Texas Instruments, Inc.
C2000 Systems and Applications

Sensorless Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors with CLA

Authors: Bilal Akin, Manish Bhardwaj

Contents

Abstract

This application note presents a solution to control a permanent magnet synchronous motor (PMSM) using the Control Law Accelerator (CLA) which is a small foot print co processor present which is present on some of the microcontrollers from the C2000 family of MCU. TMS320F2803x devices are part of the family of C2000 microcontrollers which enable cost-effective design of intelligent controllers for three phase motors by reducing the system components and increase efficiency. With these devices it is possible to realize far more precise digital vector control algorithms like the Field Orientated Control (FOC). This algorithm's implementation is discussed in this document. The FOC algorithm maintains efficiency in a wide range of speeds and takes into consideration torque changes with transient phases by processing a dynamic model of the motor. Among the solutions proposed are ways to eliminate the phase current sensors and use an observer for speed sensorless control.

This application note covers the following:

- A theoretical background on field oriented motor control principle.
- Incremental build levels based on modular software blocks
- Experimental results

Introduction

A brushless Permanent Magnet Synchronous motor (PMSM) has a wound stator, a permanent magnet rotor assembly and internal or external devices to sense rotor position. The sensing devices provide position feedback for adjusting frequency and amplitude of stator voltage reference properly to maintain rotation of the magnet assembly. The combination of an inner permanent magnet rotor and outer windings offers the advantages of low rotor inertia, efficient heat dissipation, and reduction of the motor size. Moreover, the elimination of brushes reduces noise, EMI generation and suppresses the need of brushes maintenance.

This document presents a solution to control a permanent magnet synchronous motor using the TMS320F2803x. This new family of DSPs enables cost-effective design of intelligent controllers for brushless motors which can fulfill enhanced operations, consisting of fewer system components, lower system cost and increased performances. The control method presented relies on the field orientated control (FOC). This algorithm maintains efficiency in a wide range of speeds and takes into consideration torque changes with transient phases by controlling the flux directly from the rotor coordinates. This application report presents the implementation of a control for sinusoidal PMSM motor. The sinusoidal voltage waveform applied to this motor is created by using the Space Vector modulation technique. Minimum amount of torque ripple appears when driving this sinusoidal BEMF motor with sinusoidal currents.

Permanent Magnet Motors

There are primarily two types of three-phase permanent magnet synchronous motors (PMSM). One uses rotor windings fed from the stator and the other uses permanent magnets. A motor fitted with rotor windings, requires brushes to obtain its current supply and generate rotor flux. The contacts are made of rings and have any commutator segments. The drawbacks of this type of structure are maintenance needs and lower reliability. Replacing the common rotor field windings and pole structure with permanent magnets puts the motor into the category of brushless motors. It is possible to build brushless permanent magnet motors with any even number of magnet poles. The use of magnets enables an efficient use of the radial space and replaces the rotor windings, therefore suppressing the rotor copper losses. Advanced magnet materials permit a considerable reduction in motor dimensions while maintaining a very high power density.

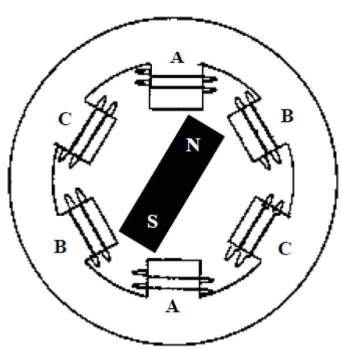


Fig. 1 A three-phase synchronous motor with a one permanent magnet pair pole rotor

Synchronous Motor Operation

- Synchronous motor construction: Permanent magnets are rigidly fixed to the rotating axis to create a constant rotor flux. This rotor flux usually has a constant magnitude. The stator windings when energized create a rotating electromagnetic field. To control the rotating magnetic field, it is necessary to control the stator currents.
- The actual structure of the rotor varies depending on the power range and rated speed of the machine. Permanent magnets are suitable for synchronous machines ranging up-to a few Kilowatts. For higher power ratings the rotor usually consists of windings in which a DC current circulates. The mechanical structure of the rotor is designed for number of poles desired, and the desired flux gradients desired.
- The interaction between the stator and rotor fluxes produces a torque. Since the stator is firmly mounted to the frame, and the rotor is free to rotate, the rotor will rotate, producing a useful mechanical output.
- The angle between the rotor magnetic field and stator field must be carefully controlled to produce maximum torque and achieve high electromechanical conversion efficiency. For this purpose a fine tuning is needed after closing the speed loop using sensorless algorithm in order to draw minimum amount of current under the same speed and torque conditions.
- The rotating stator field must rotate at the same frequency as the rotor permanent magnetic field; otherwise the rotor will experience rapidly alternating positive and negative torque. This will result in less than optimal torque production, and excessive mechanical vibration, noise, and mechanical stresses on the machine parts. In addition, if the rotor inertia prevents the rotor from being able to respond to these oscillations, the rotor will stop rotating at the synchronous frequency, and respond to the average torque as seen by the stationary rotor: Zero. This means that the machine experiences a phenomenon known as 'pull-out'. This is also the reason why the synchronous machine is not self starting.
- The angle between the rotor field and the stator field must be equal to 90º to obtain the highest mutual torque production. This synchronization requires knowing the rotor position in order to generate the right stator field.
- The stator magnetic field can be made to have any direction and magnitude by combining the contribution of different stator phases to produce the resulting stator flux.

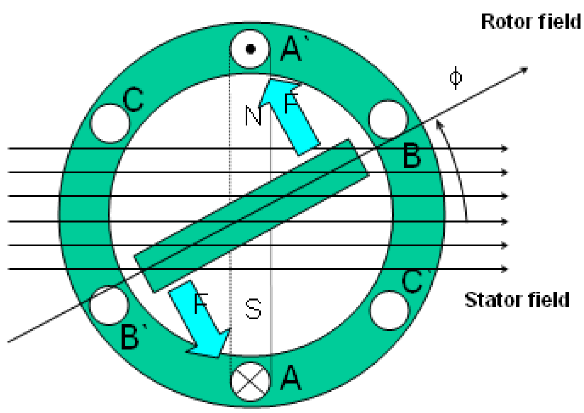


Fig. 2 The interaction between the rotating stator flux, and the rotor flux produces a torque which will cause the motor to rotate.

Field Oriented Control

Introduction

In order to achieve better dynamic performance, a more complex control scheme needs to be applied, to control the PM motor. With the mathematical processing power offered by the microcontrollers, we can implement advanced control strategies, which use mathematical transformations in order to decouple the torque generation and the magnetization functions in PM motors. Such de-coupled torque and magnetization control is commonly called rotor flux oriented control, or simply Field Oriented Control (FOC).

The main philosophy behind the FOC

In order to understand the spirit of the Field Oriented Control technique, let us start with an overview of the separately excited direct current (DC) Motor. In this type of motor, the excitation for the stator and rotor is independently controlled. **Electrical study of the DC motor shows that the produced torque and the flux can be independently tuned**. The strength of the field excitation (i.e. the magnitude of the field excitation current) sets the value of the flux. The current through the rotor windings determines how much torque is produced. The commutator on the rotor plays an interesting part in the torque production. The commutator is in contact with the brushes, and the mechanical construction is designed to switch into the circuit the windings that are mechanically aligned to produce the maximum torque. This arrangement then means that the torque production of the machine is fairly near optimal all the time. **The key point here is that the windings are managed to keep the flux produced by the rotor windings orthogonal to the stator field**.

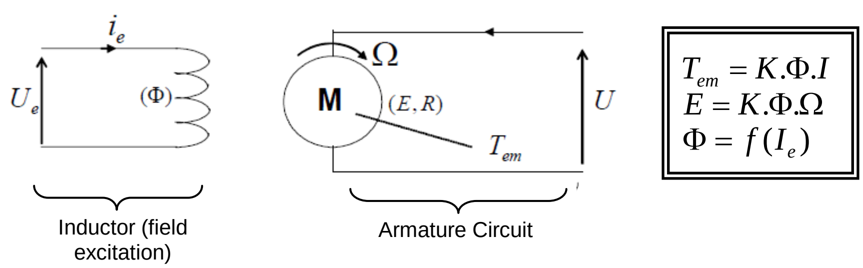


Fig 3. Separated excitation DC motor model, flux and torque are independently controlled and the current through the rotor windings determines how much torque is produced.

AC machines do not have the same key features as the DC motor. In both cases we have only one source that can be controlled which is the stator currents. On the synchronous machine, the rotor excitation is given by the permanent magnets mounted onto the shaft. On the synchronous motor, the only source of power and magnetic field is the stator phase voltage. Obviously, as opposed to the DC motor, flux and torque depend on each other.

The goal of the FOC (also called vector control) on synchronous and asynchronous machine is to be able to separately control the torque producing and magnetizing flux components. The control technique goal is to (in a sense), imitate the DC motor's operation. FOC control will allow us to decouple the torque and the magnetizing flux components of stator current. With decoupled control of the magnetization, the torque producing component of the stator flux can now be thought of as independent torque control. To decouple the torque and flux, it is necessary to engage several mathematical transforms, and this is where the microcontrollers add the most value. The processing capability provided by the microcontrollers enables these mathematical transformations to be carried out very quickly. This in turn implies that the entire algorithm controlling the motor can be executed at a fast rate, enabling higher dynamic performance. In addition to the decoupling, a dynamic model of the

motor is now used for the computation of many quantities such as rotor flux angle and rotor speed. This means that their effect is accounted for, and the overall quality of control is better.
According to the electromagnetic laws, the torque produced in the synchronous machine is equal to vector cross product of the two existing magnetic fields:

$$T_{em} = \vec{B}_{stator} \times \vec{B}_{rotor}$$

This expression shows that the torque is maximum if stator and rotor magnetic fields are orthogonal meaning if we are to maintain the load at 90 degrees. If we are able to ensure this condition all the time, if we are able to orient the flux correctly, we reduce the torque ripple and we ensure a better dynamic response. However, the constraint is to know the rotor position: this can be achieved with a position sensor such as incremental encoder. For low-cost application where the rotor is not accessible, different rotor position observer strategies are applied to get rid of position sensor.

In brief, the goal is to maintain the rotor and stator flux in quadrature: the goal is to align the stator flux with the q axis of the rotor flux, i.e. orthogonal to the rotor flux. To do this the stator current component in quadrature with the rotor flux is controlled to generate the commanded torque, and the direct component is set to zero. The direct component of the stator current can be used in some cases for field weakening, which has the effect of opposing the rotor flux, and reducing the back-emf, which allows for operation at higher speeds.

Technical Background

The Field Orientated Control consists of controlling the stator currents represented by a vector. This control is based on projections which transform a three phase time and speed dependent system into a two co-ordinate (d and q co-ordinates) time invariant system. These projections lead to a structure similar to that of a DC machine control. Field orientated controlled machines need two constants as input references: the torque component (aligned with the q co-ordinate) and the flux component (aligned with d co-ordinate). As Field Orientated Control is simply based on projections the control structure handles instantaneous electrical quantities. This makes the control accurate in every working operation (steady state and transient) and independent of the limited bandwidth mathematical model. The FOC thus solves the classic scheme problems, in the following ways:

- The ease of reaching constant reference (torque component and flux component of the stator current)
- The ease of applying direct torque control because in the (d,q) reference frame the expression of the torque is:

$$m \propto \psi_R \, i_{Sq}$$

By maintaining the amplitude of the rotor flux (ψ_R) at a fixed value we have a linear relationship between torque and torque component (i_{Sq}). We can then control the torque by controlling the torque component of stator current vector.

Space Vector Definition and Projection

The three-phase voltages, currents and fluxes of AC-motors can be analyzed in terms of complex space vectors. With regard to the currents, the space vector can be defined as follows. Assuming that i_a, i_b, i_c are the instantaneous currents in the stator phases, then the complex stator current vector $\bar{i}_s$ is defined by:

$$\bar{i}_s = i_a + \alpha \, i_b + \alpha^2 i_c$$

where $\alpha = e^{j\frac{2}{3}\pi}$ and $\alpha^2 = e^{j\frac{4}{3}\pi}$, represent the spatial operators. The following diagram shows the stator current complex space vector:

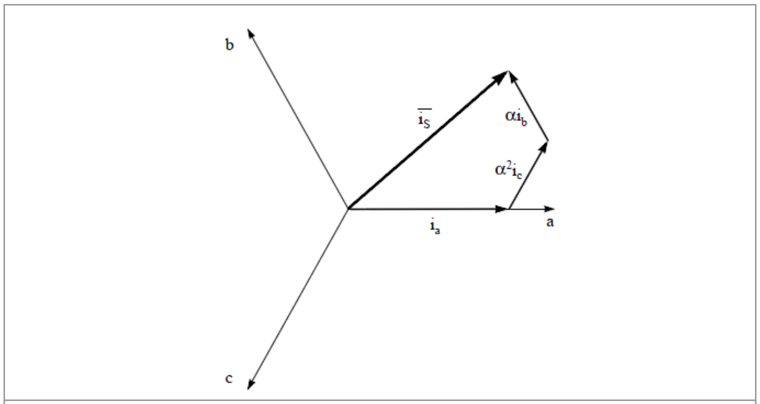


Fig.4 Stator current space vector and its component in (a,b,c)

where (a,b,c) are the three phase system axes. This current space vector depicts the three phase sinusoidal system. It still needs to be transformed into a two time invariant co-ordinate system. This transformation can be split into two steps:

- (a,b,c) $\Rightarrow (\alpha, \beta)$ (the Clarke transformation) which outputs a two co-ordinate time variant system
- $(\alpha, \beta) \Rightarrow$ (d,q) (the Park transformation) which outputs a two co-ordinate time invariant system

The (a,b,c) $\Rightarrow (\alpha, \beta)$ Projection (Clarke transformation)

The space vector can be reported in another reference frame with only two orthogonal axis called (α, β). Assuming that the axis a and the axis α are in the same direction we have the following vector diagram:

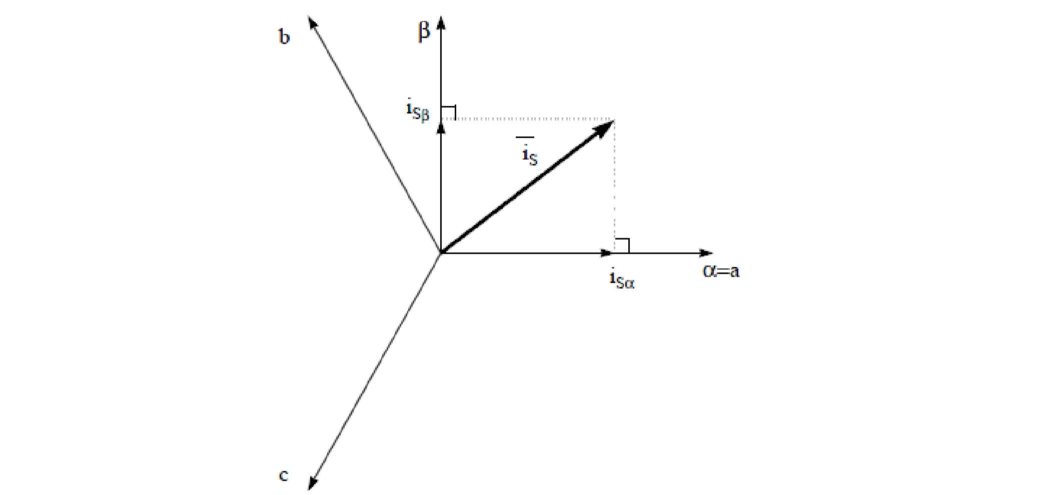


Fig.5 Stator current space vector and its components in the stationary reference frame

The projection that modifies the three phase system into the (α, β) two dimension orthogonal system is presented below.

$$\begin{cases} i_{s\alpha} = i_a \\ i_{s\beta} = \dfrac{1}{\sqrt{3}} i_a + \dfrac{2}{\sqrt{3}} i_b \end{cases}$$

The two phase (α, β) currents are still depends on time and speed.

The $(\alpha, \beta) \Rightarrow$ (d,q) Projection (Park Transformation)

This is the most important transformation in the FOC. In fact, this projection modifies a two phase orthogonal system (α, β) in the d,q rotating reference frame. If we consider the d axis aligned with the rotor flux, the next diagram shows, for the current vector, the relationship from the two reference frame:

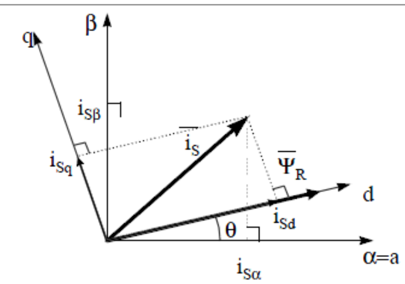


Fig.6 Stator current space vector and its component in (α, β) and in the d,q rotating reference frame

where θ is the rotor flux position. The flux and torque components of the current vector are determined by the following equations:

$$\begin{cases} i_{sd} = i_{s\alpha} \cos\theta + i_{s\beta} \sin\theta \\ i_{sq} = -i_{s\alpha} \sin\theta + i_{s\beta} \cos\theta \end{cases}$$

These components depend on the current vector (α, β) components and on the rotor flux position; if we know the right rotor flux position then, by this projection, the d,q component becomes a constant. Two phase currents now turn into dc quantity (time-invariant). At this point the torque control becomes easier where constant i_{sd} (flux component) and i_{sq} (torque component) current components controlled independently.

The Basic Scheme for the FOC

The following diagram summarizes the basic scheme of torque control with FOC:

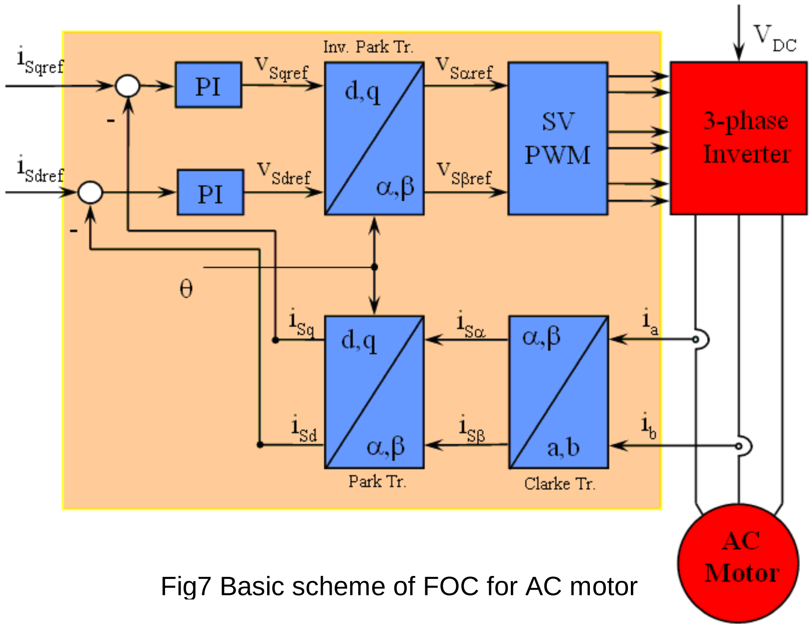


Fig7 Basic scheme of FOC for AC motor

Two motor phase currents are measured. These measurements feed the Clarke transformation module. The outputs of this projection are designated $i_{s\alpha}$ and $i_{s\beta}$. These two components of the current are the inputs of the Park transformation that gives the current in the d,q rotating reference frame. The i_{sd} and i_{sq} components are compared to the references i_{sdref} (the flux reference) and i_{sqref} (the torque reference). At this point, this control structure shows an interesting advantage: it can be used to control either synchronous or HVPM machines by simply changing the flux reference and obtaining rotor flux position. As in synchronous permanent magnet a motor, the rotor flux is fixed determined by the magnets; there is no need to create one. Hence, when controlling a PMSM, i_{sdref} should be set to zero. As HVPM motors need a rotor flux creation in order to operate, the flux reference must not be zero. This conveniently solves one of the major drawbacks of the "classic" control structures: the portability from asynchronous to synchronous drives. The torque command i_{sqref} could be the output of the speed regulator when we use a speed FOC. The outputs of the current regulators are V_{sdref} and V_{sqref}; they are applied to the inverse Park transformation. The outputs of this projection are $V_{s\alpha ref}$ and $V_{s\beta ref}$ which are the components of the stator vector voltage in the (α, β) stationary orthogonal reference frame. These are the inputs of the Space Vector PWM. The outputs of this block are the signals that drive the inverter. Note that both Park and inverse Park transformations need the rotor flux position. Obtaining this rotor flux position depends on the AC machine type (synchronous or asynchronous machine). Rotor flux position considerations are made in a following paragraph.

Rotor Flux Position

Knowledge of the rotor flux position is the core of the FOC. In fact if there is an error in this variable the rotor flux is not aligned with d-axis and i_{sd} and i_{sq} are incorrect flux and torque components of the stator current. The following diagram shows the (a,b,c), (α, β) and (*d,q*) reference frames, and the correct position of the rotor flux, the stator current and stator voltage space vector that rotates with d,q reference at synchronous speed.

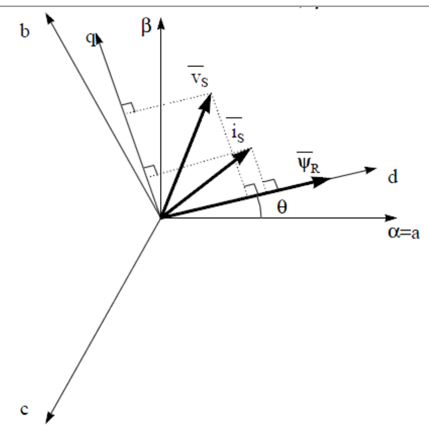


Fig.8 Current, voltage and rotor flux space vectors in the d,q rotating reference frame and their relationship with a,b,c and (α, β) stationary reference frame

The measure of the rotor flux position is different if we consider synchronous or asynchronous motor:

- In the synchronous machine the rotor speed is equal to the rotor flux speed. Then θ (rotor flux position) is directly measured by position sensor or by integration of rotor speed.
- In the asynchronous machine the rotor speed is not equal to the rotor flux speed (there is a slip speed), then it needs a particular method to calculate θ. The basic method is the use of the current model which needs two equations of the motor model in *d,q* reference frame.

Theoretically, the field oriented control for the PMSM drive allows the motor torque be controlled independently with the flux like DC motor operation. In other words, the torque and flux are decoupled from each other. The rotor position is required for variable transformation from stationary reference frame to synchronously rotating reference frame. As a result of this transformation (so called Park transformation), q-axis current will be controlling torque while d-axis current is forced to zero. Therefore, the key module of this system is the estimation of rotor position using Sliding-mode observer. The overall block diagram of this project is depicted in Fig. 9.

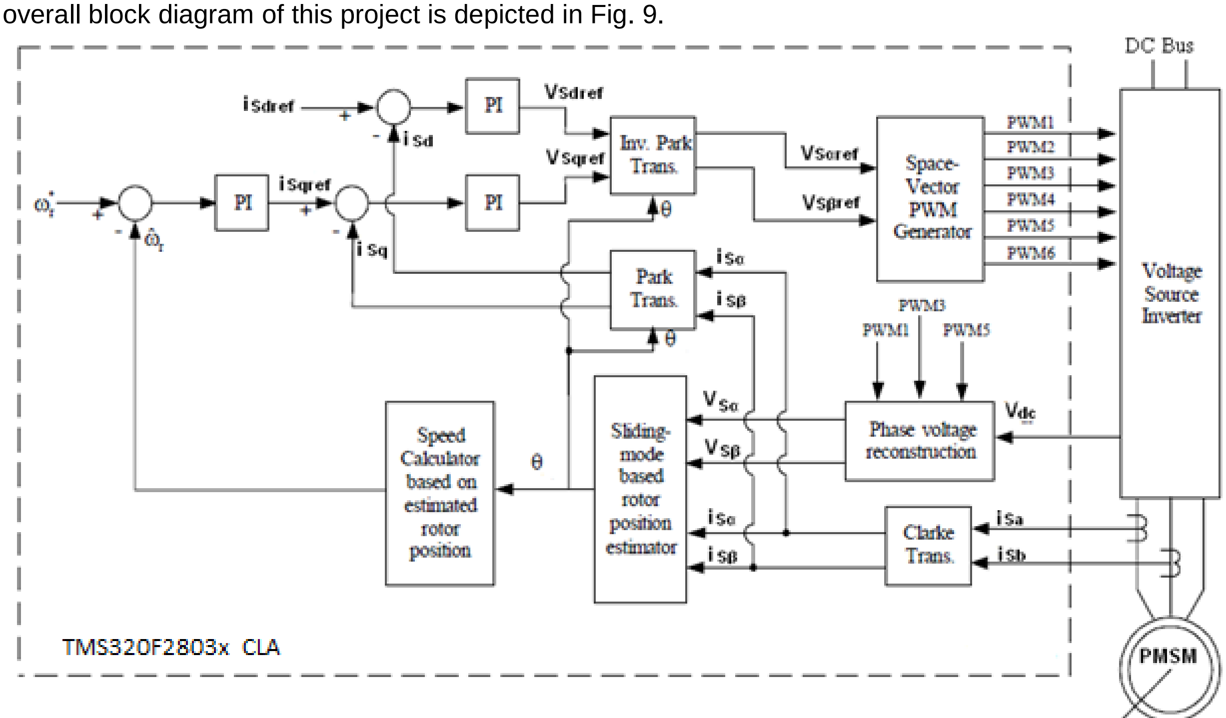


Fig.9 Overall block diagram of sensorless field oriented control of PMSM

Benefits of 32-bit C2000 Controllers for Digital Motor Control (DMC)

C2000 family of devices posses the desired computation power to execute complex control algorithms along with the right mix of peripherals to interface with the various components of the DMC hardware like the ADC, ePWM, QEP, eCAP etc. These peripherals have all the necessary hooks for implementing systems which meet safety requirements, like the trip zones for PWMs and comparators. Along with this the C2000 ecosystem of software (libraries and application software) and hardware (application kits) help in reducing the time and effort needed to develop a Digital Motor Control solution. The DMC Library provides configurable blocks that can be reused to implement new control strategies.

Thus, with C2000 family of devices it is easy and quick to implement complex control algorithms (sensored and sensorless) for motor control. The use of C2000 devices and advanced control schemes provides the following system improvements

- Favors system cost reduction by an efficient control in all speed range implying right dimensioning of power device circuits
- Use of advanced control algorithms it is possible to reduce torque ripple, thus resulting in lower vibration and longer life time of the motor
- Advanced control algorithms reduce harmonics generated by the inverter thus reducing filter cost.
- Use of sensorless algorithms eliminates the need for speed or position sensor.
- Decreases the number of look-up tables which reduces the amount of memory required
- The Real-time generation of smooth near-optimal reference profiles and move trajectories, results in better-performance
- Generation of high resolution PWM's is possible with the use of ePWM peripheral for controlling the power switching inverters
- Provides single chip control system

For advanced controls, C2000 controllers can also perform the following:

- Enables control of multi-variable and complex systems using modern intelligent methods such as neural networks and fuzzy logic.
- Performs adaptive control. C2000 controllers have the speed capabilities to concurrently monitor the system and control it. A dynamic control algorithm adapts itself in real time to variations in system behaviour.
- Performs parameter identification for sensorless control algorithms, self commissioning, and online parameter estimation update.
- Performs advanced torque ripple and acoustic noise reduction.
- Provides diagnostic monitoring with spectrum analysis. By observing the frequency spectrum of mechanical vibrations, failure modes can be predicted in early stages.
- Produces sharp-cut-off notch filters that eliminate narrow-band mechanical resonance. Notch filters remove energy that would otherwise excite resonant modes and possibly make the system unstable.

TI Literature and CLA DMC Library

Literature distinguishes two types of FOC control:

- Direct FOC control: In this case we try to directly estimate the rotor flux based upon the measurements of terminal voltages and currents.

- Indirect FOC control: in this case the goal is to estimate the slip based upon the motor model in FOC condition and to recalculate the rotor flux angle from the integration of estimated slip and measured rotor speeds. Again knowing the motor parameters, especially rotor time constant, is key in order to achieve the FOC control

In this document the Direct FOC Control is discussed.

The CLA Digital Motor Control (DMC) library is composed of functions represented as blocks. These blocks are categorized as Transforms & Estimators (Clarke, Park, Sliding Mode Observer, Phase Voltage Calculation, and Flux, and Speed Calculators and Estimators), Control (Signal Generation, PID, Space Vector Generation), and Peripheral Drivers (PWM abstraction for multiple topologies and techniques, ADC drivers). Each block is a modular software macro is separately documented with source code, use, and technical theory. Check the folders below for the source codes and explanations of macro blocks:

- C:\TI\controlSUITE\libs\app_libs\motor_control\math_blocks\CLA_v1.0

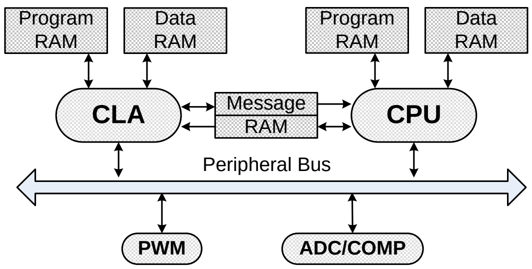
These modules allow users to quickly build, or customize, their own systems. The Library supports the three motor types: ACI, PMSM and comprises both peripheral dependent (software drivers) and target dependent modules.

The DMC Library components have been used by TI to provide system examples. At initialization all DMC Library variables are defined and inter-connected. At run-time the macro functions are called in order. Each system is built using an incremental build approach, which allows some sections of the code to be built at a time, so that the developer can verify each section of their application one step at a time. This is critical in real-time control applications where so many different variables can affect the system and many different motor parameters need to be tuned.

Digital Motor Control on CLA

Control Law Accelerator (CLA) is a small footprint floating point co processor which is present on some microcontrollers from the C2000 family of MCU's by Texas Instruments. Most control algorithms can be split into three tasks: excite the system, sample the system and control the system. Exciting the system for motor control type application implies changing duty cycle of the PWM waveform. Sampling the system involves reading the ADC results values and controlling the system implies computing the control effort. The excite sample and control loop is run at the switching rate of the power stage. Given the complexity of the FOC algorithm, which is typically used for efficient motor control, the CPU is left with little bandwidth to do other tasks such as diagnostics, monitoring etc.

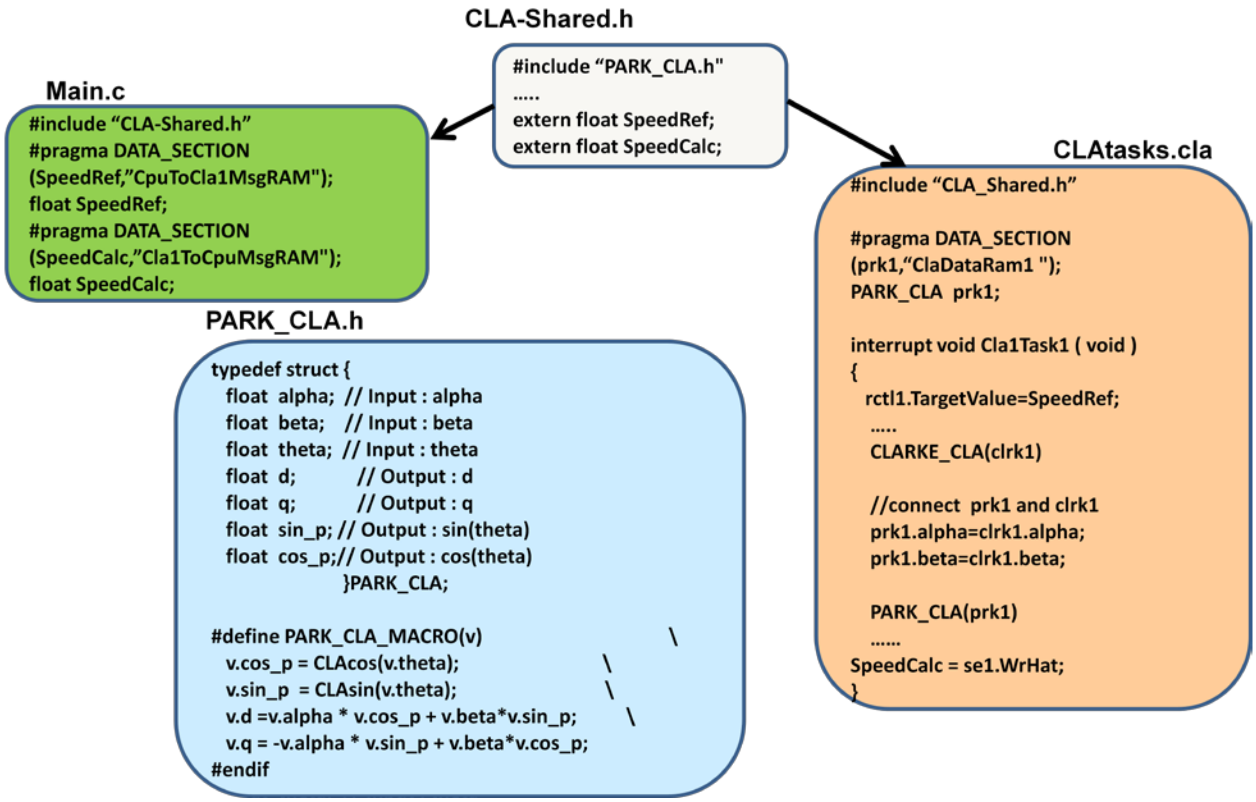
CLA is designed to offload the control task burden from the CPU, thus freeing up bandwidth on the main CPU i.e. (C28x) core. It has access to the control peripherals such as PWM and the ADCs which it shares with the main CPU. The CLA has it's own program and data bus as shown in figure below, and executes independently of the main core. The CLA interacts with the main core with use of message rams and has access to the control peripheral simultaneous to the main CPU.



The CLA however does not have hardware support to support a full standard C- compiler, for example the CLA does not have a stack. Also the instruction set of the CLA is reduced and cannot support all the functions of the standard C – compiler. Therefore a "CLA-C Compiler" which supports most but not all C compiler functions is available for programming the CLA. For example the CLA only supports one level nesting of function calls to avoid overhead in function calls. More details about how to set up the CLA C compiler on your machine visit the following wiki page which details the CCS environment setup and codegen tool necessary for the CLA C compiler.

http://processors.wiki.ti.com/index.php/C2000_CLA_C_Compiler

The CLA-C compiler is integrated into the CCS IDE and files meant to be executed by the cla are identified with *.cla extension. The DMC library for CLA follows the same format as the DMC library for the C28x.



The DMC CLA project is composed of three key files

{ProjectName}-Main.c : This file comprises of all the peripheral initialization and sets up the CLA by assigning it program RAM and dataRAM.

{ProjectName}-Shared.h : This file consists of key variables that are shared between the CPU and the CLA. These variables must be declared in the message RAM's

{ProjectName}-tasks_C.cla : This is the run time file of the CLA, it has 8 task interrupts defined which can be used to execute the algorithm in this task the Task 8 is reserved for initialization of the variables that are in the CLA writable data space.

The DMC blocks are in form of macros that are similar to the DMC C28x library with changes made to the trigonometric functions. The macro approach matches well with the CLA C compiler as it only allows 2 level of nesting. The CLA math library is used to realize the trigonometric functions.

All the DMC library variables are declared in the CLA data ram, as these need to be computed by the CLA. {Note these variables will not be visible on the C28x side, however if the JTAG is connected the variables can be observed on the watch window}. The variable such as SpeedRef, lsw(LoopSwitch) are declared in the CpuToCLAMsgRAM, whereas the variables such as calculated speed that needs to be monitored by the CPU is declared in the CLAtoCPUMsgRAM.

System Overview

This document describes the “C” real-time control framework used to demonstrate the sensorless field oriented control of HVPM motors. The “C” framework is designed to run on TMS320C2803x based controllers on Code Composer Studio. The framework uses the following modules[1]:

Macro Names	Explanation
CLARKE	Clarke Transformation
PARK / IPARK	Park and Inverse Park Transformation
PI	PI Regulators
RC	Ramp Controller (slew rate limiter)
RG	Ramp / Sawtooth Generator
SE	Speed Estimation (based on sensorless position estimation)
SMO	Sliding Mode Observer for Sensorless Applications
SVGEN	Space Vector PWM with Quadrature Control (includes IClarke Trans.)
PHASEVOLT	Phase Voltage Calculator
PWM / PWMDAC	PWM and PWMDAC Drives
[1] Please refer to pdf documents in motor control folder explaining the details and theoretical background of each macro	

In this system, the sensorless Field Oriented Control (FOC) of Permanent Magnet Synchronous Motor (PMSM) will be experimented with and will explorethe performance of speed control. The PM motor is driven by a conventional voltage-source inverter. The TMS320x2803x control card is used to generate three pulse width modulation (PWM) signals. The motor is driven by an integrated power module by means of space vector PWM technique. Two phase currents of HVPM motor (ia and ib) are measured from the inverter and sent to the TMS320x2803x via two analog-to-digital converters (ADCs). In addition, the DC-bus voltage in the inverter is measured and sent to the TMS320x2803x via an ADC. This DC-bus voltage is necessary to calculate the three phase voltages when the switching functions are known.

HVPM_Sensorless_CLA project has the following properties:

C Framework		
System Name	**Program Memory Usage 2803x**	**Data Memory Usage 2803x**
HVPM_Sensorless_CLA	2956 words	1464 words

CPU Utilization – PMSM Sensorless	
Total Number of Cycles	**818***
CPU Utilization @ 60 Mhz	**13.6%**
CPU Utilization @ 40 Mhz	**20.4%**

* At 10 kHz ISR frequency. Debug macros excluded (i.e. PWMDAC). IQSin/Cos tables used.

System Features	
Development /Emulation	Code Composer Studio V4.0(or above) with Real Time debugging
Target Controller	TMS320F2803x CLA
PWM Frequency	10kHz PWM (Default), 60kHz PWMDAC
PWM Mode	Symmetrical with a programmable dead band
Interrupts	ADC, end of conversion – Implements 10 kHz ISR execution rate
Peripherals Used	PWM 1 / 2 / 3 for motor control PWM 6A, 6B, 7A & 7B for DAC outputs ADC A7 for DC Bus voltage sensing, A1 & B1 for phase current sensing

The overall system implementing a 3-ph HVPM motor control is depicted in Fig.10. The HVPM motor is driven by the conventional voltage-source inverter. The TMS320F2803x is being used to generate the six pulse width modulation (PWM) signals using a space vector PWM technique, for six power switching devices in the inverter. Two input currents of the HVPM motor (ia and ib) are measured from the inverter and they are sent to the TMS320F2803x via two analog-to-digital converters (ADCs). In addition, the DC-bus voltage in the inverter is measured and sent to the TMS320F2803x via an ADC as well. This DC-bus voltage is necessary in order to calculate three phase voltages of HVPM motor when the switching functions are known.

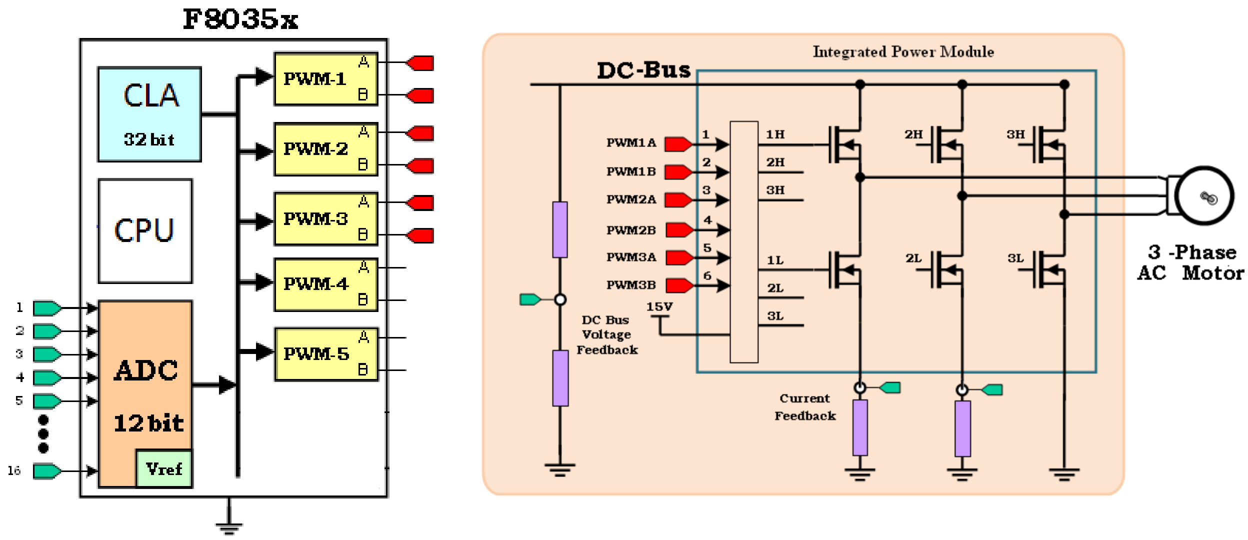


Fig 10 A 3-ph PM motor drive implementation

Hardware Configuration (HVDMC Kit R1.1)

Please refer to the HVMotorCtrl+PFC How to Run Guide found:

C:\TI\controlSUITE\development_kits\HVMotorCtrl+PfcKit_v2.0\~Docs

for an overview of the kit's hardware and steps on how to setup this kit. Some of the hardware setup instructions are captured below for quick reference

HW Setup Instructions

1. Open the Lid of the HV Kit
2. Install the Jumpers [Main]-J3, J4 and J5, J9 for 3.3V, 5V and 15V power rails and JTAG reset line.
3. Unpack the DIMM style controlCARD and place it in the connector slot of [Main]-J1. Push vertically down using even pressure from both ends of the card until the clips snap and lock. (to remove the card simply spread open the retaining clip with thumbs)
4. Connect a USB cable to connector [M3]-JP1. This will enable isolated JTAG emulation to the C2000 device. [M3]-LD1 should turn on. Make sure [M3]-J5 is not populated. If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will automatically be installed. If a windows installation window appears try to automatically install drivers from those already on your computer. The emulation drivers are found at http://www.ftdichip.com/Drivers/D2XX.htm. The correct driver is the one listed to support the FT2232.
5. If a third party JTAG emulator is used, connect the JTAG header to [M3]-J2 and additionally [M3]-J5 needs to be populated to put the onboard JTAG chip in reset.
6. Ensure that [M6]-SW1 is in the "Off" position. Connect 15V DC power supply to [M6]-JP1.
7. Turn on [M6]-SW1. Now [M6]-LD1 should turn on. Notice the control card LED would light up as well indicating the control card is receiving power from the board.
8. Note that the motor should be connected to the [M5]-TB3 terminals after you finish with the first incremental build step.
9. Note the DC Bus power should only be applied during incremental build levels when instructed to do so. The two options to get DC Bus power are discussed below,

(i) To use DC power supply, set the power supply output to zero and connect [Main]-BS5 and BS6 to DC power supply and ground respectively.

(ii) To use AC Mains Power, Connect [Main]-BS1 and BS5 to each other using banana plug cord. Now connect one end of the AC power cord to [Main]-P1. The other end needs to be connected to output of a variac. Make sure that the variac output is set to zero and it is connected to the wall supply through an isolator.

[1] Since the motor is rated at 200V, the motor can run only at a certain speed and torque range properly without saturating the PI regulators in the control loop when the DC bus is fed from 110V AC entry. As an option, the user can run the PFC on HV DMC drive platform as boost converter to increase the DC bus voltage level or directly connect a DC power supply.

For reference the pictures below show the jumper and connectors that need to be connected for this lab.

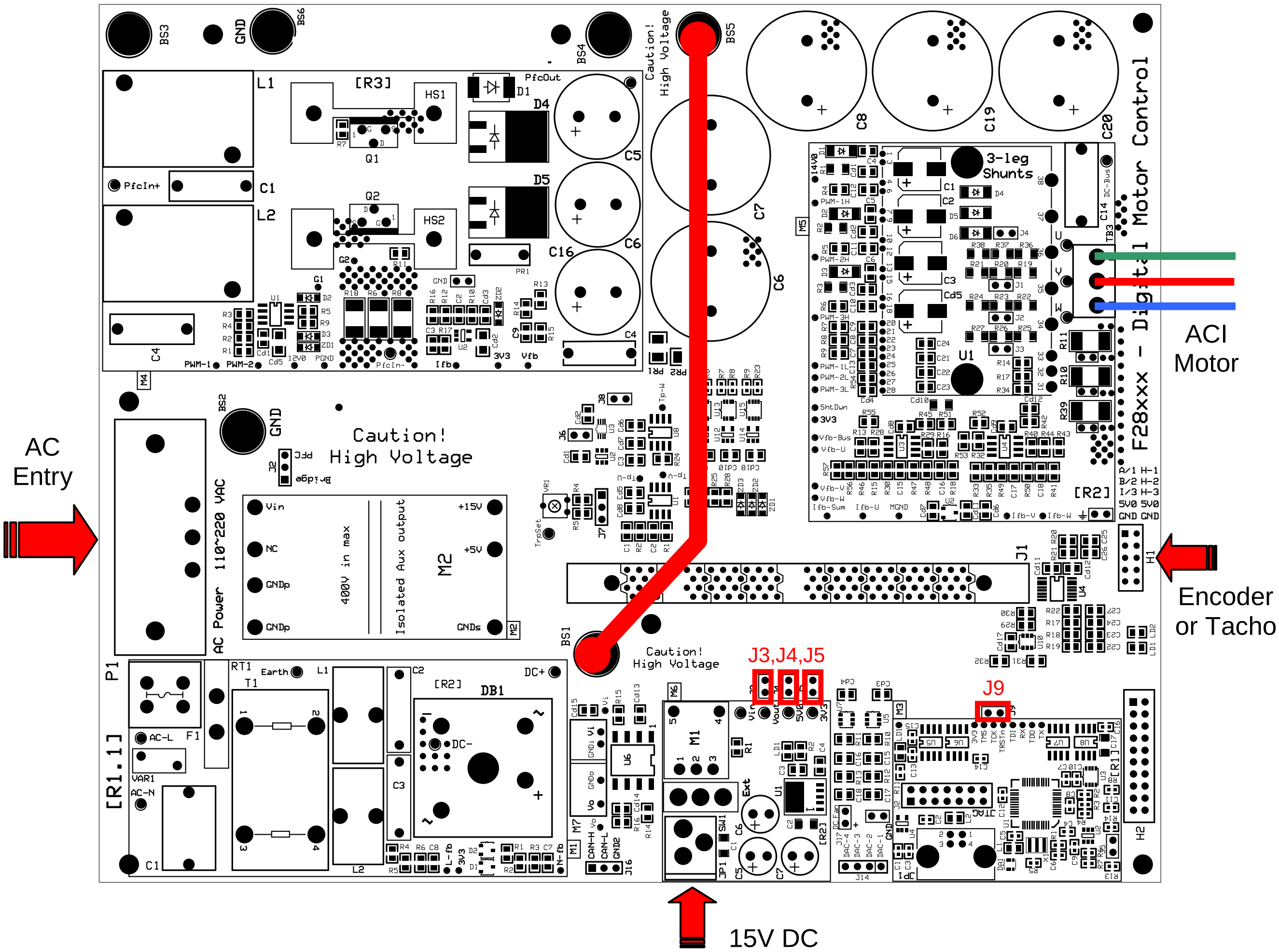


Fig. 12 Using AC Power to generate DC Bus Power

CAUTION: The inverter bus capacitors remain charged for a long time after the high power line supply is switched off/disconnected. Proceed with caution!

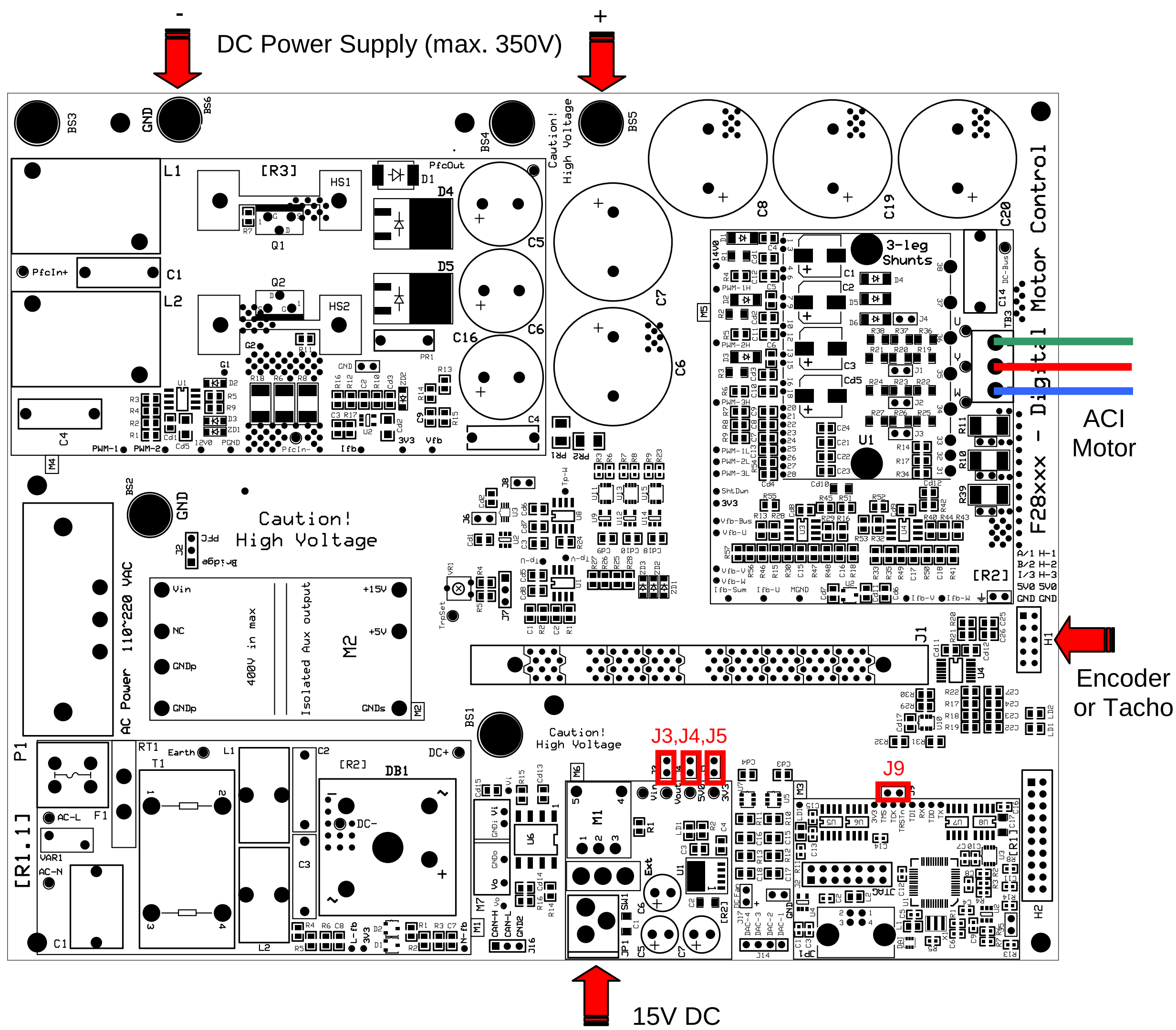


Fig. 13 Using External DC power supply to generate DC-Bus for the inverter

CAUTION: The inverter bus capacitors remain charged for a long time after the high power line supply is switched off/disconnected. Proceed with caution!

Software Setup Instructions to Run HVPM_Sensorless_CLA Project

Please refer to the “Generic Steps for Software Setup for HVMotorCtrl+PFC Kit Projects” section in the HVMotorCtrl+PFC Kit How To Run Guide

C:\TI\controlSUITE\development_kits\HVMotorCtrl+PfcKit_v2.0\~Docs

Select the HVPM_Sensorless_CLA as the active project. Verify that the build level is set to 1. Make sure the code gen version 6.1.0 or later is installed and verify that the CCS environment recognizes *.cla extension (Windows ->preferences -> CCS-> file type). Right click on the project name and select “Rebuild Project”. Once build completes, launch a debug session to load the code into the controller. Now open a watch window and add the critical variables as shown in the table below.

Local (1) | Watch (1)

Name	Value	Address	Type	Format
(x)= IsrTicker	1253.0	0x00008A0C@Data	float	Natural
(x)= EnableFlag	1.0	0x00001500@Data	float	Natural
(x)= Isw	2	0x00001504@Data	int	Natural
(x)= IqRef	0.1	0x00001508@Data	float	Natural
(x)= IdRef	0.0	0x00001506@Data	float	Natural
(x)= SpeedRef	0.3	0x00001502@Data	float	Natural
<new>				

Table 1 Watch Window Variables

Click on Continuous Refresh button on the top left corner of the graph tab to enable periodic capture of data from the microcontroller.

Level 1 Incremental Build

At this step keep the motor disconnected. Assuming the load and build steps described in the "HVMotorCtrl+PFC Kit How To Run Guide" completed successfully, this section describes the steps for a "minimum" system check-out which confirms operation of system interrupt, the peripheral & target independent I_PARK_CLA_MACRO (inverse park transformation) and SVGEN_CLA_MACRO (space vector generator) modules and the peripheral dependent PWMDRV_3PHINV_CLA_MACRO (PWM initializations and update) modules. Open {App Name}_CLA-Shared_C.h and select level 1 incremental build option by setting the BUILDLEVEL to LEVEL1 (#define BUILDLEVEL LEVEL1). Now right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will now keep on increasing, confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are summarized below.
- SpeedRef (float): for changing the rotor speed in per-unit.
- VdTesting (float): for changing the d-qxis voltage in per-unit.
- VqTesting (float): for changing the q-axis voltage in per-unit.

Level 1A (SVGEN_CLA_MACRO Test)

The SpeedRef value is specified to the RG_CLA_MACRO module via RC_CLA_MACRO module. The IPARK_CLA_MACRO module is generating the outputs to the SVGEN_CLA_MACRO module. Three outputs from SVGEN_CLA_MACRO module are monitored via the graph window as shown in Fig. 14 where Ta, Tb, and Tc waveform are 120^o apart from each other. Specifically, Tb lags Ta by 120^o and Tc leads Ta by 120^o. Check the PWM test points on the board to observe PWM pulses (PWM-1H to 3H and PWM-1L to 3L) and make sure that the PWM module is running properly.

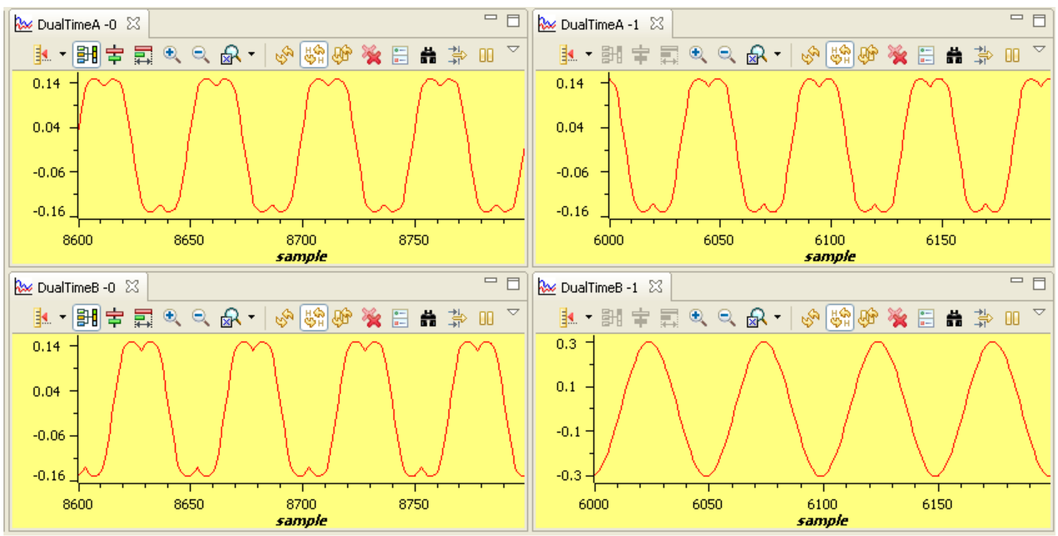


Fig 14 Output of SVGEN, Ta, Tb, Tc and Tb-Tc waveforms*

*Note that the graph window snapshots belong to C28x application. Please use PWMDACs to obtain the same graphs for debugging purposes while using CLA.

Level 1B (testing The PWMDAC Macro)

To monitor internal signal values in real time PWM DACs are very useful tools. Present on the HV DMC board are PWM DAC's which use an external low pass filter to generate the waveforms ([Main]-J14, DAC-1 to 4). A simple 1^{st}–order low-pass filter RC circuit is placed on the board to filter out the high frequency components. The selection of R and C value (or the time constant, τ) is based on the cut-off frequency (f_c), for this type of filter the relation is as follows:

$$\tau = RC = \frac{1}{2\pi f_c}$$

For example, R=1.8kΩ and C=100nF, it gives f_c = 884.2 Hz. This cut-off frequency has to be below the PWM frequency. Using the formula above, one can customize low pass filters used for signal being monitored. The DAC circuit low pass filters ([Main]-R10 to13 & [Main]-C15 to18) is shipped with 2.2kΩ and 220nF on the board. Refer to application note *SPRAA88A* for more details at TI website.

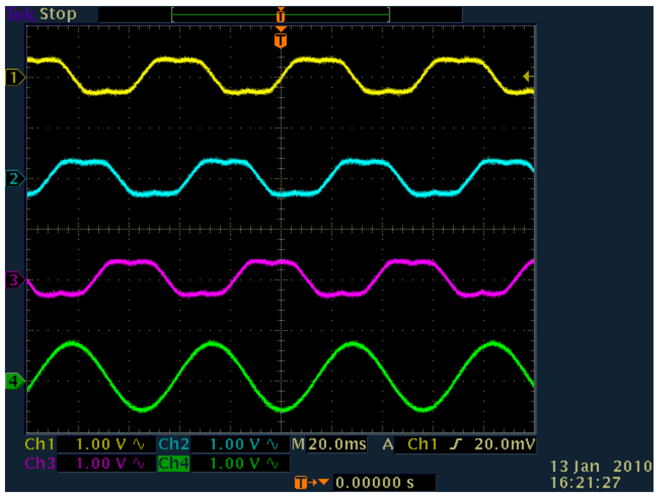


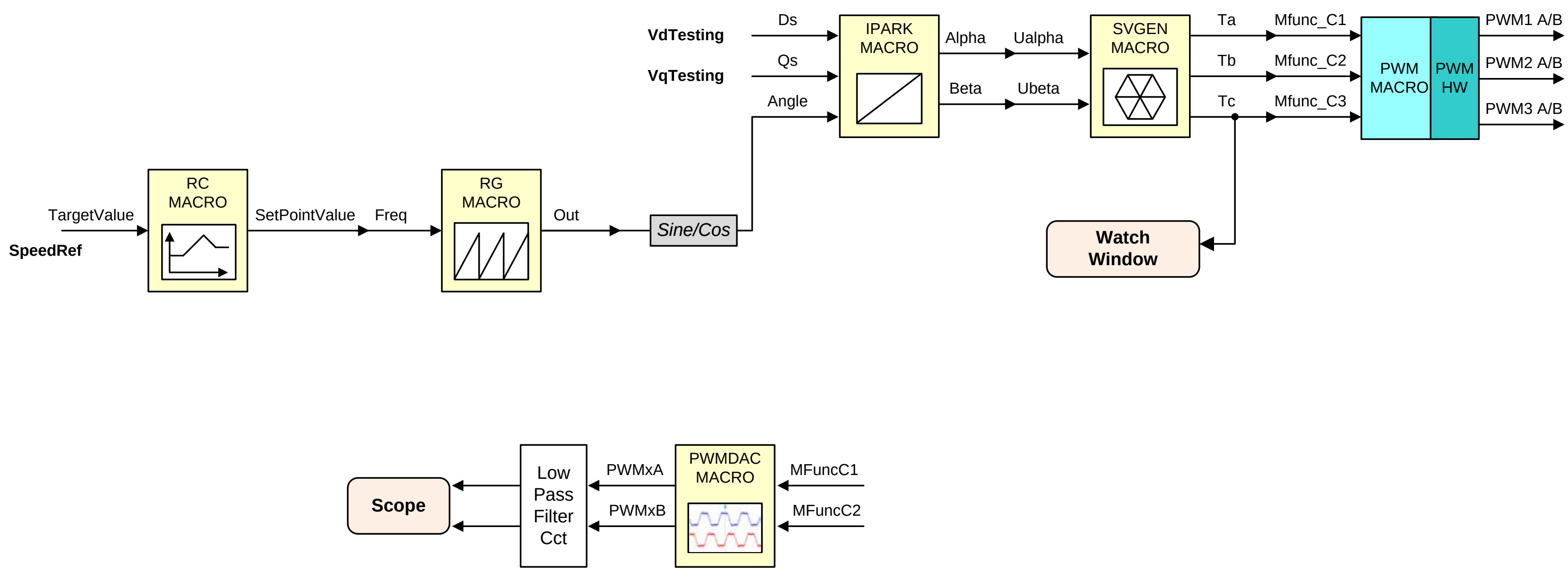
Fig.15 DAC 1-4 outputs showing Ta, Tb, Tc and Tb-Tc waveforms

Level 1C (PWMDRV_3PHINV_CLA_MACRO and INVERTER Test)

After verifying SVGEN_CLA_MACRO module in level 1a, the PWMDRV_3PHINV_CLA_MACRO software module and the 3-phase inverter hardware are tested by looking at the low pass filter outputs. For this purpose, If using the external DC power supply gradually increase the DC bus voltage and check the Vfb-U, V and W test points using an oscilloscope or if using AC power entry slowly change the variac to generate the DC bus voltage. Once the DC Bus voltage is greater than 15 to 20V you would start observing the Inverter phase voltage dividers and waveform monitoring filters (Vfb-U, Vfb-V, Vfb-W) enable the generation of the waveform and ensures theta the inverter is working appropriately. Note that the default RC values are optimized for AC motor state observers employing phase voltages.

After verifying this, reduce the DC Bus voltage, take the controller out of real time mode (disable), reset the processor (see "HVMotorCtrl+PFC Kit How To Run Guide" for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, hence caution needs to be taken while doing these experiments.

Level 1 - Incremental System Build Block Diagram



Level 1 verifies the target independent modules, duty cycles and
PWM updates. The motor is disconnected at this level.

Level 2 Incremental Build

Assuming section BUILD 1 is completed successfully, this section verifies the analog-to-digital conversion, Clarke / Park transformations and phase voltage calculations. Now the motor can be connected to HVDMC board since the PWM signals are successfully proven through level 1 incremental build. Note that the open loop experiments are meant to test the ADCs, inverter stage, sw modules etc. Therefore running motor under load or at various operating points is not recommended.

Open {App Name}_CLA-Shared_C.h and select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL LEVEL2). Now right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will now keep on increasing, confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are summarized below.

- SpeedRef (float): for changing the rotor speed in per-unit.
- VdTesting(float): for changing the d-qxis voltage in per-unit.
- VqTesting(float): for changing the q-axis voltage in per-unit.

During the open loop tests, VqTesting, SpeedRef and DC Bus voltages should be adjusted carefully for PM motors so that the generated Bemf is lower than the average voltage applied to motor winding. This will prevent the motor from stalling or vibrating.

Level 2A – Testing the Phase Voltage module

In this part, the phase voltage calculation module, VOLT_CALC_CLA_MACRO, will be tested. Now, gradually increase the DC bus voltage. The outputs of this module can be checked via the graph window as follows:

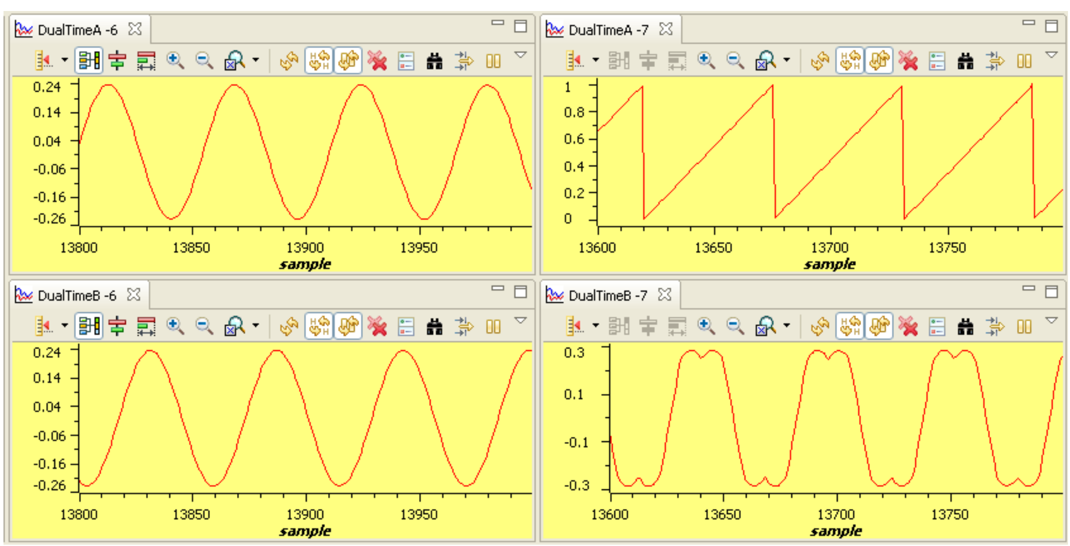


Fig 16 Calculated phase A&B voltages by volt1 module, rg1.Out and svgen_dq1.Ta

- The VphaseA, VphaseB, and VphaseC waveforms should be 120^{o} apart from each other. Specifically, VphaseB lags VphaseA by120^{o} and VphaseC leads VphaseA by 120.
- The Valpha waveform should be same as the VphaseA waveform.
- The Valpha waveform should be leading the Vbeta waveform by 90^{o} at the same magnitude.

Phase 2B – Testing the Clarke module

In this part the Clarke module will be tested. The three measured line currents are transformed to two phase dq currents in a stationary reference frame. The outputs of this module can be checked from graph window.

- The clarke1.Alpha waveform should be same as the clarke1.As waveform.
- The clarke1.Alpha waveform should be leading the clarke1.Beta waveform by 90^{o} at the same magnitude.

It is important that the measured line current must be lagging with the reconstructing phase voltage because of the nature of the AC motor. This can be easily checked as follows:

- The clarke1.Alpha waveform should be lagging the Valpha waveform at an angle by nature of the reactive load of motor.
- The clarke1.Beta waveform should be lagging the Vbeta waveform at the same angle.

If the clarke1.Alpha and Valpha or clarke1.Beta and Vbeta waveforms in the previous step are not truly affecting the lagging relationship, then set OutofPhase to 1 at the beginning of the VOLT_CALC_CLA_MACRO module. The outputs of this test can be checked via the graph window.

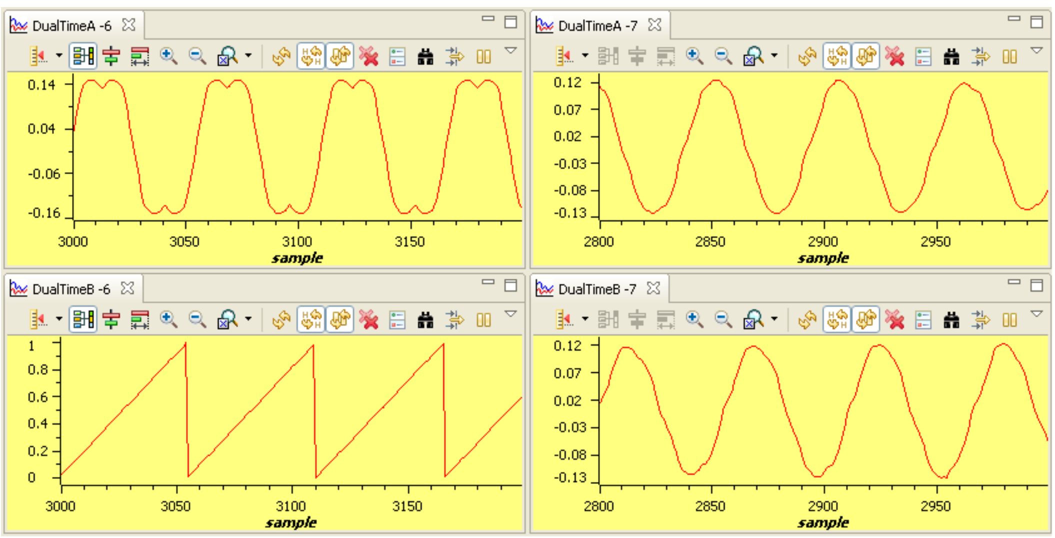


Fig 17 The waveforms of Svgen_dq1.Ta, rg1.Out, and phase A&B currents*

* Deadband = 1.66 usec, Vdcbus=300V , dlog.prescalar=3

Level 2C – Adjusting PI Limits

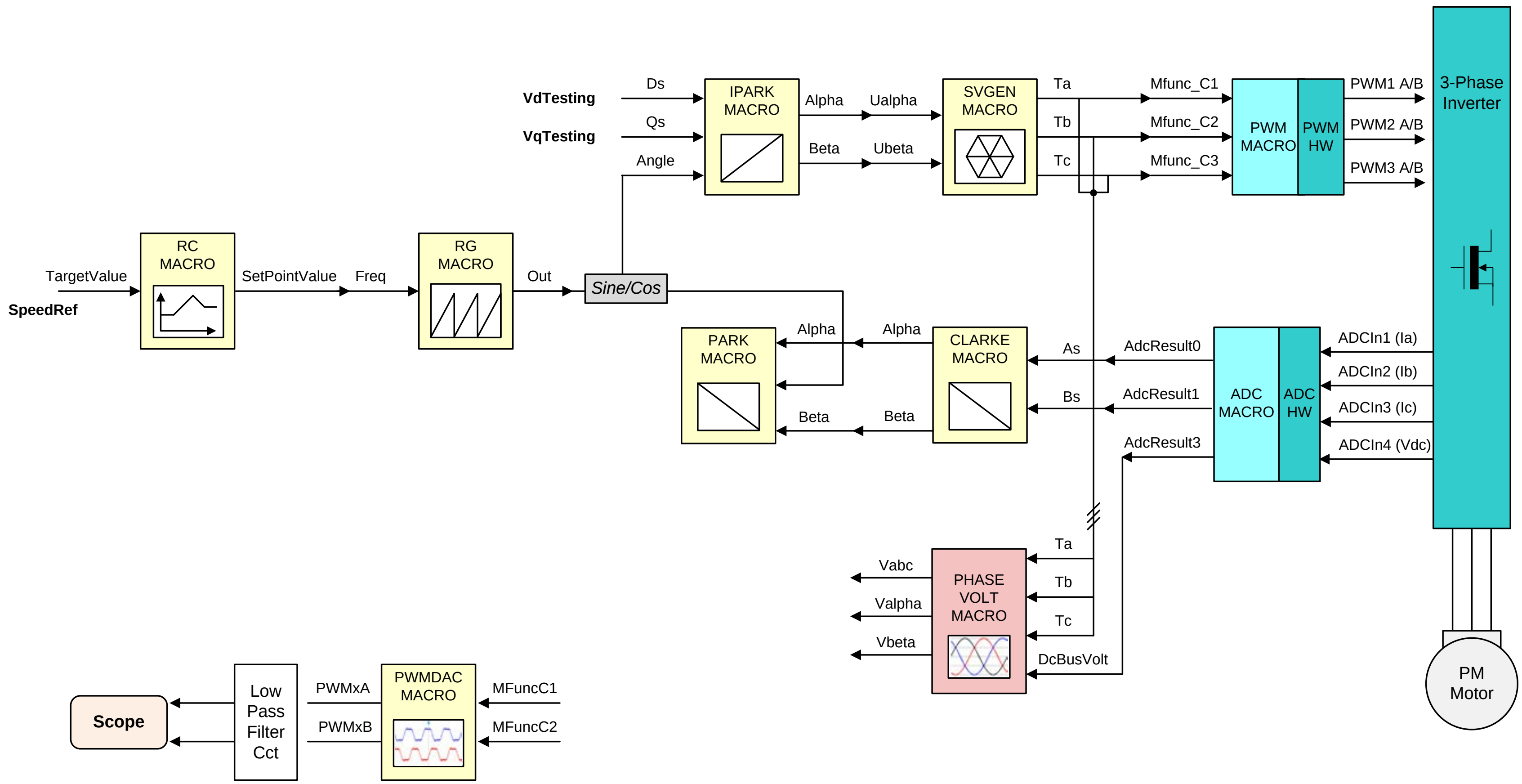
Note that the vectorial sum of d-q PI outputs should be less than 1.0 which refers to maximum duty cycle for SVGEN macro. Another duty cycle limiting factor is the current sense through shunt resistors which depends on hardware/software implementation. Depending on the application requirements 3, 2 or a single shunt resistor can be used for current waveform reconstruction. The higher number of shunt resistors allow higher duty cycle operation and better dc bus utilization.

Run the system with default VdTesting, VqTesting and SpeedRef and gradually increase VdTesting and VqTesting values. Meanwhile, watch the current waveforms in the graph window. Keep increasing until you notice distorted current waveforms and write down the maximum allowed VdTesting and VqTesting values. Make sure that these values are consistent with expected d-q current component maximums while running the motor. After this build level, PI outputs will automatically generate the voltage reference and determine the PWM duty cycle depending on the d-q current demand, therefore set pi_id.Umax/min and pi_iq.Umax/min according to recorded VdTesting and VqTesting values respectively.

Running motor without proper PI limits can yield distorted current waveforms and unstable closed loop operations which may damage the hardware.

Bring the system to a safe stop as described at the end of build 1 by reducing the bus voltage, taking the controller out of realtime mode and reset.

Level 2 - Incremental System Build Block Diagram



Level 2 verifies the analog-to-digital conversion, offset compensation, clarke / park transformations, phase voltage calculations.

Level 3 Incremental Build

Assuming the previous section is completed successfully, this section verifies the dq-axis current regulation performed by PI modules and speed measurement modules (optional). To confirm the operation of current regulation, the gains of these two PI controllers are necessarily tuned for proper operation.

Open {App Name}_CLA-Shared_C.h and select level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3). Now right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will now keep on increasing, confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are summarized below.

- SpeedRef (float): for changing the rotor speed in per-unit.
- IdRef(float): for changing the d-qxis voltage in per-unit.
- IqRef(float): for changing the q-axis voltage in per-unit.

In this build, the motor is supplied by AC input voltage and the (AC) motor current is dynamically regulated by using PI module through the park transformation on the motor currents.

The steps are explained as follows:

- Compile/load/run program with real time mode.

- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different), Idref to zero and Iqref to 0.05 pu.

- Gradually increase voltage at variac / dc power supply to get an appropriate DC-bus voltage.

- Add the soft-switch variable “lsw” to the watch window in order to switch from current loop to speed loop. In the code lsw manages the loop setting as follows:

- lsw=0, lock the rotor of the motor.
- lsw=1, run the motor with closed current loop.

- Check pi_id.Fdb in the watch windows with continuous refresh feature whether or not it should be keeping track pi_id.Ref for PI module. If not, adjust its PI gains properly.

- Check pi_iq.Fdb in the watch windows with continuous refresh feature whether or not it should be keeping track pi_iq.Ref for PI module. If not, adjust its PI gains properly.

- To confirm these two PI modules, try different values of pi_id.Ref and pi_iq.Ref or SpeedRef.

- For both PI controllers, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied responses.

- Bring the system to a safe stop as described at the end of build 1 by reducing the bus voltage, taking the controller out of realtime mode and reset. Now the motor should stop, once stopped terminate the debug session.

During running this build, the current waveforms in the CCS graphs should appear as follows*:

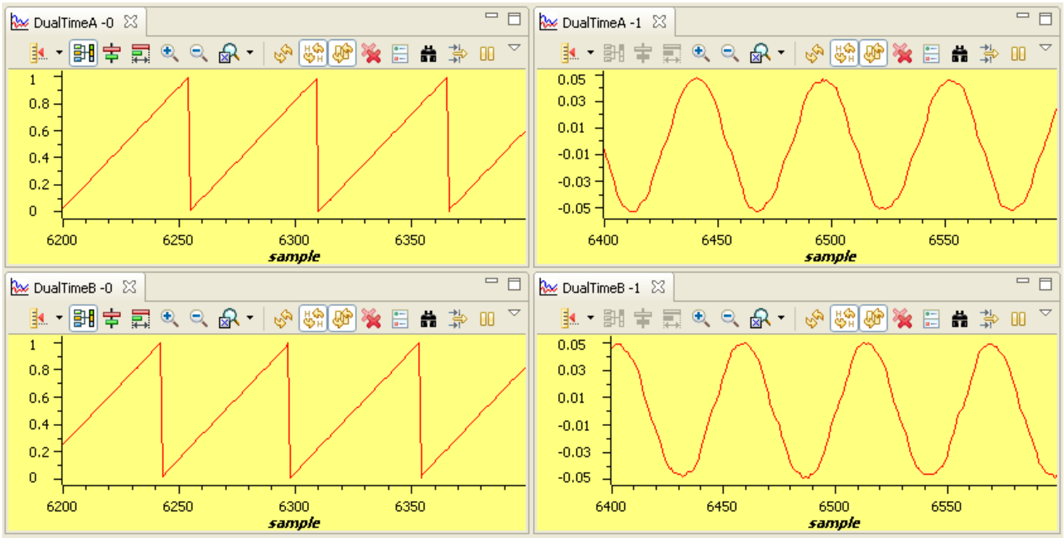
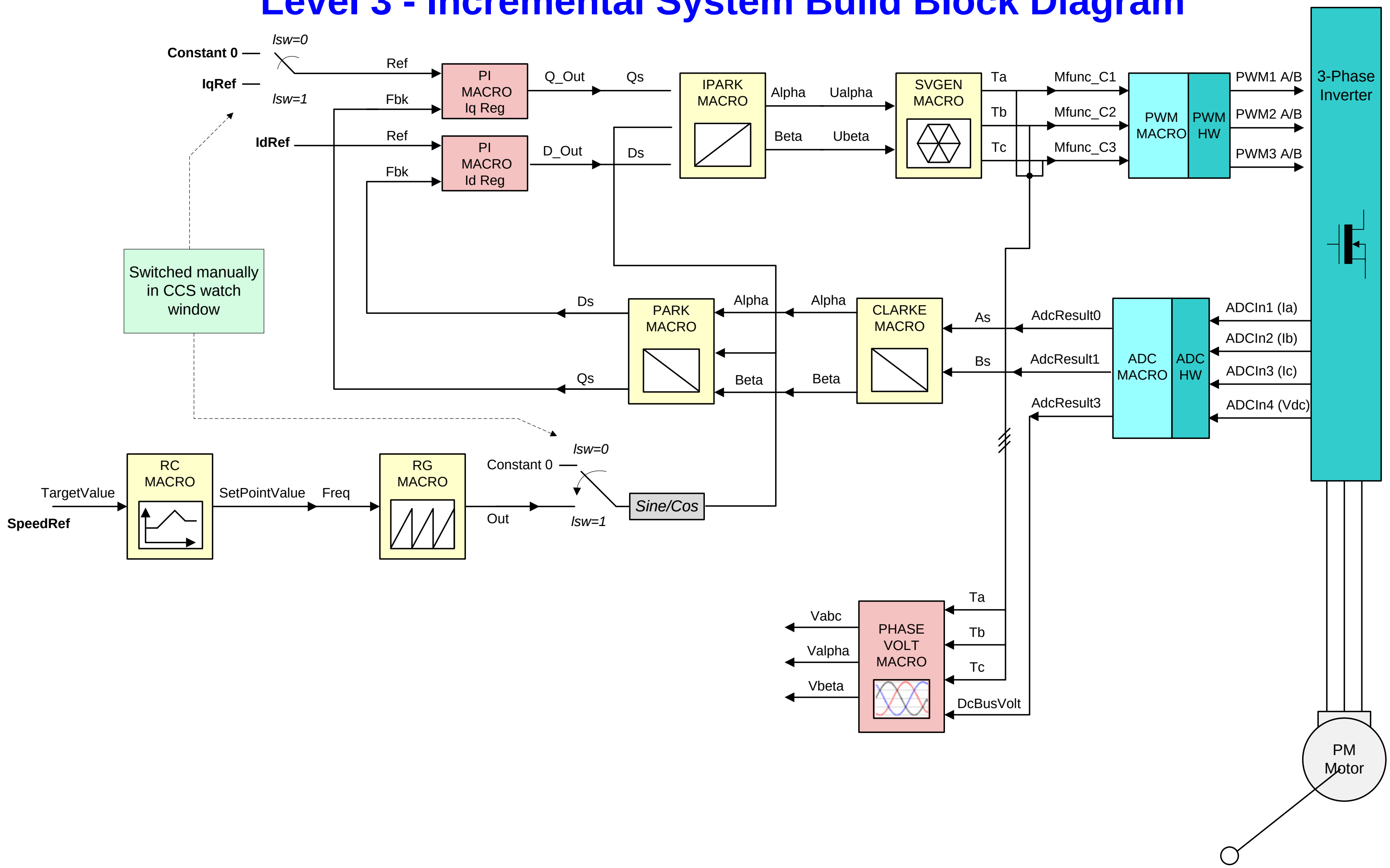


Fig.18 rg1.Out, measured theta and Phase A & B current waveforms.

* Deadband = 1.66 usec, Vdcbus=300V , dlog.trig_value= 100

Level 3 - Incremental System Build Block Diagram



Level 3 verifies the dq-axis current regulation performed by PI macros and speed measurement macros

Level 4 Incremental Build

Assuming the previous section is completed successfully; this section verifies the estimated rotor position and speed estimation performed by SMOPOS_CLA_MACRO (sliding mode observer) and SE_CLA_MACRO (speed estimation modules, respectively.

Open {App Name}_CLA-Shared_C.h and select level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4). Now right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will now keep on increasing, confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.
In the software, the key variables to be adjusted are summarized below.

- SpeedRef (float): for changing the rotor speed in per-unit.
- IdRef (float): for changing the d-qxis voltage in per-unit.
- IqRef (float): for changing the q-axis voltage in per-unit.

The tuning of sliding-mode and low-pass filter gains (Kslide and Kslf) inside the rotor position estimator may be critical for low speed operation. The key steps can be explained as follows:

- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).

- Compile/load/run program with real time mode and then increase voltage at variac / dc power supply to get the appropriate DC-bus voltage.

- Add the soft-switch variable “lsw” to the watch window in order to switch from current loop to speed loop. In the code lsw manages the loop setting as follows:

- lsw=0, lock the rotor of the motor.
- lsw=1, close the current loop.

- Set lsw to 1. Now the motor is running close to reference speed.Compare smo1.Theta with rg1.Out via PWMDAC with external low-pass filter and an oscilloscope. They should be identical with a small phase shift.

- If smo1.Theta does not give the sawtooth waveform, the Kslide and Kslf inside the sliding mode observer are required to be re-tuned.

- To confirm rotor position estimation, try different values of SpeedRef.

- Compare se1.WrHat (estimated speed) with reference speed or measured speed in the watch windows with continuous refresh feature whether or not it should be nearly the same.

- To confirm this open-loop speed estimator, try different values of SpeedRef.

- Bring the system to a safe stop as described at the end of build 1 by reducing the bus voltage, taking the controller out of realtime mode and reset.

During running this build, the current waveforms in the CCS graphs should appear as follows *:

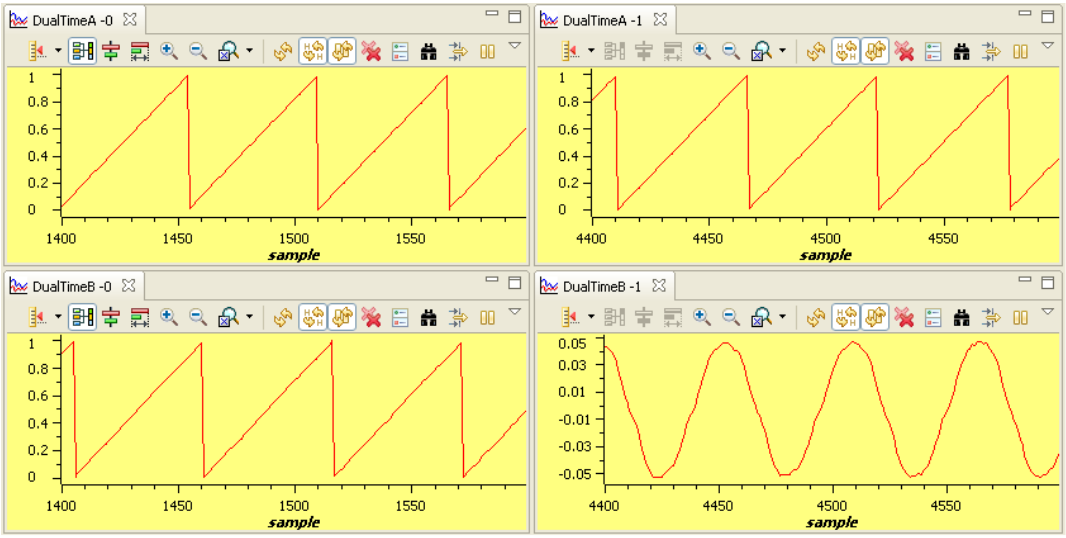
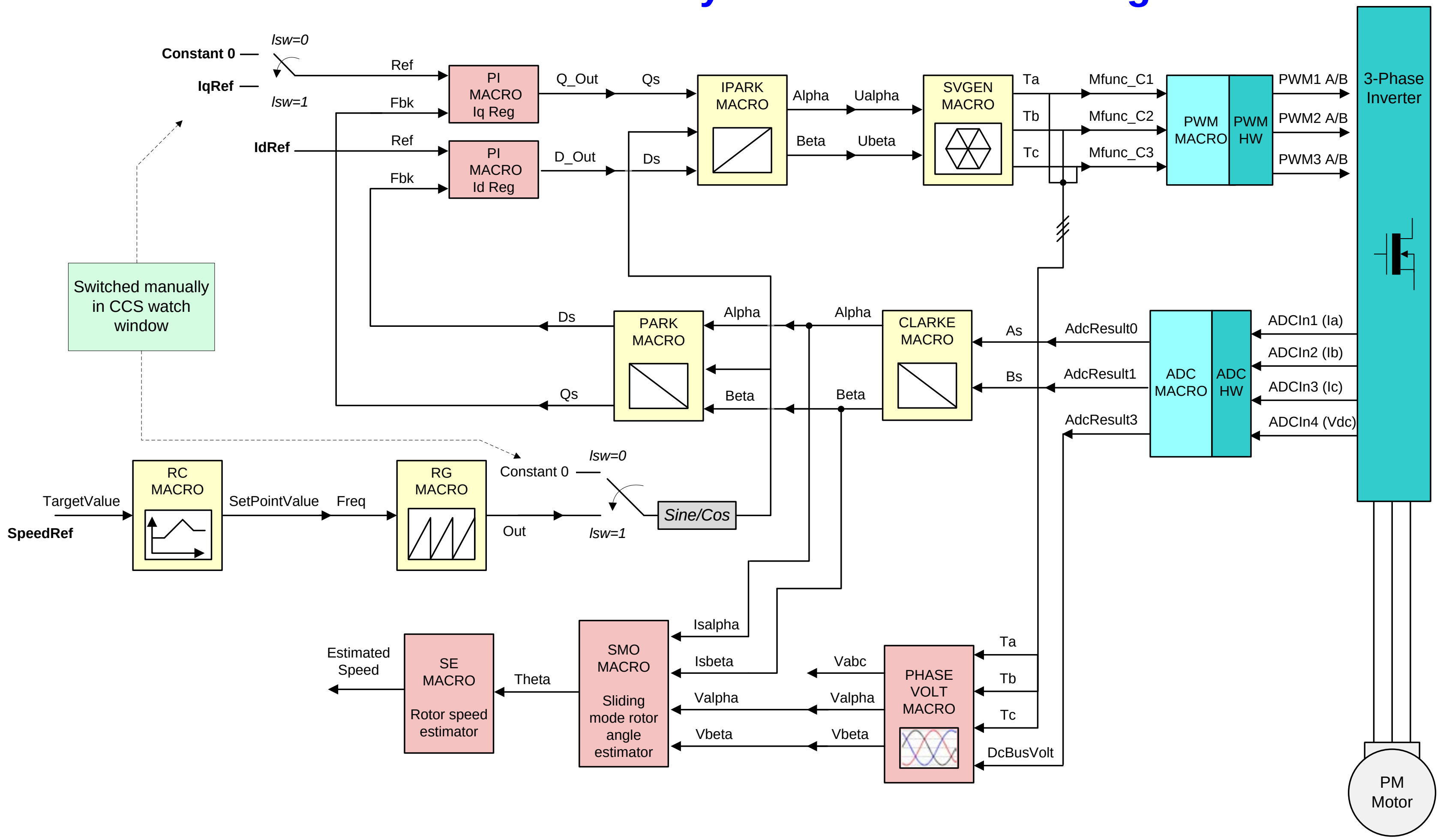


Fig.19 Measured theta, estimated theta (SMO), rg1. Out and Phase A current

* dlog.trig_value=100, deadband=1.66 usec, Vdcbus=300V

Level 4 - Incremental System Build Block Diagram



Level 4 verifies the rotor position and speed estimation performed by SMO and SE macros

Level 5 Incremental Build

Assuming the previous section is completed successfully, this section verifies the speed regulator performed by PI module. The system speed loop is closed by using the estimated speed as a feedback.

Open {App Name}_CLA-Shared_C.h and select level 6 incremental build option by setting the BUILDLEVEL to LEVEL5 (#define BUILDLEVEL LEVEL5). Now right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will now keep on increasing, confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly. In the software, the key variables to be adjusted are summarized below.

- SpeedRef (float): for changing the rotor speed in per-unit.
- IdRef (float): for changing the d-qxis voltage in per-unit.

The speed loop is closed by using estimated speed. The key steps can be explained as follows:

- Compile/load/run program with real time mode.

- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different) and Iqref to 0.1 pu.

- Add the soft-switch variable“lsw” to the watch window in order to switch from current loop to speed loop. In the code lsw manages the loop setting as follows:

- lsw=0, lock the rotor of the motor.
- lsw=1, close the current loop.
- lsw=2, close the speed loop.

- Set lsw to 1. Gradually increase voltage at variac / dc power supply to get an appropriate DC-bus voltage and now the motor is running with this reference speed (0.3 pu). Then, set lsw to 2 to close the speed loop. After a few tests, the user can determine the best time to close the speed loop depending on the load-speed profile and then close the speed loop in the code. For most of the applications, the speed loop can be closed before the motor speed reaches to SpeedRef.

- Compare se1.WrHat with SpeedRef in the watch windows with continuous refresh feature whether or not it should be nearly the same.

- To confirm this speed PI module, try different values of SpeedRef.

- For speed PI controller, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied responses.

- At very low speed range, the performance of speed response relies heavily on the good rotor flux angle computed by flux estimator.

- Bring the system to a safe stop as described at the end of build 1 by reducing the bus voltage, taking the controller out of realtime mode and reset.

Note: The first order low-pass filter inside the SMO module causes small amount of estimated angle delay. In order to achieve accurate field orientation, it is recommended to compensate this delay. Once the delays are detected for different operating points, they can be interpolated by means of a simple second or third order equation and this equation can be added to the code. Please refer to the smopos.pdf for the details of SMO: ..controlSUITE\libs\app_libs\motor_control\math_blocks\v4.0\~Docs.

During running this build, the current waveforms in the CCS graphs should appear as follows *:

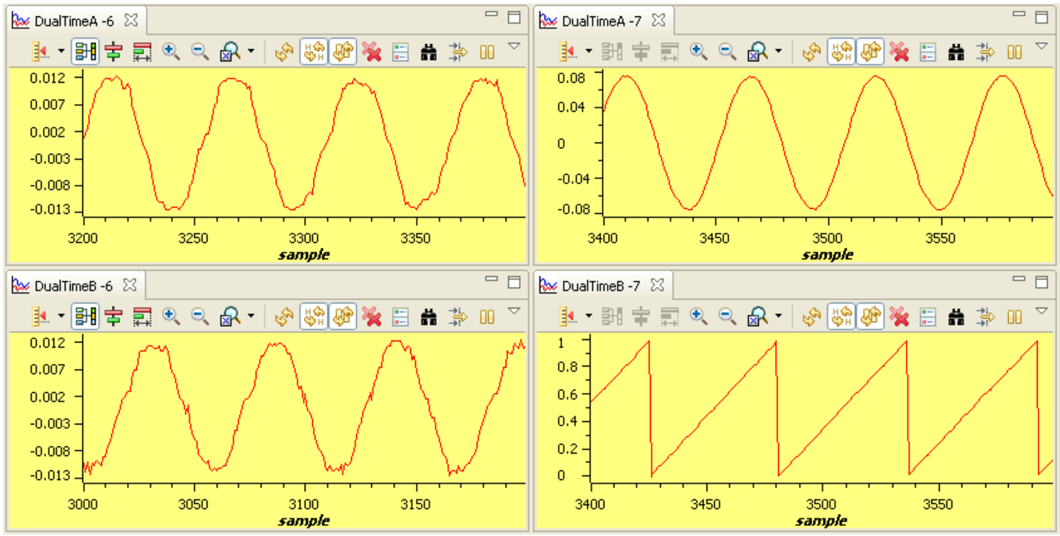


Fig.20 Waveforms of Phase A&B currents, calculated Phase A voltage, and estimated theta by SMO under no-load and 0.3pu speed

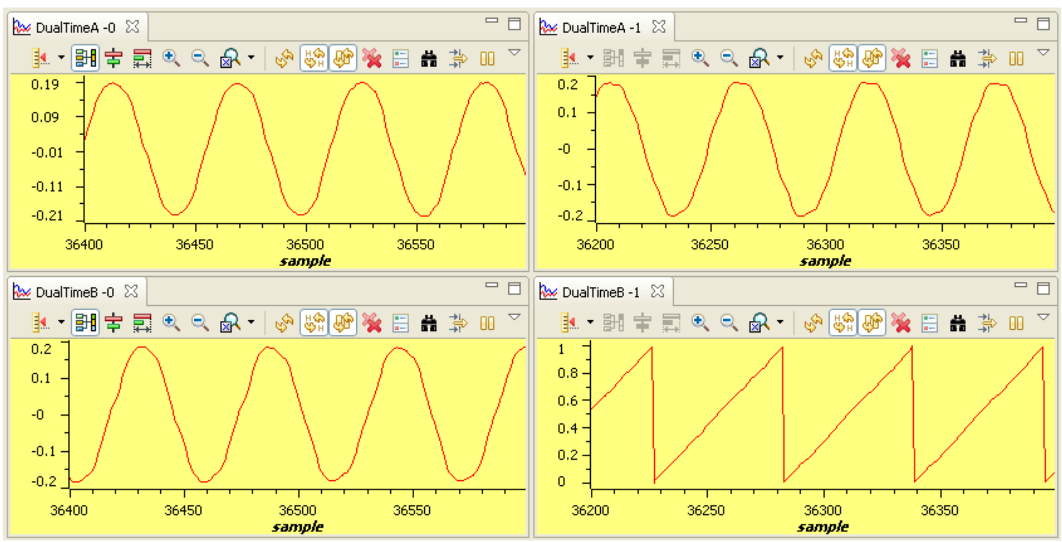


Fig.21 Waveforms of Phase A&B currents, calculated Phase A voltage, and estimated theta by SMO under 0.33pu-load and 0.5pu speed

* dlog.trig_value=100, deadband=1.66 usec, Vdcbus=300V, pi_spd.Kp=1.0

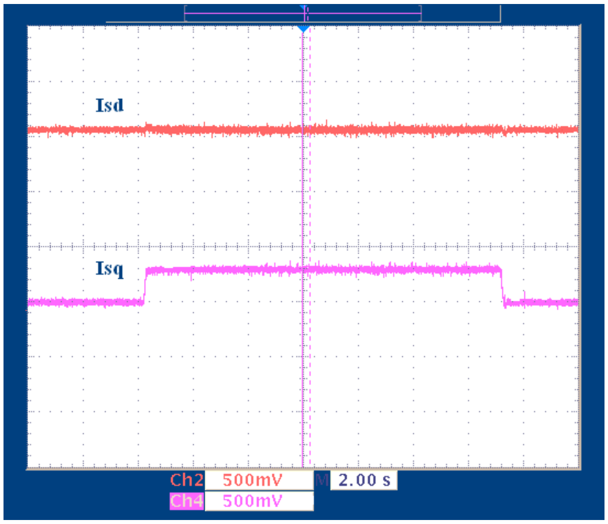
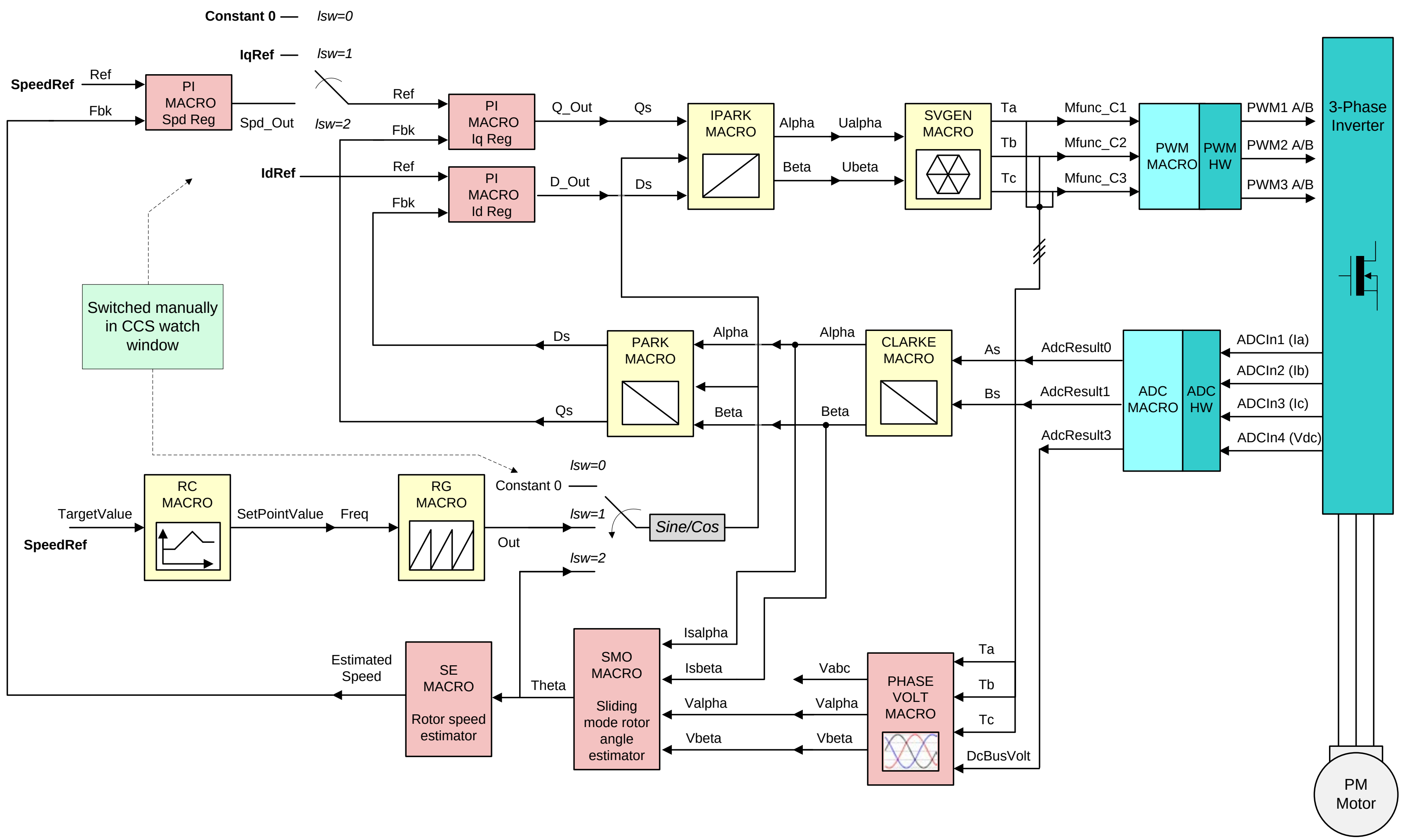


Fig22 Flux and torque components of the stator current in the synchronous reference frame under 0.33pu step- load and 0.5 pu speed monitored from PWMDAC output

Level 5 - Incremental System Build Block Diagram



Level 5 verifies the complete system