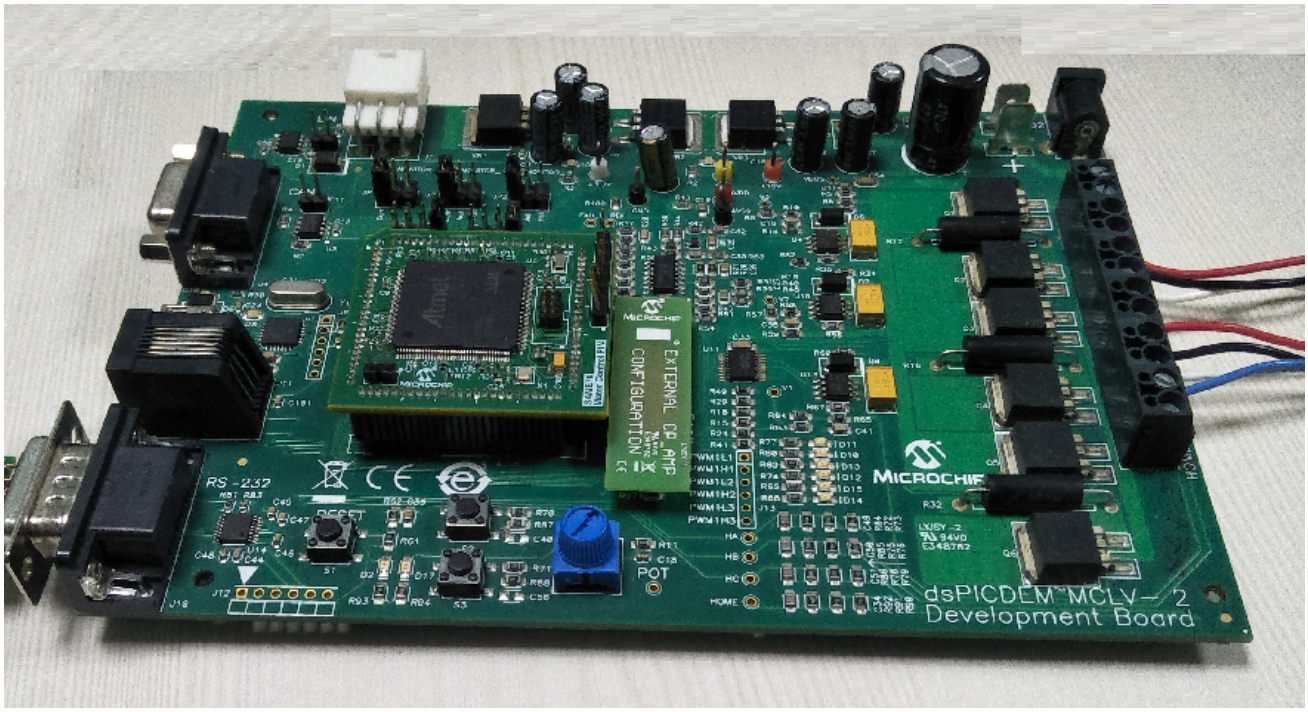
- JP1 - Curr, JP2 - Curr, JP3 - Curr
- In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**

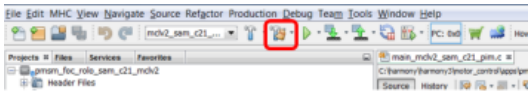


Running The Application

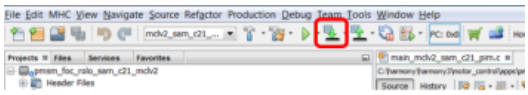
This section provides information on how to run the application using MPLAB X IDE.

Description

1. **Running the Application : pmsm_foc_encoder_sam_e70 - MCLV2**
1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_encoder_sam_e70\firmware\mclv2_sam_e70_pim.X
2. Clean and Build Project



3. Program the project



4. Press the switch S2 to start/stop the motor.
5. Rotate the potentiometer knob from mid point to extreme left to control the speed of the motor in counter-clockwise direction.
6. Rotate the potentiometer knob from mid point to extreme right to control the speed of the motor in clock-wise direction.

Project Name	Output
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 reset switch to stop the motor.

pmsm_foc_encoder_sam_e54

This motor control example project shows how to control the Permanent Magnet Synchronous Motor (PMSM) using an Encoder based Field Oriented Control (FOC) on a SAME54 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 the PMSM motor for applications which requires high dynamic performance.

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

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 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_e54

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

Project Name	Description
mclv2_sam_e54_pim.X	MCLV2 Development Board (Low Voltage)

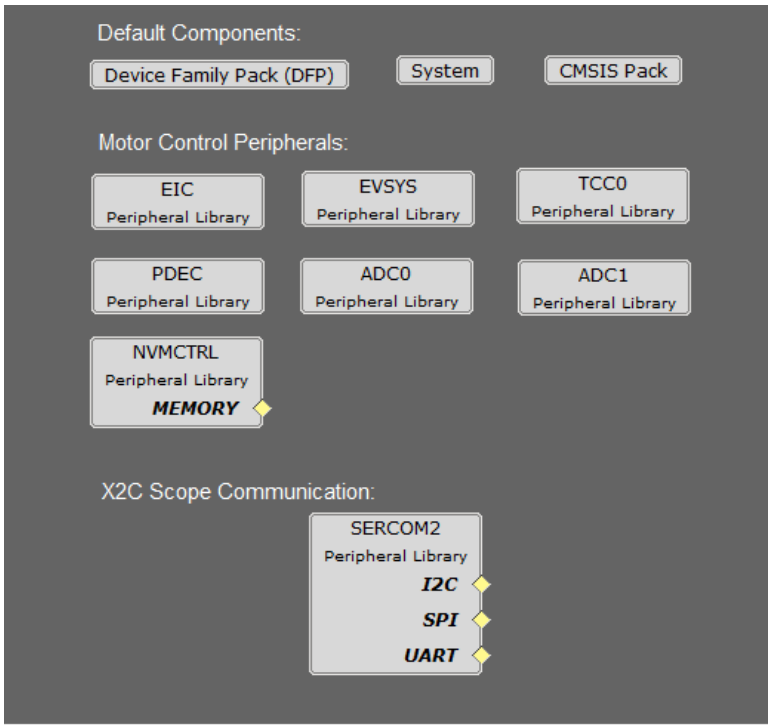
MPLAB Harmony Configurations

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_e54_pim.X



Components Used

TCC0:

- Configured to generate three pairs of complimentary PWM signals at a frequency of 20 kHz in "Dual Slope PWM with interrupt/event when counter = TOP" i.e. "Center Aligned Mode"
- In "Dual Slop PWM" mode, the PWM duty cyle has an inverse relation with the compare value. For example, for a Period Count (PER) = 3000, Compare Count (CCx) = 2000 (66%); would result in a PWM with a 33% duty cycle. In order to achieve a direct relation between compare count and duty cycle, the PWMxH and PWMxL outputs are swapped
- In dual shunt configuration, phase current measurements are performed at the end of the PWM cycle. Since the PWM outputs are swapped, the ADC conversion event is generated when the counter reaches TOP
- Dead-time is enabled and set to

- mclv2_sam_e54_pim.X - 1uS
- 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 the same resolution and sampling time, both ADCs complete conversion at the same time

PDEC:

- PDEC is configured to Quadrature decoder direction (X4) in QDEC mode to detect the quadrature encoder signals
- All the 16-bits of the QDEC counter are used to measure the angular position (CTRLA.ANGULAR = 0x7)

EIC:

- External Interrupt Controller detects a 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 an intermediary between event generator and event users
- Event generated by the TCC0 when the counter reaches TOP, is used by the ADC0 as a hardware trigger source via the 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

SERCOM2:

- SERCOM2 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 the X2CScope

Control Algorithm

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

Description

The project implements an encoder based Field Oriented Control (FOC) algorithm on the SAME54 32-bit micro-controller to regulate the speed of the PMSM motor. For details of encoder based FOC, refer to Application Note AN2757. The following section describes briefly about the FOC algorithm, software design and implementation.

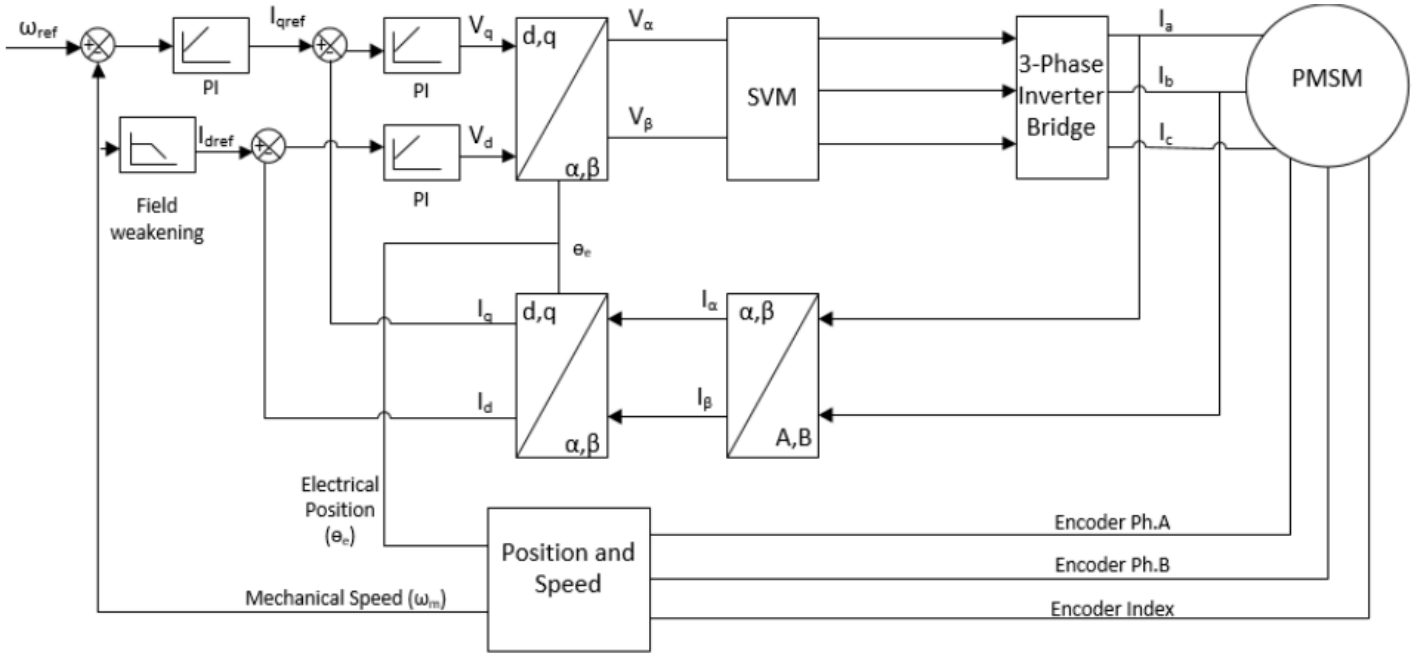
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from a stationary reference frame to torque and flux producing currents components in a rotating reference frame using mathematical 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 the Clarke transformation.
3. Calculate the rotor position angle.
4. Transform stator currents into the d,q-coordinate system using the 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 the Inverse Park transformation.
7. Using the space vector modulation, the three-phase output 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.

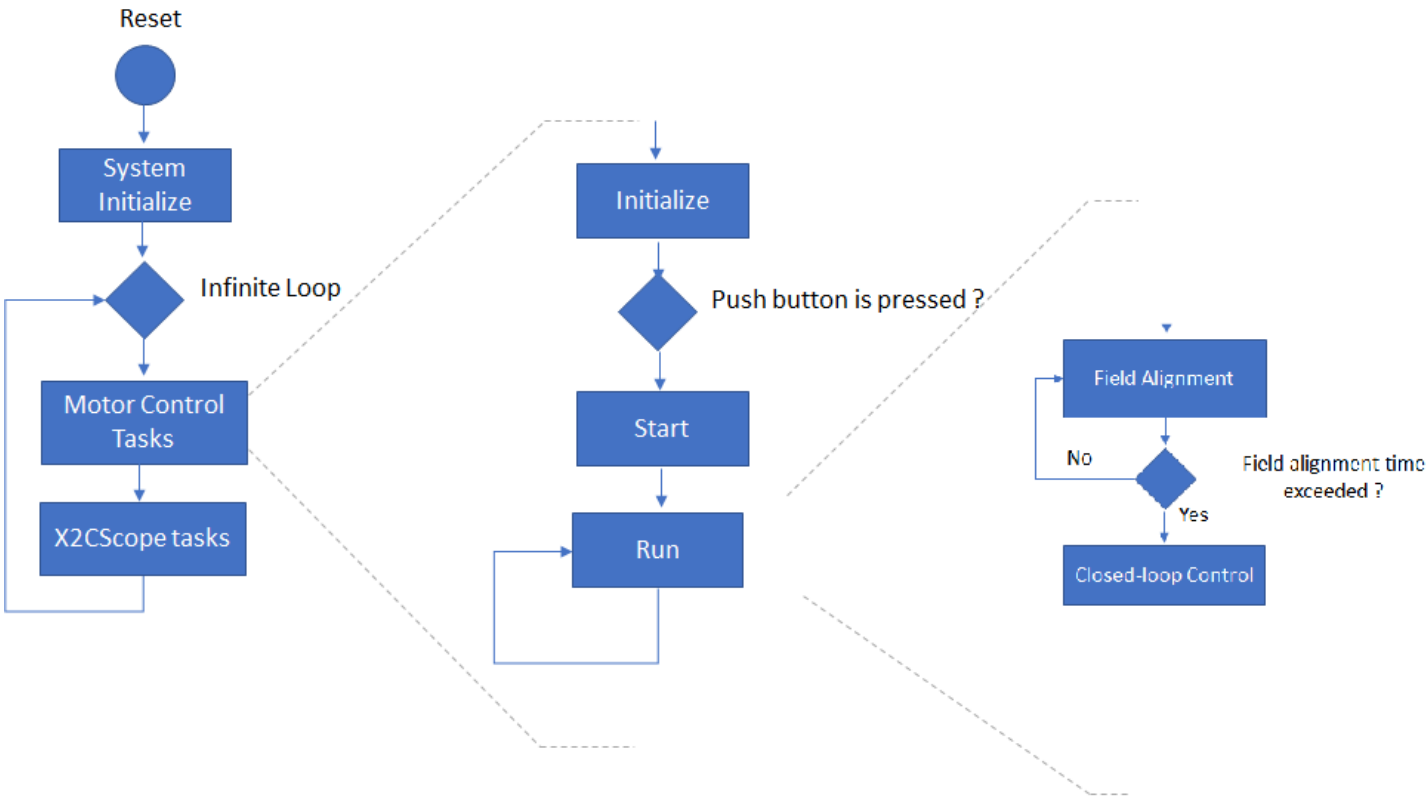
The rotor position angle is determined from the Quadrature encoder.

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 previous figure. The following sections briefly describes the various states in the PMSM speed control task:

1. Initialize:

In this state, following tasks are performed:

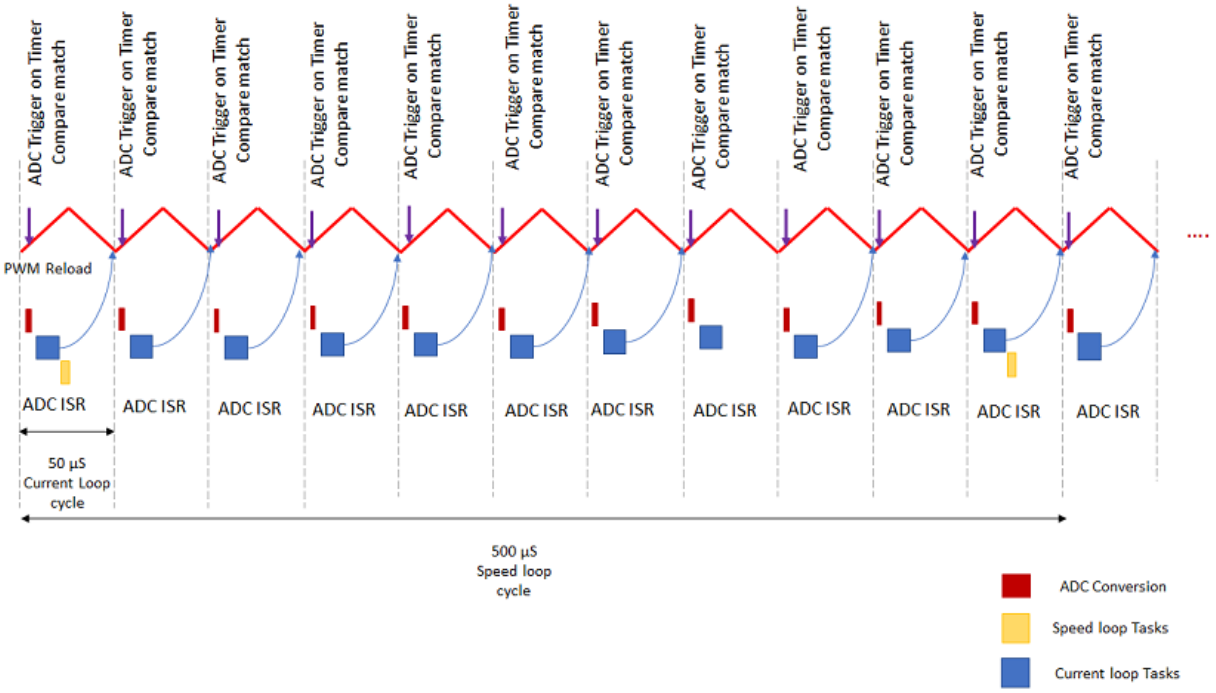
- Initialization and configuration of motor control peripherals for generation of periodic ADC triggers and ADC conversion interrupt
- Current Offset measurement and calibration
- Initialize PI controller parameters for speed and current control loops

2. Start:

In this state, the motor control state variables are reset and the 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; the 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. Therefore, the ADC interrupt cycle indicates the current control frequency.

- **Field Alignment:**

The quadrature encoder is an incremental encoder. It is important to know the initial position of the rotor to maintain orthogonal rotor and stator flux. To achieve this, the q-axis is excited with a small open loop current while the d-axis is excited with zero current.

In order to avoid pseudo-alignment, i.e. rotor alignment at 180 degrees out of phase, dual step alignment approach is used. The rotor is aligned to an intermediate angle before aligning it to the zero position.

After the initial alignment, an electrical angle is shifted by 90 degrees to obtain the maximum torque.

- **Close Loop Control:**

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

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 need 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 the following macro in `userparams.h` file.

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

Macro	Description
PWM_FREQUENCY	Current controller and PWM frequency in Hz.

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

Set the 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 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. Debugging Features:

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

Macro	Description
TORQUE_MODE	This is 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).

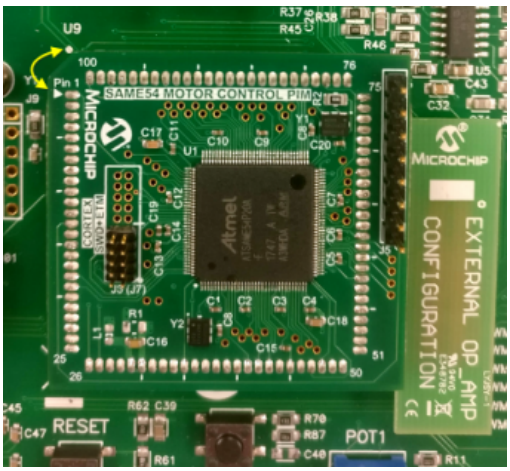
Hardware Setup

This section describes how to configure the supported hardware.

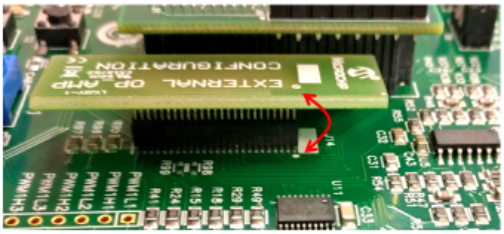
Description

2. **Project mclv2_sam_e54_pim.X.**

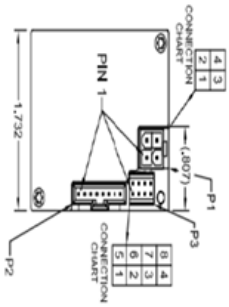
- **Hardware Used**
  - dsPICDEMTM MCLV2 Development Board (Low Voltage)
  - ATSAME54 Motor Control Plug In Module
  - Long Hurst DMA0204024B101 Motor
- **Hardware Setup**
  1. Mount the ATSAME54 Motor Control Plug In Module on U9 header.

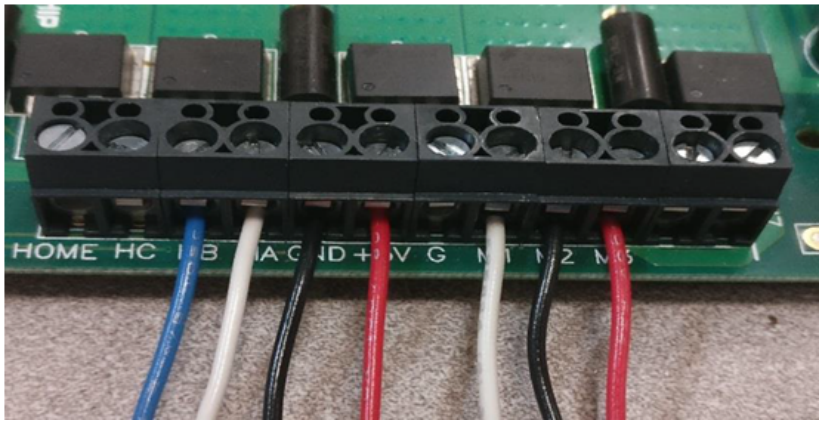


2. Place the "External Opamp Configuration" Matrix board at J14

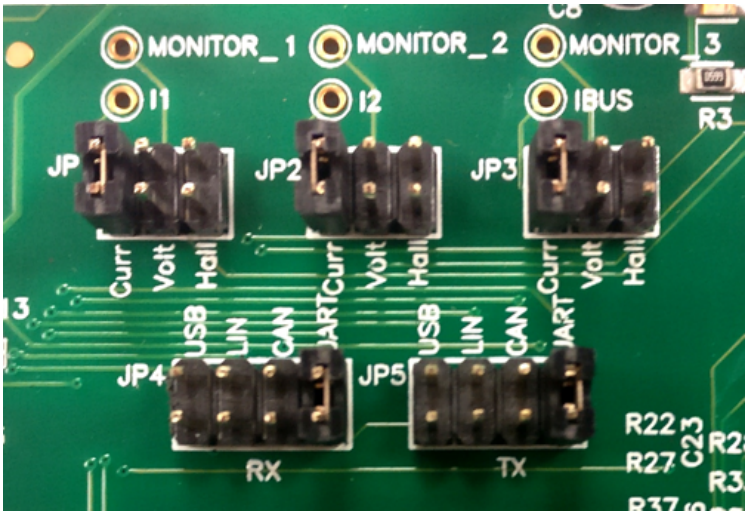


3. Connect motor and encoder terminals.

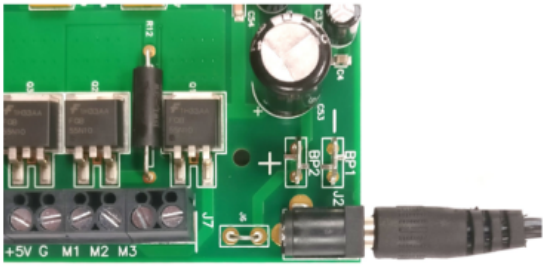




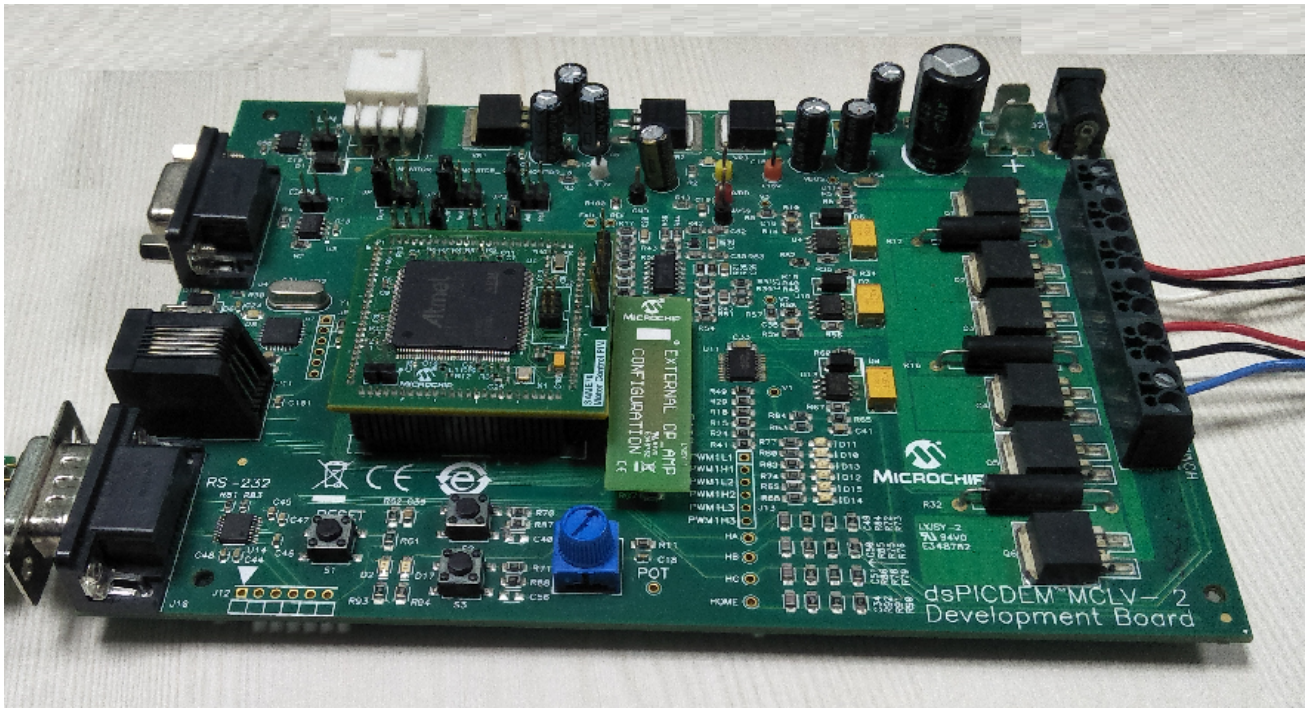
4. Jumper Settings:
   - JP1 - Curr, JP2 - Curr, JP3 - Curr
   - In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
   - In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**



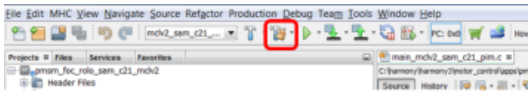
Running The Application

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

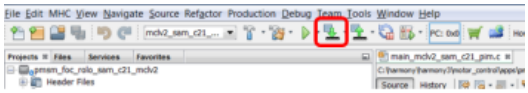
Description

Running the Application : pmsm_foc_encoder_sam_e54 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_encoder_sam_e54\firmware\mclv2_sam_e54_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.

Fault Indication and Recovery : pmsm_foc_encoder_sam_e54 - MCLV2

If LED D17 is ON, it indicates a non-recoverable fault caused by over-current condition. The MCU needs to be reset in order to recover from this fault mode.

Project Name	Output
`mclv2_sam_e54_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

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 demonstrates the Reduced Order Luenberger Observer (ROLO) based FOC algorithm on a SAMC21 micro-controller.

Description

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)

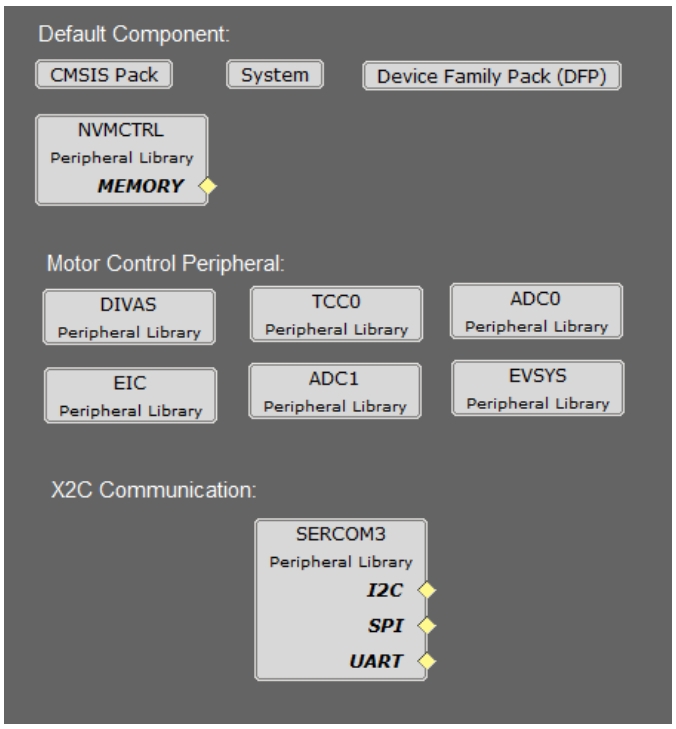
MPLAB Harmony Configurations

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 three pairs of complimentary PWM signals at a 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
  - mchv3_sam_c21_pim.X. - 2uS
  - mclv2_sam_c21_pim.X. - 1uS
- 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 a 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 an intermediary 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 the Event System.
- Event generated by the EIC upon over-current fault, is used by the TCC0 as a non-recoverable fault event via the 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 the 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 more details about 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 current components in rotating reference frame using mathematical transformations. The Field Oriented Control is implemented as follows:

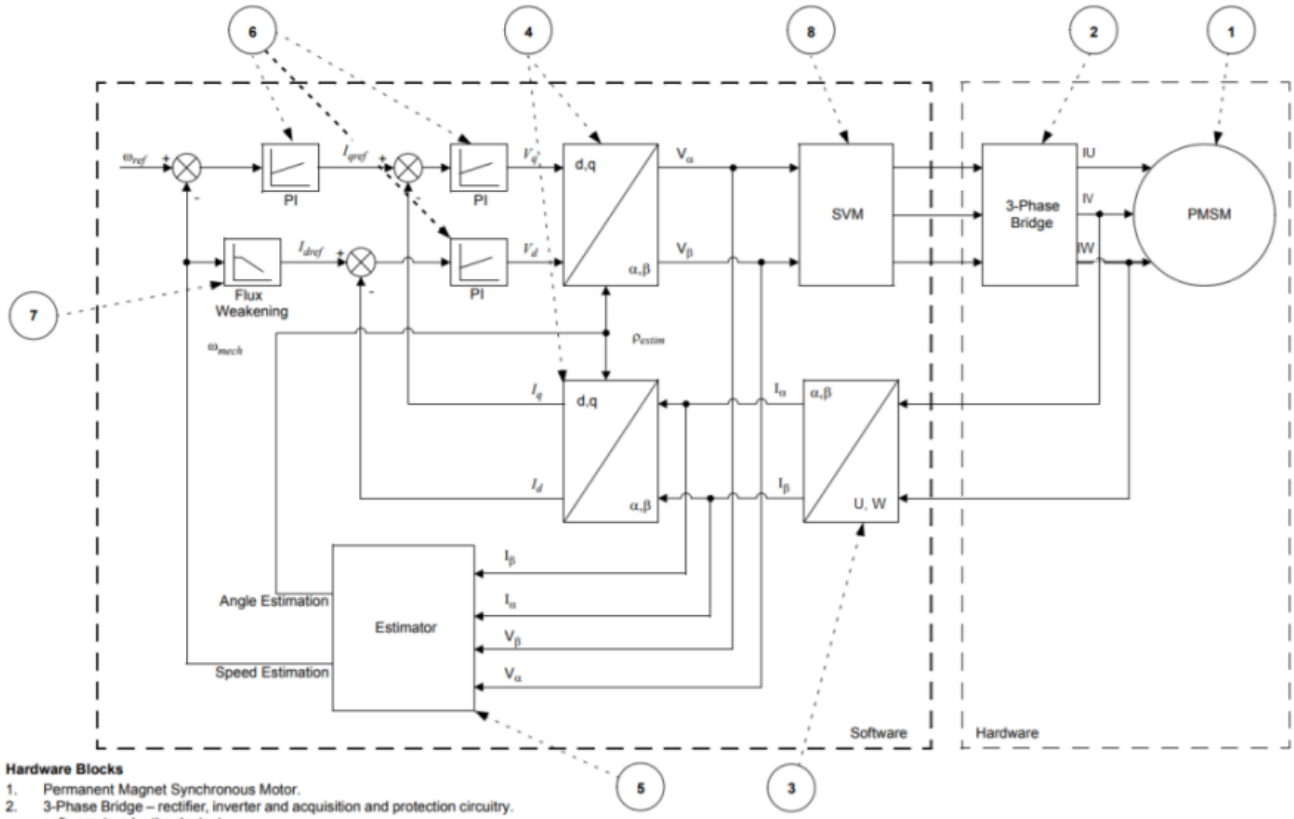
1. Measure the motor phase currents.
2. Transform the three phase currents from three phase stationary reference frame (u,v,w) into two phase stationary reference frame (alpha, beta) using the Clarke transformation.
3. Estimate (sensorless) or measure (sensored) the rotor position angle.
4. Transform 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 the 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.

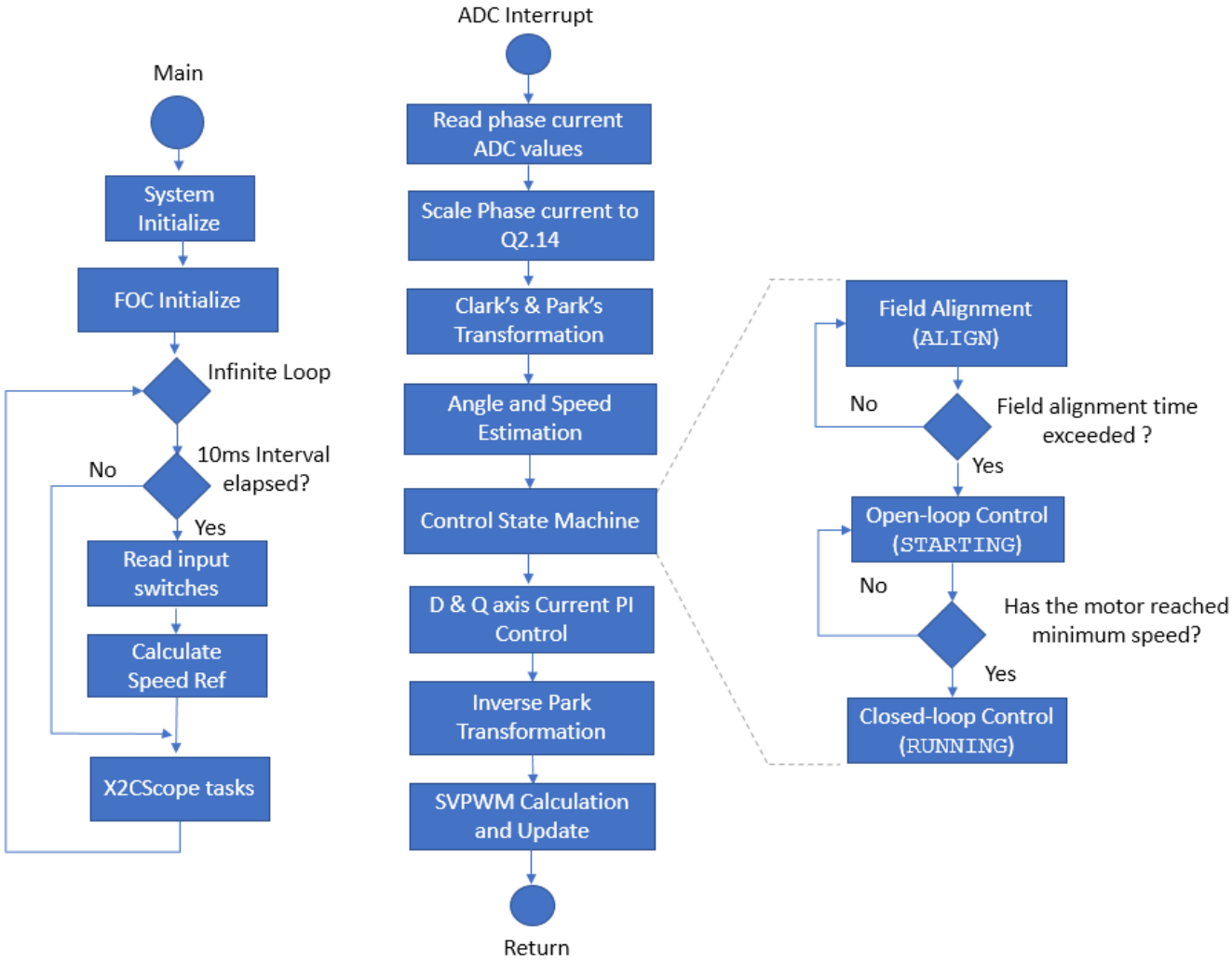


Hardware Blocks

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`:

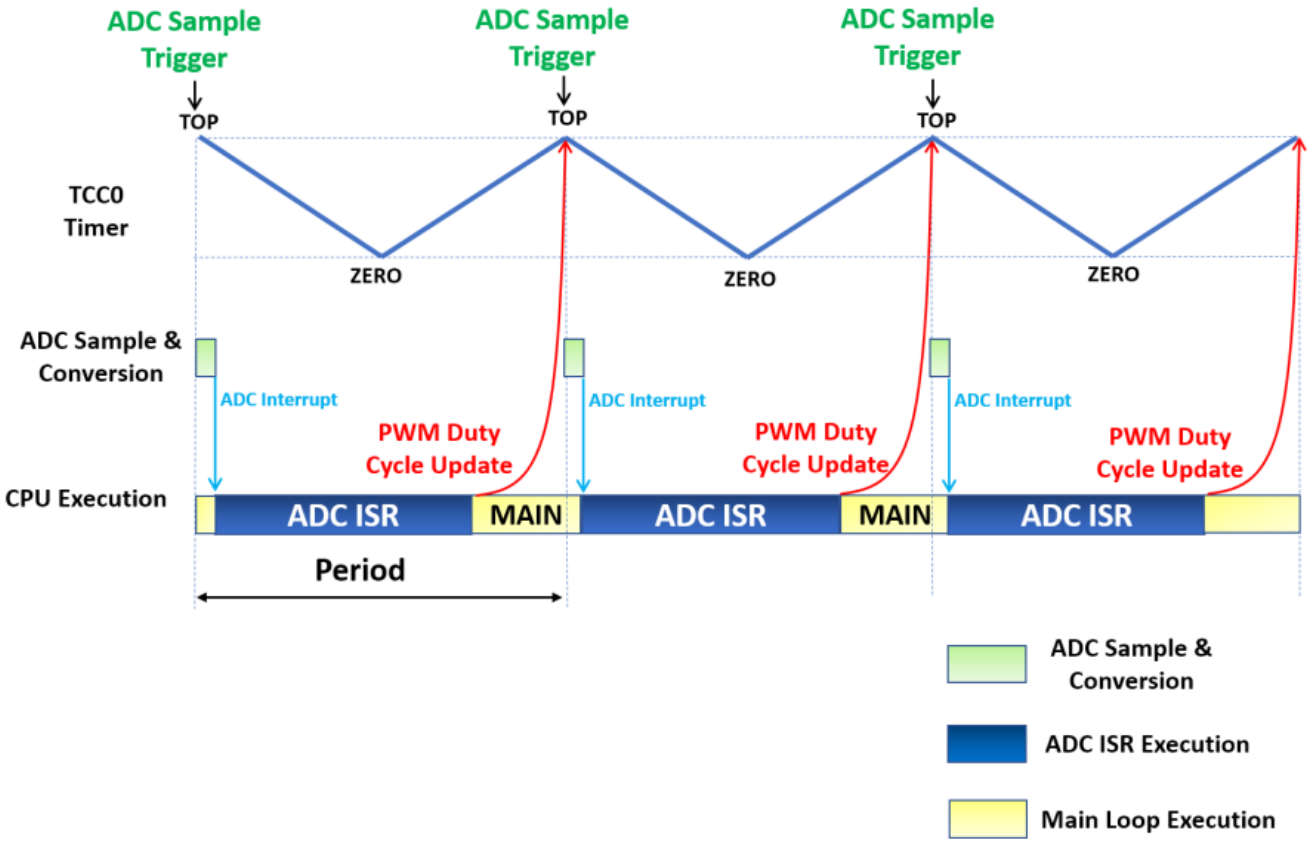
In this mode, the estimated rotor angle obtained from ROLO-based estimator is used to perform field oriented control of the PMSM motor.

Timing Diagram:

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

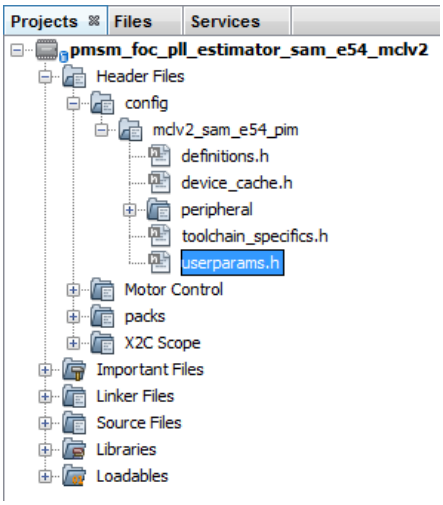
- ADC - It measures motor phase currents at the start of the PWM cycle.
- CPU - Based on motor phase currents measurements, the CPU executes the control algorithm to calculate the PWM duty cycle needed to apply required voltage across motor windings.
- TCC0 - It generates the PWM signals needed to apply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

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



Software Configuration:

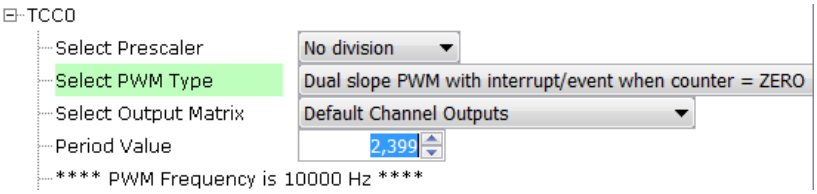
The FOC algorithm is used across different application fields. In order to get the optimal control of the PMSM motor, the motor specific parameters, board specific parameters and application parameter macros have to be updated in the software. The following section describes how to update both motor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

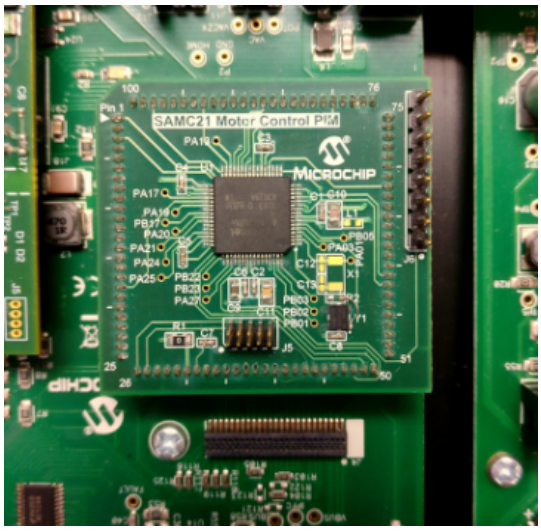
the rotor magnetic axis, resulting in lower torque generation.
  - This mode can be enabled by defining the macro `OPEN_LOOP_FUNCTIONING` in `userparams.h` header file
  - The torque current reference in this mode is defined by the macro `START_CUR_AMP`
  - Depending upon the load inertia and startup acceleration time (`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.
  - Instruction breakpoint will not be asserted on an instruction which is being executed from RAM. It is recommended to disable this macro while debugging this algorithm using instruction breakpoints.
  - 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

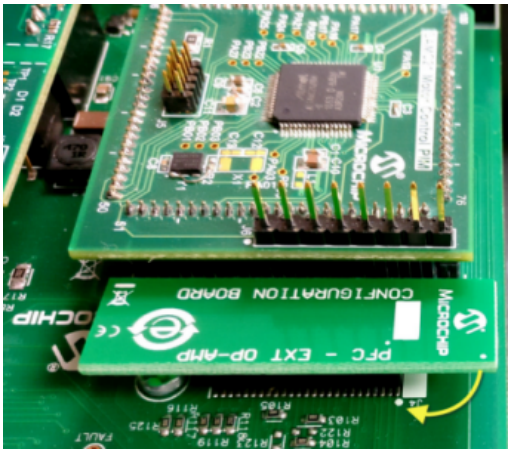
This section describes how to configure the supported hardware.

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.

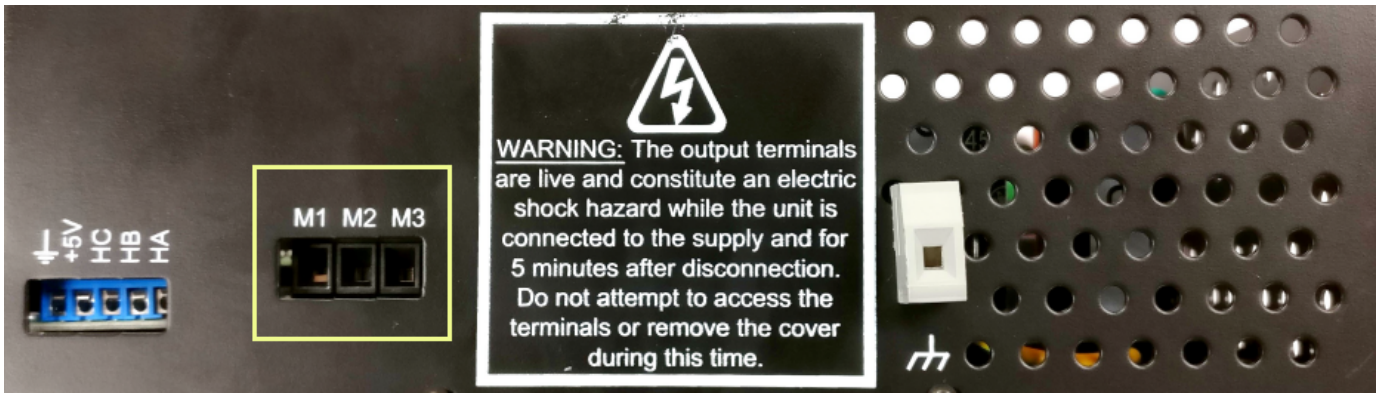


2. Place the "PFC - External Opamp Configuration" Matrix board at J4.



3. Motor Connections:

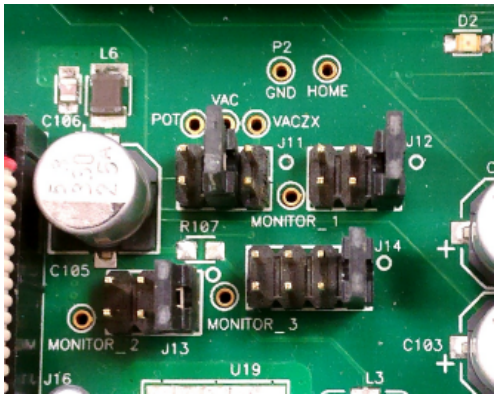
- Phase U - M1
- Phase V - M2
- Phase W - M3



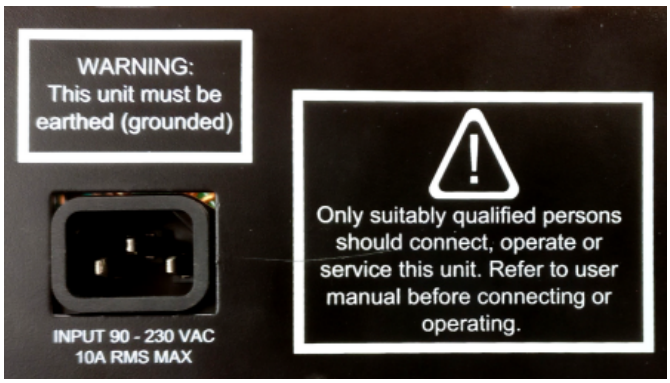
4. Jumper Settings:

- J11 - VAC (Short Pin 3 - 4)
- J12 - IA (Short Pin 1 - 2)

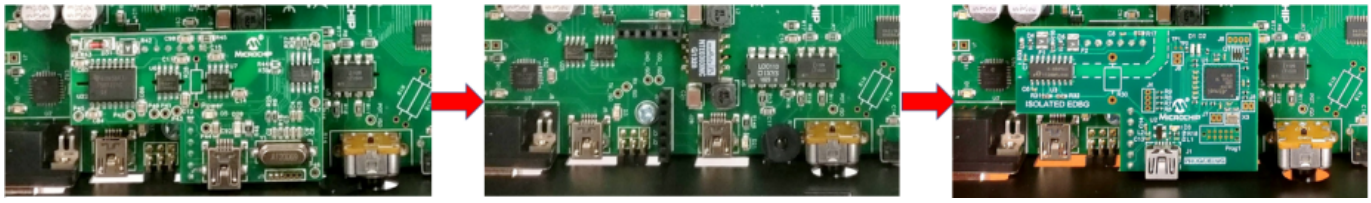
- J13 - IB (Short Pin 1 - 2)
- J14 - Fault_IP/IBUS (Short Pin 1 - 2)



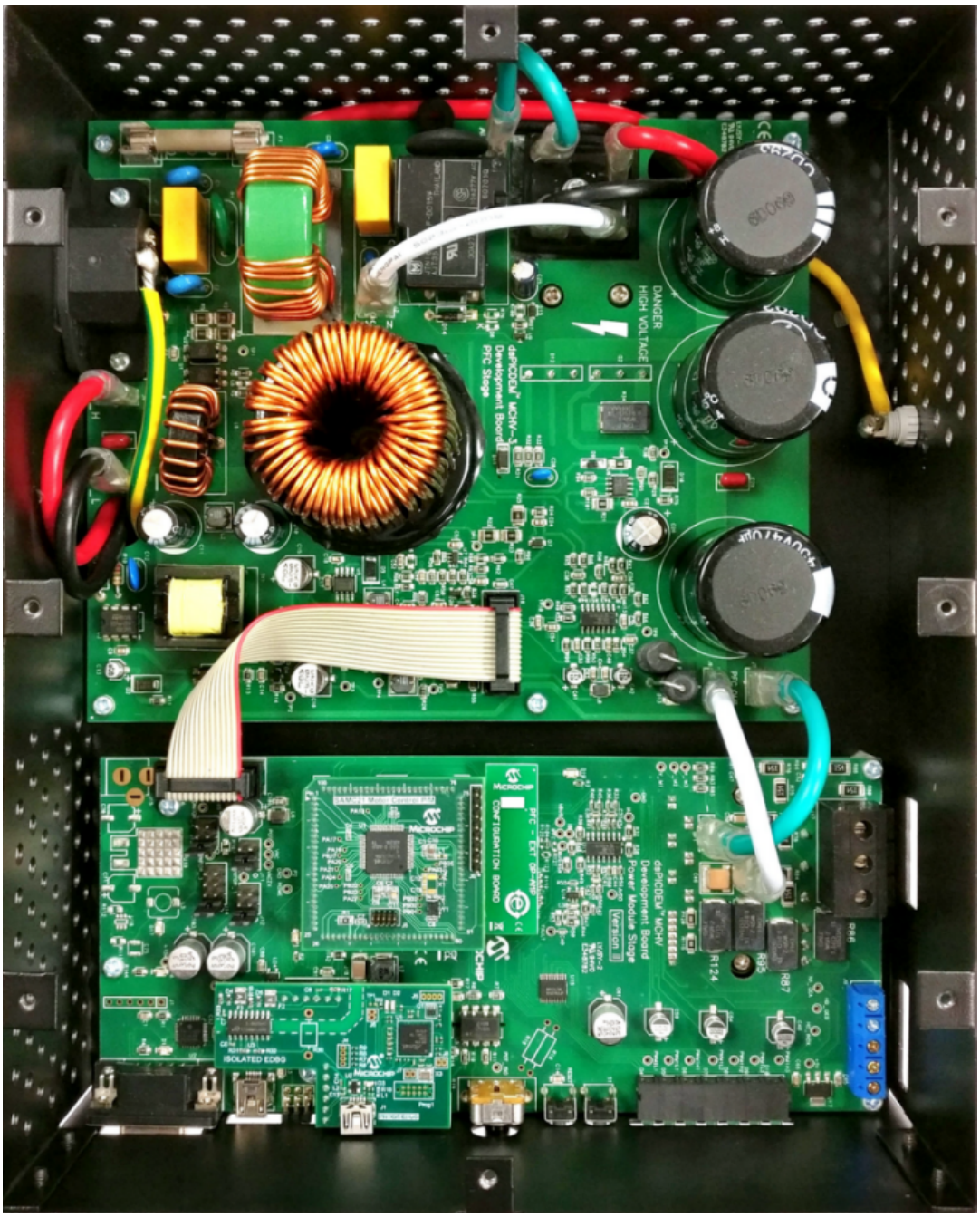
5. Power the board with (110V/220V) AC mains. For additional safety, it is recommended to use a current limited power supply while testing this software demonstration on a non-default hardware and motor.



- **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.

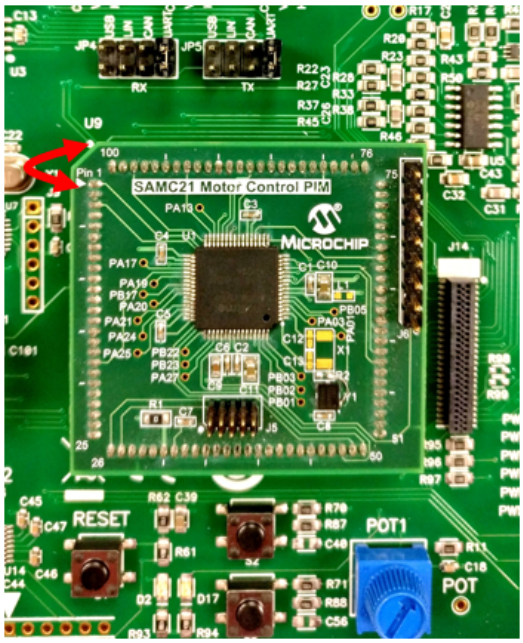


- Complete Setup

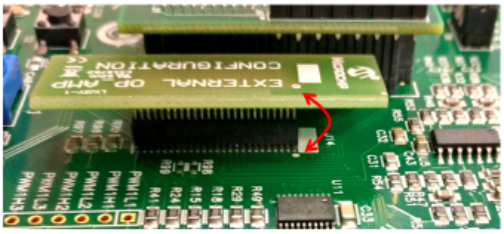


2. **Project mclv2_sam_c21_pim.X.**

- **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.

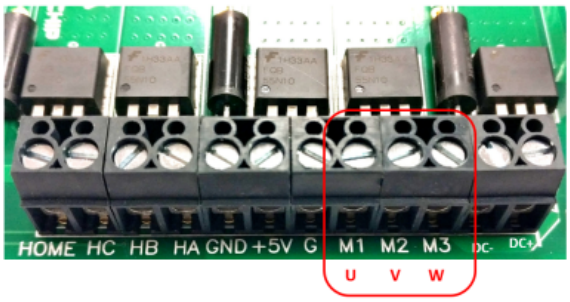


2. Place the "External Opamp Configuration" Matrix board at J14.



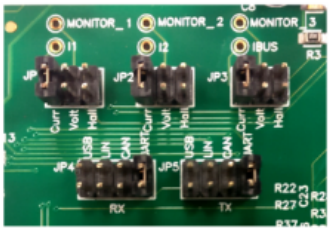
3. Motor Connections:

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

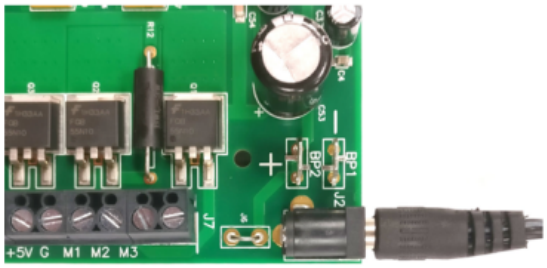


4. Jumper Settings:

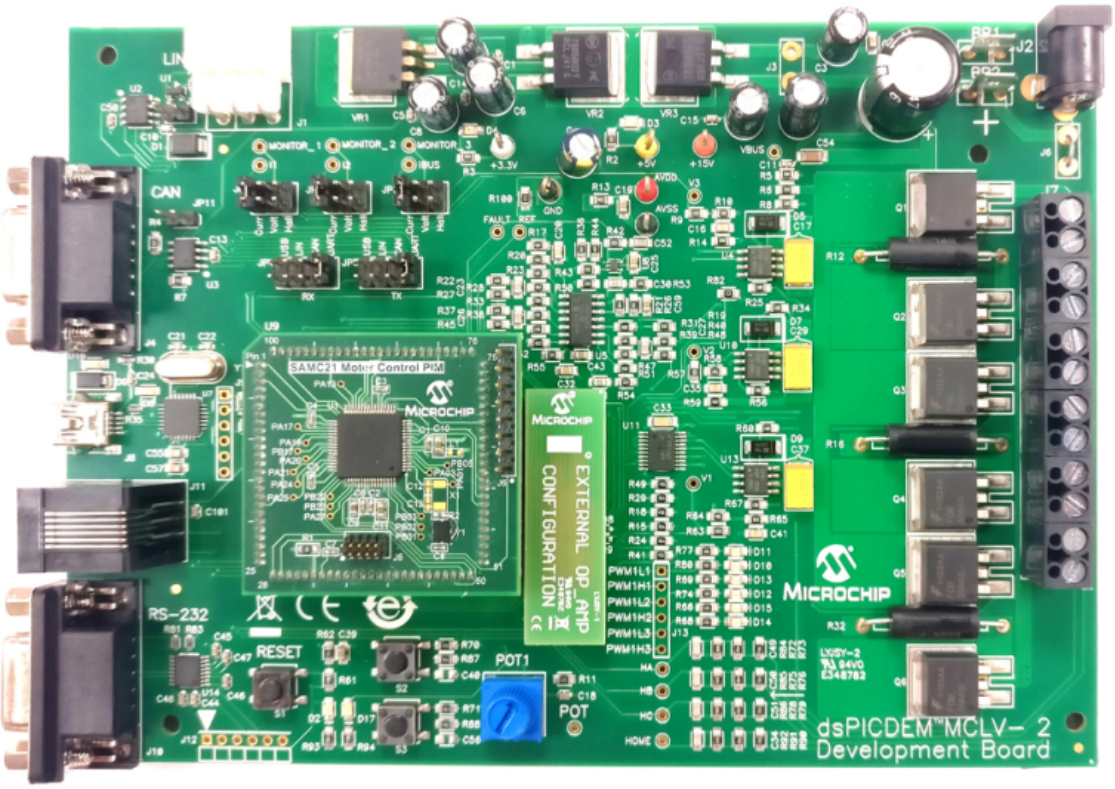
- JP1 - Curr, JP2 - Curr, JP3 - Curr
- In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**



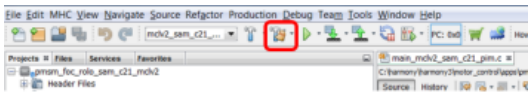
Running The Application

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

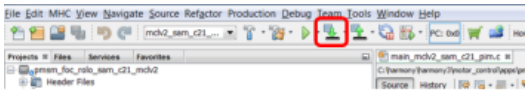
Description

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:



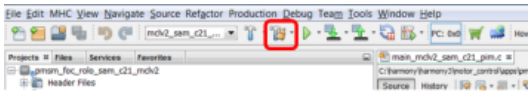
4. Press the switch "Push Button" to start/stop the motor.
5. Vary Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_rolo_wm_sam_c21 - MCHV3

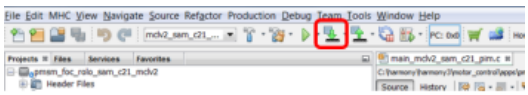
If LED D19 is ON, it indicates a non-recoverable fault caused by either over-current, phase loss or due to erroneous rotor position estimation. The MCU needs to be reset in order to recover from this fault mode.

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 demonstrates the windmilling feature with Reduced Order Luenberger Observer (ROLO) based FOC algorithm on a SAMC21 micro-controller.

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). 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)

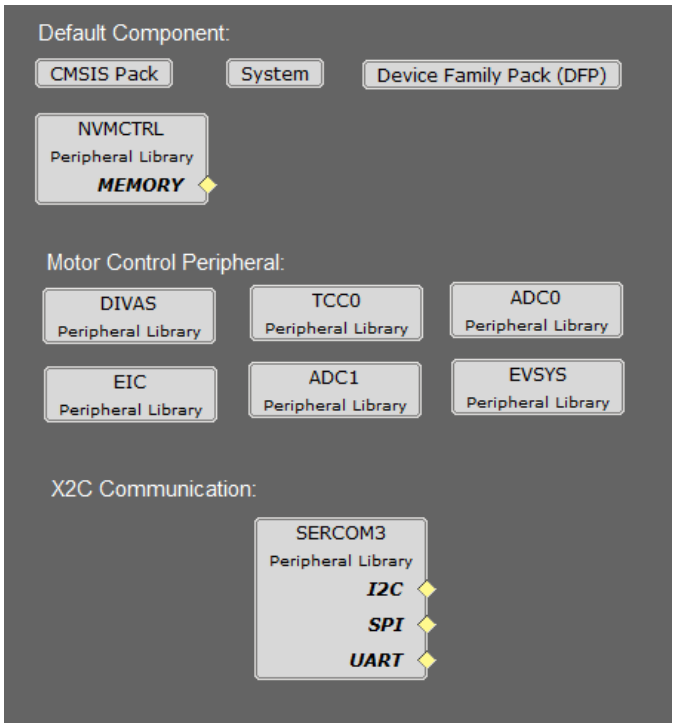
MPLAB Harmony Configurations

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 three pairs of complimentary PWM signals at the 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
  - mchv3_sam_c21_pim.X. - 2uS
  - mclv2_sam_c21_pim.X. - 1uS
- 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 a 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 an intermediary 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 the Event System.
- Event generated by the EIC upon over-current fault, is used by the TCC0 as a non-recoverable fault event via the 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 the 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 current 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 the 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.

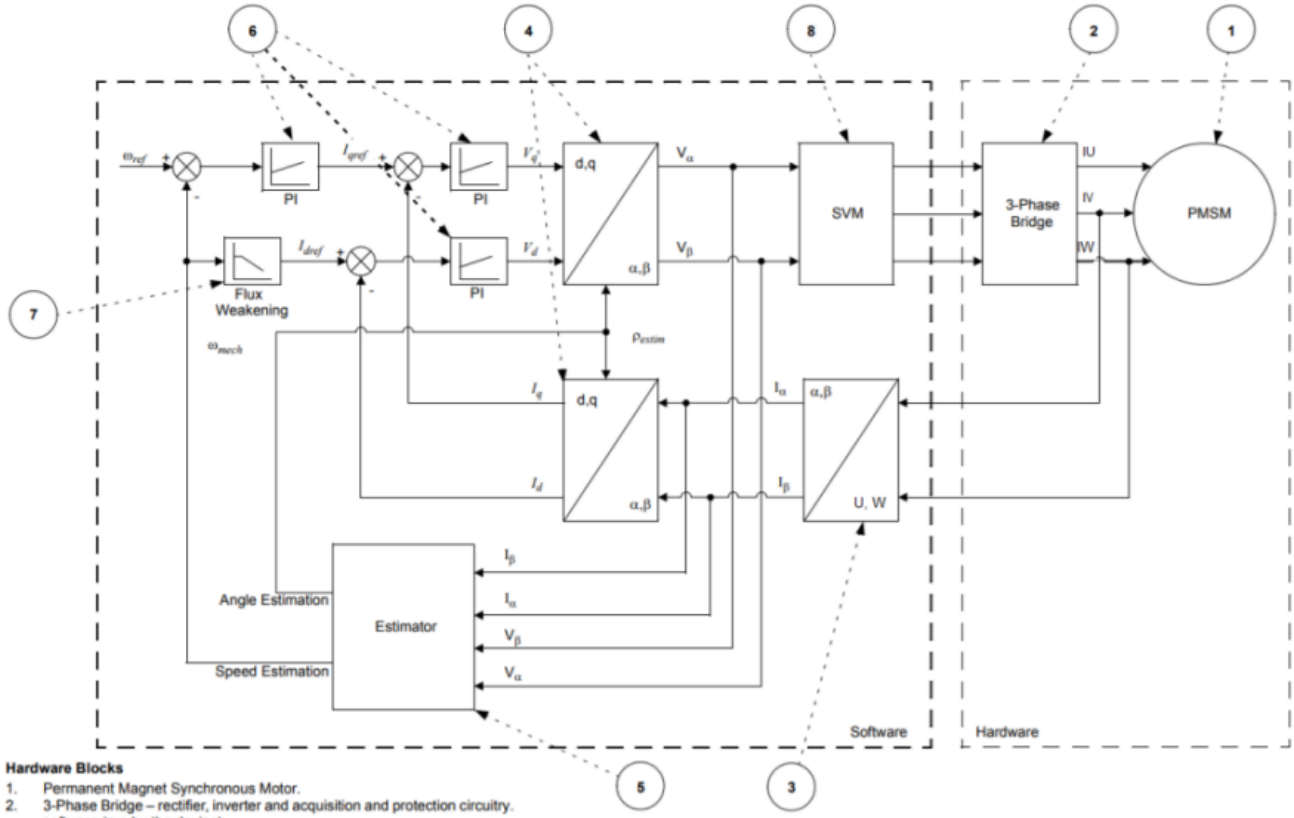
The phase currents are measured using two shunt resistors connected to Phase U and Phase V inverter legs respectively. The W phase current is determined based on Kirchoff's current law.

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.

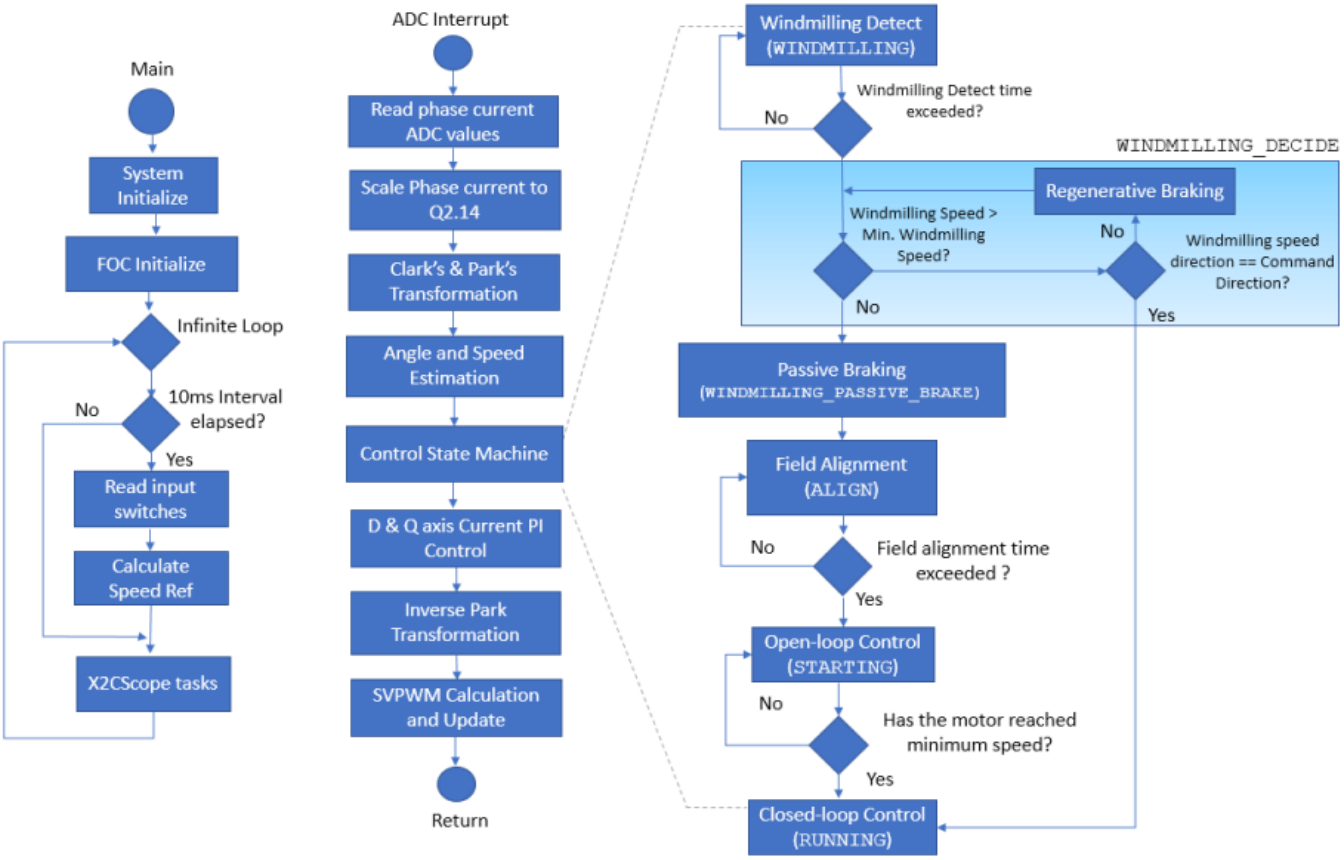


Hardware Blocks

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`:

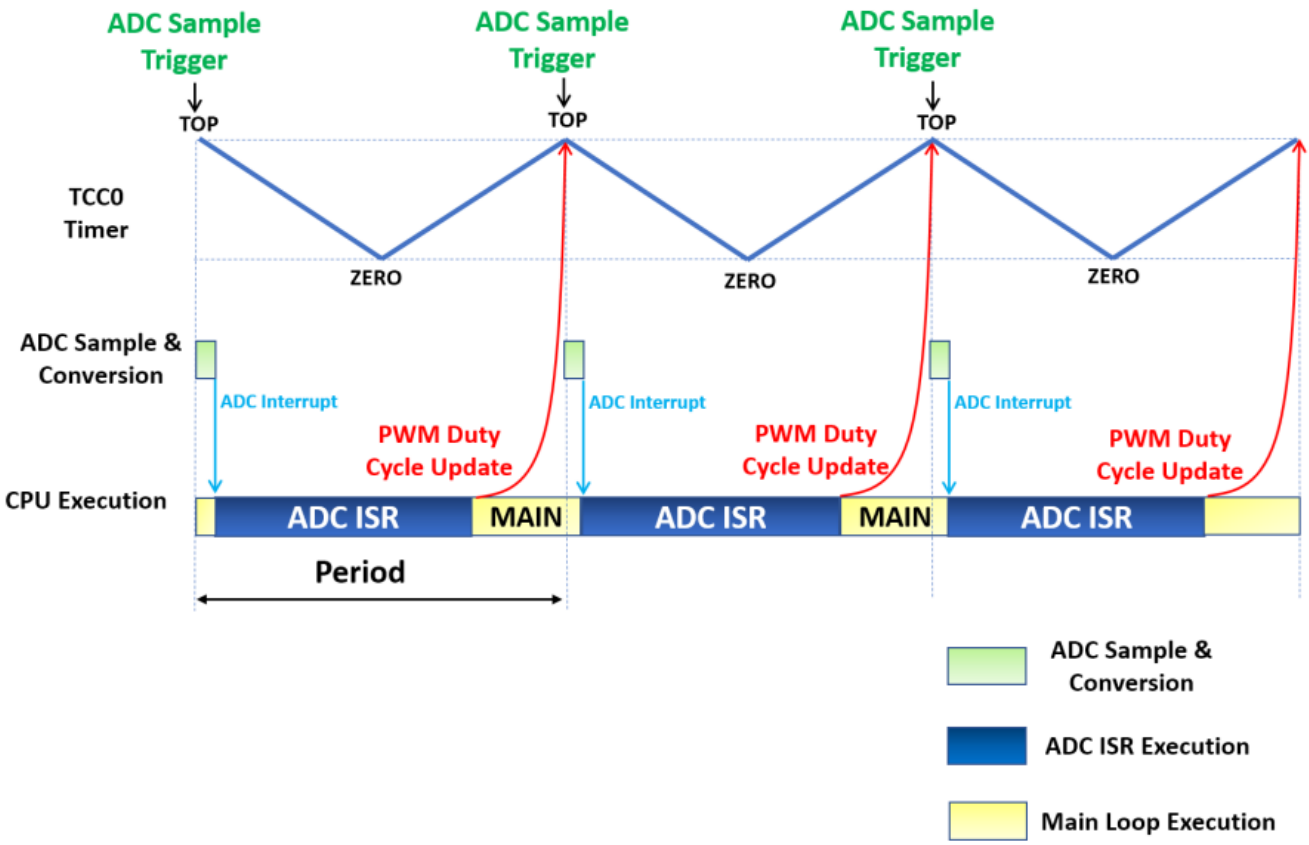
In this mode, the estimated rotor angle obtained from ROLO-based estimator is used to perform field oriented control of the PMSM motor.

Timing Diagram:

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

- ADC - It measures motor phase currents at the start of the PWM cycle.
- CPU - Based on motor phase currents measurements, the 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.

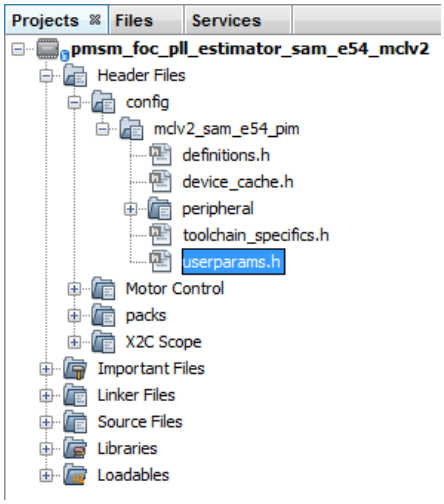
The following timing diagram, shows the chronological role played by the ADC, CPU and TCC0 in the execution of FOC algorithm.



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 >*

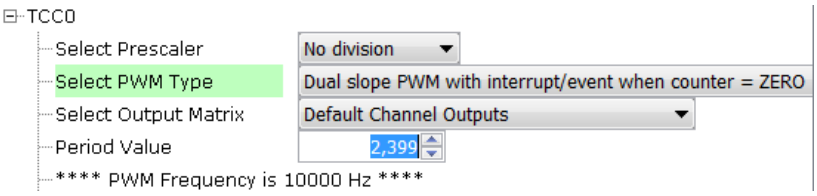
config > *<config_name>* >*userparams.h* in MPLABX IDE.



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
  - Instruction breakpoint will not be asserted on an instruction which is being executed from RAM. It is recommended to

disable this macro while debugging this algorithm using instruction breakpoints.

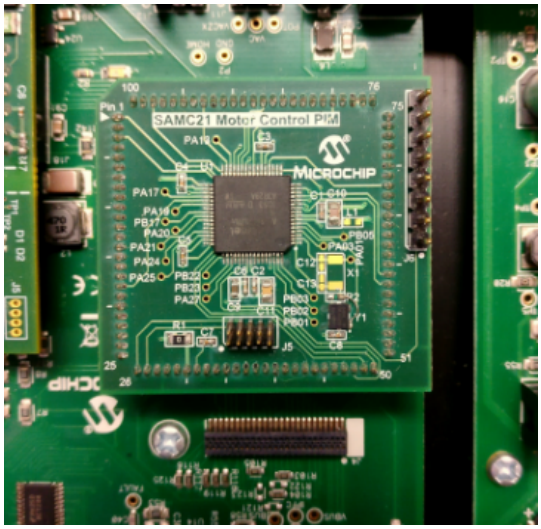
- Exercise caution while using this feature as the functions in RAM are vulnerable to get overwritten due to stack overruns or unguarded buffer overruns

Hardware Setup

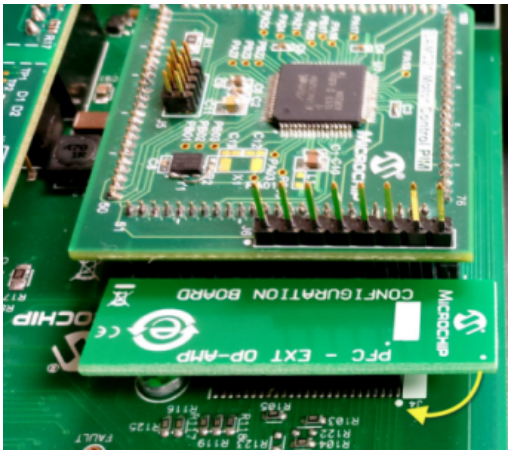
This section describes how to configure the supported hardware.

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.

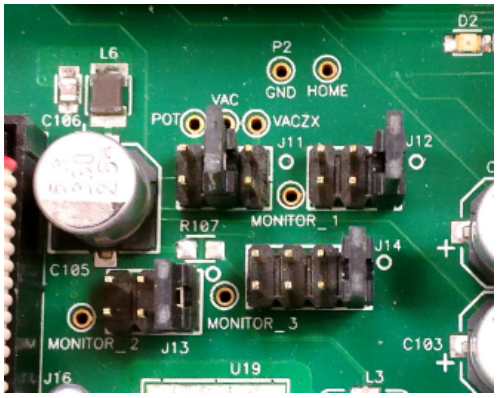


3. Motor Connections:

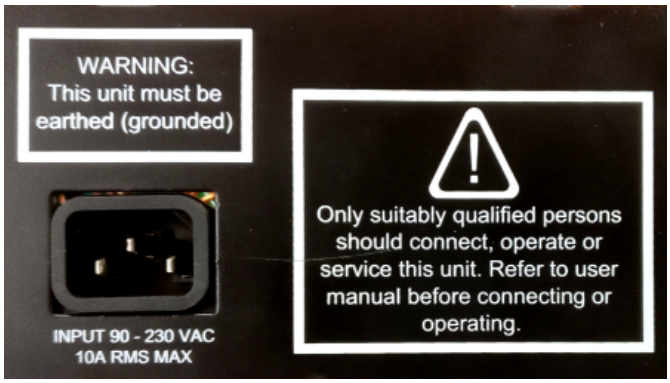
- Phase U - M1
- Phase V - M2
- Phase W - M3

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)

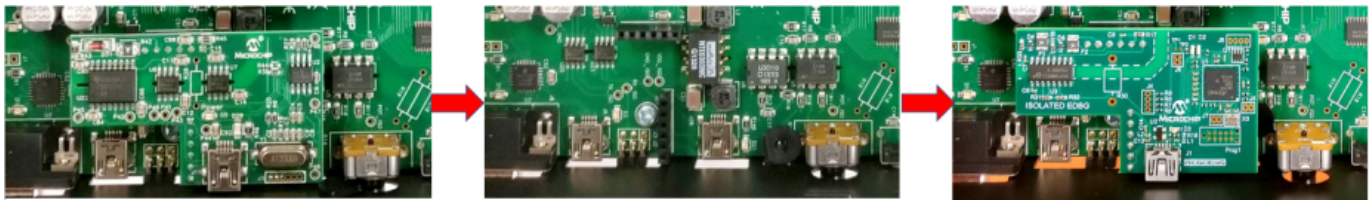


5. Power the board with (110V/220V) AC mains. For additional safety, it is recommended to use a current limited power supply while testing this demo on a non-default hardware and motor.

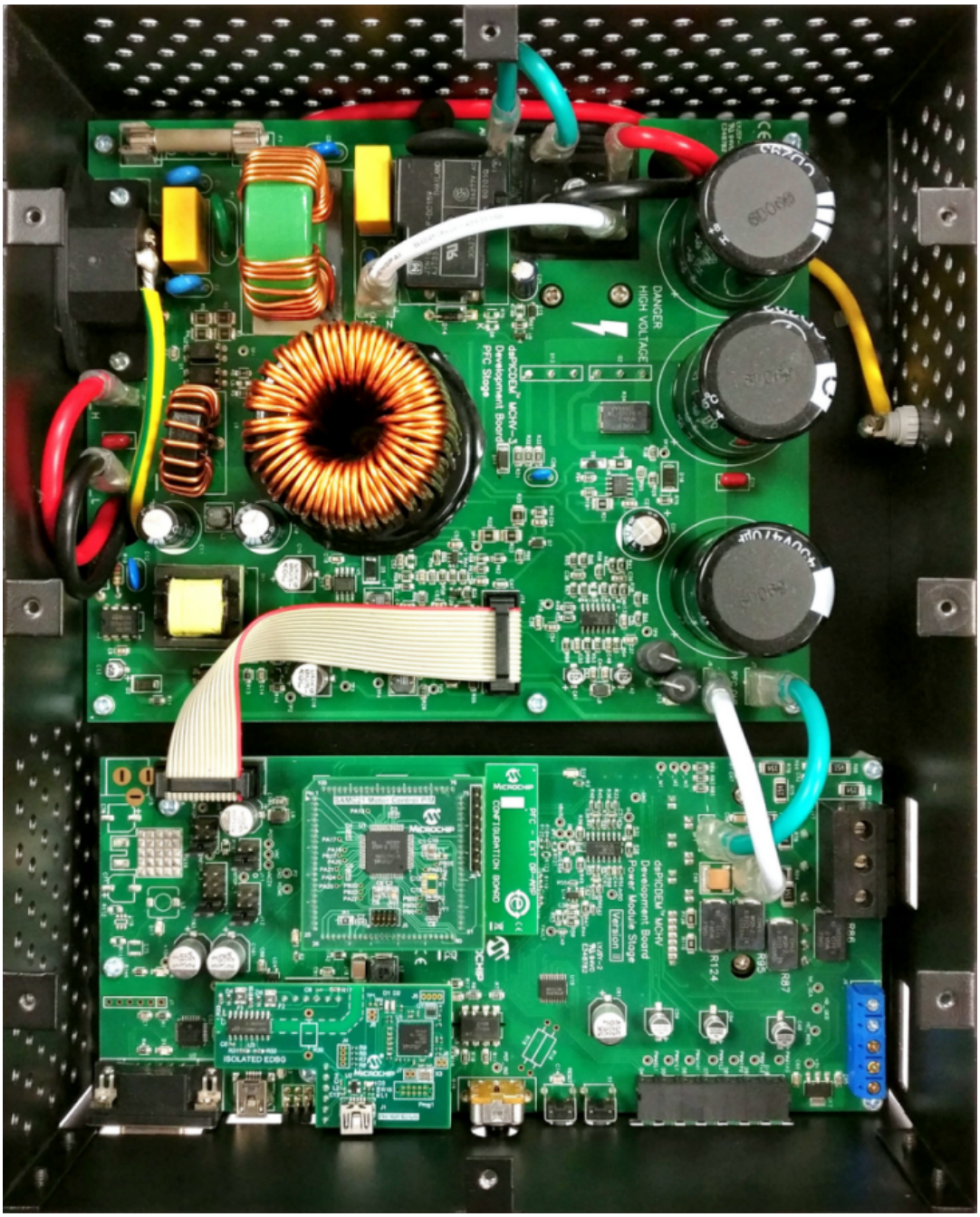


- Installing Isolated Embedded Debugger

- 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.



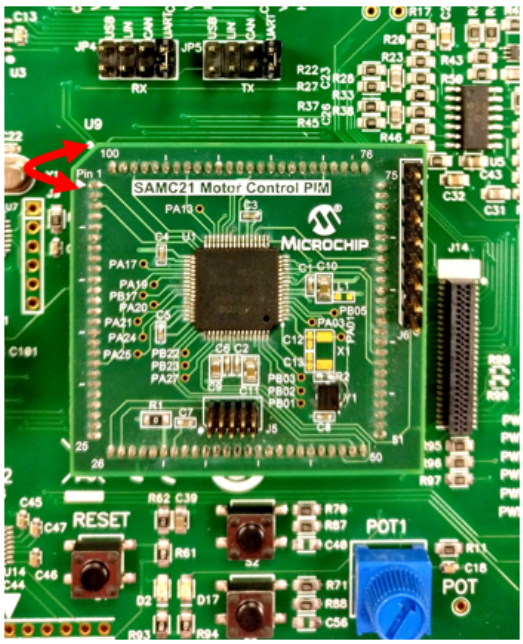
- Complete Setup



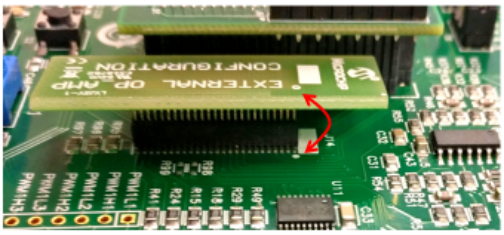
2. **Project mclv2_sam_c21_pim.X.**

- **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.

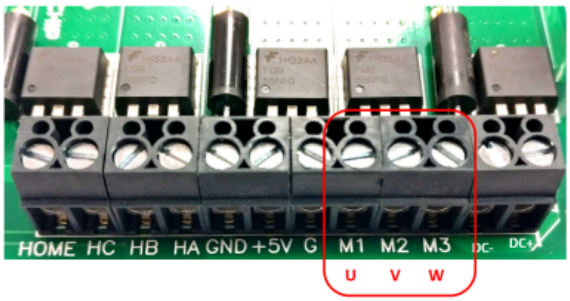


2. Place the "External Opamp Configuration" Matrix board at J14.



3. Motor Connections:

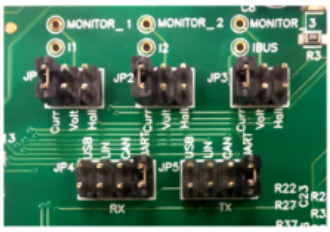
- Phase U - M1
- Phase V - M2
- Phase W - M3



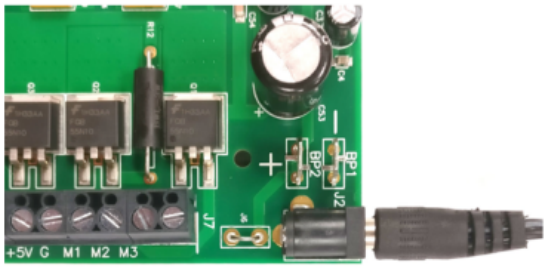
4. Jumper Settings:

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

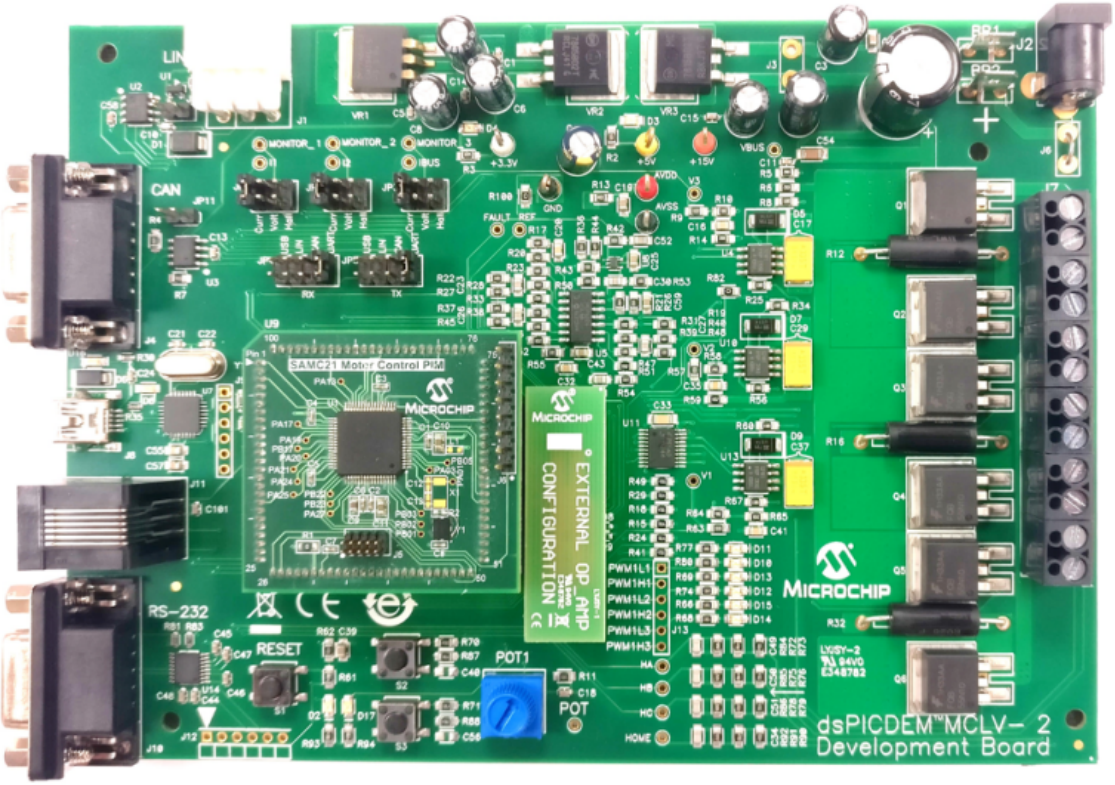
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**



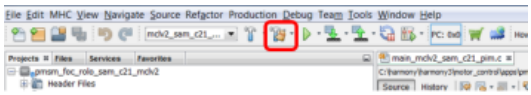
Running The Application

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

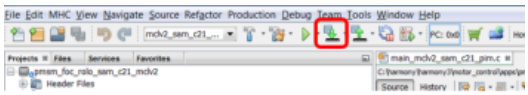
Description

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:



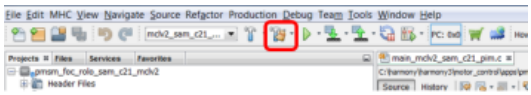
4. Press the switch "Push Button" to start or stop the motor.
5. Vary the Potentiometer to change the speed of the motor.

Fault Indication and Recovery : pmsm_foc_rolo_wm_sam_c21 - MCHV3

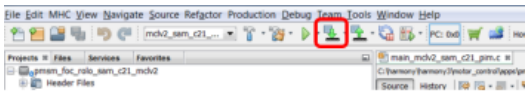
If LED D19 is ON, it indicates a non-recoverable fault caused by either over-current, phase loss or due to erroneous rotor position estimation. The MCU needs to be reset in order to recover from this fault mode.

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.

PFC in PLL based PMSM drive

This Section provides help for the Power Factor Correction application.

pmsm_pfc_foc_pll_estimator_sam_e70

This document is to guide the user to configure, build and run the power factor correction in Microchip's 32-bit SAME70 microcontroller in MCHV3 development board.

Description

Power Factor Correction (PFC) shapes the input current drawn from the AC mains to be in synchronization with the input AC voltage. This project uses Boost-Converter topology to carry out the Power Factor Correction using Microchip's 32-bit SAME70 microcontroller in MCHV3.

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 the following table and open the appropriate project file in MPLAB X IDE.

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

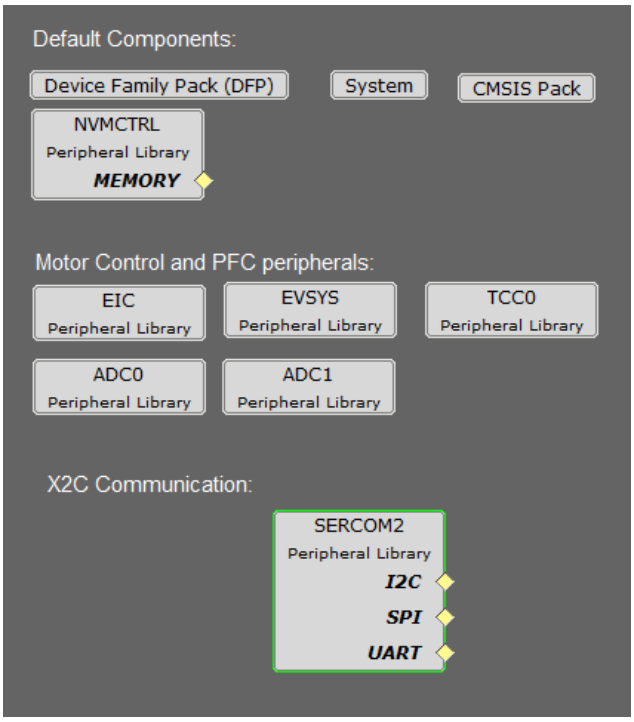
MPLAB Harmony Configurations

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



Components Used

1. AFEC Peripheral:
   - The AFEC0 peripheral is used to measure analog quantities for motor control operation. 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)
   - The AFEC1 peripheral is used to measure analog quantities for power factor correction. Three channels are used to measure the input rectified voltage, input rectified current, and the output DC link inverter voltage. Conversion is triggered at the PWM period match

2. PWM Peripheral:
   - The PWM0 peripheral is used to generated three phase synchronous PWM waveforms. Fault functionality is also enabled to switch off the output waveforms asynchronously
   - The PWM1 peripheral is used to generate the PWM waveform for the boost converter switch for current waveform shaping

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

Control Algorithm

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

Description

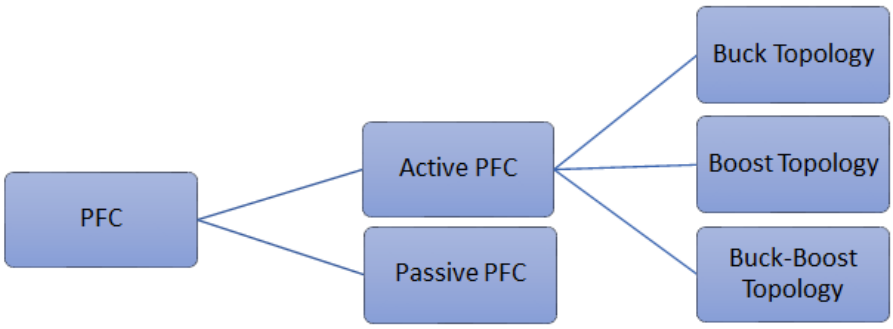
This project implements a active Power Factor Correction (PFC) control algorithm on SAME70 32-bit micro-controller. The following section introduces briefly to PFC theory, PFC algorithm, software design and implementation.

The Power Factor Correction is a technique of increasing the power factor of the input AC power supply, thereby providing

following advantages:

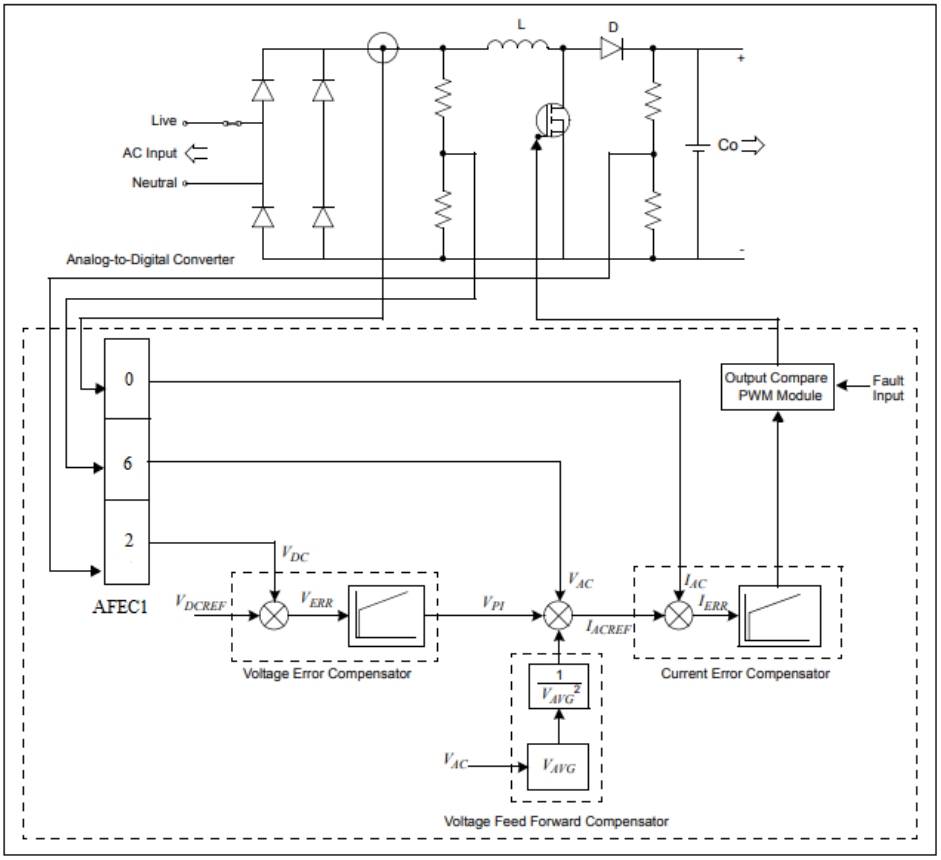
1. Low line harmonics.
2. Reduced input current.
3. Reduced kVA requirement.
4. Improved line efficiency, and improved electromagnetic compatibility.

The Power Factor Correction techniques can be classified as - Passive and Active. The Passive Power Factor Correction uses passive components, while the active Power Factor Correction Technique uses power electronic switches to carry out Power Factor Correction. Following figure shows the various Power Factor Correction techniques classification:



This project uses Boost topology for Power Factor Correction. A boost converter is placed between the rectifier and output inverter stage to synchronize input AC current to the input AC voltage. For details, refer to application note AN1106. The block diagram of a typical boost converter topology based Power Factor Correction is shown as follows:

Block Diagram:



The power factor correction is implemented in following steps:

1. Measure the rectifier stage rectified AC voltage and rectified AC current, and output inverter stage DC bus voltage via high speed ADC channels.
2. Regulate the DC bus voltage using the Voltage Error Compensator.
3. Determine the reference input for Current Error Compensator based on the output of Voltage Error Compensator and Voltage Feed-Forward Compensator.
4. Track the rectified reference AC current using a Current Error Compensator.

Current Error Compensator:

The inner loop in the control block forms the current loop. The input to the current loop is the reference current signal I_{ACREF} and the actual inductor current I_{AC}. The current error compensator is designed to produce a control output such that the inductor current I_{AC} follows the reference current I_{ACREF}. The current loop should run at a much faster rate when compared to the voltage loop. The bandwidth of the current compensator should be higher for correctly tracking the semi-sinusoidal waveform at twice the input frequency. The current controller GI produces a duty cycle value after appropriate scaling to drive the gate of the boost converter MOSFET.

Voltage Error Compensator:

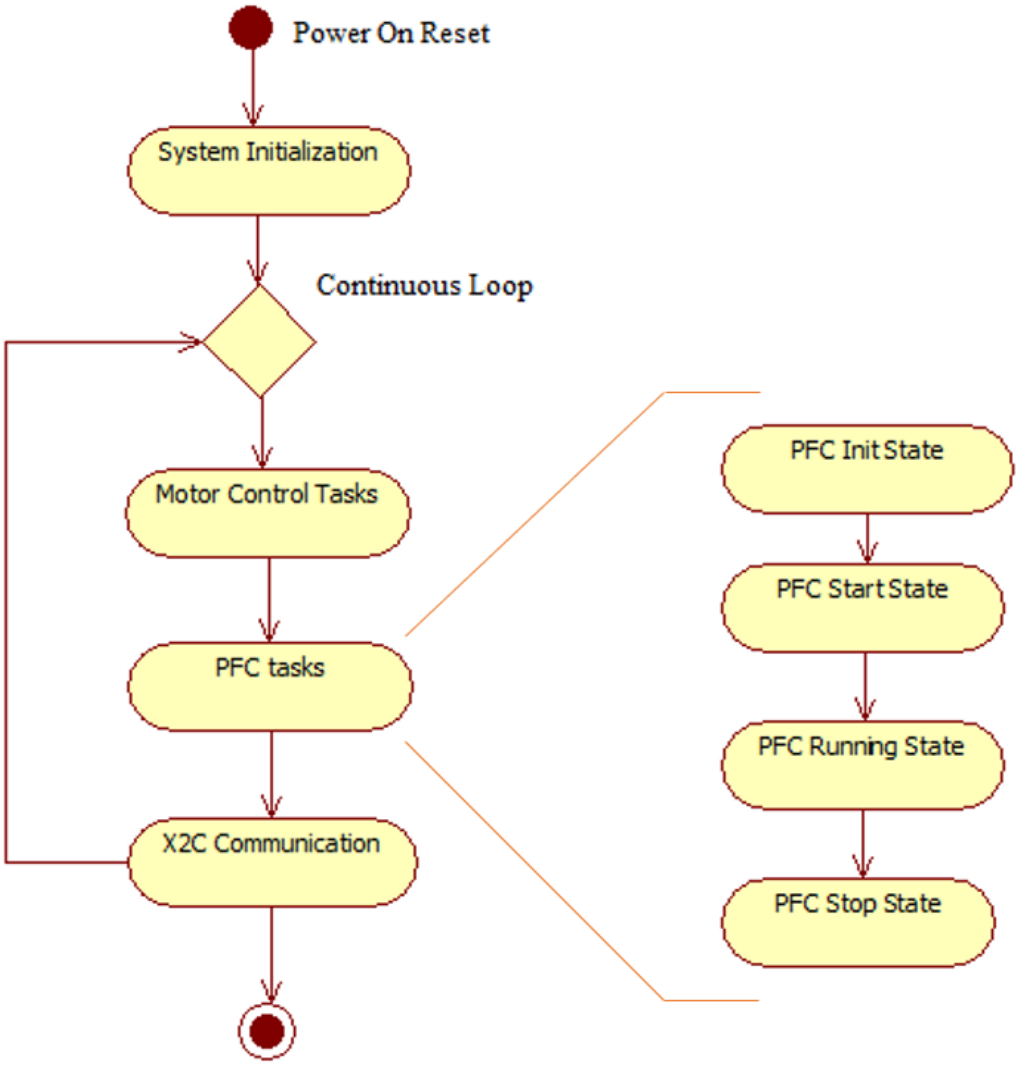
The outer loop in the control block forms the voltage loop. The input to the voltage loop is the reference DC voltage V_{DCREF} and the actual sensed output DC voltage V_{DC}. The voltage error compensator is designed to produce a control output such that the DC bus voltage V_{DC} remains constant at the reference value V_{DCREF} regardless of variations in the load current I_O and the supply voltage V_{AC}. The voltage controller GV produces a control signal, which determines the reference current I_{ACREF} for the inner current loop. The output voltage is controlled by the voltage error compensator. When the input voltage increases, the product of V_{AC} and V_{PI} increases, and thereby increasing the programming signal. When this signal is divided by the square of the average voltage signal, it results in the current reference signal being reduced proportionally. The outcome is that the current is reduced proportionally to the increase in voltage, thereby keeping the input power constant. This ensures that the reference control output I_{ACREF} from the voltage compensator is maximum such that the rated output power is delivered at minimum input voltage.

Voltage Feed-Forward Compensator:

If the voltage decreases, the product ($V_{AC} \cdot V_{PI}$), which determines I_{ACREF}, also proportionally decreases. However, to maintain a constant output power at reduced input voltage, the term I_{ACREF} should proportionally increase. The purpose of having an input voltage feed-forward, is to maintain the output power constant as determined by the load regardless of variations in the input line voltage. This compensator implemented digitally by calculating the average value of the input line voltage, squaring this average value and using the result as a divider for the input reference current, which is fed to the current error compensator.

Software Design:

The following section describes the software flow diagram for the power factor correction application project. The power factor correction algorithm in this example project is realized by following state machines.



1. PFC Init State

In this state following tasks are performed:

- Initialization and configuration of NVIC, AFEC, PWM peripherals for generation of periodic ADC triggers and ADC conversion interrupt.
- Calibration of input rectified AC current sense amplifiers.
- Initialization of reference DC voltage and reference rectified AC current PI controllers used for Power Factor Correction.

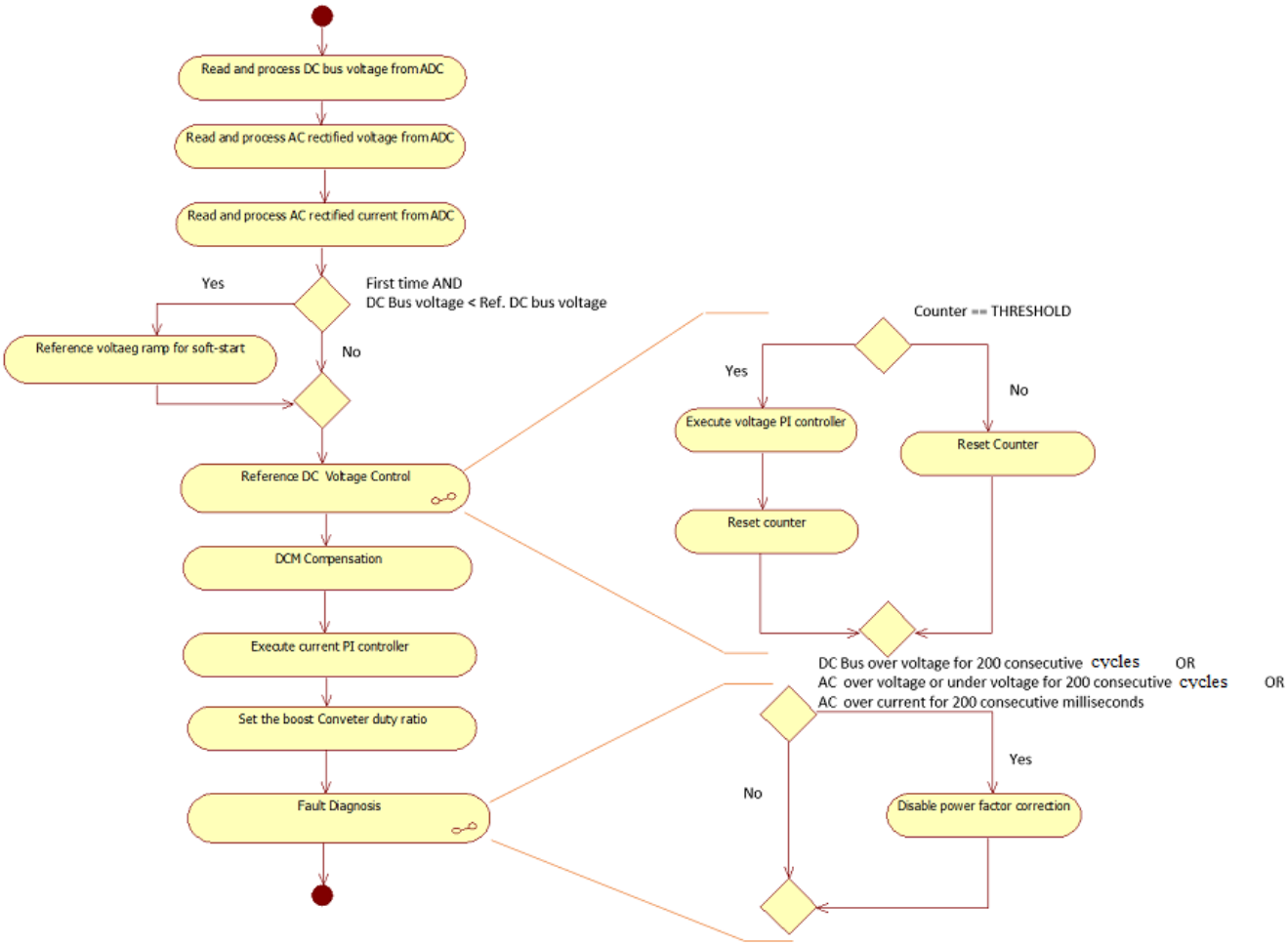
2. PFC Start state

In this state following tasks are performed:

- Start PWM peripherals timer to trigger ADC conversion interrupt.

3. PFC Running State

In this state, the power factor correction ISR is executed.



4. PFC Stop State

In this state, the power factor correction is disabled.

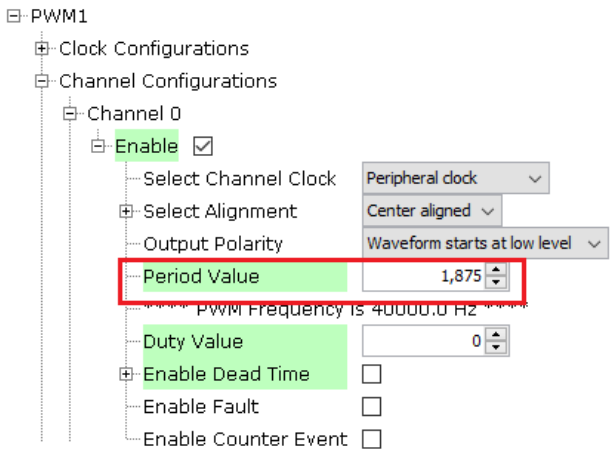
Software Configuration:

The following section describes user specific parameters used the the project.

1. Setting the PFC boost converter stage frequency:

The boost converter stage PWM frequency (in Hz) can be configured by setting the following macro in `userparams.h` file.

This period timer ticks should be configured properly for PWM peripheral in the MHC to get the desired frequency as shown below:



Macro	Description
PFC_PWM_FREQUENCY	Boost converter stage PWM frequency

2. Setting PI Controller parameters of Boost Converter Controller:

The voltage and current controllers used in the PFC algorithm has to be tuned to get the optimal control.

Parameters for reference DC bus voltage PI Controller:

Macro	Description
PFC_VOLTAGE_PTERM	Proportional gain of voltage control loop
PFC_VOLTAGE_ITERM	Integral gain of voltage control loop
PFC_VOLTAGE_CTERM	Anti-windup term of voltage control loop
PFC_VOLTAGE_OUTMAX	Maximum controller output of voltage control loop

Parameters for reference rectified AC current PI Controller:

Macro	Description
PFC_CURRCNTR_PTERM	Proportional gain of input current control loop
PFC_CURRCNTR_ITERM	Integral gain of input current control loop
PFC_CURRCNTR_CTERM	Anti-windup term of input current control loop
PFC_CURRCNTR_OUTMAX	Maximum controller output of input current control loop

Hardware Setup

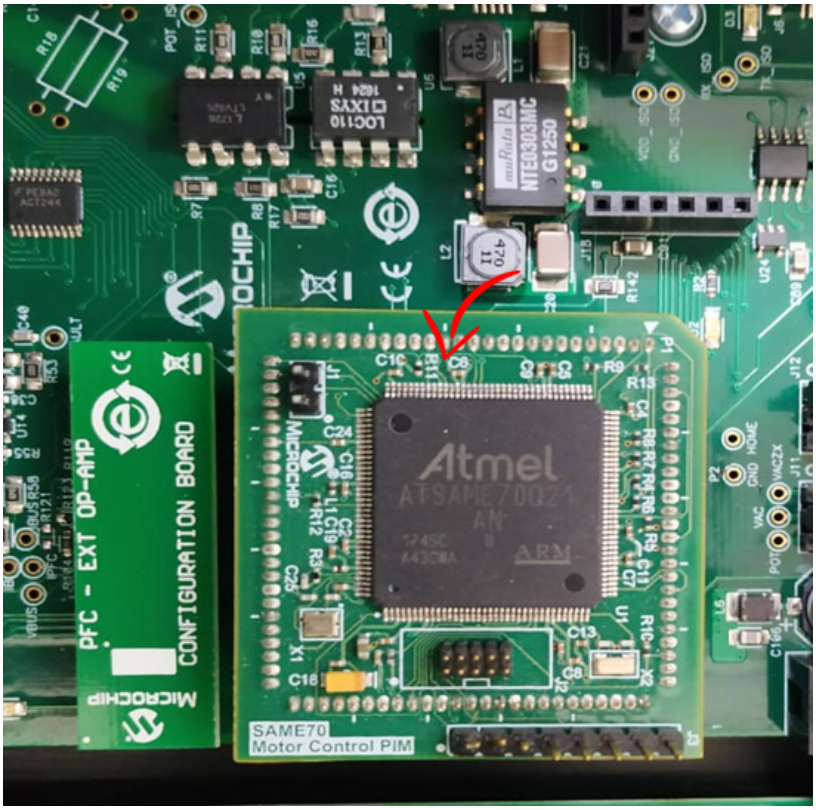
This section describes how to configure the supported hardware.

Description

1. **Project mchv3_sam_e70_pim.X.**
   - **Hardware Used**
     - MCHV3 Development Board (High Voltage)
     - Isolated Embedded Debugger Interface
     - ATSAME70 PIM

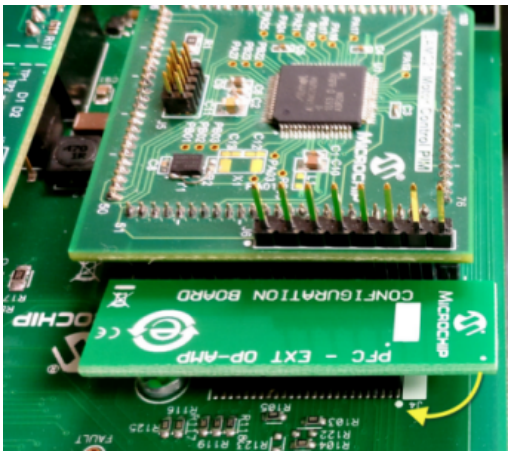
- Leadshine EL5-M0400-1-24 Motor

- **Hardware Setup**
  1. Mount the SAME70 144-pin Motor Control PIM on U9 header of the dsPICDEM MCHV-3 Development Board.

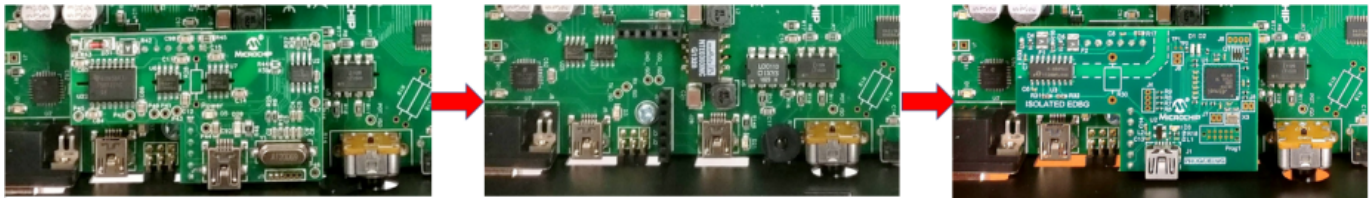


2. Connect the three-phase wires of the PMSM motor to M1, M2 and M3.

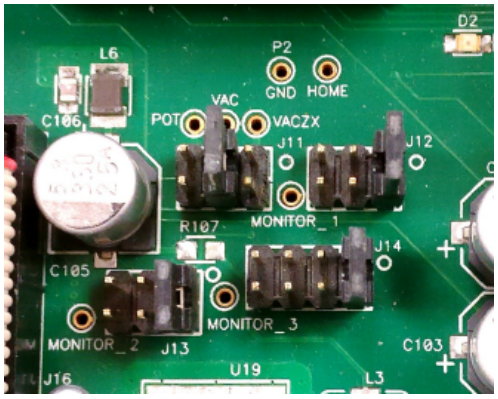
3. Connect External Op amp configuration Matrix board to header J4.



4. Connect Isolated EDBG card as shown in the following figure. Refer to Isolated EDBG Card Information Sheet for instructions.



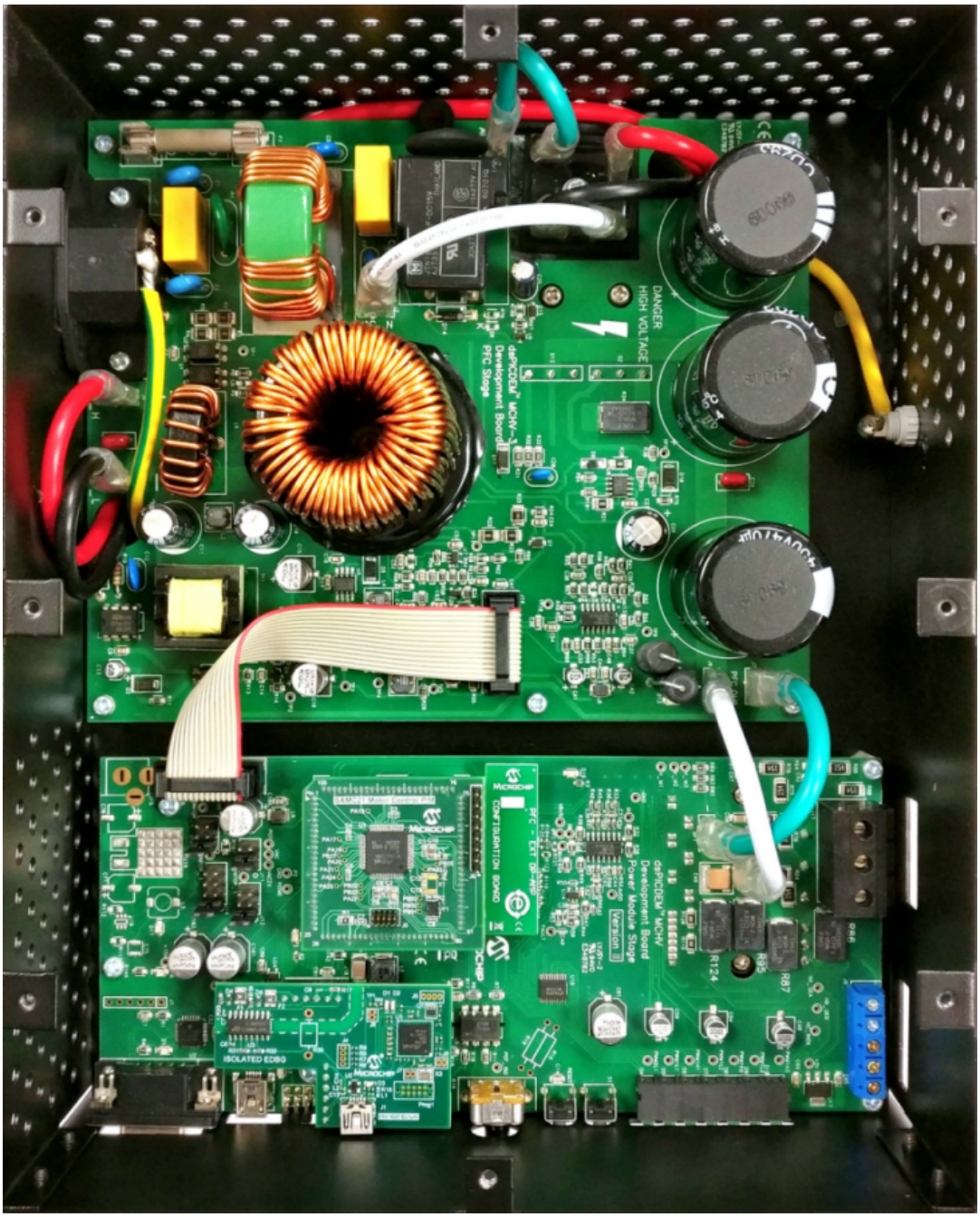
5. Connect the jumpers J11, J12, J13 and J14.



6. Connect the AC mains using J1.

7. 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.

- **Complete Setup**

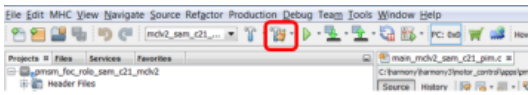


Running The Application

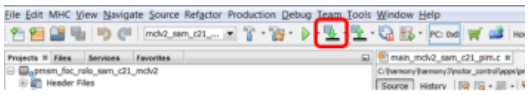
This section provides information on how to run the application using MPLAB X IDE.

Description

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_pfc_foc_pll_estimator_sam_e70\firmware\mchv3_sam_e70_pim.X.
2. Clean and Build Project:



3. Program the project:



4. Press the control push button to start the power factor correction.

Project Name	Output
`mchv3_sam_c21_pim.X`	1. Press the push button to start the power factor correction. 2. Observe the input rectified current and rectified voltage waveforms on X2CScope scope view.

BLDC Block Commutation Using Hall Sensors

This Section provides help for the BLDC Block Commutation using Hall Sensors application.

bldc_bc_hall_sam_c21

This document serves as a help document for Microchip's "BLDC block commutation using hall sensor" example project.

Description

Brushless Direct Current electric motors, or BLDC motors for short, are electronically commutated motors powered by a DC electric source via an external motor controller. This document serves as a guidance to build, configure and run Block Commutation algorithm for BLDC motors on Microchip's 32-bit SAMC21 microcontroller devices.

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 the following table and open the appropriate project file in MPLAB X IDE.

Project Name	Description
mclv2_sam_c21_pim.X	MCLV2 Development Board (Low Voltage)

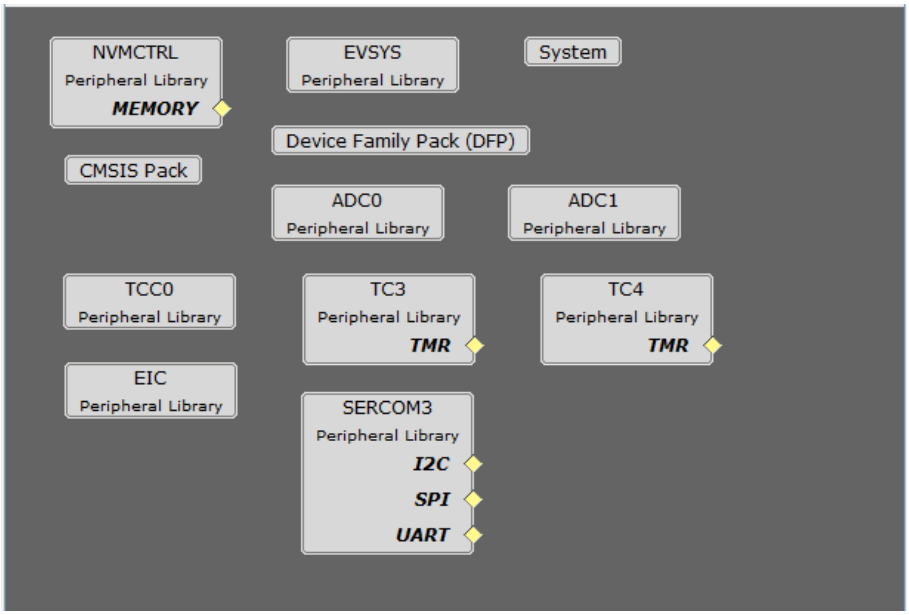
MPLAB Harmony Configurations

This topic provides information on the MHC configurations.

Description

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

bldc_bc_hall_sam_c21



Components Used

Components Used

TCC0:

- Configured to generate 3 pairs of single PWM signals at frequency of 20 kHz in Edge Aligned Mode
- Event output is enabled which is generated for counter overflow
- 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. DC link current is sampled and converted by ADC0 which can be used for current control. Potentiometer 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

TC3-TC4:

- TC3 peripheral is used as a timer to measure the time elapsed between two consecutive hall edges
- TC4 peripheral is used as an internal 1 mS timer counter

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
- External Interrupt control also detects hall sensor edges allowing the software to update the commutation pattern

EVSYS:

- Event System acts as a traffic cop between event generator and event users
- Event generated by the TCC0 when the counter overflows, 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

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 briefly introduces to the BLDC block commutation technique, and its implementation in the example project.

Description

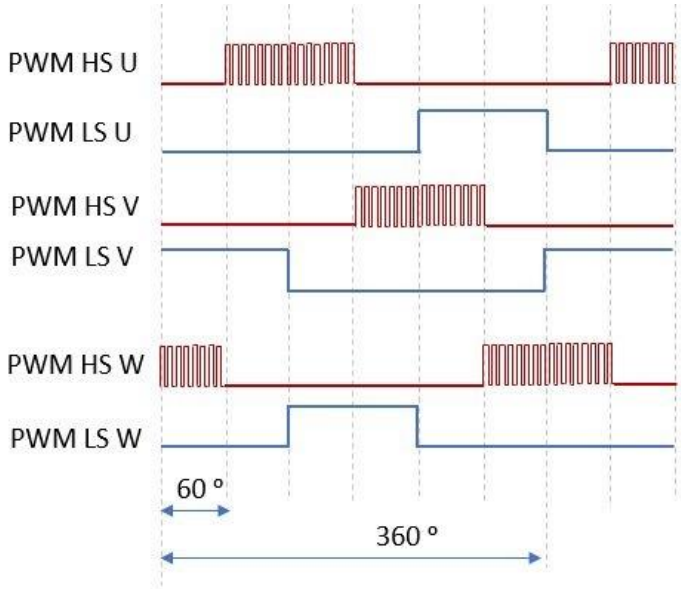
Brushless Direct Current electric motors, or BLDC motors for short, are electronically commutated motors powered by a DC electric source via an external motor controller. The electronic commutation to rotate the motor is achieved by a three phase inverter. The commutation technique are broadly classified as block Commutation and Sinusoidal commutation.

The block commutation has reduced system complexity as compared to the sinusoidal commutation. Hence, the block commutation technique is quite popular for the low cost applications where control precision, reduced efficiency, and higher acoustic noise are permitted.

The BLDC motor control using block commutation is done as follows:

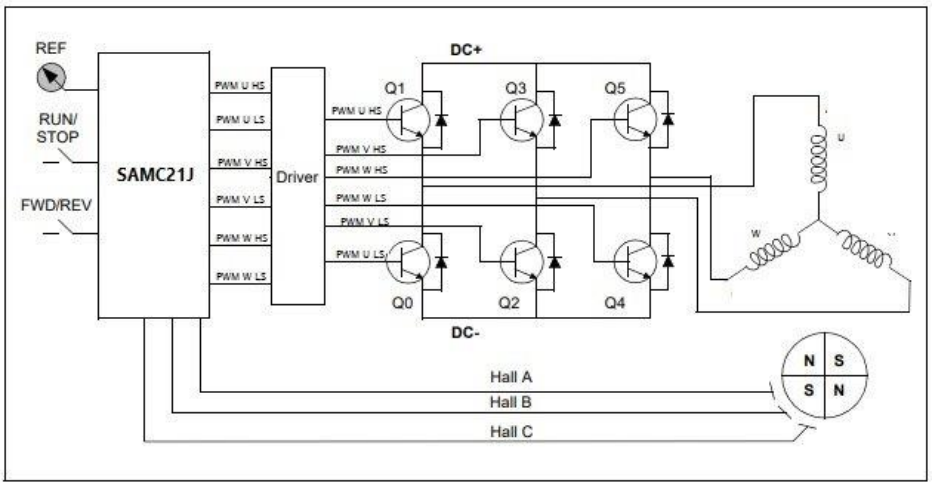
1. Read the hall sensor input pattern from three 120 degree spatially oriented hall sensor.
2. Use the hall sensor pattern to determine the three phase inverter commutation pattern.

Below diagram shows the high side modulated PWM.



The simplified block diagram for BLDC block commutation is shown below:

Block Diagram:



Software Design:

This section describes the software design of BLDC block commutation application. The first part of this section, numerical scaling for fixed-point implementation is briefly described. The later section, the software implementation of the example project is described.

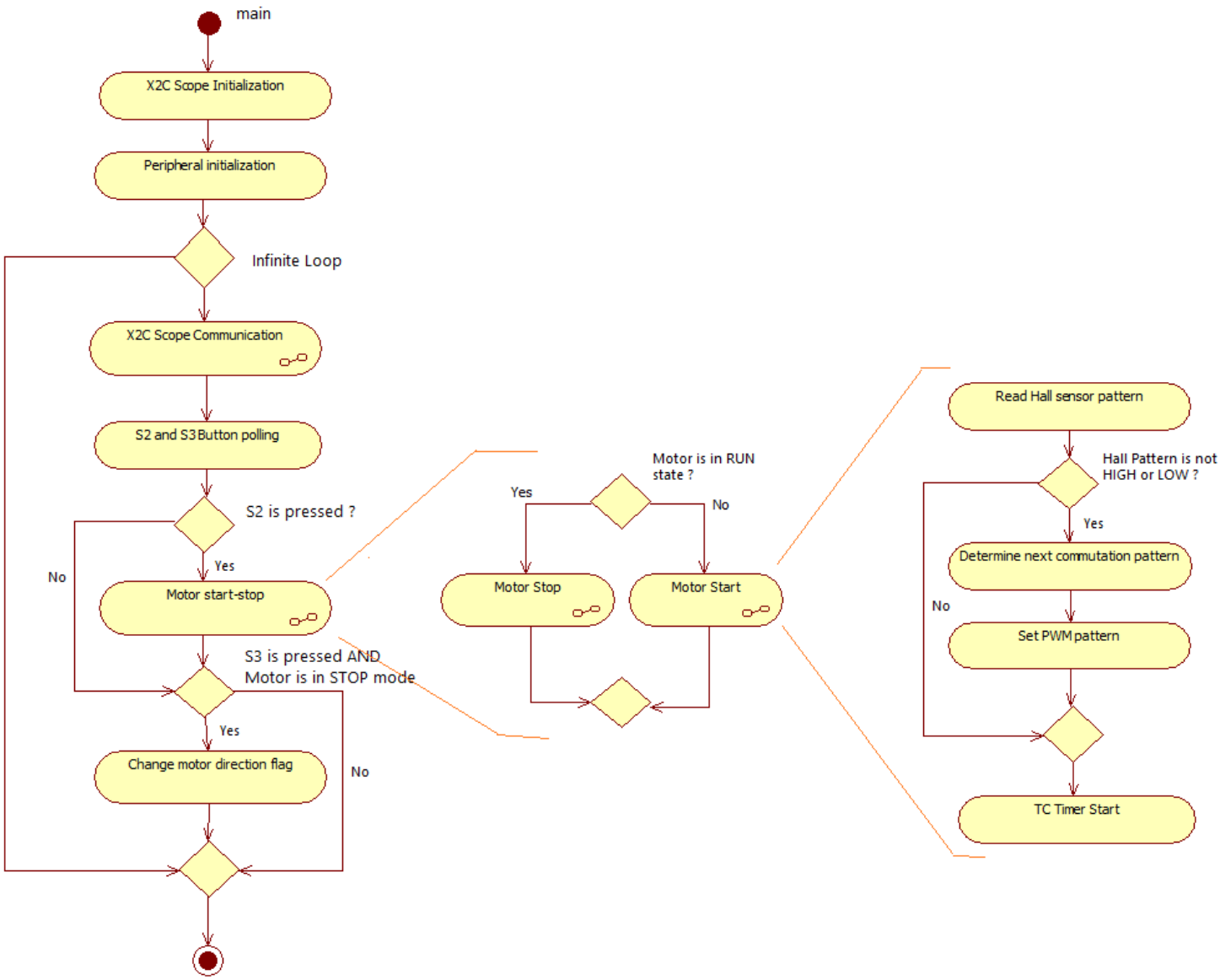
Scaling:

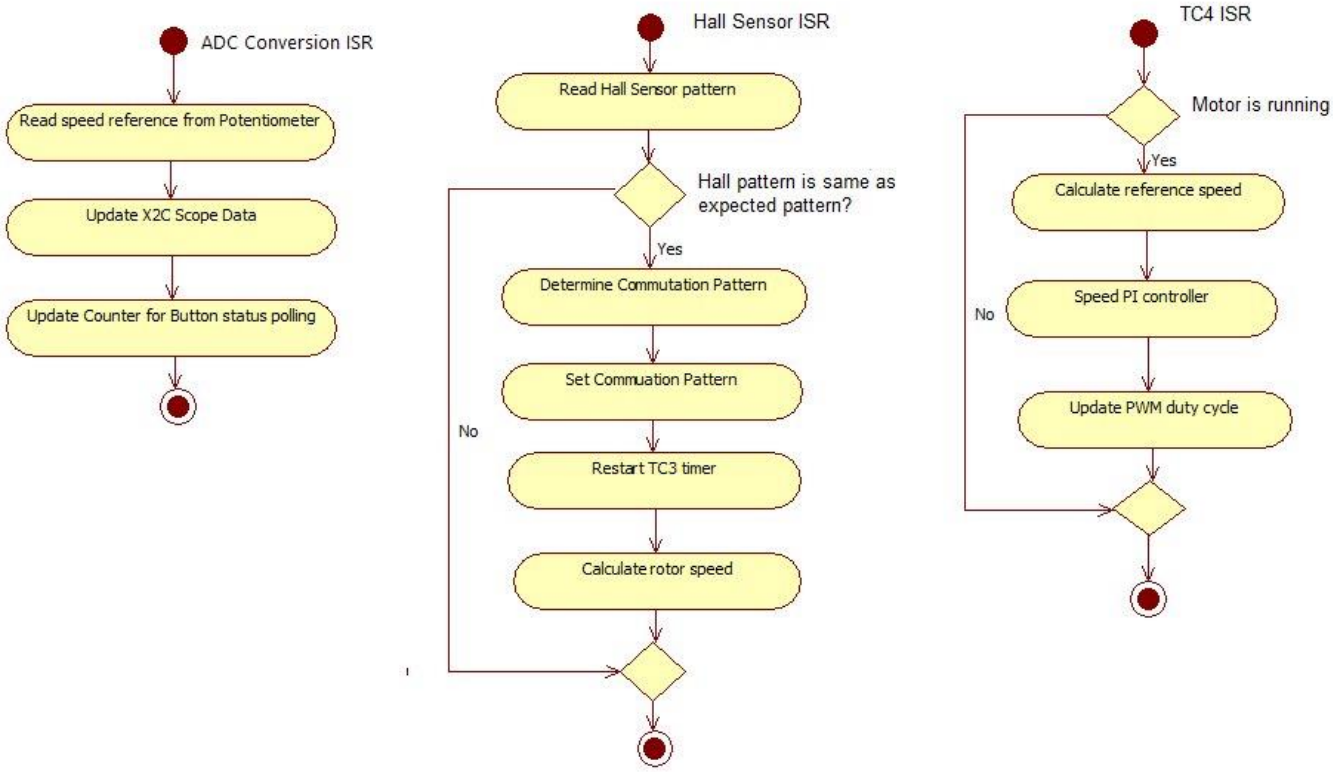
The example project uses Q14 format to represent the real-time physical quantities as follows:

$$Scaled\ value = \frac{Real\ value}{Real\ value\ range} * 2^{14}$$

The real value range of voltage, current and speed is chosen as independent base quantities for deriving the range of other physical quantities. The independent base quantities are determined by considering the electrical constraints of the MCLV2 development board and the motor used.

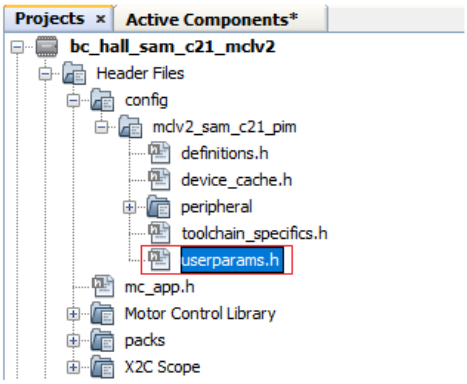
Software Implementation:


main
X2C Scope Initialization
Peripheral initialization
Infinite Loop
X2C Scope Communication
S2 and S3 Button polling
S2 is pressed ?
Yes
No
Motor start-stop
S3 is pressed AND
Motor is in STOP mode
Yes
No
Change motor direction flag
Motor is in RUN
state ?
Yes
No
Motor Stop
Motor Start
Read Hall sensor pattern
Hall Pattern is not
HIGH or LOW ?
Yes
No
Determine next commutation pattern
Set PWM pattern
TC Timer Start



Software Configuration:

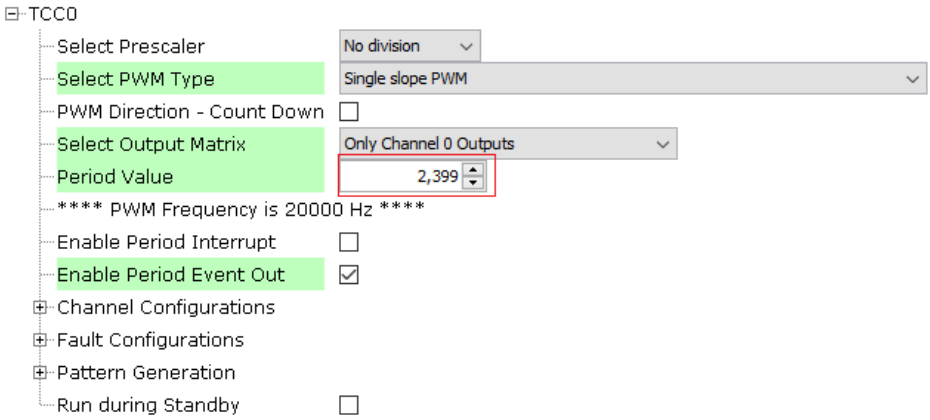
The motor specific parameters, board specific parameters and application parameter macros have to be updated in the software for optimal control of the BLDC motor. The following section briefly describes motor and application specific parameters used in the project. These parameter macros can be found 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:**

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 20 KHz in edge 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.



Macro	Description
PWM_PERIOD	PWM Timer period value = "Period value" defined in MHC + 1

2. **Setting motor specific parameters:**

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

Macro	Description	Units
MOTOR_POLE_PAIRS	Number of Pole Pairs	-
MAX_MOTOR_SPEED	Rated speed of the motor	RPM

3. **Setting hall pattern and commutation pattern:**

Below table gives the hall and commutation pattern relation for long hurst and small hurst motors.

HA	HB	HC	Pattern	Ph U	Ph V	Ph W
0	0	1	V+W-	0	H	L
1	0	1	U-V+	L	H	0
1	0	0	U-W+	L	0	H
1	1	0	V-W+	0	L	H
0	1	0	U+V-	H	L	0
0	1	1	U+W-	H	0	L

Hall pattern and commutation patterns are defined in the **mc_app.c** file.

HALL_ARRAY[16] array contains the hall pattern for both the directions - first 8 entries are for clockwise direction and next 8 entries are for anti-clockwise direction.

COMMUTATION_ARRAY[16] array contains commutation pattern corresponding to the hall pattern - first 8 entries are for clockwise direction and next 8 entries are for anti-clockwise direction.

e.g The entry in the commutation array for V+W- is 0x4075.

This is given in terms of the pattern enable register TCC_PATT.

Pattern value configuration for V+W- is 0x40. This means low side switch of Phase W is continuously ON.

X	W-	V-	U-	X	W+	V+	U+
0	1	0	0	0	0	0	0

Pattern enable configuration is 0x75. Pattern output is disabled for Phase V high side switch where PWM will be applied as per speed command.

X	W-	V-	U-	X	W+	V+	U+
0	1	1	1	0	1	0	1

Hardware Setup

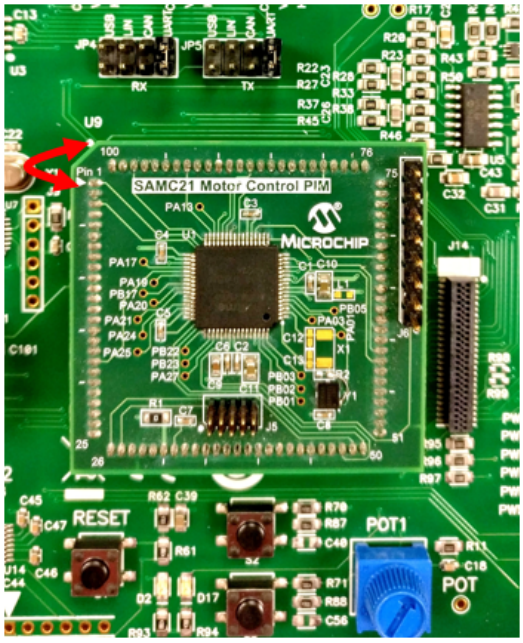
This section describes how to configure the supported hardware.

Description

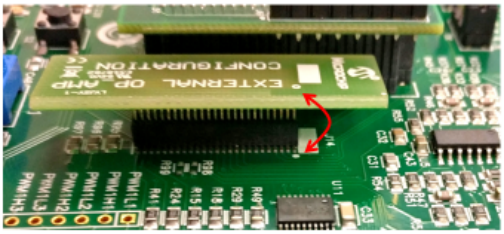
Project mclv2_sam_c21_pim.X.

- **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. Connect the motor phases and hall sensor interfaces. Hall and phase connections for long hurst motor are given below:

Hall Connector	
Red	+5 V
Black	GND
White	HA
Brown	HB
Green	HC

Phase Connections	
Red	M3
Black	M2
White	M1

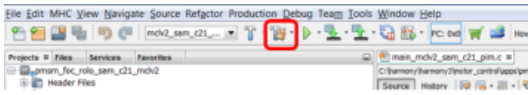
Running The Application

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

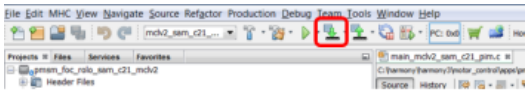
Description

Running the Application : bldc_bc_hall_sam_c21

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\bldc_bc_hall_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 the motor direction.

LED D2 Status	Description
OFF	Motor spin direction is "forward"
ON	Motor spin direction is "reverse"

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 :

If LED D17 is ON, it indicates a non-recoverable fault caused by over-current. The MCU needs to be reset in order to recover from this fault mode.

Project Name	Output
`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.

FOC Based Position Control of PMSM Using Quadrature Encoder

This Section provides help for the FOC Based Position Control of PMSM using Quadrature Encoder.

pmsm_foc_encoder_position_sam_e54

This motor control example project shows how to control the position of Permanent Magnet Synchronous Motor (PMSM) using an Encoder based Field Oriented Control (FOC) on SAME54 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 the PMSM motor for applications which requires high dynamic performance.

This example shows how to configure motor control peripherals ADC, PDEC and TCC for the control operation. The control strategy is the sensored FOC, in which rotor position is determined by the Quadrature Encoder. Position is obtained using quadrature encoder and speed is calculated from the position.

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 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_position_sam_e54

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

Project Name	Description
mclv2_sam_e54_pim.X	MCLV2 Development Board (Low Voltage)

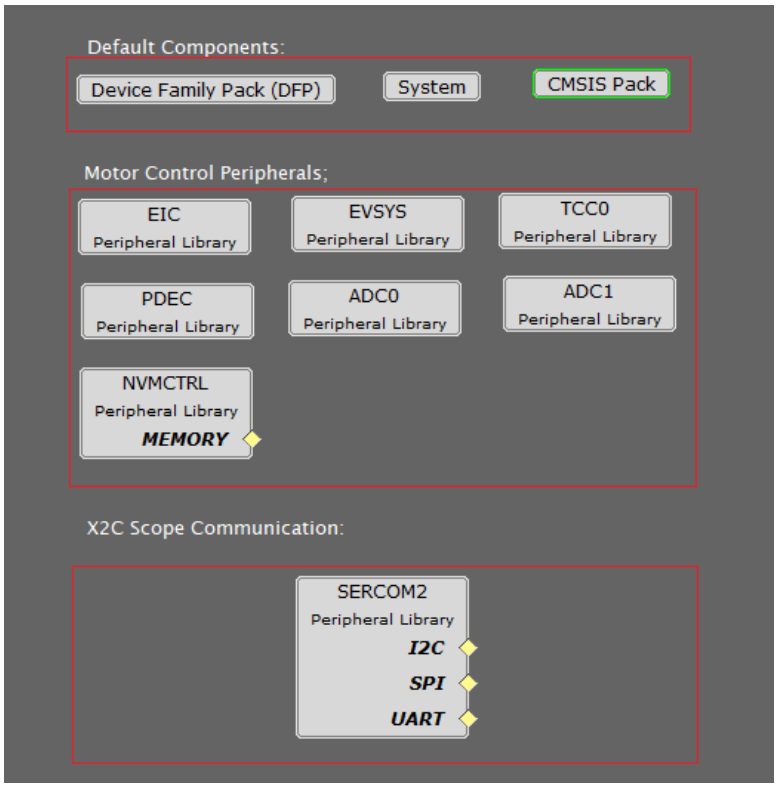
MPLAB Harmony Configurations

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_e54_pim.X



Components Used

TCC0:

- Configured to generate three pairs of complimentary PWM signals at a frequency of 20 kHz in "Dual Slope PWM with interrupt/event when counter = TOP" i.e. "Center Aligned Mode"
- In "Dual Slop PWM" mode, the PWM duty cyle has an inverse relation with the compare value. For example, for a Period Count (PER) = 3000, Compare Count (CCx) = 2000 (66%); would result in a PWM with a 33% duty cycle. In order to achieve a direct relation between compare count and duty cycle, the PWMxH and PWMxL outputs are swapped
- In dual shunt configuration, phase current measurements are performed at the end of the PWM cycle. Since the PWM outputs are swapped, the ADC conversion event is generated when the counter reaches TOP
- Dead-time is enabled and set to
  - mclv2_sam_e54_pim.X - 1uS
- 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 the same resolution and sampling time, both ADCs complete conversion at the same time

PDEC:

- PDEC is configured to Quadrature decoder direction (X4) in QDEC mode to detect the quadrature encoder signals
- All the 16-bits of the QDEC counter are used to measure the angular position (CTRLA.ANGULAR = 0x7)

EIC:

- External Interrupt Controller detects a 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 an intermediary between event generator and event users
- Event generated by the TCC0 when the counter reaches TOP, is used by the ADC0 as a hardware trigger source via the 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

SERCOM2:

- SERCOM2 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 the X2CScope

Control Algorithm

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

Description

The project implements an encoder based Field Oriented Control (FOC) algorithm on the SAME54 32-bit micro-controller to track the position of the PMSM motor. The following section describes briefly about the FOC algorithm, software design and implementation.

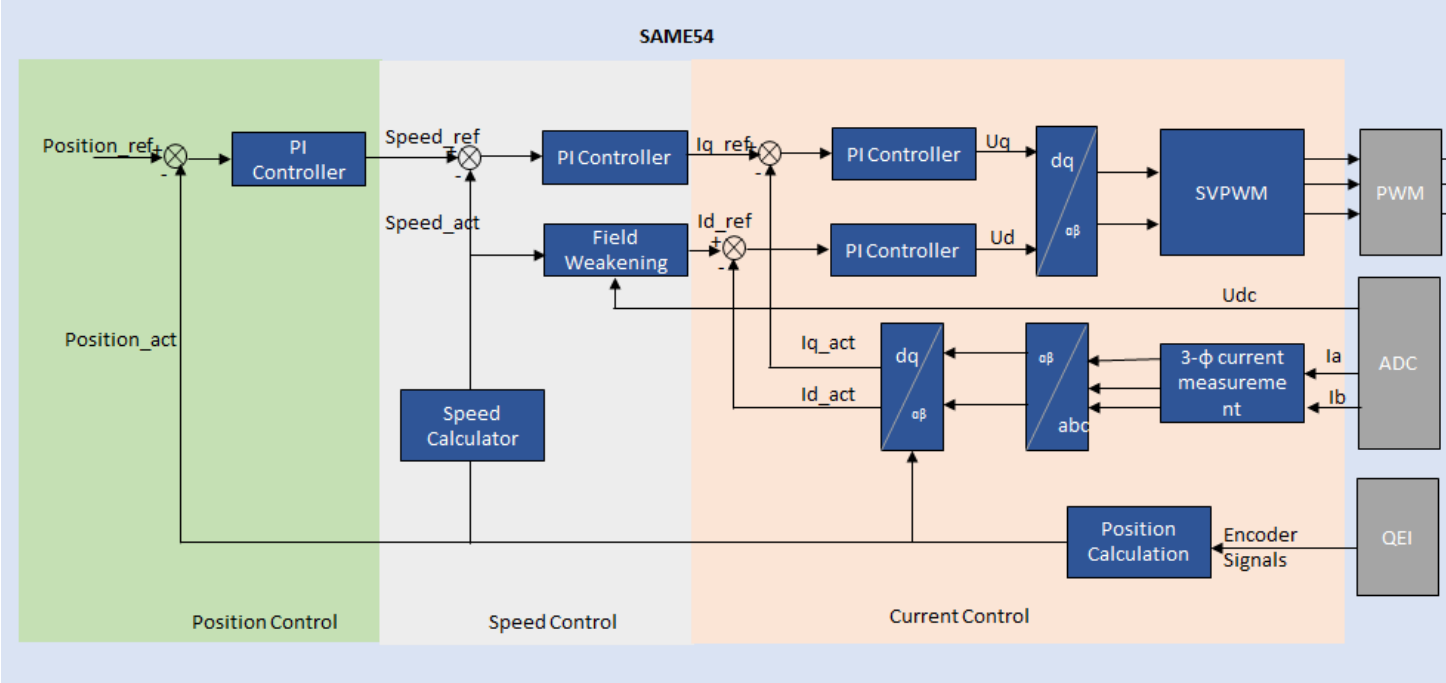
Field Oriented Control is the technique used to achieve the decoupled control of torque and flux. This is done by transforming the stator current quantities (phase currents) from a stationary reference frame to torque and flux producing currents components in a rotating reference frame using mathematical 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 the Clarke transformation.
3. Calculate the rotor position angle.
4. Transform stator currents into the d,q-coordinate system using the Park transformation.
5. Position and speed are controlled by position and speed PI controllers respectively.
6. The stator current torque (iq) and flux (id) producing components are controlled separately by the corresponding PI controllers.
7. 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 the Inverse Park transformation.
8. Using the space vector modulation, the three-phase output 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.

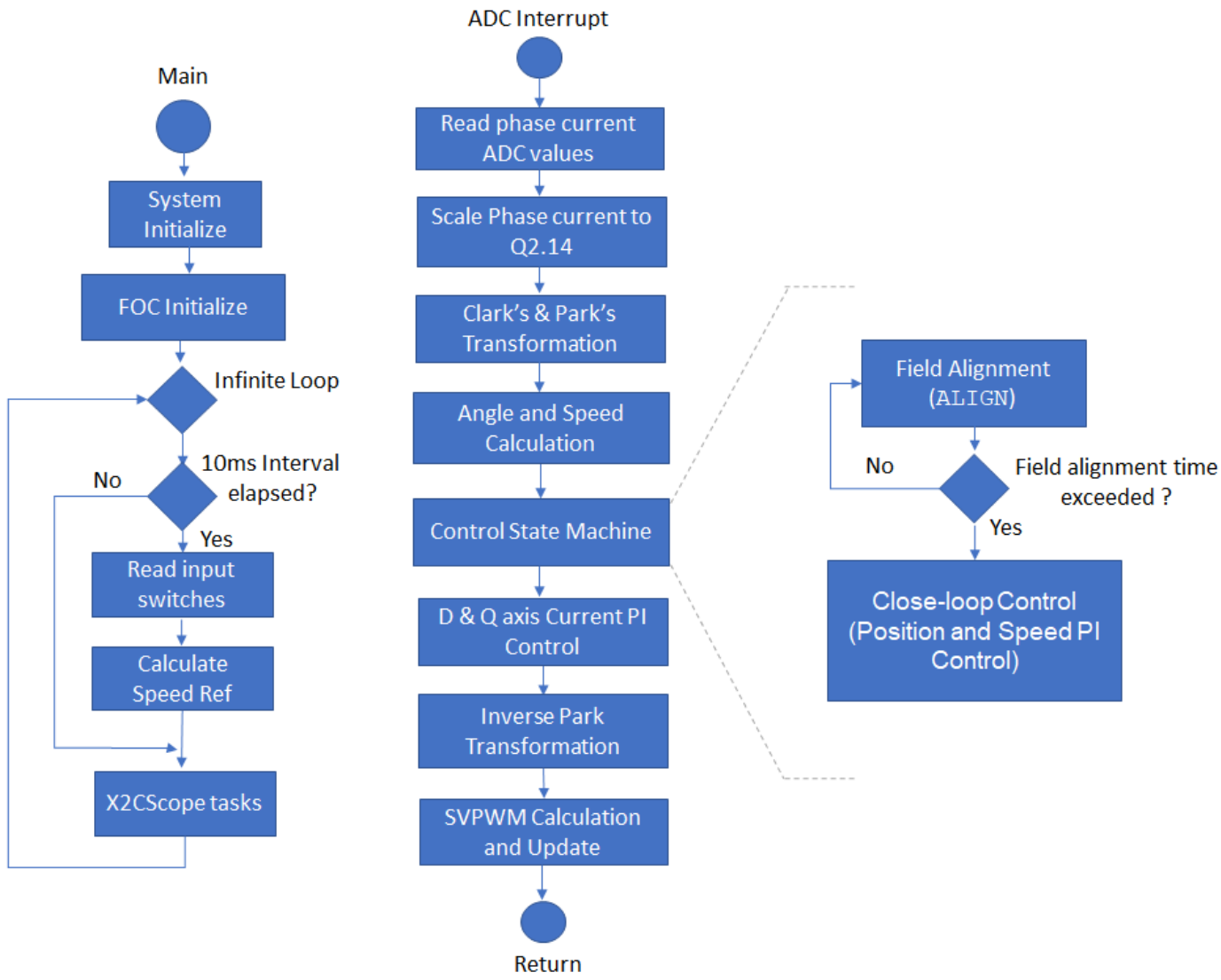
The rotor position angle is determined from the Quadrature encoder.

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 position control task is realized by a state machine as shown in the previous figure. The following sections briefly describes the various states in the PMSM position control task:

1. Initialize:

In this state, following tasks are performed:

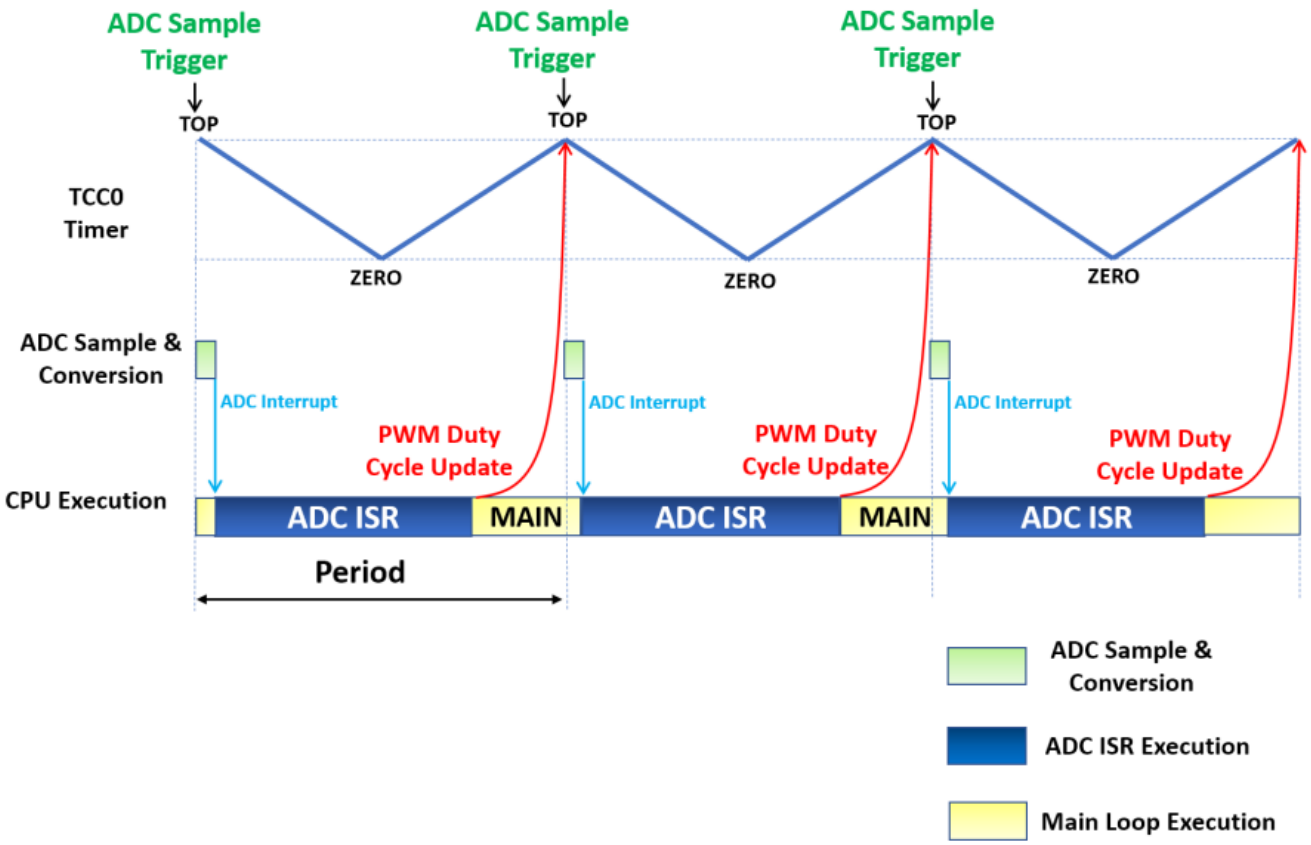
- Initialization and configuration of motor control peripherals for generation of periodic ADC triggers and ADC conversion interrupt
- Current Offset measurement and calibration
- Initialize PI controller parameters for speed and current control loops

2. Start:

In this state, the motor control state variables are reset and the 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:



The position control is carried out in the ADC interrupt task.

- **Field Alignment:**

The quadrature encoder is an incremental encoder. It is important to know the initial position of the rotor to maintain orthogonal rotor and stator flux. To achieve this, the q-axis is excited with a small open loop current while the d-axis is excited with zero current.

In order to avoid pseudo-alignment, i.e. rotor alignment at 180 degrees out of phase, dual step alignment approach is used. The rotor is aligned to an intermediate angle before aligning it to the zero position.

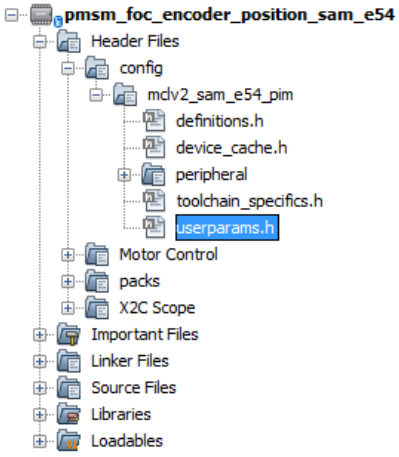
After the initial alignment, an electrical angle is shifted by 90 degrees to obtain the maximum torque.

- **Close Loop Control:**

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

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:**

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

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

Macro	Description
PWM_FREQUENCY	Current controller and PWM frequency in Hz.

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

Set the 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 should to be updated in userparams.h file.

Parameters for position Control loop:

Macro	Description
POSCNTR_PTERM	Proportional gain of position control loop
POSCNTR_ITERM	Integral gain of speed position loop
POSCNTR_CTERM	Anti-windup term of position control loop

POSCNTR_OUTMAX	Maximum controller output of position control loop

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

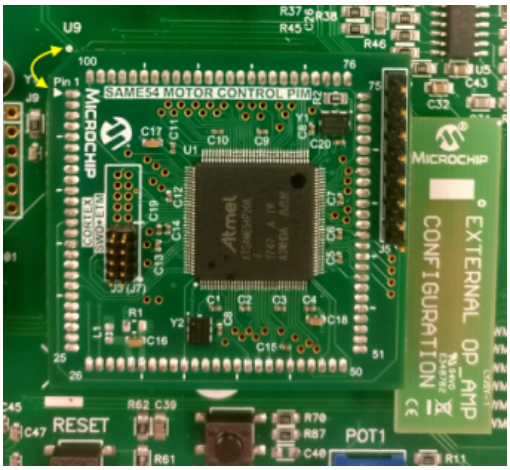
Hardware Setup

This section describes how to configure the supported hardware.

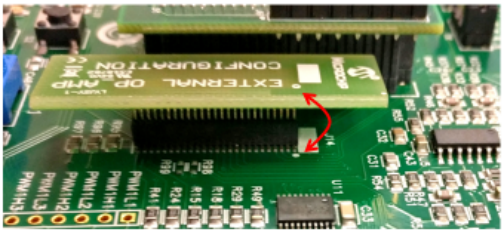
Description

Project mclv2_sam_e54_pim.X.

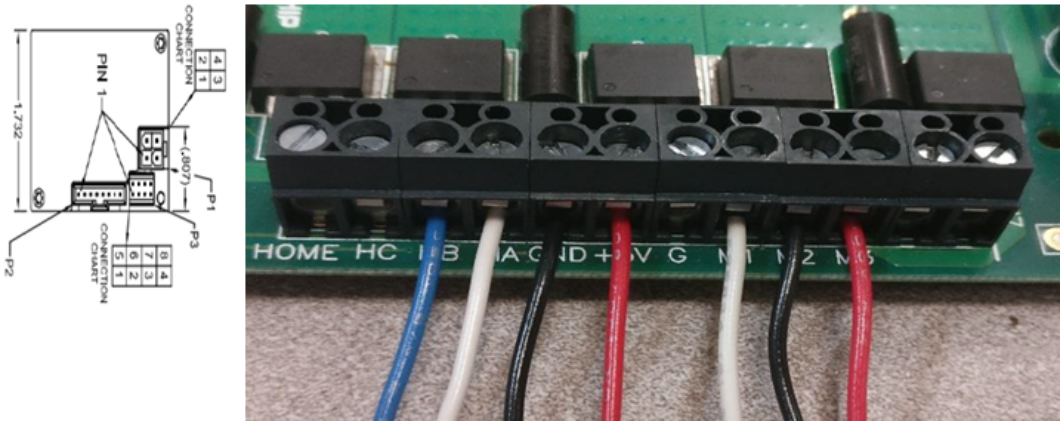
- **Hardware Used**
  - dsPICDEMTM MCLV2 Development Board (Low Voltage)
  - ATSAME54 Motor Control Plug In Module
  - Long Hurst DMA0204024B101 Motor
- **Hardware Setup**
  1. Mount the ATSAME54 Motor Control Plug In Module on U9 header.



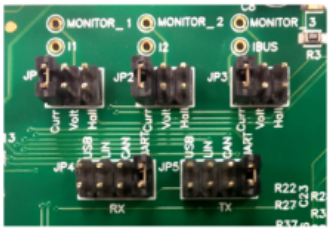
2. Place the "External Opamp Configuration" Matrix board at J14



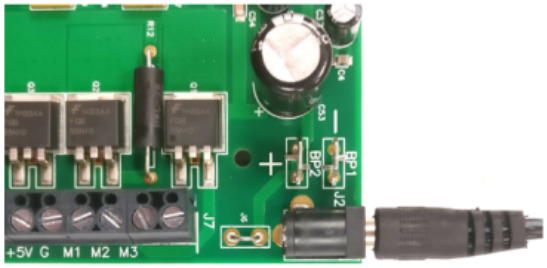
3. Connect motor and encoder terminals.



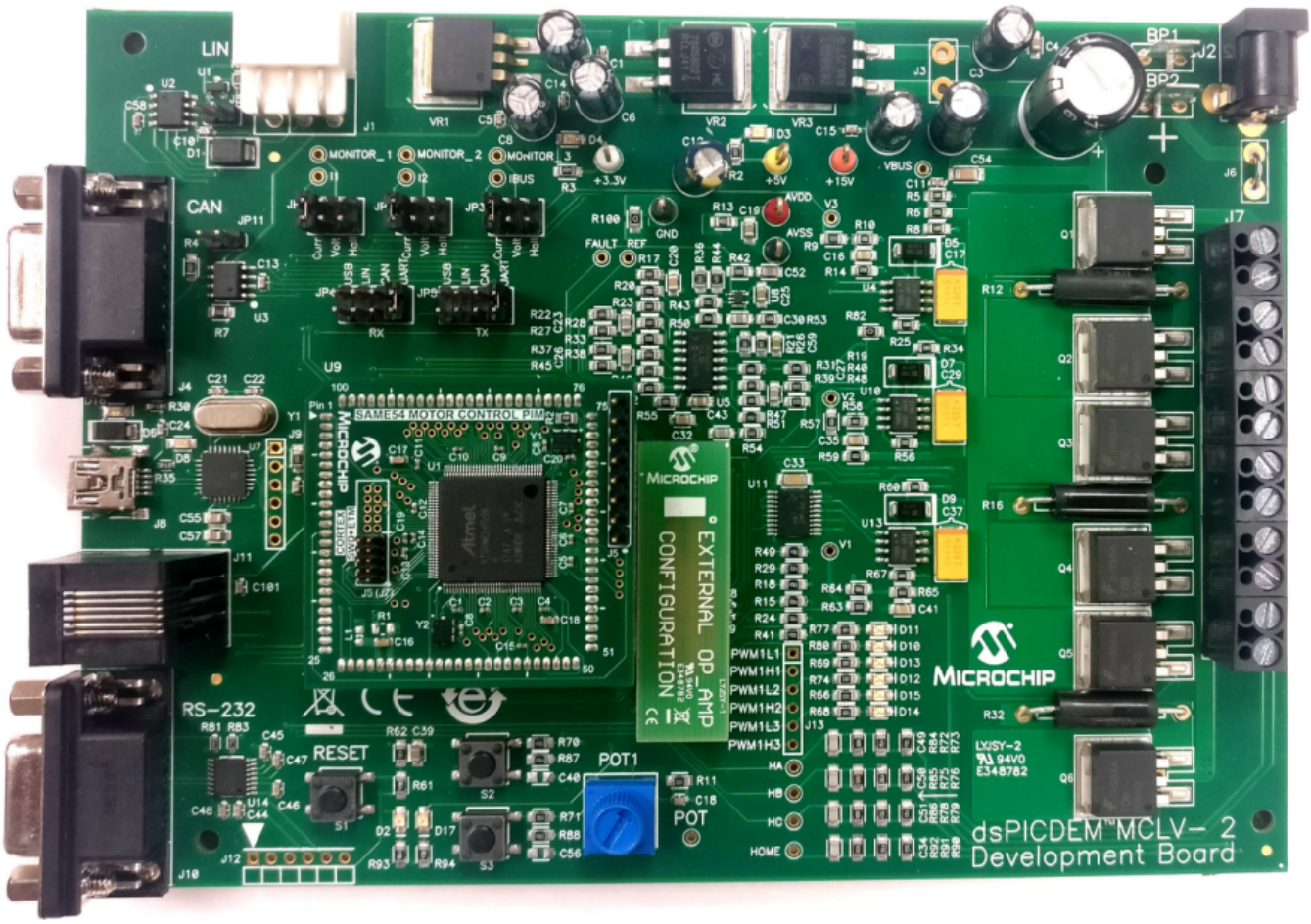
4. Jumper Settings:
   - JP1 - Curr, JP2 - Curr, JP3 - Curr
   - In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
   - In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**



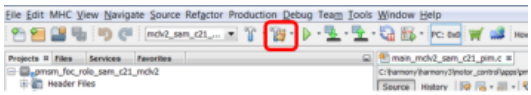
Running The Application

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

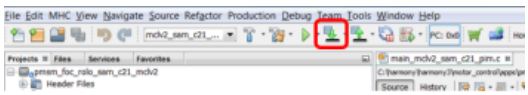
Description

Running the Application : pmsm_foc_encoder_position_sam_e54 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_encoder_position_sam_e54\firmware\mclv2_sam_e54_pim.X
2. Clean and Build Project



3. Program the project



4. Put the potentiometer knob at mid-point for zero position.
5. Press the switch S2 to start/stop the motor.
6. Rotate the potentiometer knob for clockwise position control, and anti-clockwise for anti-clockwise position control.

Fault Indication and Recovery : pmsm_foc_encoder_position_sam_e54 - MCLV2

If LED D17 is ON, it indicates a non-recoverable fault caused by over-current condition. The MCU needs to be reset in order to recover from this fault mode.

Project Name	Output
`mclv2_sam_e54_pim.X`	1. Press the Switch S2 to start the motor. 2. Vary Potentiometer to change the reference rotor position of the motor. 3. Observe waveforms on X2CScope scope view. 4. Press the Switch S2 to stop the motor.

Note: The range of rotor position control is in terms of the number of encoder pulses.

PMSM FOC Using ROLO with Field Weakening and MTPA Capability

This Section provides help for PMSM sensor-less FOC using ROLO observer with Field Weakening and MTPA capability.

pmsm_foc_rolo_fw_mtpa_sam_c21

This application demonstrates the field weakening and MTPA features with the Reduced Order Luenberger Observer (ROLO) based FOC on SAMC21 micro-controller.

Description

This application implements the sensor-less field oriented control (FOC) of a permanent magnet synchronous motor (PMSM) with Field Weakening and MTPA capability. 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
mclv2_sam_c21_pim.X	pmsm_foc_rolo_fw_mtpa_sam_c21 is setup to work with ATSAMC21 Motor Control Plug In Module + dsPICDEM™ MCLV-2 Development Board (Low Voltage)

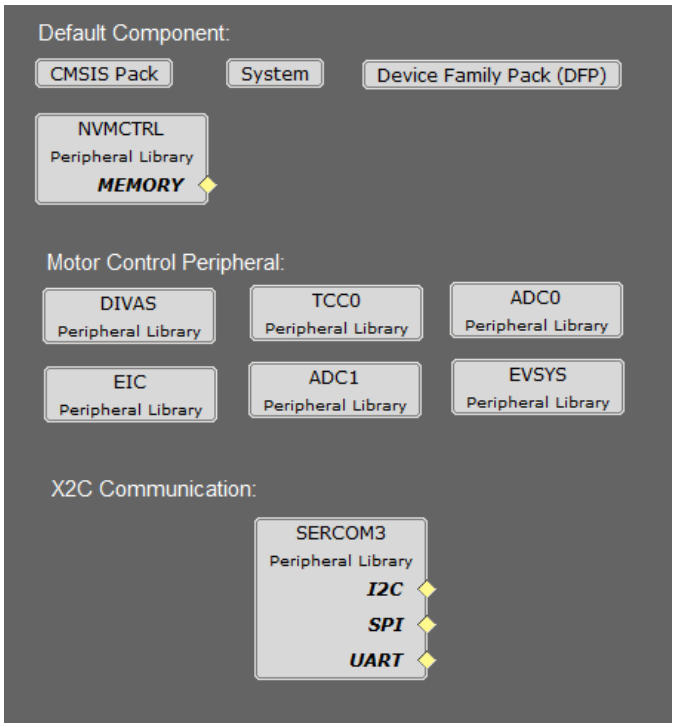
MPLAB Harmony Configurations

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_fw_mtpa_sam_c21



Components Used

TCC0:

- Configured to generate three pairs of complimentary PWM signals at a frequency of 10 kHz in "Dual Slope PWM with interrupt/event when counter = ZERO" i.e. "Center Aligned Mode"
- Event output is enabled which is generated when the counter reaches ZERO
- Dead-time is enabled and set to
  - mclv2_sam_c21_pim.X. - 1uS
- 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 a 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 an intermediary 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 the Event System
- Event generated by the EIC upon over-current fault, is used by the TCC0 as a non-recoverable fault event via the 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 the X2CScope

Control Algorithm

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

Description

Summary

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 Field Weakening and MTPA capability on the SAMC21J18A MCU. It estimates rotor position from measured phase currents and a reduced order luenberger observer (ROLO). For more details about 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 current 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 the 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 clark 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.

Field Weakening:

The field weakening for PMSM implies imposing a negative value of d-axis current, which has the role of weakening the air gap flux linkage. Field weakening is required to enable the PMSM motor to rotate above its rated speed. In this example project, an equation based field weakening is implemented.

$$V_{qs_{ref}} = \sqrt{V_{max}^2 - V_{ds}^2}$$

$$i_{ds_{flux_weakening}} = \frac{V_{qs_{ref}} - R_s \cdot i_{qs} - \omega \cdot \psi_{PM}}{\omega \cdot L_s}$$

For more details refer "Equation-based Flux weakening" section of Application note AN2520.

The expression $\omega\psi_{PM}$ equals the back EMF generated by the motor. Hence the term can be replaced by a back EMF term which is readily available in ROLO.

Maximum Torque Per Ampere:

The Maximum Torque per ampere (MTPA) for PMSM implies imposing a negative value of d-axis current in order to minimize the copper losses. Minimizing the copper losses increases the efficiency of the overall motor drive. It has to be noted that MTPA is only applicable for anisotropic motors.

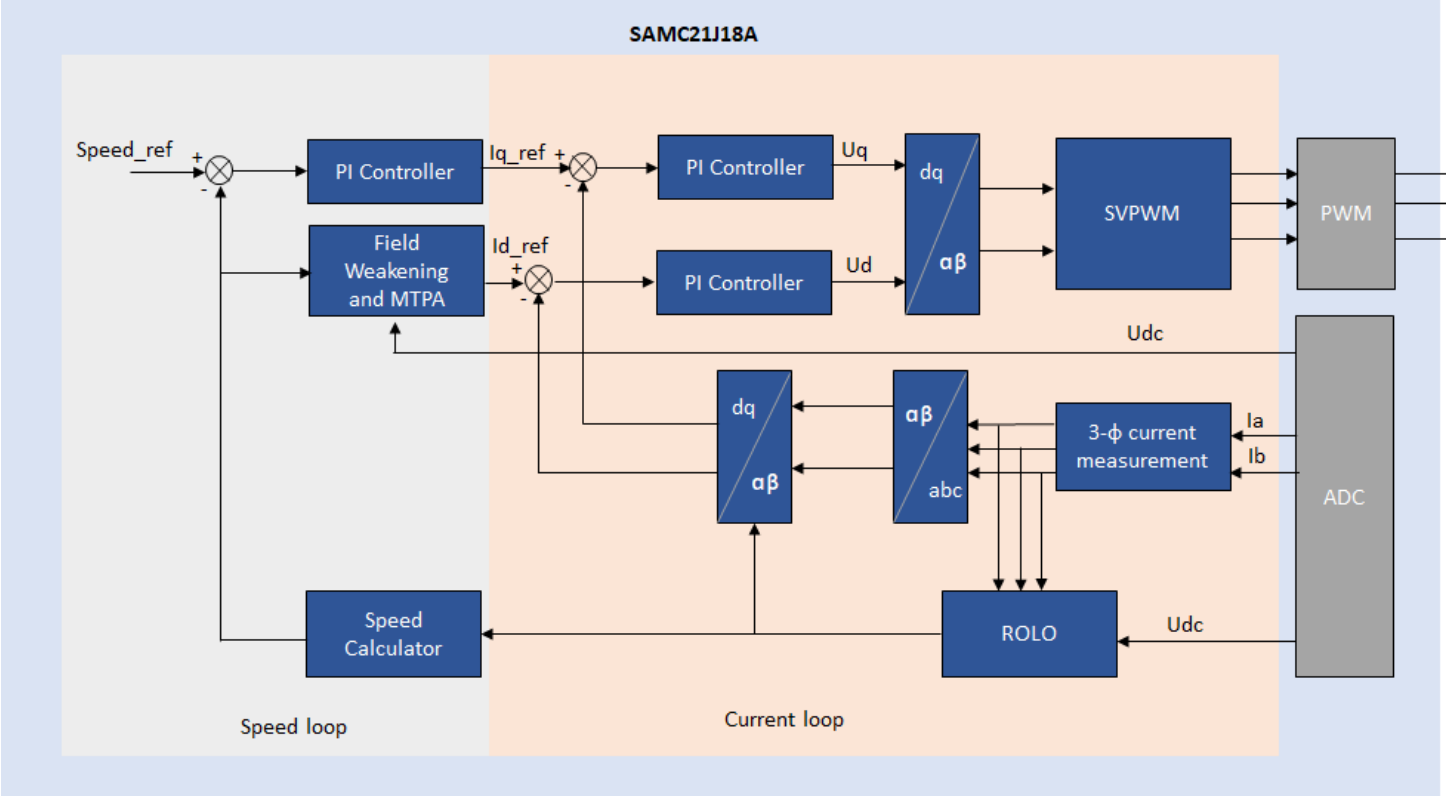
$$T_e = \frac{3}{2}p\left(\varphi_f i_q + (L_d - L_q)i_d i_q\right)$$

$$\min\left\{P_{Cu} = \frac{3}{2}R_s\left(i_d{}^2 + i_q{}^2\right) = \frac{3}{2}R_s\left[i_d{}^2 + \left(\frac{2T_e}{3p\left(\varphi_f + (L_d - L_q)i_d\right)}\right)^2\right]\right\}$$

Solving for the above equation,

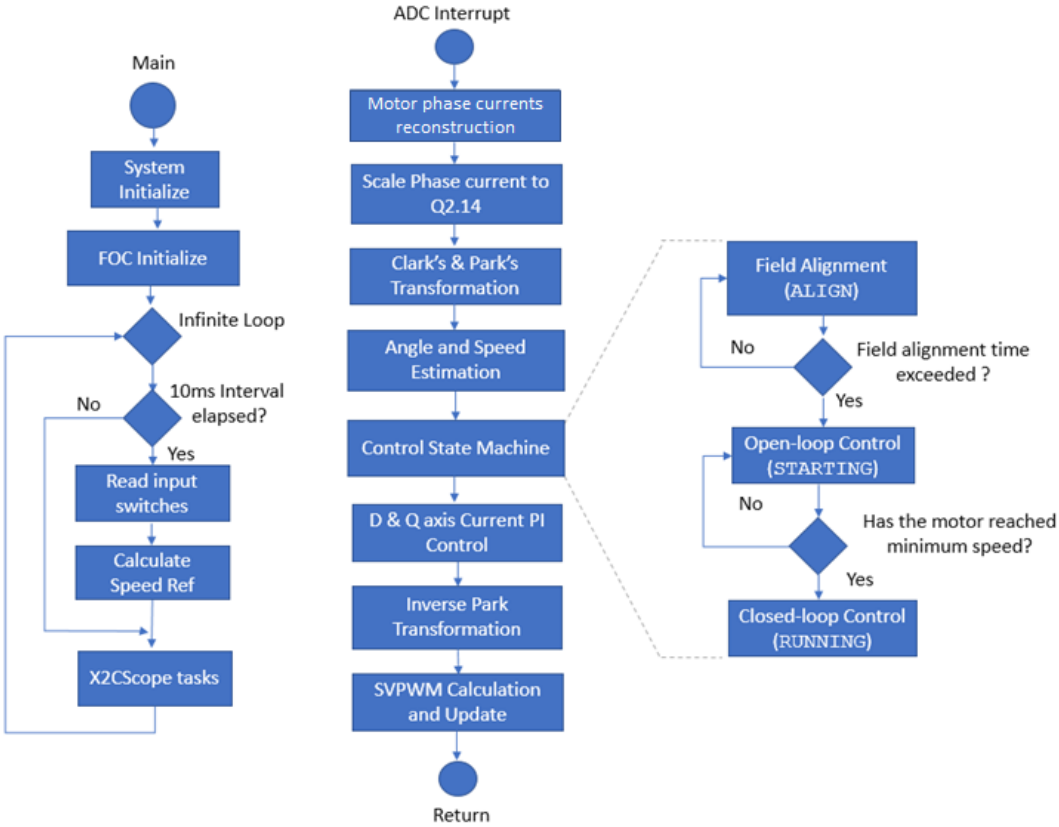
$$id_{MTPA} = \frac{\varphi_f}{2(L_d - L_q)} - \sqrt{\left[\frac{\varphi_f}{2(L_d - L_q)}\right]^2 - i_q^2}$$

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



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:

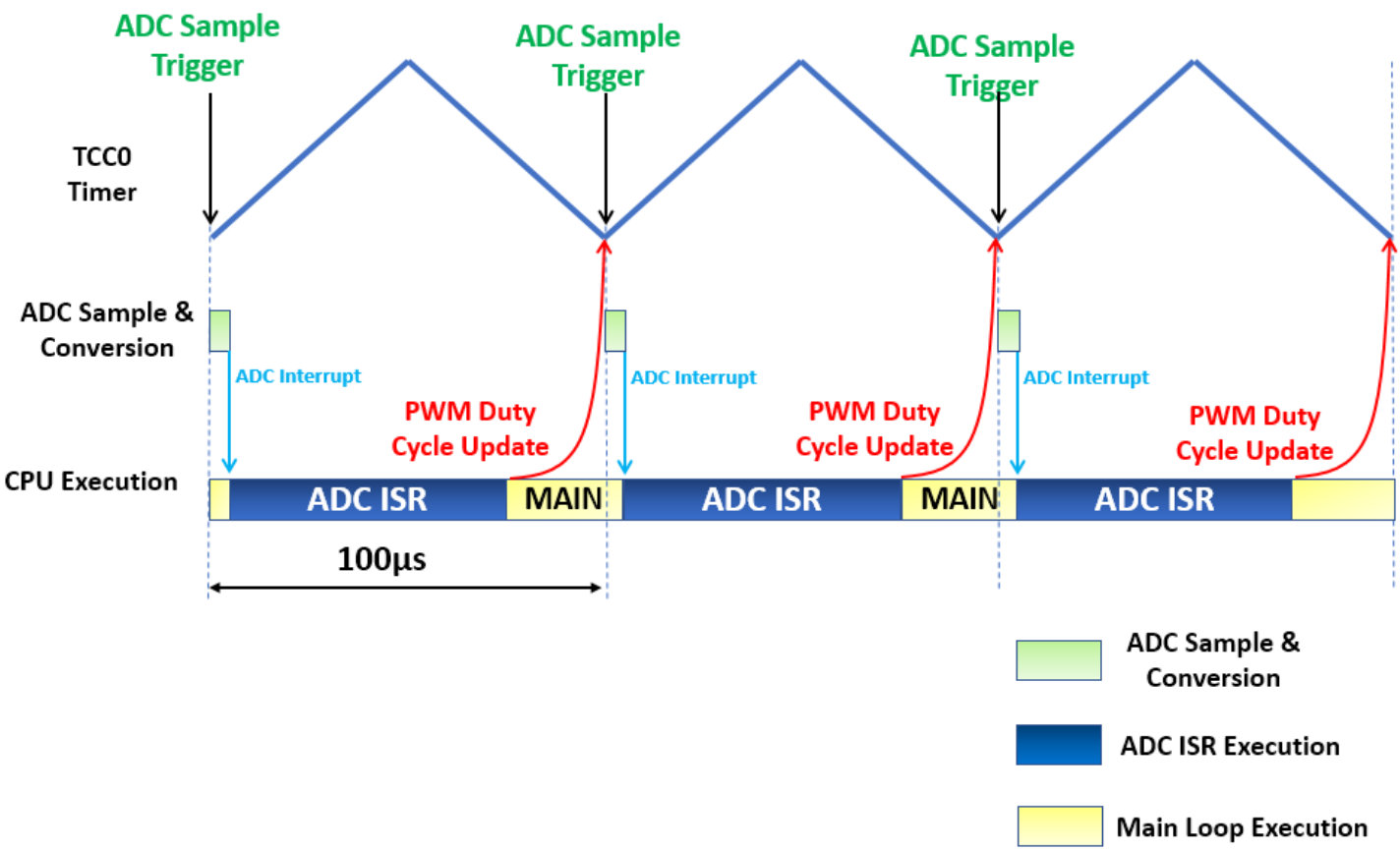
In this mode, the estimated rotor angle obtained from ROLO-based estimator is used to perform field oriented control of the PMSM motor.

Timing Diagram:

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

- ADC - It measures motor phase currents at the start of the PWM cycle.
- CPU - Based on motor phase currents measurements, the 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.

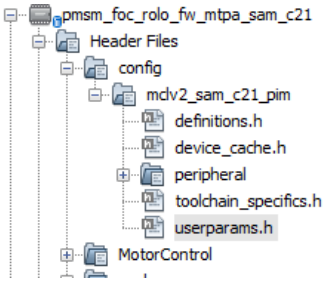
The following timing diagram, shows the chronological role played by the ADC, CPU and TCC0 in the execution of FOC algorithm.



Software Configuration:

The FOC algorithm is used across different application fields. In order to get the optimal control of the PMSM motor, the

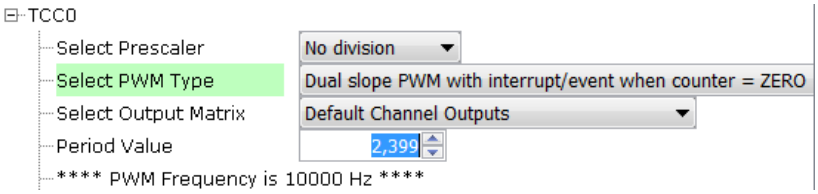
motor specific parameters, board specific parameters and application parameter macros have to be updated in the software. The following section describes how to update both motor and application specific parameters in the the project. These parameter macros can be found in userparams.h header file which 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.

- Field Weakening - FIELD_WEAKENING
  - This optimization feature allows the PMSM motor to rotate at speed higher than rated speed
  - This feature is enabled by setting FIELD_WEAKENING to 1 in userparams.h header file
  - When this feature is enabled, the run-time of the application project increases. Hence, the motor control functions has to be has to be called form RAM by defining RAM_EXECUTE.
- Maximum Torque Per Ampere (MTPA): ENABLE_MTPA
  - This optimization feature allows the PMSM motor to operate at maximum efficiency
  - This feature is enabled by setting ENABLE_MTPA to 1 in userparams.h header file
  - When this feature is enabled, the run-time of the application project increases. Hence, the motor control functions has to be called form RAM by defining RAM_EXECUTE.
- 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 (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.
  - Instruction breakpoint will not be asserted on an instruction which is being executed from RAM. It is recommended to disable this macro while debugging this algorithm using instruction breakpoints.
  - Exercise caution while using this feature as the functions in RAM are vulnerable to get overwritten due to stack overruns or unguarded buffer overruns.

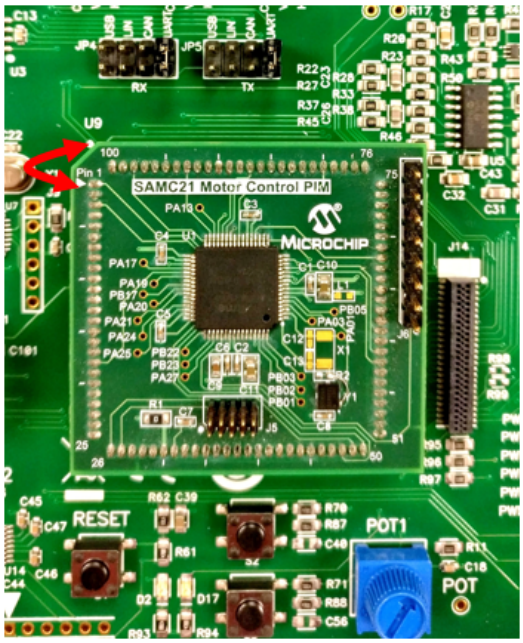
Hardware Setup

This section describes how to configure the supported hardware.

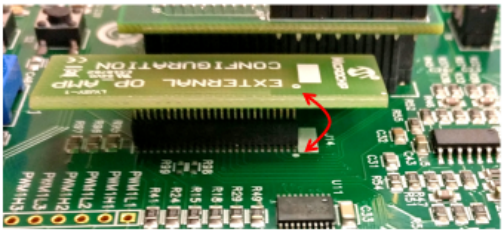
Description

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.

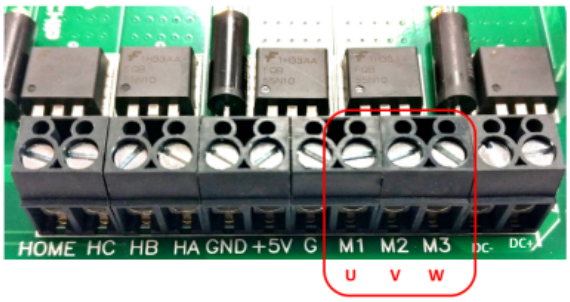


2. Place the "External Opamp Configuration" Matrix board at J14.



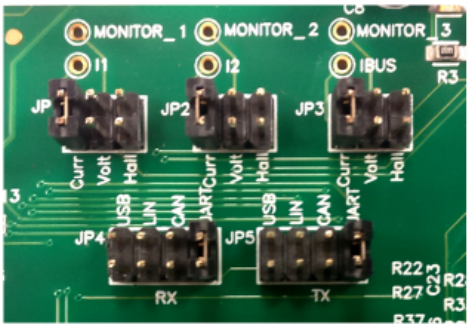
3. Motor Connections:

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

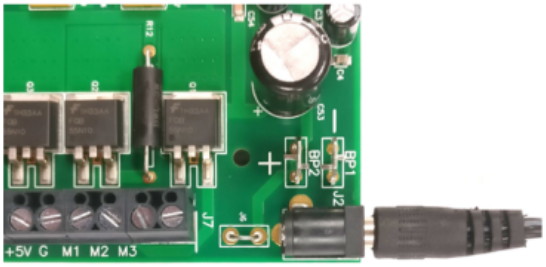


4. Jumper Settings:

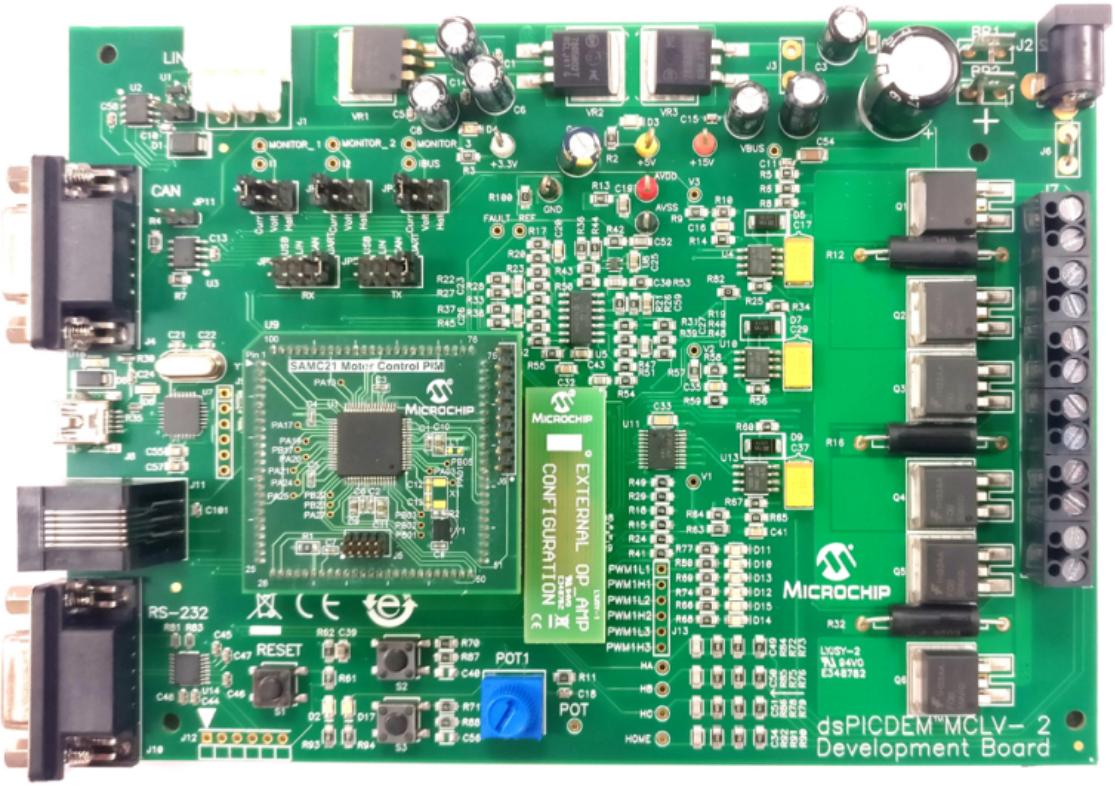
- JP1 - Curr, JP2 - Curr, JP3 - Curr
- In order to use RS232 port for X2CScope Communication JP4 - UART, JP5 - UART
- In order to use USB port for X2CScope Communication JP4 - USB, JP5 - USB



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**



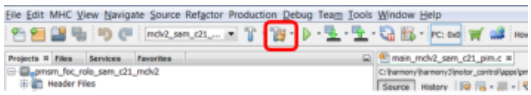
Running The Application

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

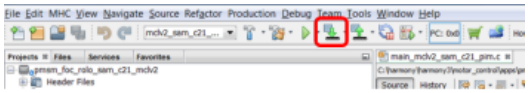
Description

Running the Application : pmsm_foc_rolo_fw_mtpa_sam_c21 - MCLV2

1. Open the project in MPLAB X IDE from :
   <install-directory>\motor_control\apps\pmsm_foc_rolo_fw_mtpa_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_fw_mtpa_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
`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

ACIM VHz Open loop 4
acim_vhz_sam_c21 4
Applications Help 4

B

BLDC Block Commutation Using Hall Sensors 105
bldc_bc_hall_sam_c21 105
Building The Application 5, 16, 31, 47, 56, 65, 80, 95, 105, 114, 124

C

Control Algorithm 6, 17, 33, 48, 58, 67, 82, 96, 107, 116, 126

F

FOC Based Position Control of PMSM Using Quadrature Encoder 114

H

Hardware Setup 12, 23, 40, 53, 62, 73, 89, 101, 112, 121, 132

M

Motor Control Overview 2
MPLAB Harmony Configurations 5, 17, 31, 47, 57, 66, 80, 95, 106, 115, 125

P

PFC in PLL based PMSM drive 95
PMSM FOC Using PLL Estimator 16
PMSM FOC Using Quadrature Encoder 46
PMSM FOC Using Reduced Order Luenberger Observer 65
PMSM FOC Using Reduced Order Luenberger Observer with Windmilling Capability 80
PMSM FOC Using ROLO with Field Weakening and MTPA Capability 124
pmsm_foc_encoder_position_sam_e54 114
pmsm_foc_encoder_sam_e54 56
pmsm_foc_encoder_sam_e70 47
pmsm_foc_pll_estimator_sam_e54 30
pmsm_foc_pll_estimator_sam_e70 16
pmsm_foc_rolo_fw_mtpa_sam_c21 124
pmsm_foc_rolo_sam_c21 65
pmsm_foc_rolo_wm_sam_c21 80
pmsm_pfc_foc_pll_estimator_sam_e70 95

R

Running The Application 15, 30, 45, 56, 64, 78, 93, 104, 113, 123, 135

X

X2CScope Plugin 2