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

Bilal Akin — *C2000 Systems and Applications Team*
Manish Bhardwaj
Jon Warriner — *D3 Engineering*

## Abstract

This application note presents a solution to control a permanent magnet synchronous motor (PMSM) using the TMS320F2803x microcontrollers along with the DRV830x-HC-EVMs. The two EVMs that are targeted are the DRV8301-HC-EVM and the DRV8302-HC-EVM which showcase the DRV8301 and DRV8302 Three Phase Pre-Drivers with Dual Current Shunt amplifiers and Buck Converter. 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 increasing efficiency. With these devices it is possible to realize far more precise digital vector control algorithms like 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. The DMC Library uses TI's IQ math library, which supports both fixed and floating point maths. This makes migrating from floating to fixed point devices easy. A configuration for TMS320F2806x, which is a floating point microcontroller, is available in the project to highlight this.

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

## Table of Contents

## 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 and TMS320F2806x. These controllers enable cost-effective design of intelligent controllers for PM 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.

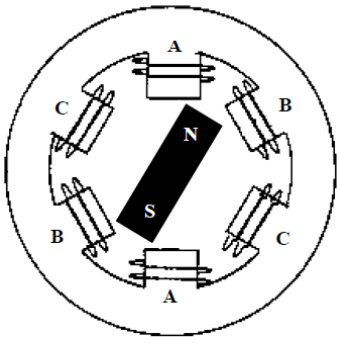



**Figure 1 A three-phase synchronous motor with a one permanent magnet pair pole rotor**

**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.

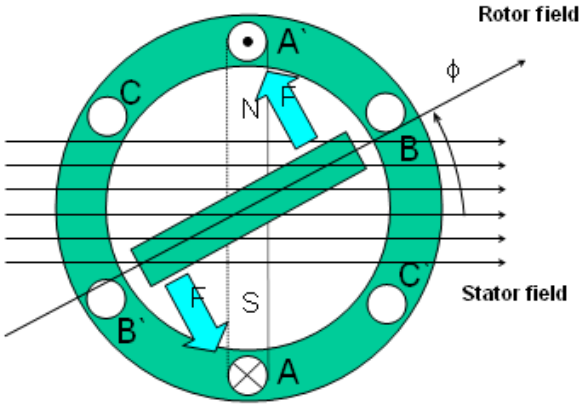



**Figure 2 The interaction between the rotating stator flux, and the rotor flux produces a torque which will cause the motor to rotate.**

## 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***.

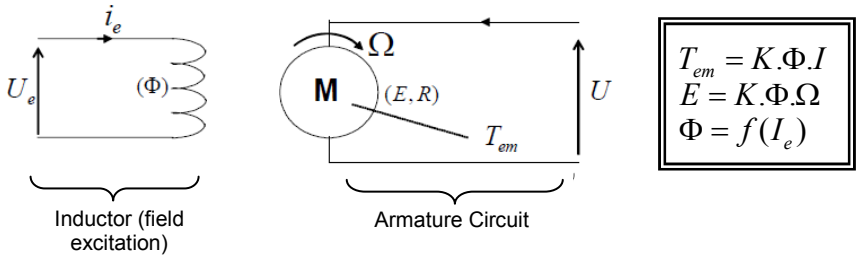



**Figure 3 Separated excitation DC motor model, flux and torque are independently controlled and the current through the rotor windings determines how much torque is produced.**

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 ($\psi_R$) 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:

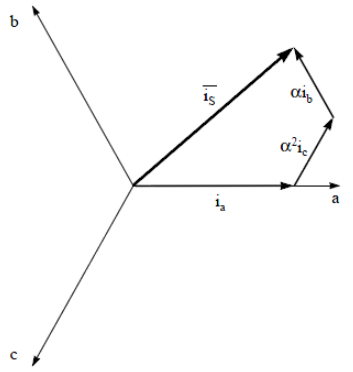



**Figure 4 Stator current space vector and its component in (a,b,c)**

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 $(\alpha, \beta)$. Assuming that the axis a and the axis $\alpha$ are in the same direction we have the following vector diagram:

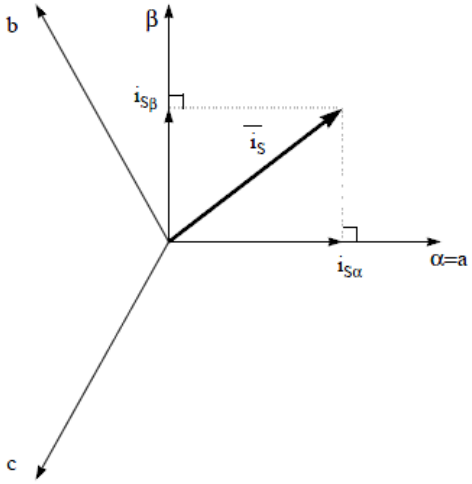



**Figure 5 Stator current space vector and its components in the stationary reference frame**

The projection that modifies the three phase system into the $(\alpha, \beta)$ 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 $(\alpha, \beta)$ 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 $(\alpha, \beta)$ 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:

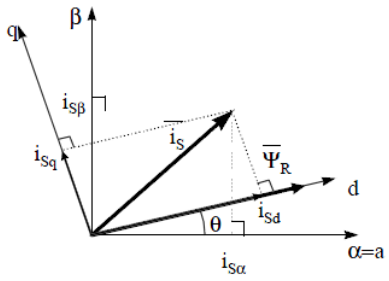



**Figure 6 Stator current space vector and its component in (α,β) and in the d,q rotating reference frame**

where $\theta$ 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 $(\alpha, \beta)$ 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:

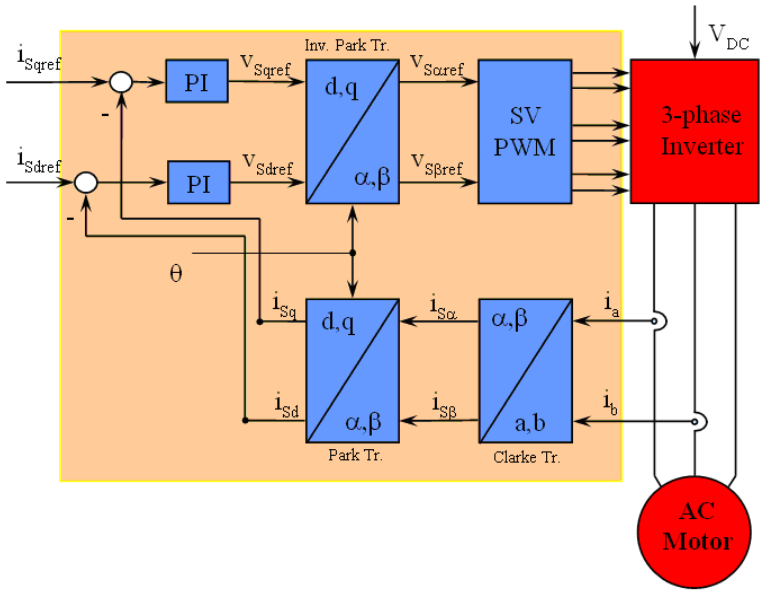



**Figure 7 Basic scheme of FOC for AC motor**

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 $(\alpha,\beta)$ 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), $(\alpha,\beta)$ 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.

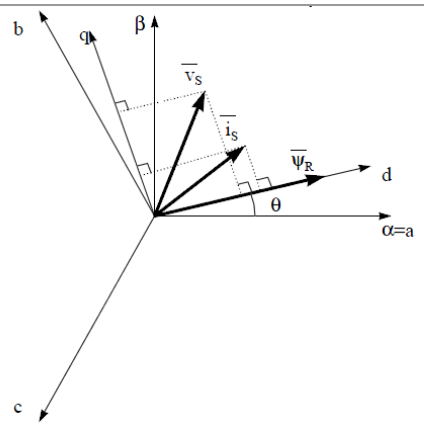



**Figure 8 Current, voltage and rotor flux space vectors in the d,q rotating reference frame and their relationship with a,b,c and (α,β) stationary reference frame**

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 Figure 9.

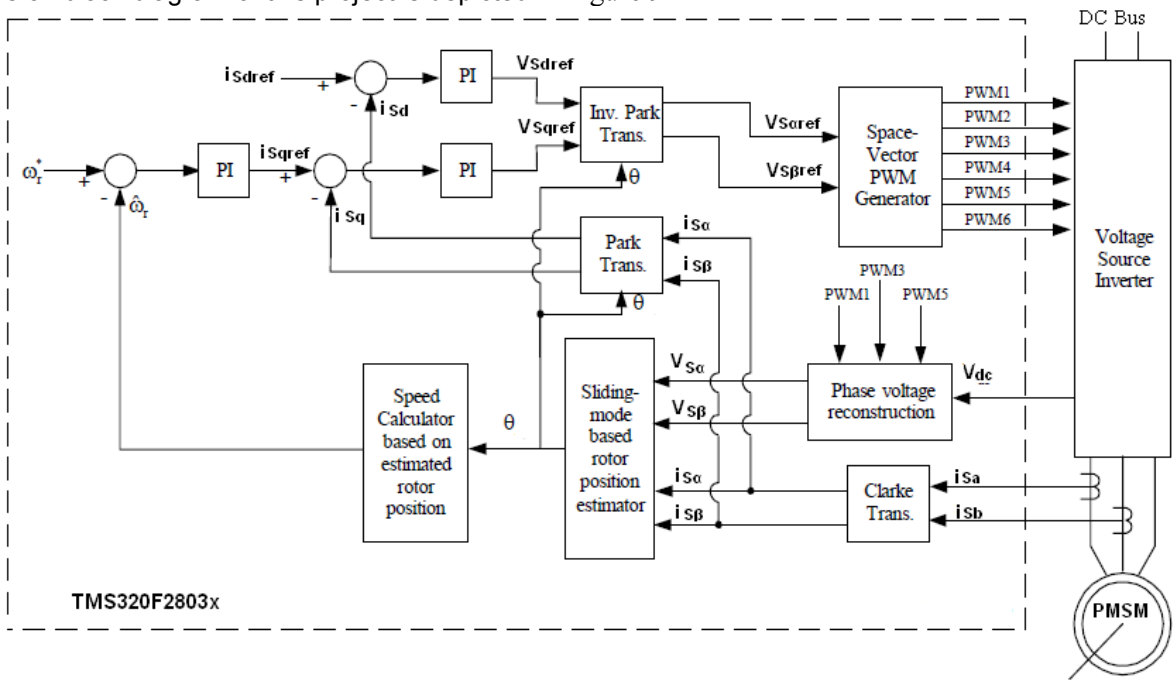



**Figure 9 Overall block diagram of sensorless field oriented control of PMSM**

## 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. IQMath Library enables easy migration from floating point algorithms to fixed point thus accelerating the development cycle.

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 Digital Motor Control (DMC) Library

The 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 Resolver, Flux, and Speed Calculators and Estimators), Control (Signal Generation, PID, BEMF Commutation, Space Vector Generation), and Peripheral Drivers (PWM abstraction for multiple topologies and techniques, ADC drivers, and motor sensor interfaces). 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\fixedv1.1
- C:\TI\controlSUITE\libs\app_libs\motor_control\math_blocks\v3.1
- C:\TI\controlSUITE\libs\app_libs\motor_control\drivers\f2806x
- C:\TI\controlSUITE\libs\app_libs\motor_control\drivers\f2803x

These modules allow users to quickly build, or customize, their own systems. The Library supports the three motor types: ACI, BLDC, 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.

**Note:** TI DMC modules are written in form of macros for optimization purposes (refer to application note *SPRAAK2* for more details at TI website). The macros are defined in the header files. The user can open the respective header file and change the macro definition, if needed. In the macro definitions, there should be a backslash "\" at the end of each line as shown below which means that the code continue in the next line. Any character including invisible ones like "space" after the backslash will cause compilation error. Therefore, make sure that the backslash is the last character in the line. In terms of code development, the macros are almost identical to C function, and the user can easily convert the macro definition to a C functions.

```
#define PARK_MACRO(v)                                                    \
                                                                         \
    v.Ds = _IQmpy(v.Alpha,v.Cosine) + _IQmpy(v.Beta,v.Sine);     \
    v.Qs = _IQmpy(v.Beta,v.Cosine)   - _IQmpy(v.Alpha,v.Sine);
```

***A typical DMC macro definition***

## System Overview

This document describes the “C” real-time control framework used to demonstrate the sensorless field oriented control of PM motors. The “C” framework is designed to run both on TMS320C2803x and TMS320C2806x 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 |
| PID_GRANDO | PID Regulators |
| RC | Ramp Controller (slew rate limiter) |
| RG | Ramp / Sawtooth Generator |
| QEP / CAP | QEP and CAP Drives (optional for speed loop tuning with a speed sensor) |
| SE | Speed Estimation (based on sensorless position estimation) |
| SPEED_FR | Speed Measurement (based on sensor signal frequency) |
| SMO | Sliding Mode Observer for Sensorless Applications |
| SVGEN | Space Vector PWM with Quadrature Control (includes IClarke Trans.) |
| VOLT | 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 explore the performance of speed control. The PM motor is driven by a DRV830x Three Phase Pre-Driver and an external three phase inverter. The TMS320x2803x or TMS320x2806x control cards can be 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 PM motor (ia and ib) are measured from the inverter and sent to the controller via two analog-to-digital converters (ADCs). In addition, the DC-bus voltage in the inverter is measured and sent to the TMS320x2803x or TMS320x2806x via an ADC. This DC-bus voltage is necessary to calculate the three phase voltages when the switching functions are known.

PM_Sensorless project has the following properties:

| C Framework | | | | |
|---|---|---|---|---|
| **System Name** | **Program Memory Usage 2806x** | **Data Memory Usage[1] 2806x** | **Program Memory Usage 2803x** | **Data Memory Usage[1] 2803x** |
| PM_Sensorless | 5000 words[3] | 1384 words | 5360 words[2] | 1400 words |

[1] Excluding the stack size
[2] Excluding “IQmath” Look-up Tables
[3] Excluding “FPUmath” Look-up Tables

| CPU Utilization of Sensorless FOC of PMSM | |
|---|---|
| **Name of Modules *** | **Number of Cycles** |
| Ramp Controller | 29 |
| Clarke Tr. | 28 |
| Park Tr. | 142 |
| I_Park Tr. | 41 |
| Sliding Mode Obs. | 273 |
| Speed Estimator | 72 |
| Phase Volt Calc. | 115 |
| 3 x Pid Grando | 167 |
| Space Vector Gen. | 137 |
| Pwm Drv | 74 |
| Contxt Save etc. | 53 |
| Pwm Dac (optional) | |
| DataLog (optional) | |
| Ramp Gen (optional) | |
| | |
| **Total Number of Cycles** | **1227 **** |
| CPU Utilization @ 60 Mhz | **20.4% ***** |
| CPU Utilization @ 40 Mhz | **30.7% ***** |

* The modules are defined in the header files as “macros”
** 1555 including the optional modules
*** At 10 kHz ISR freq.

| System Features | |
|---|---|
| **Development /Emulation** | Code Composer Studio V.4.1 (or above) with Real Time debugging |
| **Target Controller** | TMS320F2803x or TMS320F2806x |
| **PWM Frequency** | 10kHz PWM (Default), 60kHz PWMDAC |
| **PWM Mode** | Symmetrical with a programmable dead band |
| **Interrupts** | EPWM1 Time Base CNT_Zero – Implements 10 kHz ISR execution rate |
| **Peripherals Used** | PWM 1 / 2 / 3 for motor control<br>PWM 5A, 6A, 6B & 4A for DAC outputs<br>QEP1 A,B, I (optional for tuning the speed loop)<br>ADC B2 for DC Bus voltage sensing, A1 & B1 for phase current sensing<br>SPI-B for communication and configuration of the DRV8301 (DRV8302 uses discrete digital and analog I/O for configuration) |

The overall system implementing a 3-ph PM motor control is depicted in Figure 10. The PM motor is driven by the DRV830x Three Phase Pre-Driver and an external three phase inverter ver. The TMS320F2803x or TMS320F2806x is being used to generate the six pulse width modulation (PWM) signals using a space vector PWM technique, for six power switching devices connected to the DRV830x Three Phase Pre-Driver. Two input currents of the PM motor (ia and ib) are measured from the inverter and they are sent to the controller via two analog-to-digital converters (ADCs). In addition, the DC-bus voltage in the inverter is measured and sent to the TMS320F2803x or TMS320F2806x via an ADC as well. This DC-bus voltage is necessary in order to calculate three phase voltages of PM motor when the switching functions are known.

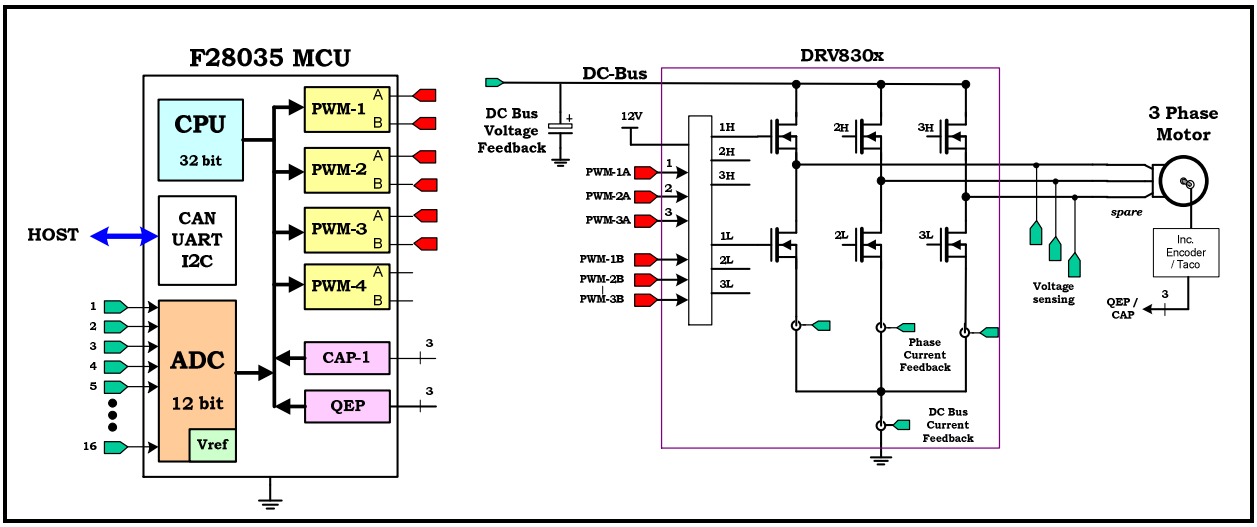



**Figure 10 A 3-ph PM motor drive implementation**

The software flow is described in the Figure 11 below.

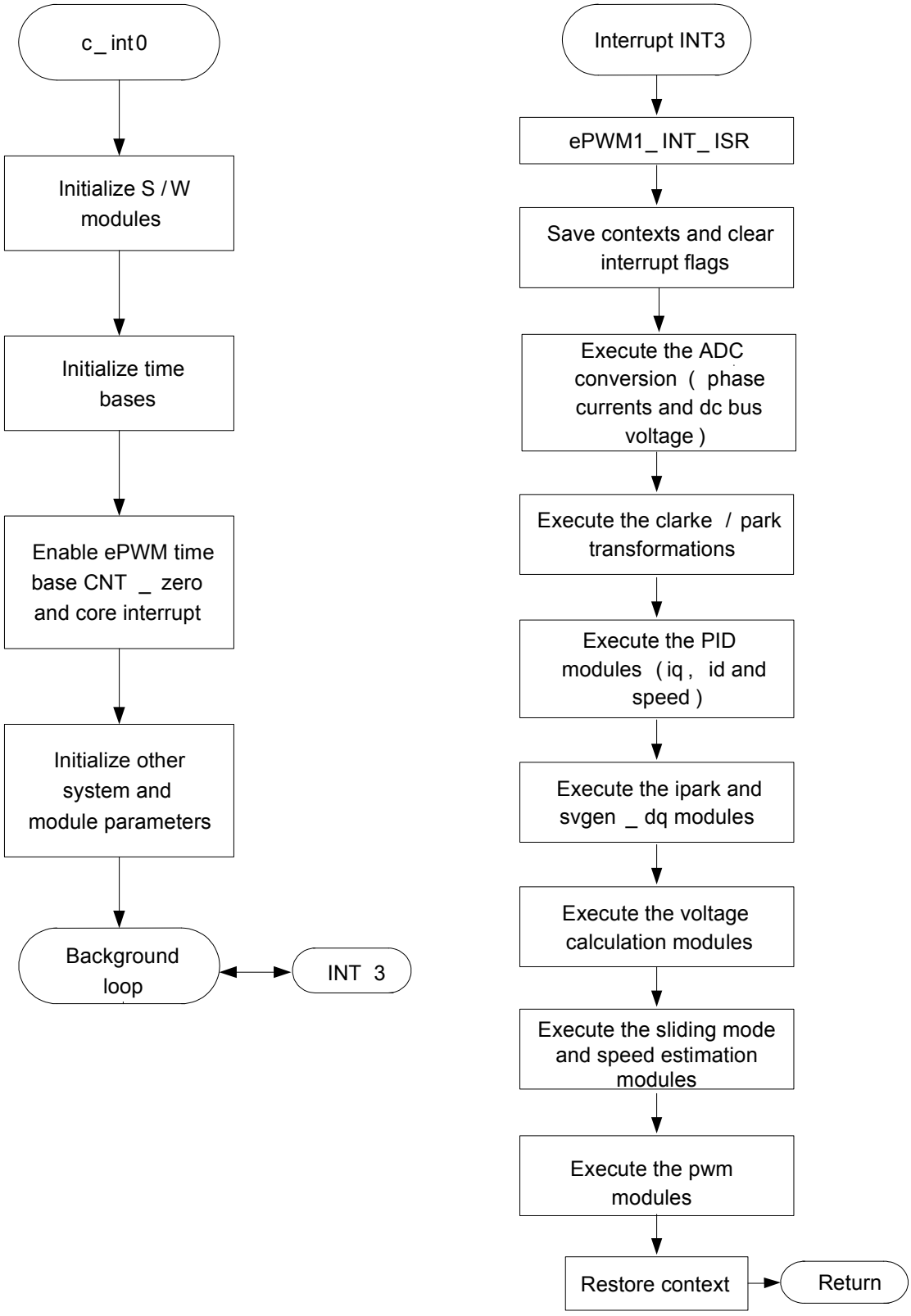



**Figure 11 System software flowchart**

## Hardware Configuration (DRV830x-HC-C2-KIT)

Please refer to the DRV830x-HC-C2-KIT How to Run Guide and HW Reference Guide found:

C:\TI\controlSUITE\development_kits\DRV830x-HC-C2-KITv*\~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. Unpack the DIMM style controlCARD and verify that the DIP switch settings match Figure 12

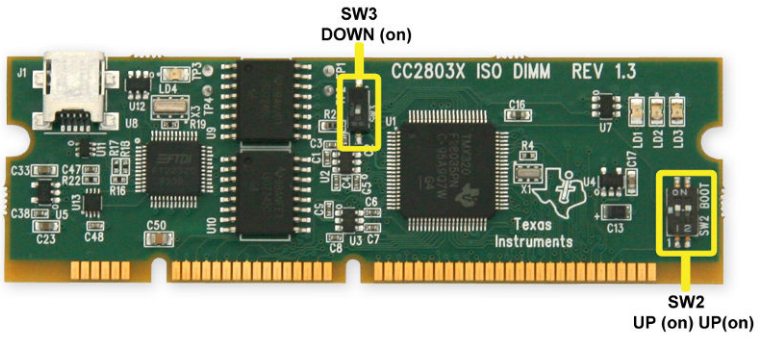



**Figure 12 controlCARD DIP Switch Settings**

2. Place the controlCARD in the connector slot of 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)

3. Connect a USB cable to connector J1 on the controlCARD. This will enable isolated JTAG emulation to the C2000 device. LD4 should turn on. 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.

4. Connect a power supply (60V max) to the PVDD and GND terminals of the DRV830x-HC-EVM. Now LED1 and LED3 should turn on. Notice the control card LED would light up as well indicating the control card is receiving power from the board.

5. Note that the motor should be connected to the OUTA, OUTB and OUTC terminals after you finish with the first incremental build step. For more details on motor wiring please refer to the datasheet provided with your motor.

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

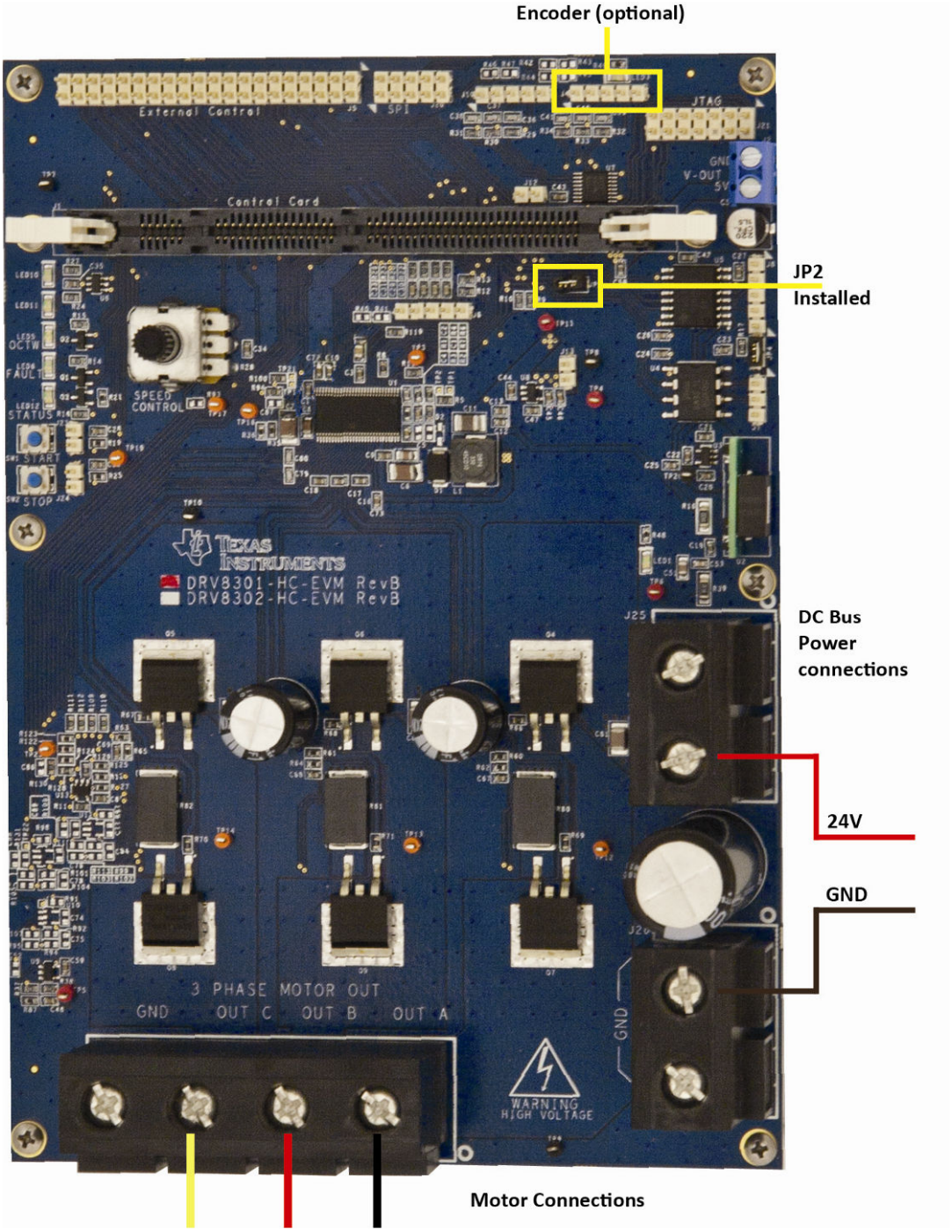



**Figure 13 EVM Connections**

**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 PM_Sensorless Project

Please refer to the “Software Setup for DRV830x-HC-C2-KIT Projects” section in the DRV830x-HC-C2-KIT How to Run Guide which can be found at

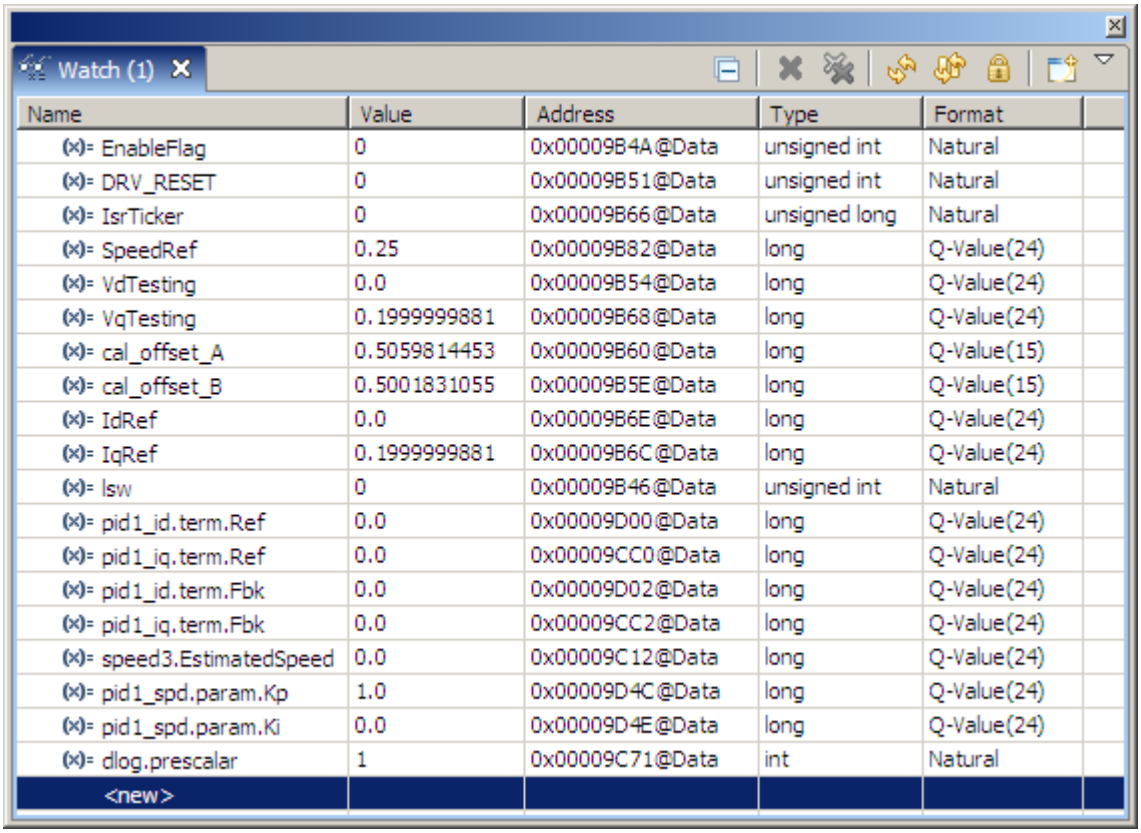
C:\TI\controlSUITE\development_kits\DRV830x-HC-C2-KITv*\~Docs

This section goes over how to install CCS and set it up to run with this project.

**The remainder of this application note will specifically discuss a hardware configuration consisting of a DRV8301-HC-EVM with a TMS320F2803x controlCARD installed. The process for other configurations, such as a DRV8302-HC-EVM and/or TMS320F2806x controlCARD, would be similar except the corresponding build configuration would need to be chosen in Code Composer Studio.**

**The default configuration of this project is optimized for running low to medium current motors. The gain of the DRV830x built-in current sense amplifiers is set to 40 giving a measurable current range of +/-20.625A. The gain can be changed by choosing the desired #define for DRV_GAIN in the file PM_Sensorless-Settings.h. Note that there are four possible settings for the DRV8301 while the DRV8302 is limited to gains of 10 or 40.**

Select PM_Sensorless as the active project. Select the active build configuration to be set as DRV8301_F2803x_RAM **(This is important as your desired build configuration may not be the default build configuration upon opening CCS.)**. Verify that the build level is set to 1 in PM_Sensorless-Settings.h, and then 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 and select the appropriate format for them.




| Name | Value | Address | Type | Format |
|---|---|---|---|---|
| (x)= EnableFlag | 0 | 0x00009B4A@Data | unsigned int | Natural |
| (x)= DRV_RESET | 0 | 0x00009B51@Data | unsigned int | Natural |
| (x)= IsrTicker | 0 | 0x00009B66@Data | unsigned long | Natural |
| (x)= SpeedRef | 0.25 | 0x00009B82@Data | long | Q-Value(24) |
| (x)= VdTesting | 0.0 | 0x00009B54@Data | long | Q-Value(24) |
| (x)= VqTesting | 0.1999999881 | 0x00009B68@Data | long | Q-Value(24) |
| (x)= cal_offset_A | 0.5059814453 | 0x00009B60@Data | long | Q-Value(15) |
| (x)= cal_offset_B | 0.5001831055 | 0x00009B5E@Data | long | Q-Value(15) |
| (x)= IdRef | 0.0 | 0x00009B6E@Data | long | Q-Value(24) |
| (x)= IqRef | 0.1999999881 | 0x00009B6C@Data | long | Q-Value(24) |
| (x)= lsw | 0 | 0x00009B46@Data | unsigned int | Natural |
| (x)= pid1_id.term.Ref | 0.0 | 0x00009D00@Data | long | Q-Value(24) |
| (x)= pid1_iq.term.Ref | 0.0 | 0x00009CC0@Data | long | Q-Value(24) |
| (x)= pid1_id.term.Fbk | 0.0 | 0x00009D02@Data | long | Q-Value(24) |
| (x)= pid1_iq.term.Fbk | 0.0 | 0x00009CC2@Data | long | Q-Value(24) |
| (x)= speed3.EstimatedSpeed | 0.0 | 0x00009C12@Data | long | Q-Value(24) |
| (x)= pid1_spd.param.Kp | 1.0 | 0x00009D4C@Data | long | Q-Value(24) |
| (x)= pid1_spd.param.Ki | 0.0 | 0x00009D4E@Data | long | Q-Value(24) |
| (x)= dlog.prescalar | 1 | 0x00009C71@Data | int | Natural |
| <new> | | | | |

**Figure 14 Watch Window Variables**

Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location C:\TI\ControlSUITE\developement_kits\DRV830x-HC-C2-KITv*\PM_Sensorless\ . Click on Continuous Refresh button  on the top left corner of the graph tab to enable periodic capture of data from the microcontroller.

## Incremental System Build

The system is gradually built up in order for the final system to be confidently operated. Seven phases of the incremental system build are designed to verify the major software modules used in the system. Table 1 summarizes the modules testing and using in each incremental system build.

| Software Module | Phase 1 | Phase 2 | Phase 3 | Phase 4 | Phase 5 | Phase 6 | Phase 7 |
|---|---|---|---|---|---|---|---|
| PWMDAC_MACRO | √ | √ | | √ | √ | √ | √ |
| RC_MACRO | √ | √ | | √ | √ | √ | √ |
| RG_MACRO | √ | √ | | √ | √ | √ | √ |
| IPARK_MACRO | √√ | √ | | √ | √ | √ | √ |
| SVGEN_MACRO | √√ | √ | | √ | √ | √ | √ |
| PWM_MACRO | √√ | √ | | √ | √ | √ | √ |
| CLARKE_MACRO | | √√ | | √ | √ | √ | √ |
| PARK_MACRO | | √√ | | √ | √ | √ | √ |
| VOLT_MACRO | | √√ | | √ | √ | √ | √ |
| Current Sensor Offset Calibration | | | √√ | √ | √ | √ | √ |
| QEP_MACRO | | | | √√ | √ | √ | √ |
| SPEED_FR_MACRO | | | | √√ | √ | √ | √ |
| PID_GRANDO_MACRO (IQ) | | | | √√ | √ | √ | √ |
| PID_GRANDO_MACRO (ID) | | | | √√ | √ | √ | √ |
| SMO_MACRO | | | | | √√ | √√ | √ |
| SE_MACRO | | | | | | √√ | √ |
| PID_GRANDO_MACRO (SPD) | | | | | | √√ | √√ |
| | | | | | | | |
| Note: the symbol √ means this module is using and the symbol √√ means this module is testing in this phase. | | | | | | | |

Table 1: Testing modules in each incremental system build

**Level 1 Incremental Build**

At this step keep the motor disconnected. Assuming the load and build steps described in the "DRV830x-HC-C2-KIT How To Run Guide" are 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_MACRO (inverse park transformation) and SVGEN_MACRO (space vector generator) modules and the peripheral dependent PWM_MACRO (PWM initializations and update) modules. Open PM_Sensorless-Settings.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: for changing the rotor speed in per-unit.
- VdTesting: for changing the d-qxis voltage in per-unit.
- VqTesting: for changing the q-axis voltage in per-unit.
- DRV_RESET: for holding the DRV830x chip in reset.

**Level 1A (SVGEN_MACRO Test)**

The SpeedRef value is specified to the RG_MACRO module via RC_MACRO module. The IPARK_MACRO module is generating the outputs to the SVGEN_MACRO module. Three outputs from SVGEN_MACRO module are monitored via the graph window as shown in Figure 15 where Ta, Tb, and Tc waveform are 120° apart from each other. Specifically, Tb lags Ta by 120° and Tc leads Ta by 120°. Check the PWM test points on the board to observe PWM pulses (PWMA, PWMB and PWMC) and make sure that the PWM module is running properly.

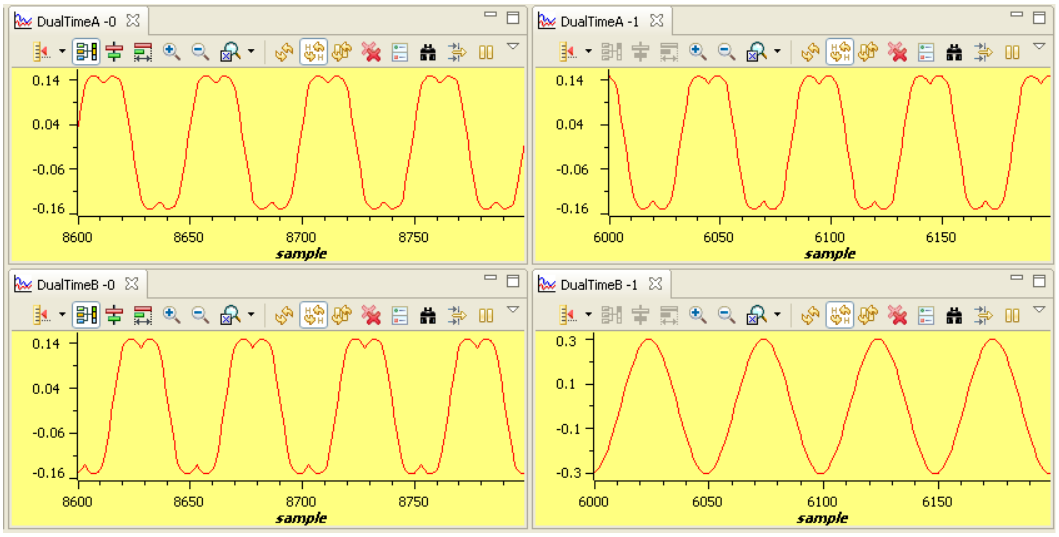



**Figure 15 Output of SVGEN, Ta, Tb, Tc and Tb-Tc waveforms**

### Level 1B (testing The PWMDAC Macro)

To monitor internal signal values in real time PWM DACs are very useful tools. Present on the DRV830x-HC-EVM are PWM DAC's which use an external low pass filter to generate the waveforms (J6, 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, $\tau$) 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 (R22, R20, R23 and R26 & C30, C29, C31 and C33) is shipped with 470Ω and 0.47μF on the board. Refer to application note *SPRAA88A* for more details at TI website.

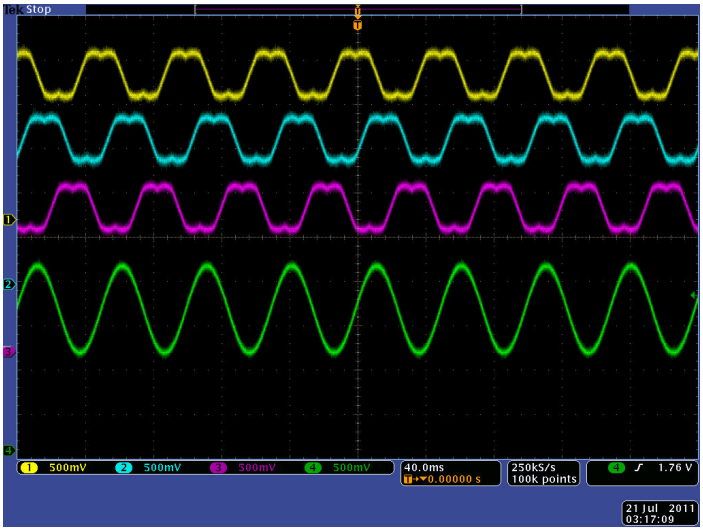


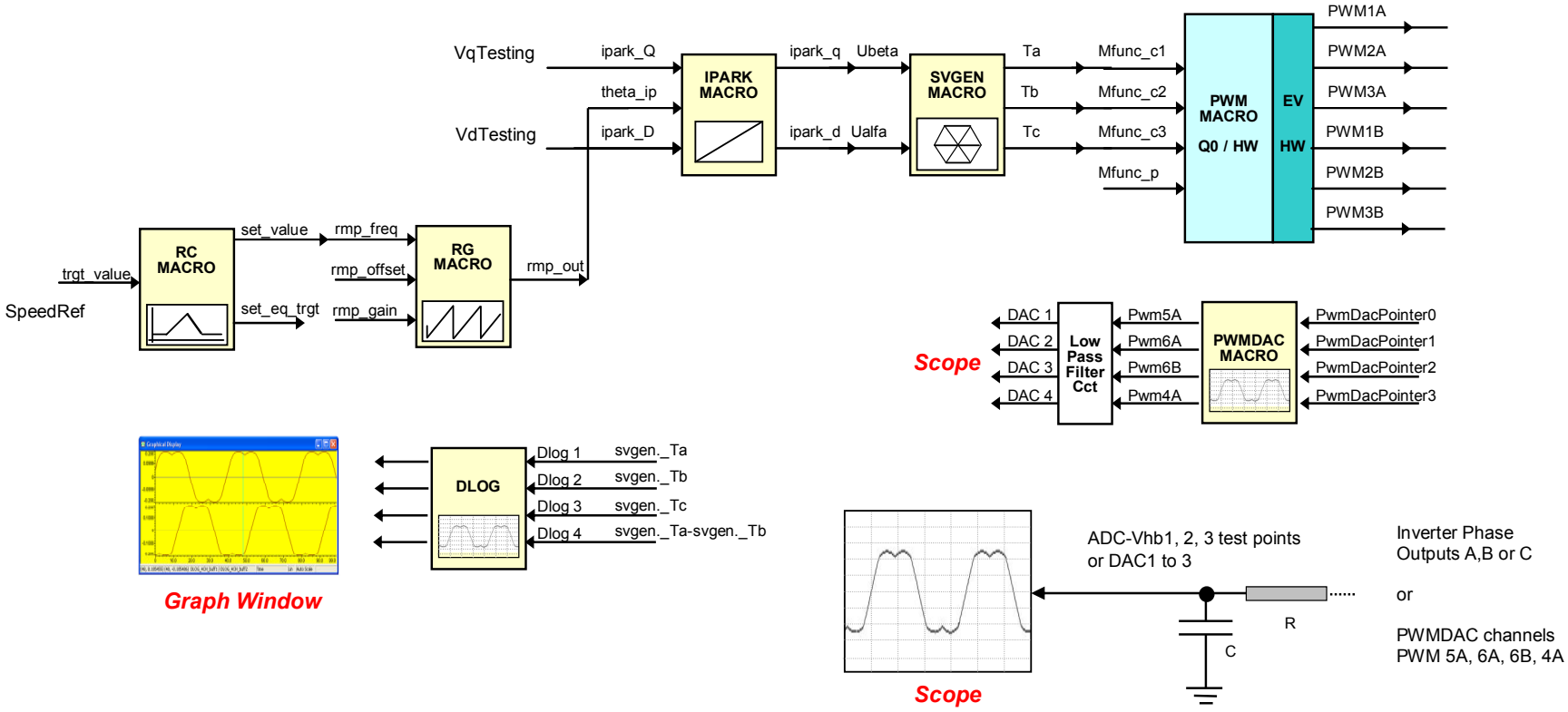

**Figure 16 DAC 1-4 outputs showing Ta, Tb Tc and Tb-Tc waveforms**

### Level 1C (PWM_MACRO and INVERTER Test)

After verifying SVGEN_MACRO module in level 1a, the PWM_MACRO software module and the DRV830x 3-phase inverter hardware are tested by looking at the low pass filter outputs. For this purpose, set the variable DRV_RESET to 0 through the watch window and check the ADC_Vhb1, ADC_Vhb2 and ADC_Vhb3 schematic nodes using an oscilloscope. You will observe that the phase voltage dividers and waveform monitoring filters enable the generation of the waveform and ensure that the DRV830x is working appropriately.

After verifying this, set the variable DRV_RESET to 1, take the controller out of real time mode (disable), reset the processor (see "DRV830x-HC-C2-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

# Level 1 - Incremental System Build Block Diagram




Level 1 verifies the target independent modules, duty cycles and PWM update. 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 DRV8301-HC-EVM 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 PM_Sensorless-Settings.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: for changing the rotor speed in per-unit.
- VdTesting: for changing the d-qxis voltage in per-unit.
- VqTesting: 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_MACRO, will be tested. The outputs of this module can be checked via the graph window as follows:

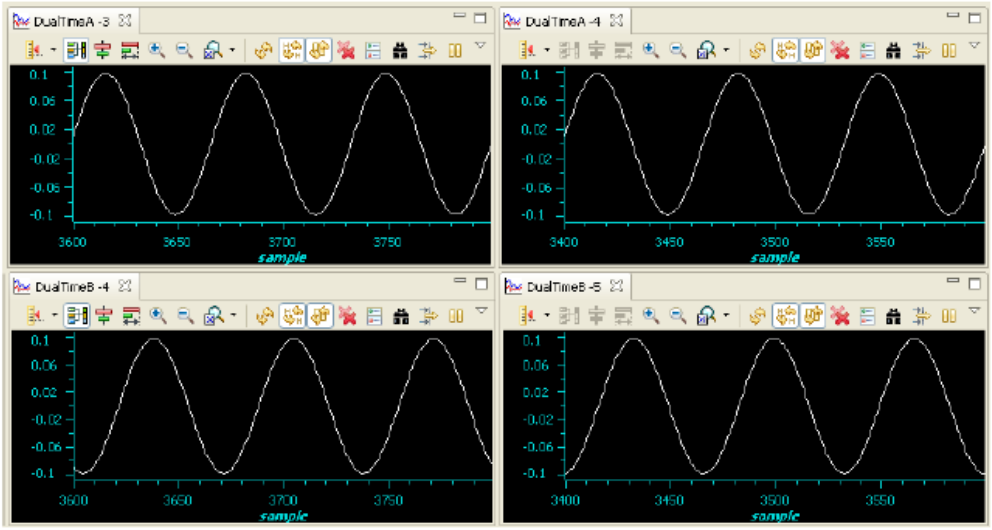



**Figure 17 The waveforms of volt1.Phase A&B and volt1.Aplha & Beta**

- 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 clark1.Alpha waveform should be same as the clark1.As waveform.
- The clark1.Alpha waveform should be leading the clark1.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 clark1.Alpha waveform should be lagging the Valpha waveform at an angle by nature of the reactive load of motor.
- The clark1.Beta waveform should be lagging the Vbeta waveform at the same angle.

If the clark1.Alpha and Valpha or clark1.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_MACRO module. The outputs of this test can be checked via the graph window.

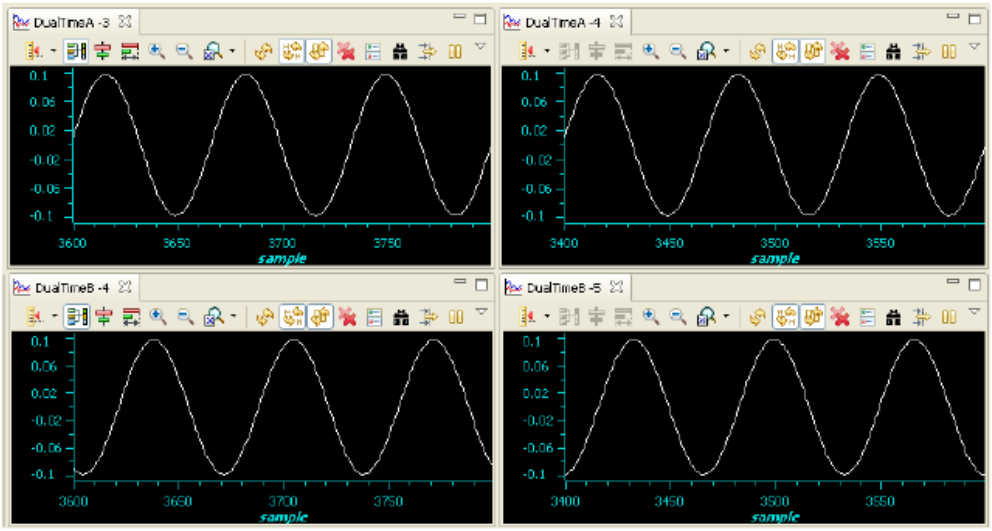
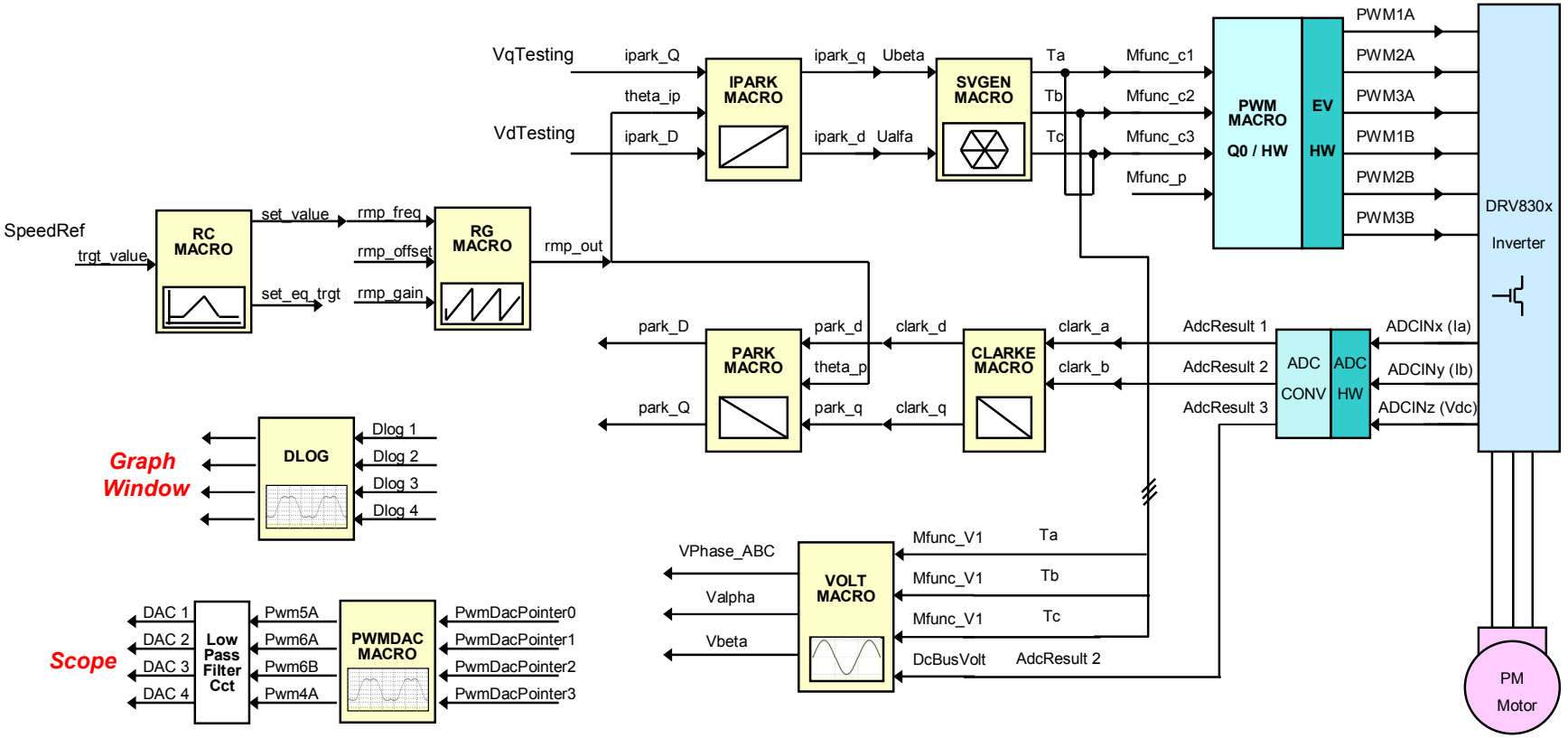



**Figure 18 The waveforms of volt1.Valpha, clarke1.Alpha and**

# 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 performs automatic calibration of the current sensor offsets.

Open PM_Sensorless-Settings.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.

- cal_offset_A: for changing the A phase current sensor offset in per-unit.
- cal_offset_B: for changing the B phase current sensor offset in per-unit.

Note that especially the low power motors draw low amplitude current after closing the speed loop under no-load. The performance of the sensorless control algorithm becomes prone to phase current offset which might stop the motors or cause unstable operation. Therefore, the phase current offset values need to be minimized at this step. The offsets will be automatically calculated by passing the measured currents through a low-pass filter to obtain the average value when zero current is flowing through the sensors.

Initialize cal_offset_A and cal_offset_B to 0.5 in the code, recompile and run the system and watch the cal_offset_A & cal_offset_B from watch window. Ideally the measured phase current offsets should be 0.5 in this case. Note the value of cal_offset_A and cal_offset_B in the watch window and change their values in the code by going to:

```
_iq cal_offset_A = _IQ15(0.5000);
_iq cal_offset_B = _IQ15(0.5000);
```

and changing `IQ15(0.5000)` offset value (e.g. `IQ15(0.5087)` or `IQ15(0.4988)` depending on the value observed in the watch window). Try to enter an offset with 4 significant digits.

These offset values will now be used for the remaining build levels.

**Note:** Piccolo devices have 12-bit ADC and 16-bit ADC registers. The `AdcResult.ADCRESULT` registers are right justified for Piccolo devices; therefore, the measured phase current value is firstly left shifted by three to convert into Q15 format (0 to 1.0), and then converted to ac quantity (± 0.5) following the offset subtraction. Finally, it is left shifted by one (multiplied by two) to normalize the measured phase current to ± 1.0 pu.

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

**Level 4 Incremental Build**

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

Open PM_Sensorless-Settings.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: for changing the rotor speed in per-unit.
- IdRef: for changing the d-qxis voltage in per-unit.
- IqRef: 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 PID_GRANDO_CONTROLLER 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 0.25 pu (or another suitable value if the base speed is different), Idref to zero and Iqref to 0.2 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, run the motor with closed current loop.

- Check pid1_id.Fdb in the watch windows with continuous refresh feature whether or not it should be keeping track pid1_id.Ref for PID_GRANDO_CONTROLLER module. If not, adjust its PID gains properly.

- Check pid1_iq.Fdb in the watch windows with continuous refresh feature whether or not it should be keeping track pid1_iq.Ref for PID_GRANDO_CONTROLLER module. If not, adjust its PID gains properly.

- To confirm these two PID modules, try different values of pid1_id.Ref and pid1_iq.Ref or SpeedRef.

- For both PID 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 setting DRV_RESET to 1, taking the controller out of realtime mode and reset.

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

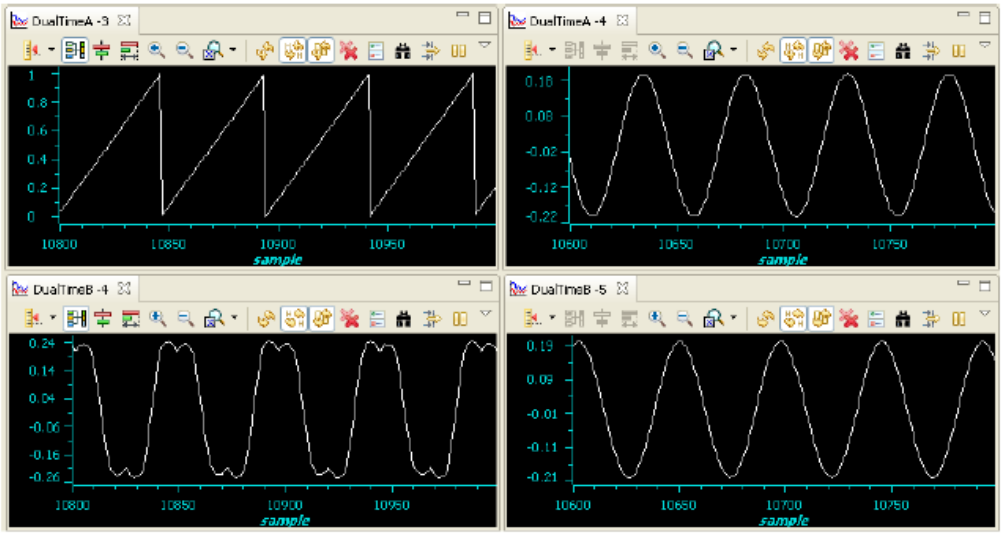



**Figure 19 rg1.Out,svgen_dq1.Ta and Phase A & B current waveforms.**

### Level 4B – QEP / SPEED_FR test

This section verifies the QEP1 driver and its speed calculation. Qep drive macro determines the rotor position and generates a direction (of rotation) signal from the shaft position encoder pulses. Make sure that the output of the incremental encoder is connected to J4 and QEP/SPEED_FR macros are initialized properly in the PM_Sensorless.c file depending on the features of the speed sensor. Refer to the pdf files regarding the details of related macros in motor control folder (C:\TI\controlSUITE\libs\app_libs\motor_control). The steps to verify these two software modules related to the speed measurement can be described as follows:

- Set SpeedRef to 0.25 pu (or another suitable value if the base speed is different).
- Compile/load/run program with real time mode.
- 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.Check the “speed1.Speed” in the watch windows with continuous refresh feature whether or not the measured speed is around the speed reference.
- To confirm these modules, try different values of SpeedRef to test the Speed.
- Use oscilloscope to view the electrical angle output, qep1.ElecTheta, from QEP_MACRO module and the emulated rotor angle, rg1.Out, from RG_MACRO at PWMDAC outputs with external low-pass filters.
- Check that both qep1.ElecTheta and rg1.Out are of saw-tooth wave shape and have the same period. If the measured angle is in opposite direction, then change the order of motor cables connected to inverter output.

- Qep1.ElecTheta should be slightly lagging rg1.out, if not adjust the calibration angle.

- Check from Watch Window that qep1.IndexSyncFlag is set back to 0xF0 every time it reset to 0 by hand. Add the variable to the watch window if it is not already in the watch window.

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

- The calibration angle of the encoder is detected in the code as detailed below. Note that this is an optional procedure for sensorless FOC speed loop tuning. If the user prefers to use encoder to tune the speed loop and apply the schemes given in level 6A then exact rotor position is not needed. However if the user tune the speed loop as in level 6C, then exact rotor position information is needed, thus the calibration angle has to be detected.

Next, the following steps are to verify and or perform calibration angle of the encoder. The steps are as follows:

- Make sure EQep1Regs.QPOSCNT, EQep1Regs.QPOSILAT, Init_IFlag, qep1.CalibratedAngle, and lsw are displayed in watch window.

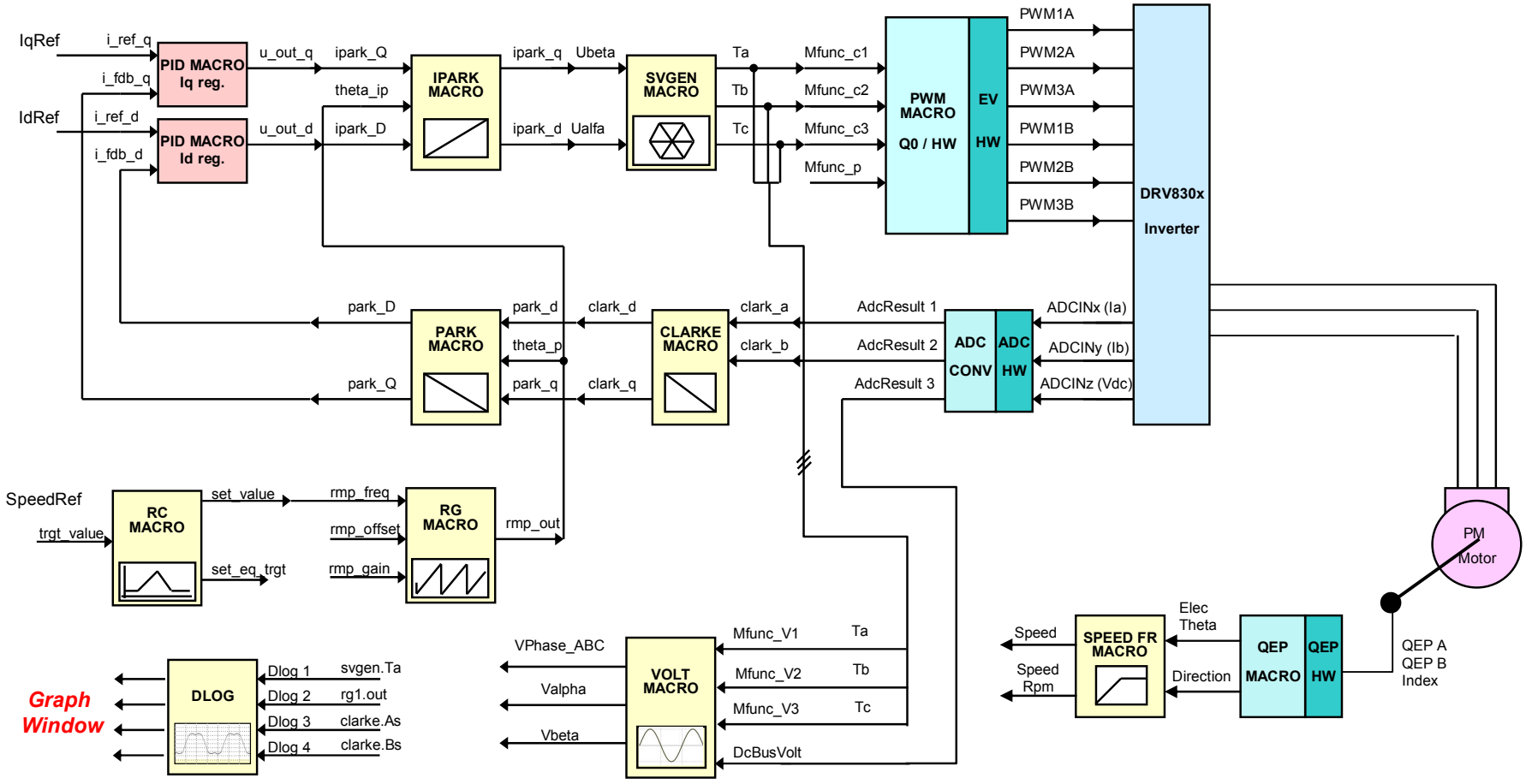
- Set SpeedRef to 0.25 pu (or another suitable value if the base speed is different).

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

- Now the rotor should be locked. Set lsw to 1 to spin the motor. When the first index signal is detected by QEP, the EQep1Regs.QPOSILAT register latches the angle offset in between initial rotor position and encoder index in the code. Later, EQep1Regs.QPOSILAT is set to maximum of EQep1Regs.QPOSCNT as it latches the counter value for each index signal. In the code qep1.CalibratedAngle keeps the initial offset value. This value can be recorded to initialize qep1.CalibratedAngle at the initialization section in PM_Sensorless.c or it can be detected in the code each time the motor is restarted. The calibration angle might be different for different start-ups and can be formulated as follows:

$$\textit{Calibration Angle} = \textit{Offset Angle} \pm n \,.\, \textit{Line Encoder}$$

# Level 4- Incremental System Build Block Diagram




Level 4verifies the dq-axis current regulation performed by pid_id, pid_iq and speed measurement modules

**Level 5 Incremental Build**

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

Open PM_Sensorless-Settings.h and select level 5 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: for changing the rotor speed in per-unit.
- IdRef: for changing the d-qxis voltage in per-unit.
- IqRef: 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 0.25 pu (or another suitable value if the base speed is different).

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

- 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 speed3.EstimatedSpeed (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 setting DRV_RESET to 1, taking the controller out of realtime mode and reset.

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

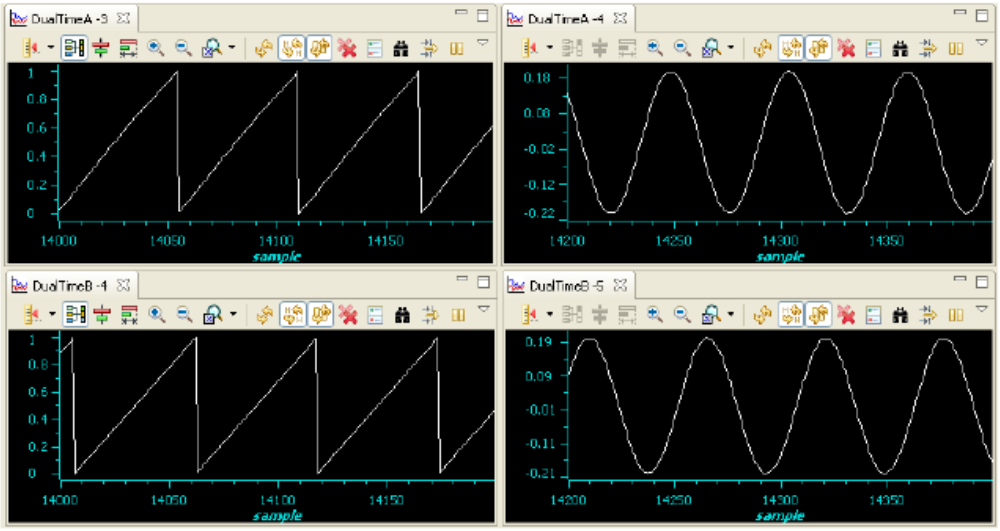
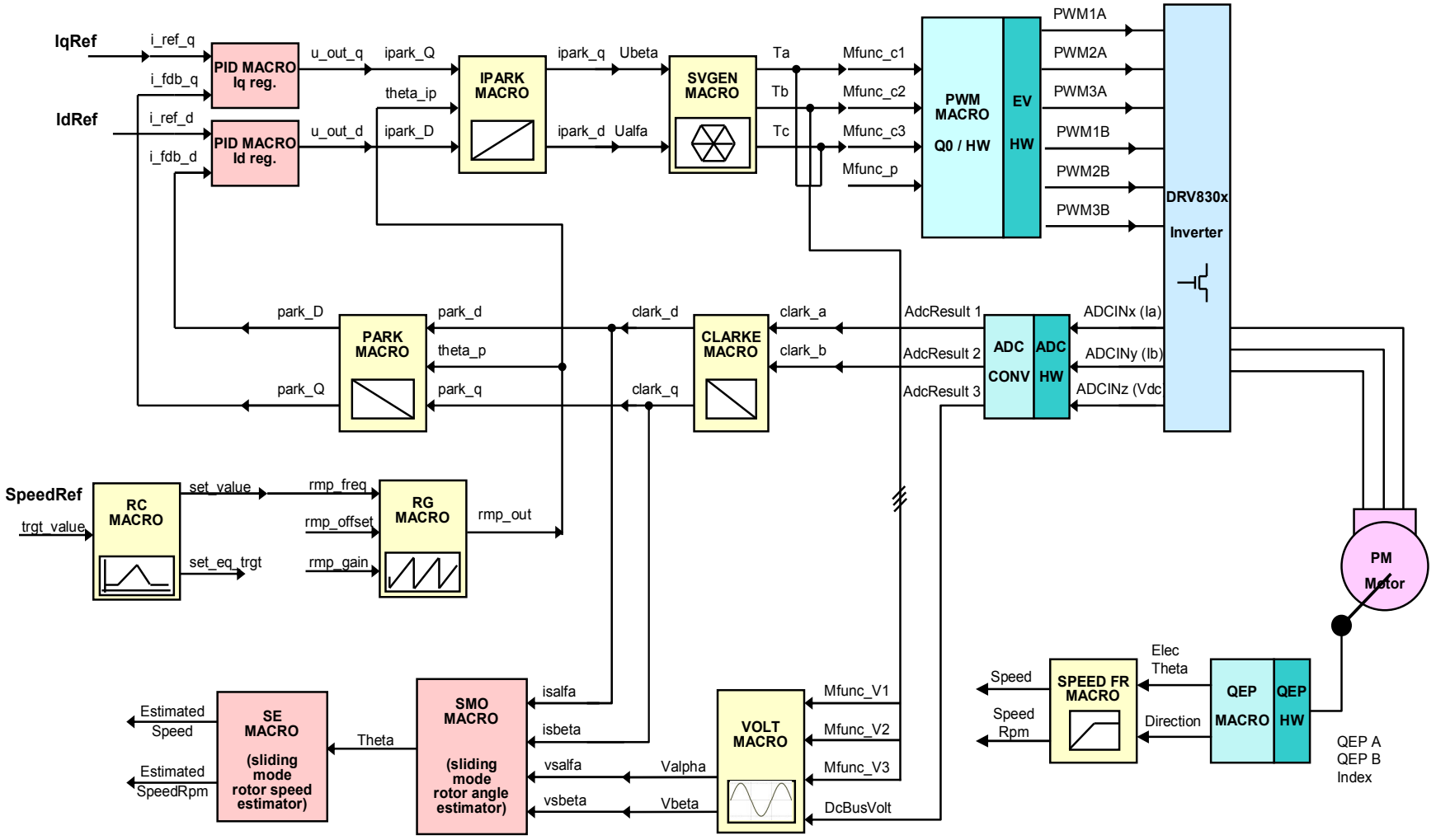



**Figure 20 Estimated theta (SMO), rg1. Out and Phase A&B currents**

# Level 5 - Incremental System Build Block Diagram




Level 5 verifies the rotor position and speed estimation performed by SMO and speed estimation modules.

**Level 6 Incremental Build**

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

Open PM_Sensorless-Settings.h and select level 6 incremental build option by setting the BUILDLEVEL to LEVEL6 (#define BUILDLEVEL LEVEL6). 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: for changing the rotor speed in per-unit.
- IdRef: for changing the d-qxis voltage in per-unit.

**Level 6A**

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

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

- Set SpeedRef to 0.25 pu (or another suitable value if the base speed is different).

- Add the soft-switch variable "lsw" to the watch window in order to switch from current loop to speed loop. In the code lsw1 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, now the motor is running around the reference speed (0.25 pu). Next, set lsw to 2 and close the speed loop.

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

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

- For speed PID 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 setting DRV_RESET to 1, taking the controller out of realtime mode and reset.

- Note that the IdRef is set to be zero all the times.

**Level 6B (Alternative method)**

Tuning both speed PID and SMO at the same time may not be easy for some applications. In order to test the SMO only without speed PID in the loop, disconnect the speed PID and Iq PID modules in the code as shown in 6B block diagram, and apply constant “Iqref” as the reference for Iq PID. After tuning SMO (Kslide), the motor should spin smoothly and the estimated angle should be clear sawtooth.

Note that, in this scheme the speed is not controlled, therefore a non-zero torque reference (Iqref) will spin the motor very fast unless loaded. Therefore keep the Iqref low initially, and load the motor using a brake, generator etc (or manually if the motor is small enough). If the motor speed is too low or the generated torque by the motor is not enough to handle the applied load, increase Iqref or reduce the amount of load. After tuning the SMO, add speed PID into the system as shown in the block diagram 7 and tune the PI coefficients, if needed. This method will help the user tune SMO and speed PID separately.

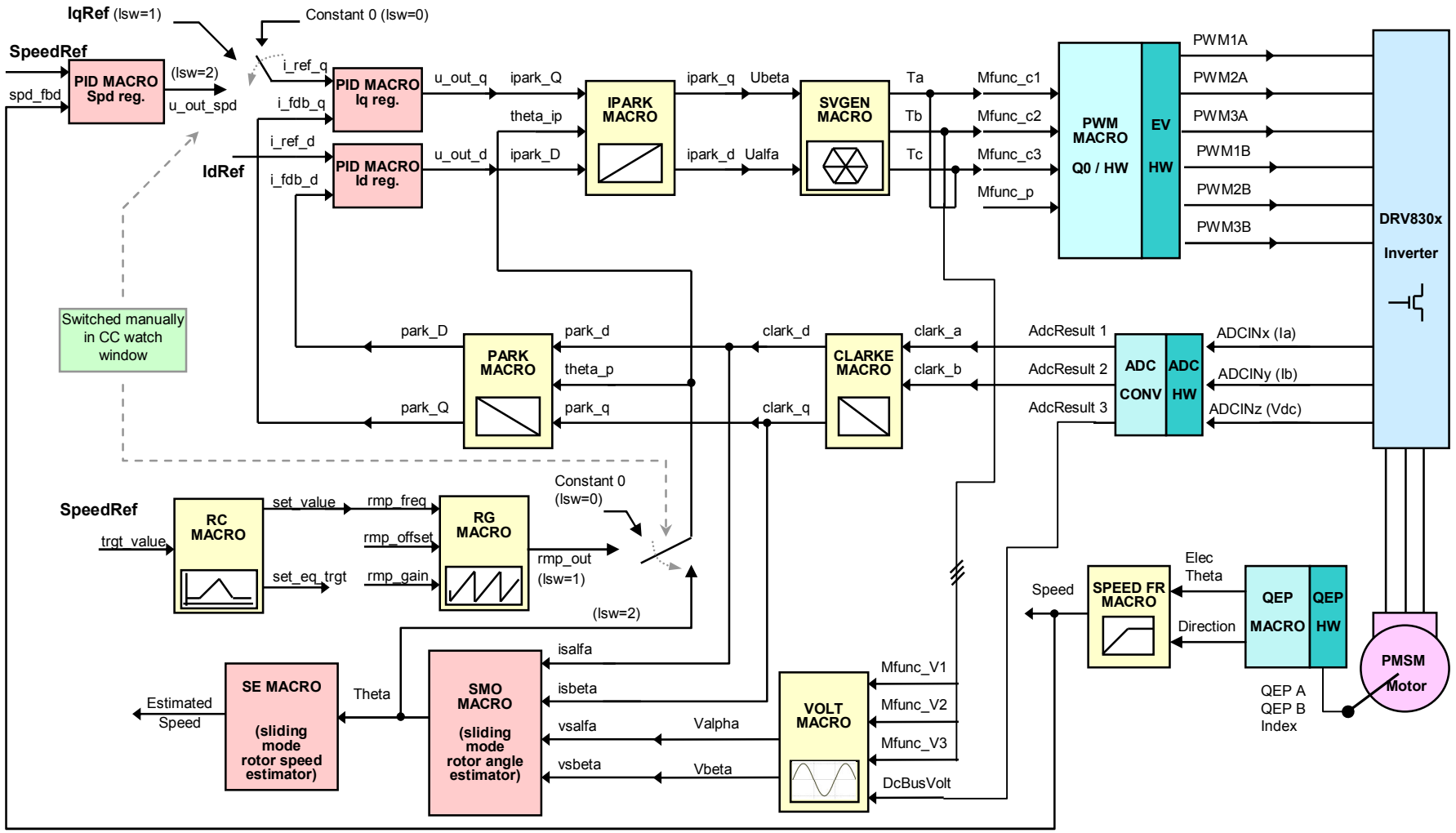
If the test is implemented on a custom inverter or a different motor is used, then

- Check the parameters in PM_Sensorless-Settings.h. Make sure that the base (pu) quantities are set to maximum measurable current, voltage etc., and motor electrical parameters are correct.

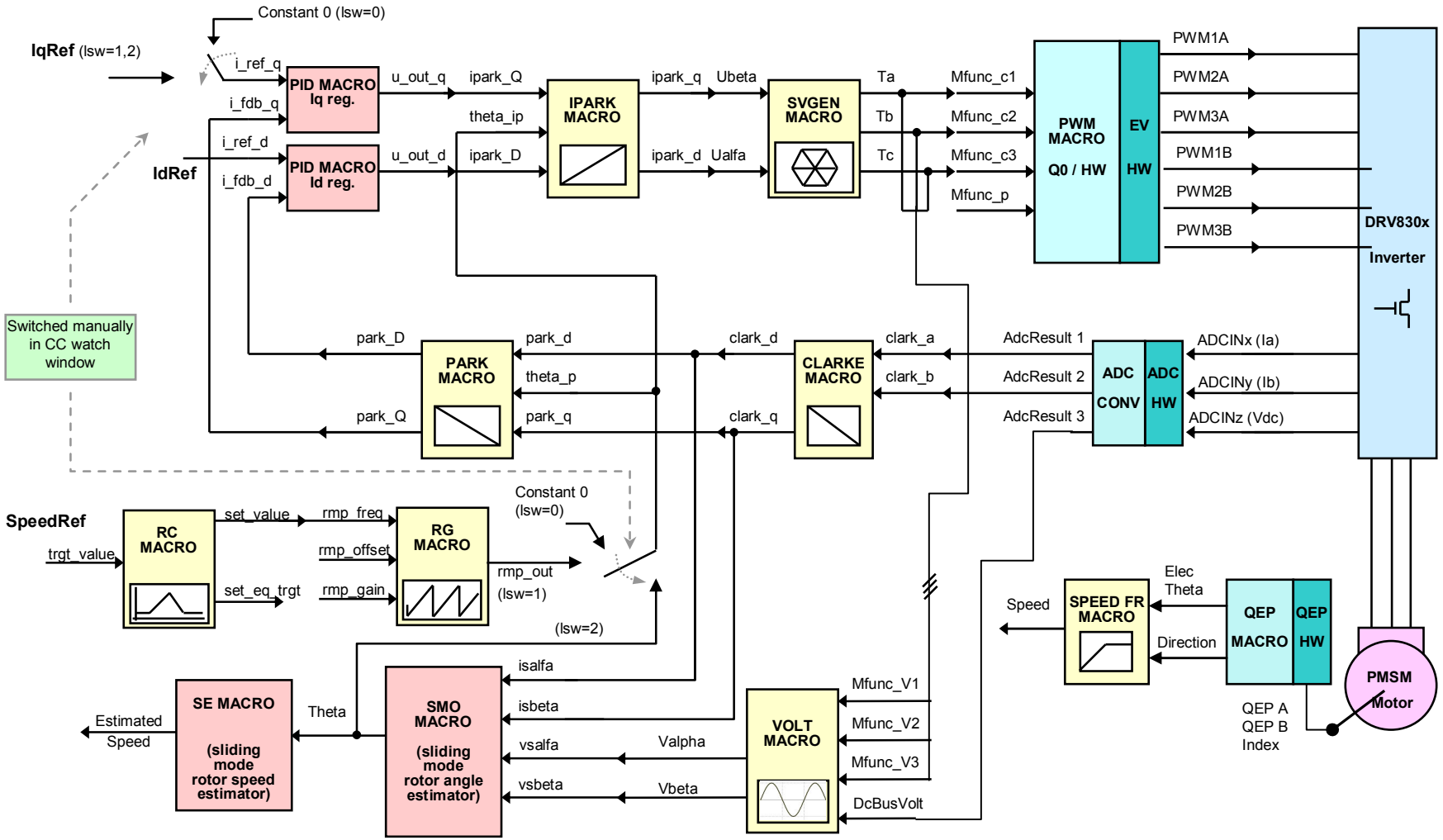
- The DC bus voltage should be high enough in order not to saturate the PID outputs.

- Run the same experiment again and keep tuning SMO gains.

# Level 6A - Incremental System Build Block Diagram



# Level 6B - Incremental System Build Block Diagram



**Level 7 Incremental Build**

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

Open PM_Sensorless-Settings.h and select level 7 incremental build option by setting the BUILDLEVEL to LEVEL7 (#define BUILDLEVEL LEVEL7). 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: for changing the rotor speed in per-unit.
- IdRef: 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 0.25 pu (or another suitable value if the base speed is different) and Iqref to 0.2 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, the motor is running with this reference speed (0.25 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 speed3.EstimatedSpeed with SpeedRef in the watch windows with continuous refresh feature whether or not it should be nearly the same.

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

- For speed PID 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 setting DRV_RESET to 1, 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\fixed\~Docs.

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

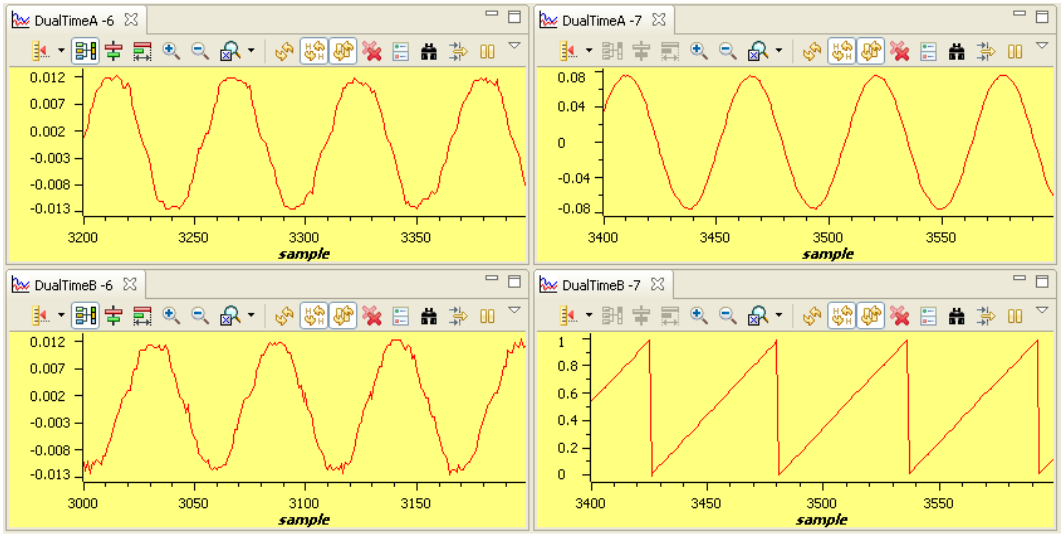



**Figure 21 Waveforms of Phase A&B currents, calculated Phase A voltage, and estimated theta by SMO under no-load and 0.3pu speed**

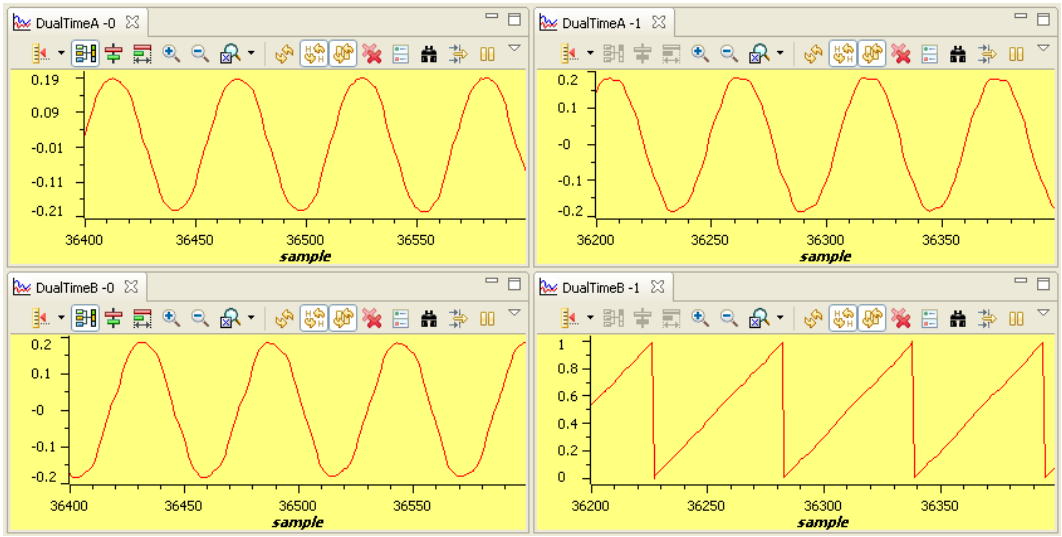



**Figure 22 Waveforms of Phase A&B currents, calculated Phase A voltage, and estimated theta by SMO under 0.33pu-load and 0.5pu speed**

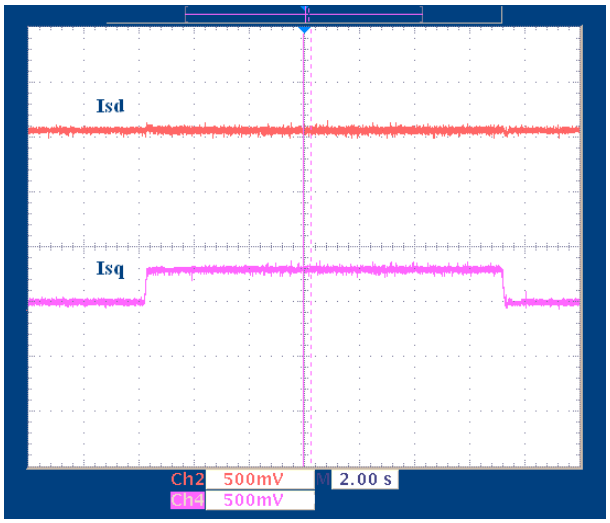
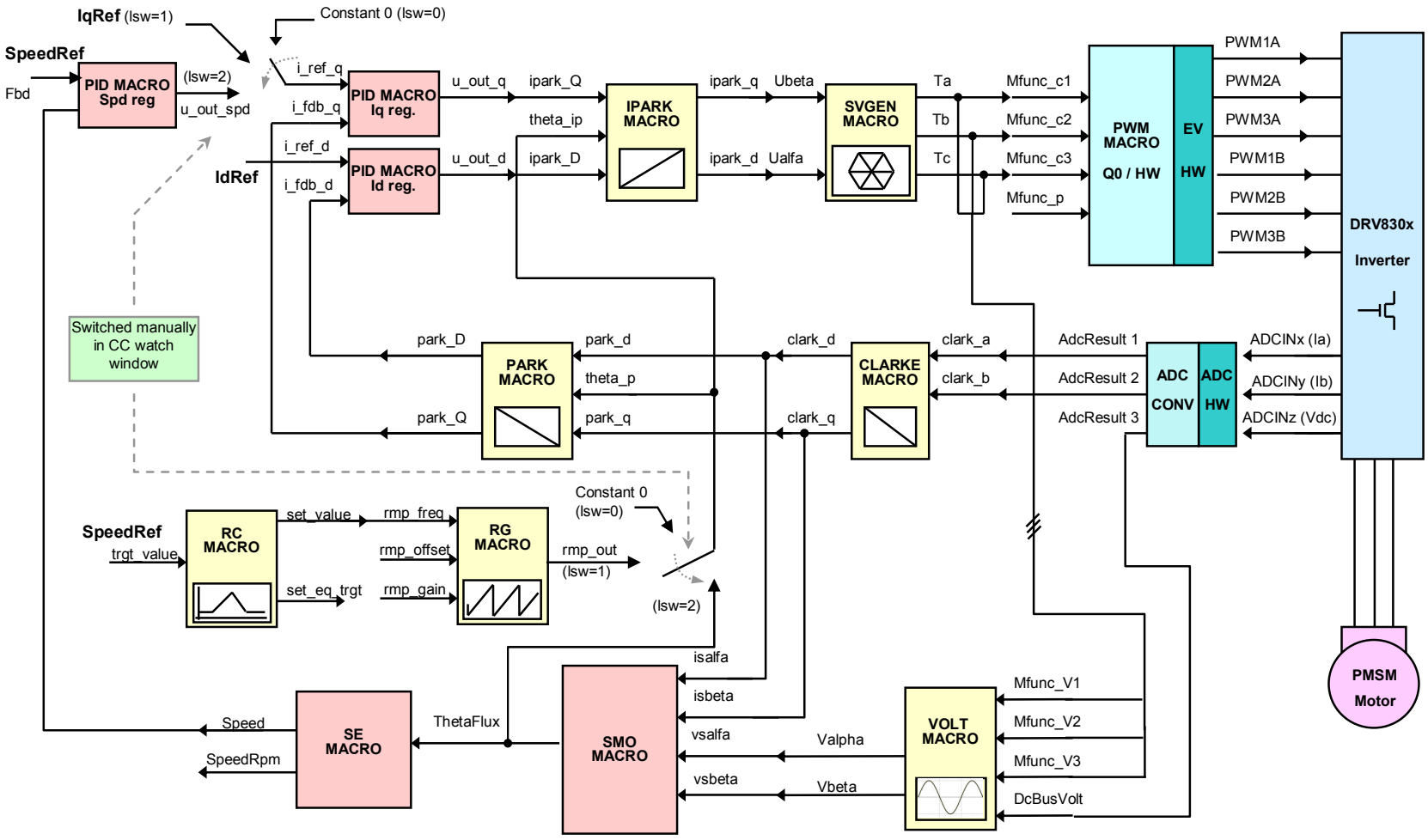



**Figure 23 Flux and torque components of the stator current in the synchronous reference frame under 0.33pu step-load and 0.5pu speed monitored from PWMDAC output**

# Level 7 Incremental System Build Block Diagram




Level 7 verifies the complete system.