

Sensored Field Oriented Control of 3-Phase Induction Motors

Bilal Akin
Manish Bhardwaj

C2000 Systems and Applications Team

Abstract

This application note presents a solution to control an AC induction motor using the TMS320F2803x microcontrollers. 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.

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	2
Induction Motors	2
Field OrientedControl.....	4
Benefits of 32-bit C2000 Controllers for Digital Motor Control	10
TI Motor Control Literature and DMC Library.....	11
System Overview	12
Hardware Configuration.....	16
Software Setup Instructions to Run HVACI_Sensored Project.....	19
Incremental System Build	20

Introduction

The motor control industry is a strong, aggressive sector. To remain competitive new products must address several design constraints including cost reduction, power consumption reduction, power factor correction, and reduced EMI radiation. In order to meet these challenges advanced control algorithms are necessary. Embedded control technology allows both a high level of performance and system cost reduction to be achieved. According to market analysis, the majority of industrial motor applications use AC induction motors. The reasons for this are higher robustness, higher reliability, lower prices and higher efficiency (up to 80%) on comparison with other motor types. However, the use of induction motors is challenging because of its complex mathematical model, its non linear behavior during saturation and the electrical parameter oscillation which depends on the physical influence of the temperature. These factors make the control of induction motor complex and call for use of a high performance control algorithms such as “vector control” and a powerful microcontroller to execute this algorithm.

During the last few decades the field of controlled electrical drives has undergone rapid expansion due mainly to the benefits of micro-controllers. These technological improvements have enabled the development of very effective AC drive control with lower power dissipation hardware and more accurate control structures. The electrical drive controls become more accurate in the sense that not only are the DC quantities controlled but also the three phase AC currents and voltages are managed by so-called vector controls. This document briefly describes the implementation of the most efficient form of a vector control scheme: the Field Orientated Control method. It is based on three major points: the machine current and voltage space vectors, the transformation of a three phase speed and time dependent system into a two coordinate time invariant system and effective Space Vector Pulse Width Modulation pattern generation. Thanks to these factors, the control of AC machine acquires every advantage of DC machine control and frees itself from the mechanical commutation drawbacks. Furthermore, this control structure, by achieving a very accurate steady state and transient control, leads to high dynamic performance in terms of response times and power conversion.

Induction Motors

Induction motors derive their name from the way the rotor magnetic field is created. The rotating stator magnetic field induces currents in the short circuited rotor. These currents produce the rotor magnetic field, which interacts with the stator magnetic field, and produces torque, which is the useful mechanical output of the machine.

The three phase squirrel cage AC induction motor is the most widely used motor. The bars forming the conductors along the rotor axis are connected by a thick metal ring at the ends, resulting in a short circuit as shown in Fig.1. The sinusoidal stator phase currents fed in the stator coils create a magnetic field rotating at the speed of the stator frequency (ω_s). The changing field induces a current in the cage conductors, which results in the creation of a second magnetic field around the rotor wires. As a consequence of the forces created by the interaction of these two fields, the rotor experiences a torque and starts rotating in the direction of the stator field.

As the rotor begins to speed up and approach the synchronous speed of the stator magnetic field, the relative speed between the rotor and the stator flux decreases, decreasing the induced voltage in the stator and reducing the energy converted to torque. This causes the torque production to drop off, and the motor will reach a steady state at a point where the load torque is matched with the motor torque. This point is an equilibrium reached depending on the instantaneous loading of the motor. In brief:

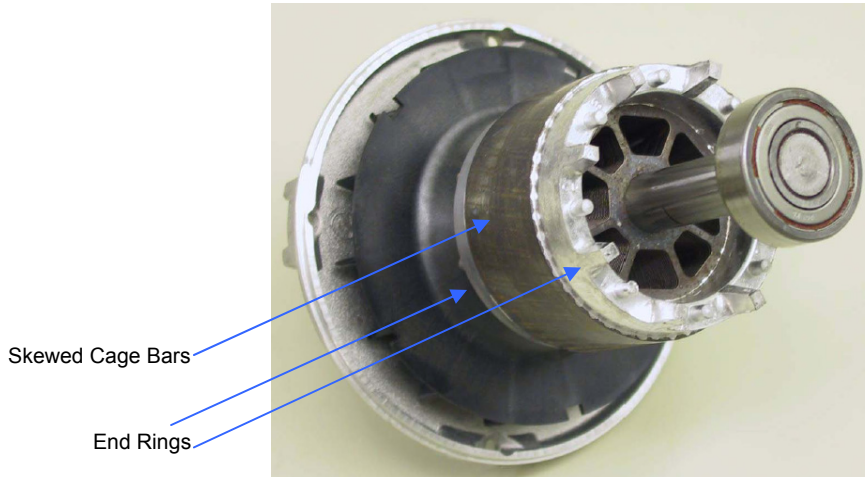


Figure 1: Induction Motor Rotor

- Owing to the fact that the induction mechanism needs a relative difference between the motor speed and the stator flux speed, the induction motor rotates at a frequency near, but less than that of the synchronous speed.
- This slip must be present, even when operating in a field-oriented control regime.
- The rotor in an induction motor is not externally excited. This means that there is no need for slip rings and brushes. This makes the induction motor robust, inexpensive and need less maintenance.
- Torque production is governed by the angle formed between the rotor and the stator magnetic fluxes.

In Fig.2 the rotor speed is denoted by Ω . Stator and rotor frequencies are linked by a parameter called the slip s , expressed in per unit as $s = (\omega_s - \omega_r) / \omega_s$.

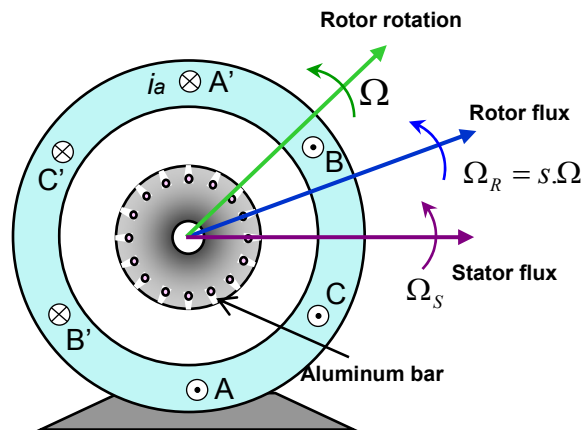


Figure 2: Squirrel cage rotor AC induction motor cutaway view

<p>- stator rotating field speed (rad / s) : $\Omega_S = \frac{\omega_S}{p} \cdot \begin{cases} \omega_S : \text{AC supply freq (rad / s)} \\ p : \text{stator poles pairs number} \end{cases}$</p> <p>- rotor rotating speed (rad / s) : $\Omega = (1 - s) \cdot \Omega_S = (1 - s) \frac{\omega_S}{p}$</p>
--

where s is called the “slip”: it represents the difference between the synchronous frequency and the actual motor rotating speed.

Field Oriented Control

Introduction

A simple control such as the V/Hz strategy has limitations on the performance. To achieve better dynamic performance, a more complex control scheme needs to be applied, to control the induction 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 an AC induction motor. Such decoupled 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.**

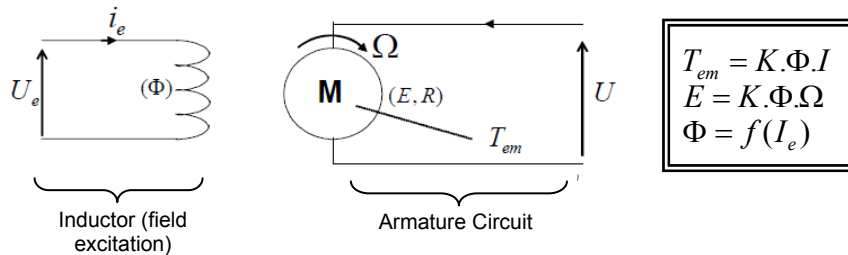


Fig 3. Separated excitation DC motor model, flux and torque are independently controlled and the current through the rotor windings determines how much torque is produced.

Induction 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.

Why Field Oriented Control

As a well known fact about the asynchronous machine, we face some natural limitations with a V/Hz control approach. FOC control will allow us to get around these limitations, by decoupling the effect of the torque and the magnetizing flux. With decoupled control of the magnetization, the torque producing component of the stator flux can now be thought of as independent torque control. Now, decoupled

control, at low speeds, the magnetization can be maintained at the proper level, and the torque can be controlled to regulate the speed.

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.

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 coordinate (d and q coordinates) 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 coordinate) and the flux component (aligned with d coordinate). 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:

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

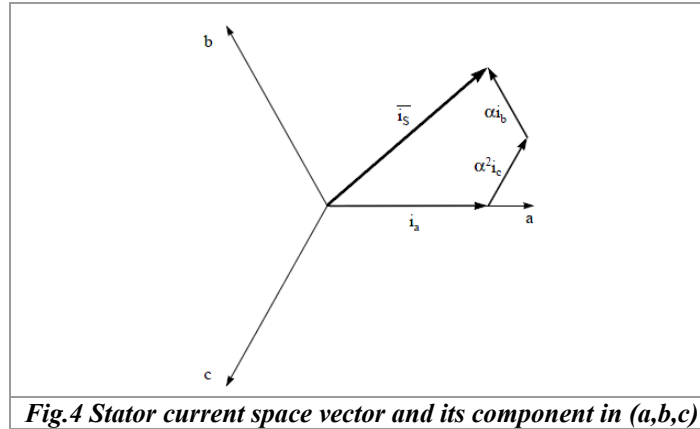
By maintaining the amplitude of the rotor flux (ψ_R) at a fixed value we have a linear relationship between torque and torque component (i_{Sq}). We can then control the torque by controlling the torque component of stator current vector.

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:

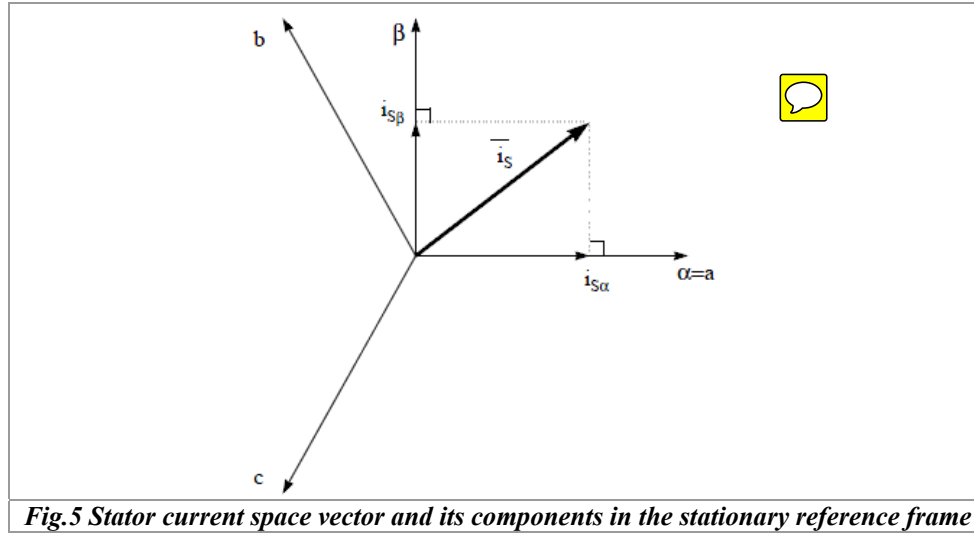


where (a,b,c) are the three phase system axes. This current space vector depicts the three phase sinusoidal system. It still needs to be transformed into a two time invariant coordinate system. This transformation can be split into two steps:

- (a,b,c) \Rightarrow (α, β) (the Clarke transformation) which outputs a two coordinate time variant system
- (α, β) \Rightarrow (d,q) (the Park transformation) which outputs a two coordinate time invariant system

The (a,b,c) \Rightarrow (α, β) 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:



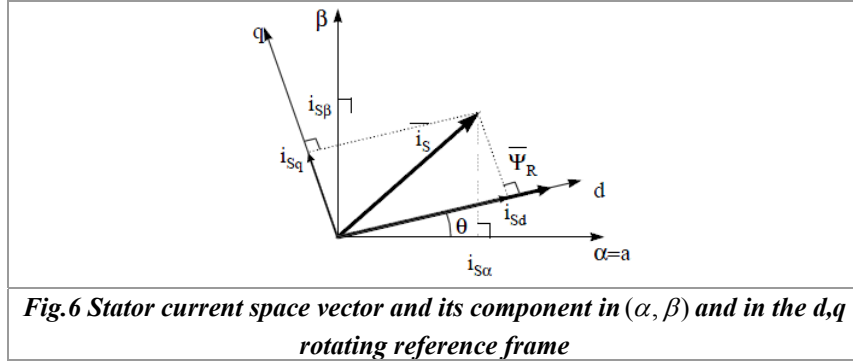
The projection that modifies the three phase system into the (α, β) two dimension orthogonal system is presented below.

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

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:



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:

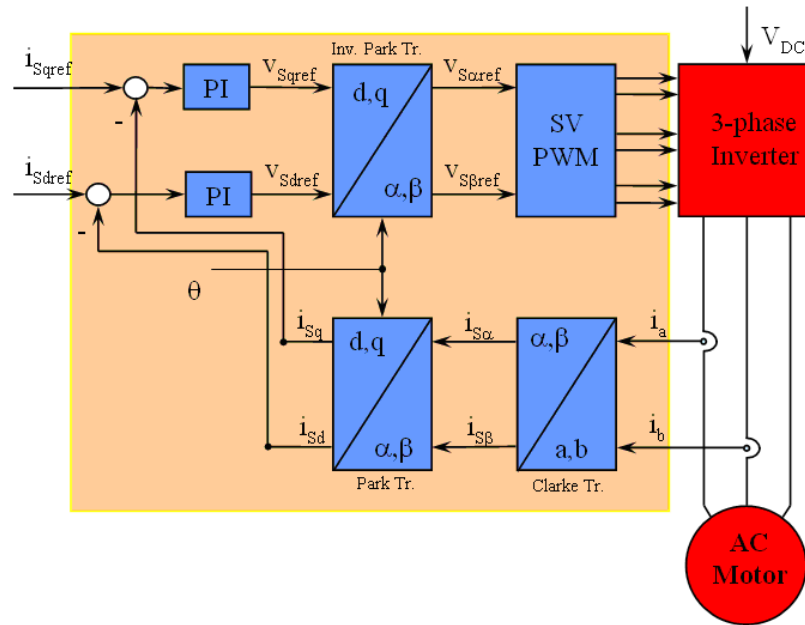
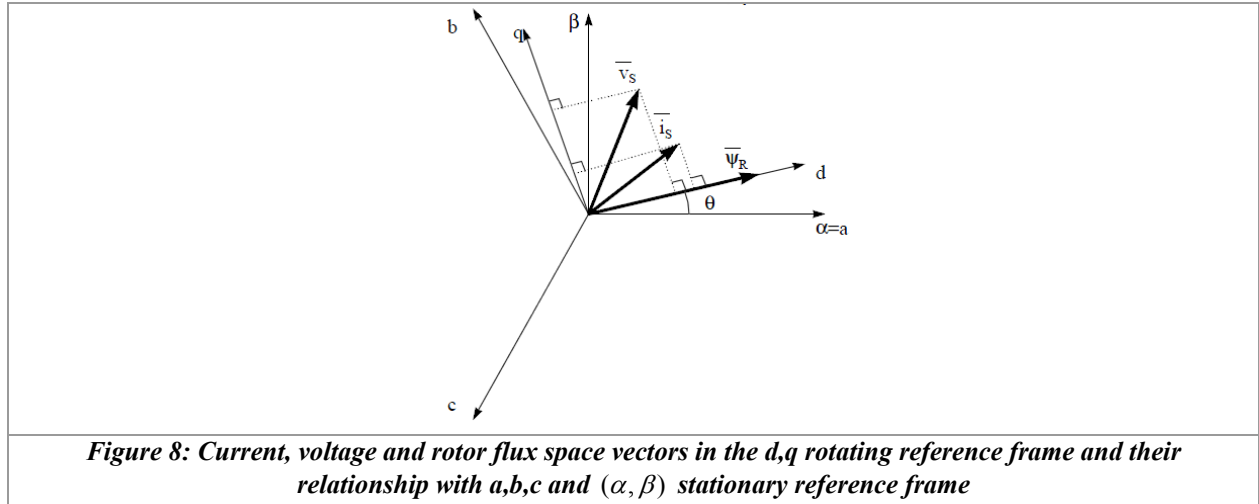


Fig7 Basic scheme of FOC for ACI motor

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_{sdrref} (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 induction 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_{sdrref} should be set to zero. As induction 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_{sdrref} and V_{sqref} ; they are applied to the inverse Park transformation. The outputs of this projection are V_{saref} and V_{sbref} 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

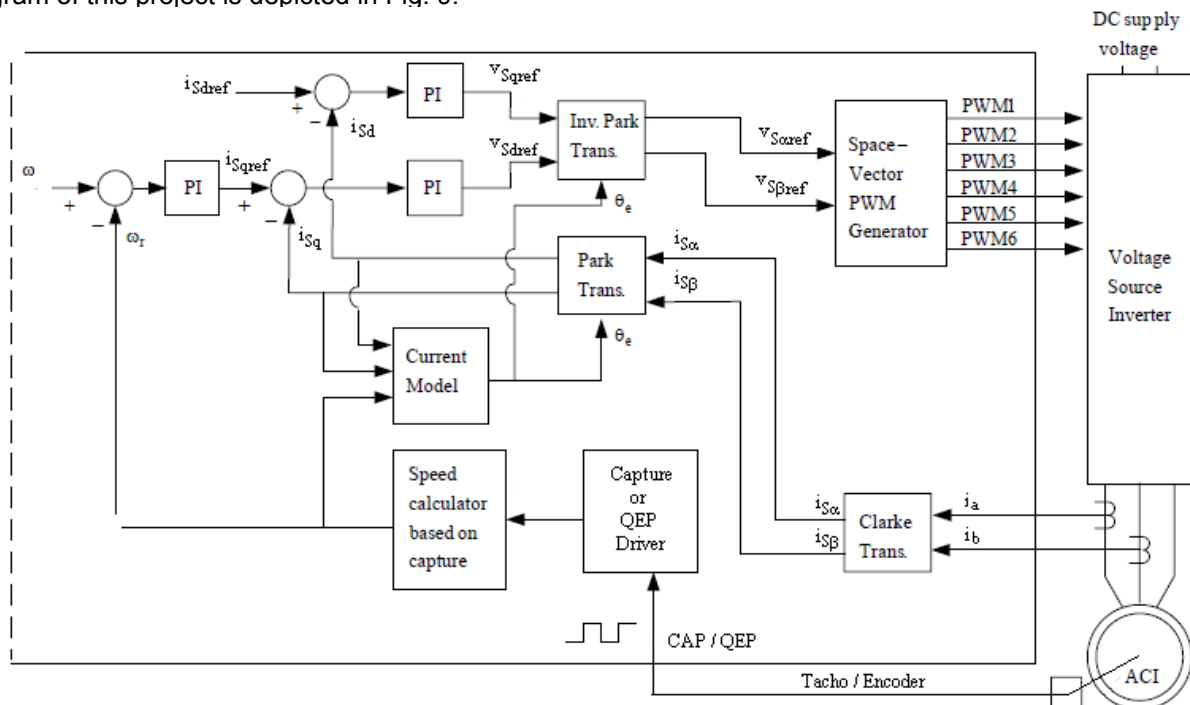
Knowledge of the rotor flux position is the core of the FOC. In fact if there is an error in this variable the rotor flux is not aligned with d -axis and i_{sd} and i_{sq} are incorrect flux and torque components of the stator current. The following diagram shows the (a,b,c) , (α,β) and (d,q) reference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



The measure of the rotor flux position is different if we consider synchronous or induction 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 induction motor drive can be mainly categorized into two types; indirect and direct schemes. The field to be oriented could be rotor, stator, or airgap flux linkage. In the indirect field oriented control, the slip estimation with measured or estimated rotor speed is required in order to compute the synchronous speed. There is no flux estimation appearing in the system. For the direct scheme, the synchronous speed is computed basing on the flux angle which is available from flux estimator or flux sensors (e.g., Hall effects). In this implementing system, the indirect flux oriented control system with measured speed based on capture is described. The overall block diagram of this project is depicted in Fig. 9.



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

C2000 family of devices possess 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, 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

Literature distinguishes two types of FOC control (for Induction Motors):

- 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.

This document discusses the indirect FOC control.

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\fixed
- 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 continues in the next line. Any character including invisible ones like “space” or “tab” 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 sensed field oriented control of induction motors. The “C” framework is designed to run on TMS320F2803x based controllers on Code Composer Studio. The framework uses the following modules¹:

Macro Names	Explanation
CLARKE	Clarke Transformation
PARK / IPARK	Park and Inverse Park Transformation
PID	PID Regulators
RC	Ramp Controller (slew rate limiter)
RG	Ramp / Sawtooth Generator
QEP / CAP	QEP and CAP Drives
SPEED_PR	Speed Measurement (based on sensor signal period)
SPEED_FR	Speed Measurement (based on sensor signal frequency)
CURMOD	Current Model for Sensed Applications
SVGEN	Space Vector PWM with Quadrature Control (includes IClarke Trans.)
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 sensed Indirect Field Oriented Control of Induction Motor will be experimented with and will explore the performance of speed control. The induction 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 induction motor (i_a and i_b) are measured from the inverter and sent to the TMS320x2803x via two analog-to-digital converters (ADCs).

HVACI_Sensored project has the following properties:

C Framework		
System Name	Program Memory Usage 2803x	Data Memory Usage ¹ 2803x
HVACI_Sensored	3901 words ²	1236 words

¹ Excluding the stack size

² Excluding “IQmath” Look-up Tables

CPU Utilization of Sensored FOC of ACI	
Name of Modules *	Number of Cycles
Ramp Controller	29
Clarke Tr.	28
Park Tr.	142
I_Park Tr.	41
Current Mode	125
Speed Freq.	67
Qep Drv	103
3 x Pid	167
Space Vector Gen.	137
Pwm Drv	74
Contxt Save etc.	53
Pwm Dac (optional)	
DataLog (optional)	
Ramp Gen (optional)	
Total Number of Cycles	966 **
CPU Utilization @ 60 Mhz	16.1% ***
CPU Utilization @ 40 Mhz	24.2% ***

* The modules are defined in the header files as “macros”

** 1154 including the optional modules

*** At 10 kHz ISR freq.

System Features	
Development /Emulation	Code Composer Studio V4.0(or above) with Real Time debugging
Target Controller	TMS320F2803x
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 PWM 5A, 6A, 7A & 7B for DAC outputs QEP1 A,B, I or CAP1 ADC A1 for DC Bus voltage sensing, B4 & B6 for phase current sensing

The overall system implementing a 3-ph induction motor control is depicted in Fig.10. The induction 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 induction motor (i_a and i_b) are measured from the inverter and they are sent to the TMS320F2803x via two analog-to-digital converters (ADCs).

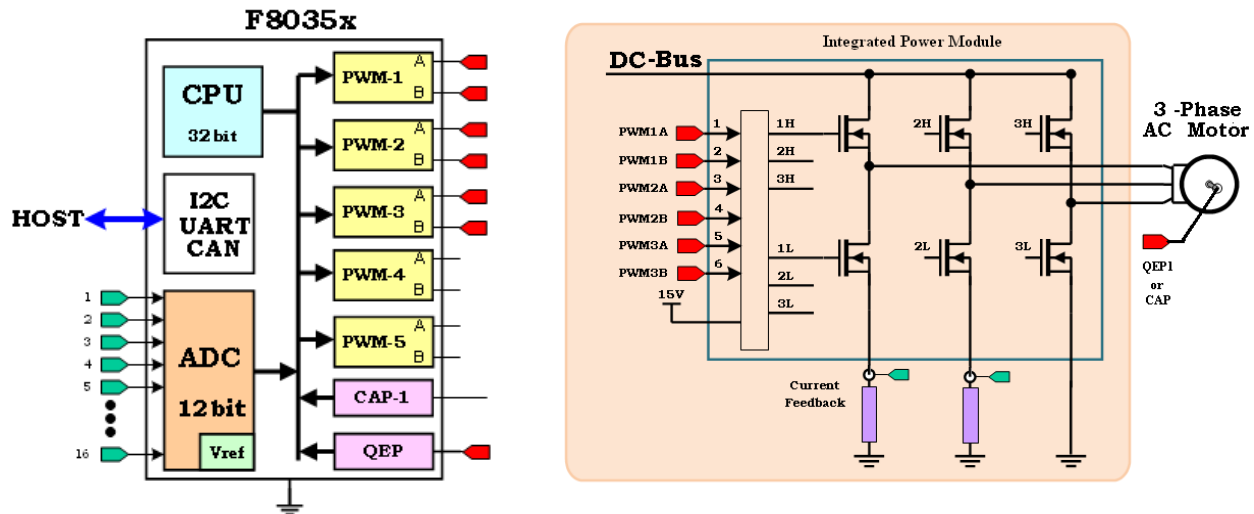


Fig 10 A 3-ph induction motor drive implementation

The software flow is described in the Figure 11 below.

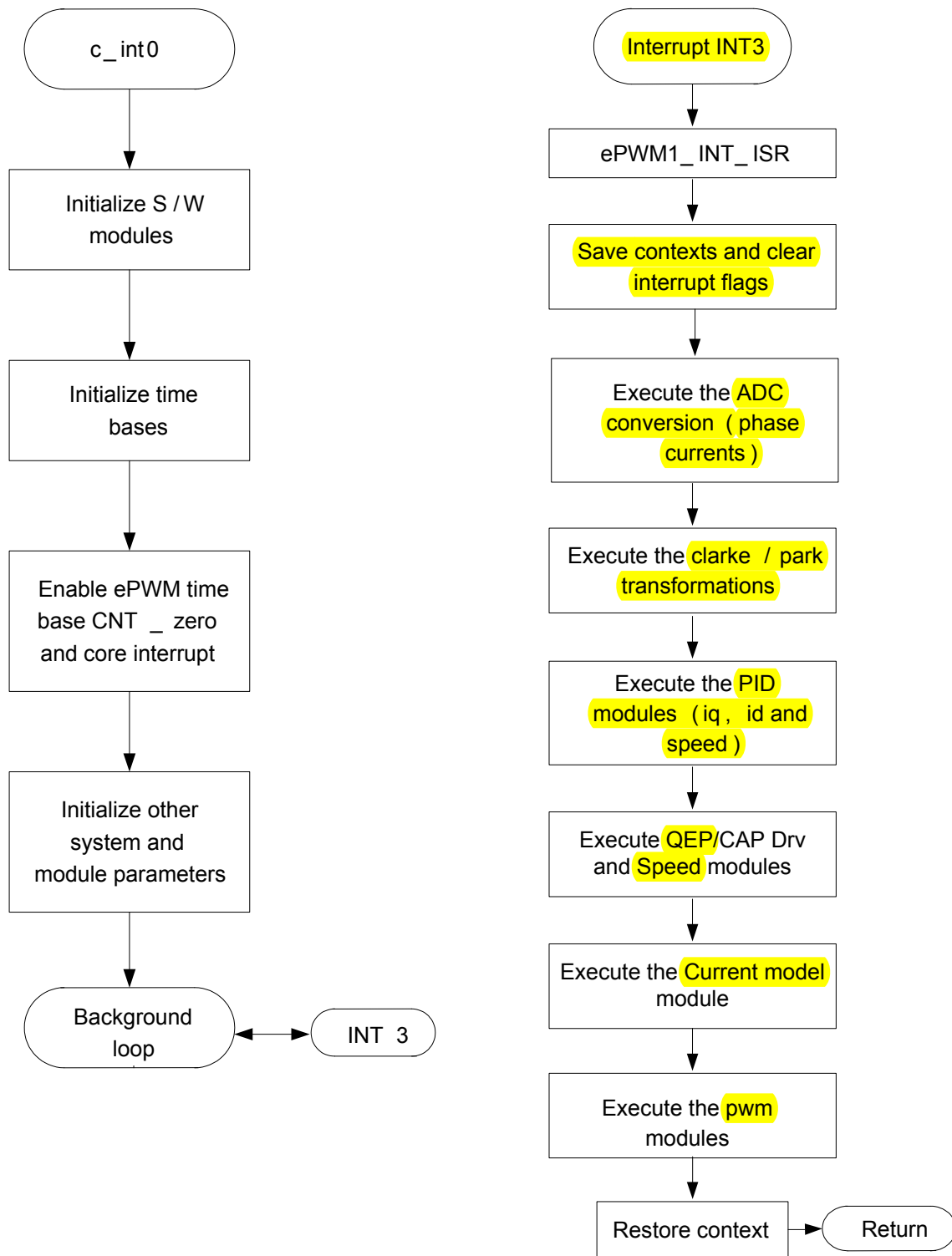


Fig.11 System software flowchart

Hardware Configuration (HVDMC Kit)

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

C:\TI\controlSUITE\development_kits\HVMotorCtrl+PfcKit\~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]-J6, J7 and J8, J9 for 3.3V, 5V and 15V power rails and JTAG reset line, make sure that the jumpers [Main]-J3, J4 and J5 are not populated.
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 level 1 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 220V, the motor can run only at a certain speed and torque range properly without saturating the PID 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.

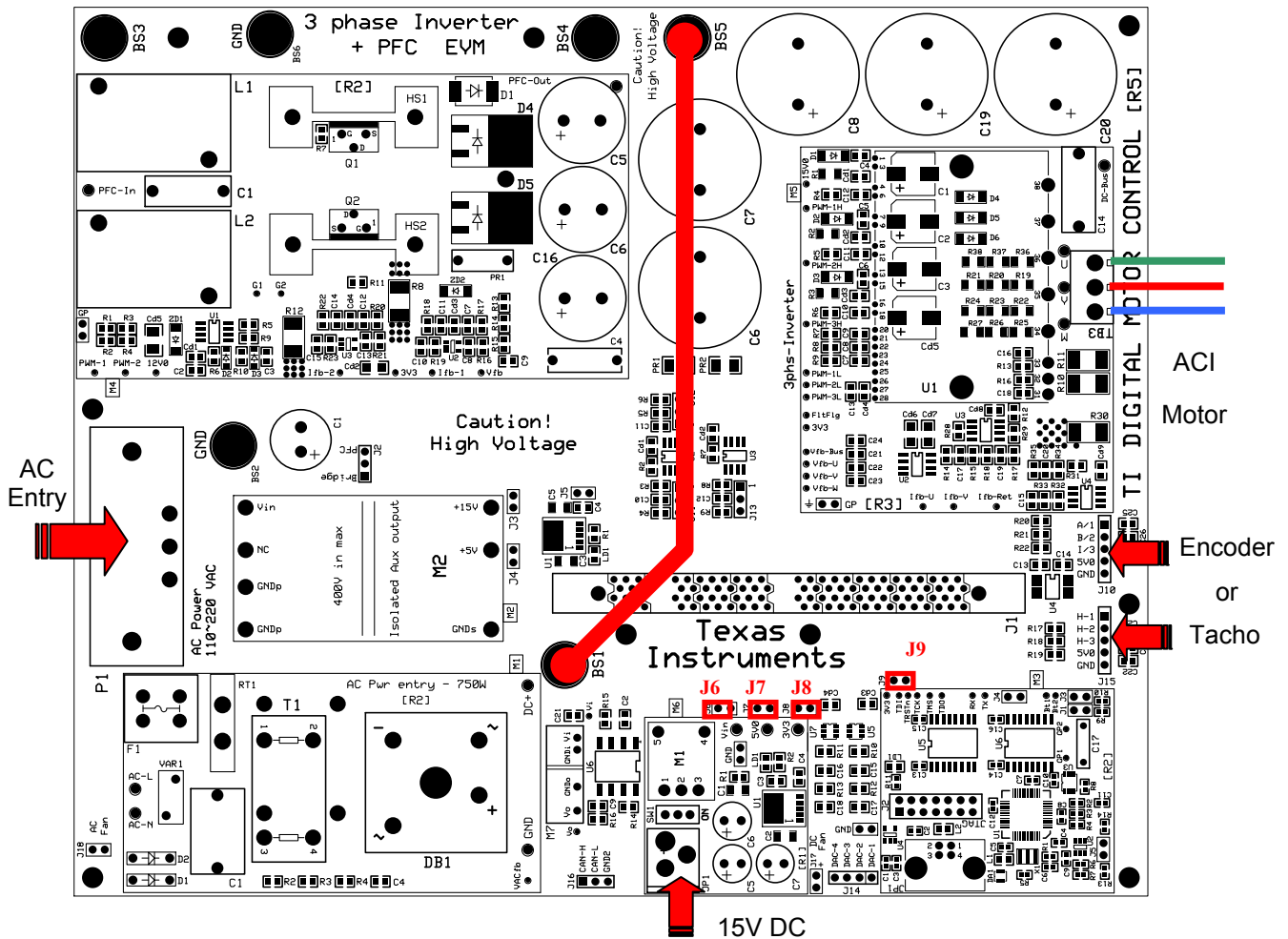



Fig. 12 Using AC Power to generate DC Bus Power



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

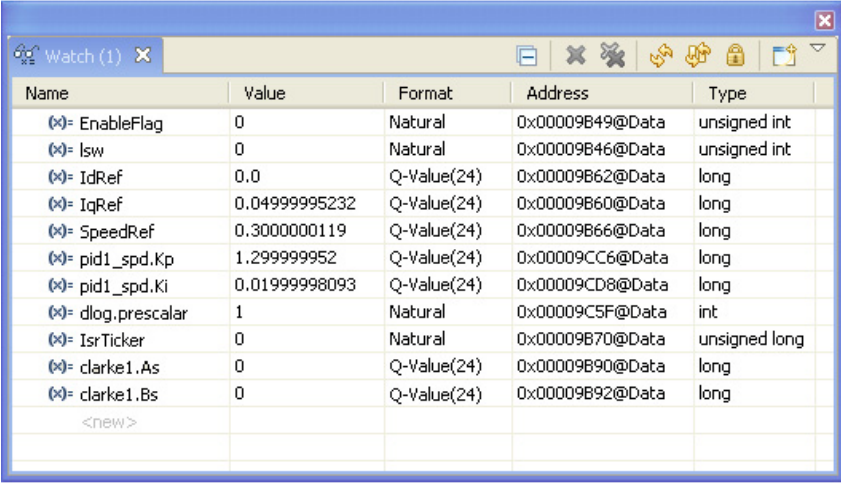
Software Setup Instructions to Run HVACI_Sensored Project

Please refer to the “Software Setup for HVMotorCtrl+PFC Kit Projects” section in the HVMotorCtrl+PFC Kit How to Run Guide which can be found at

C:\TI\controlSUITE\development_kits\HVMotorCtrl+PfcKit\~Docs


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

Select the HVACI_Sensored as the active project. Select the active build configuration to be set as F2803x_RAM. Verify that the build level is set to 1, 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 Q format for them.



Name	Value	Format	Address	Type
(*)= EnableFlag	0	Natural	0x00009B49@Data	unsigned int
(*)= lsw	0	Natural	0x00009B46@Data	unsigned int
(*)= IdRef	0.0	Q-Value(24)	0x00009B62@Data	long
(*)= IqRef	0.04999995232	Q-Value(24)	0x00009B60@Data	long
(*)= SpeedRef	0.3000000119	Q-Value(24)	0x00009B66@Data	long
(*)= pid1_spd.Kp	1.299999952	Q-Value(24)	0x00009CC6@Data	long
(*)= pid1_spd.Ki	0.01999998093	Q-Value(24)	0x00009CD8@Data	long
(*)= dlog.prescalar	1	Natural	0x00009C5F@Data	int
(*)= IsrTicker	0	Natural	0x00009B70@Data	unsigned long
(*)= clarke1.As	0	Q-Value(24)	0x00009B90@Data	long
(*)= clarke1.Bs	0	Q-Value(24)	0x00009B92@Data	long
<new>				

Table 1 Watch Window Variables

Go to Tools → Graph → Dual Time, and click the Import button at the bottom of the window. Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location C:\TI\ControlSUITE\development_kits\HVMotorCtrl+PfcKit\HVACI_sensored\. 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 so the final system can be confidently operated. Five 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
PWMDAC_MACRO	√	√	√	√	√
RC_MACRO	√	√	√	√	
RG_MACRO	√	√	√	√	
IPARK_MACRO	√√	√	√	√	√
SVGEN_MACRO	√√	√	√	√	√
PWM_MACRO	√√	√	√	√	√
CLARKE_MACRO		√√	√	√	√
PARK_MACRO		√√	√	√	√
CAP_MACRO			√√	√	
SPEED_PR_MACRO			√√	√	
QEP_MACRO			√√	√	√
SPEED_FR_MACRO			√√	√	√
PID_MACRO (IQ)			√√	√	√
PID_MACRO (ID)			√√	√	√
CURMOD				√√	√
PID_MACRO (SPD)					√√
Note: the symbol √ means this module is using and the symbol √√ means this module is testing in this phase.					

Table 2 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 “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_MACRO (inverse park transformation) and SVGEN_MACRO (space vector generator) modules and the peripheral dependent PWM_MACRO (PWM initializations and update) modules. Open HVACI_Sensored-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 (Q24): for changing the rotor speed in per-unit.
- VdTesting (Q24): for changing the d-qxis voltage in per-unit.
- VqTesting (Q24): for changing the q-axis voltage in per-unit.

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 Fig. 14 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 (PWM-1H to 3H and PWM-1L to 3L) and make sure that the PWM module is running properly.

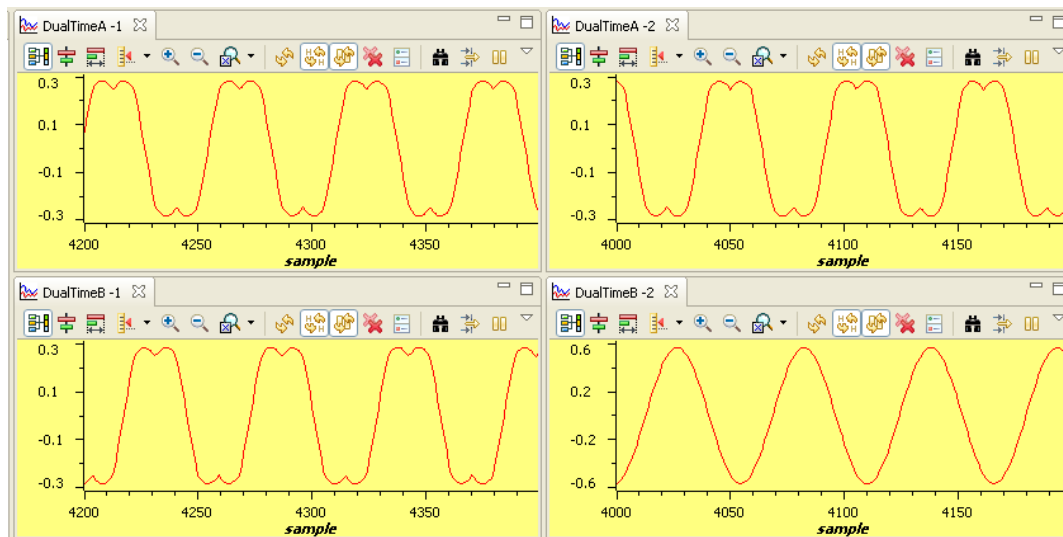


Fig.14 SVGEN duty cycle outputs Ta, Tb, Tc and Tb-Tc

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 1st-order low-pass filter RC circuit is used 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, $R=1.8k\Omega$ 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 $2.2k\Omega$ and $220nF$ on the board. Refer to application note *SPRAA88A* for more details at TI website.

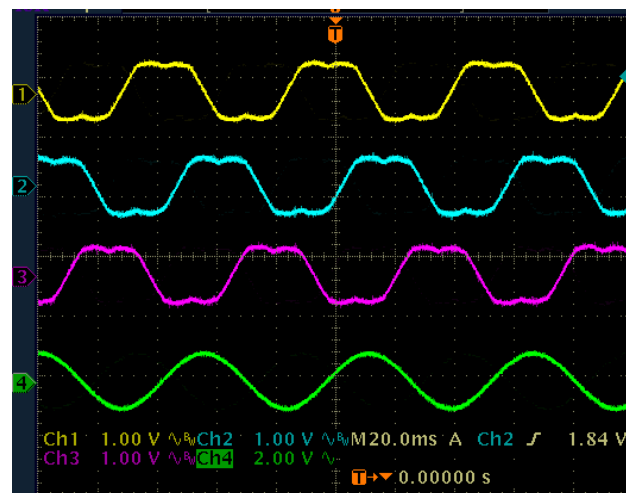


Fig.15 DAC-1-4 outputs showing Ta, Tb, Tc and Ta-Tb waveforms

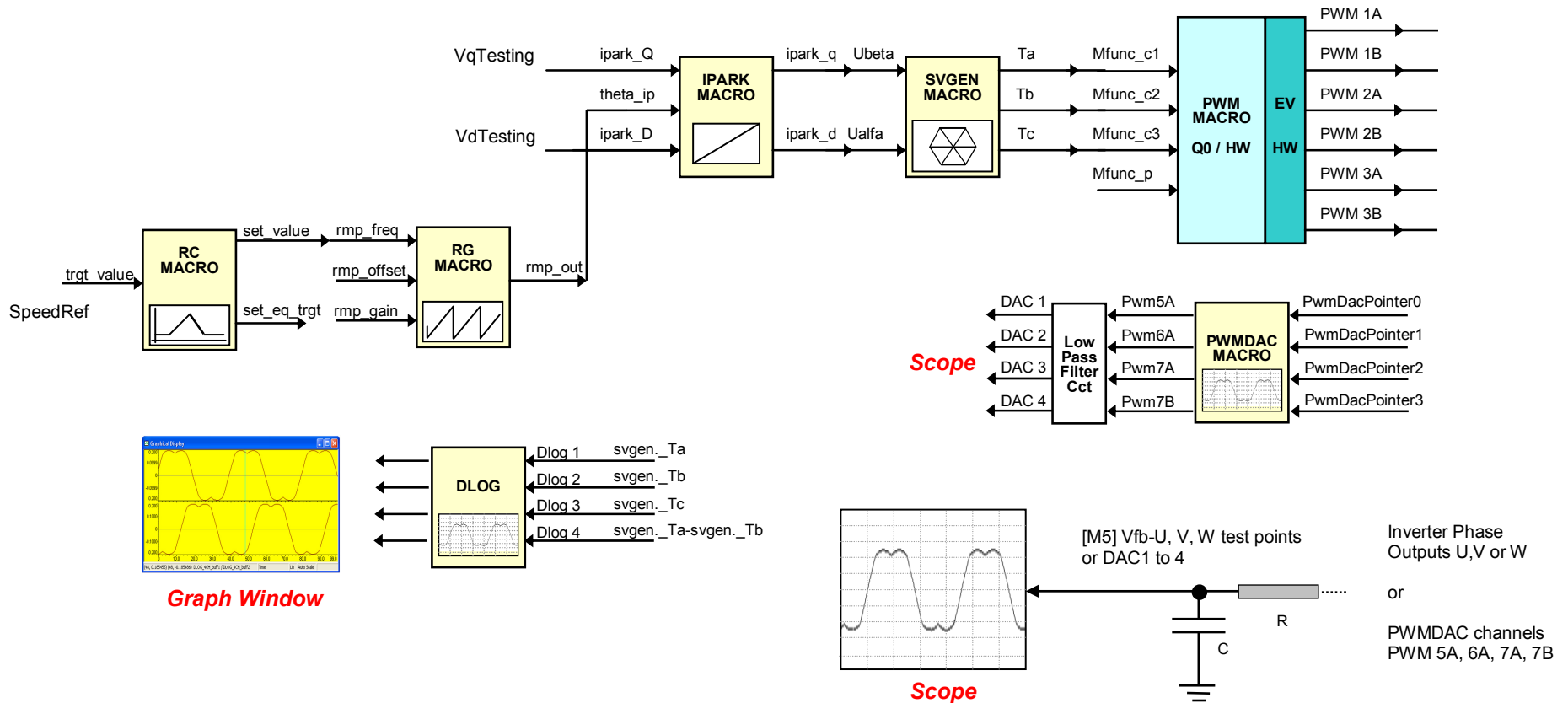
Level 1C – PWM_MACRO and INVERTER Testing

After verifying SVGEN_MACRO module in Level 1A, the PWM_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 ([M5]-R19 to27 & [M5]-C21 to 23) enable the generation of the waveform and ensures thata the inverter is working appropriately. Note that the default RC values are optimized for BLDC back-emf sensing and high frequency noise filtering. A value closer to 0.1uf for the [M5]-C21 to C23 would give better waveforms.



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 update. The motors are 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.

Open `HVACI_Sensored-Settings.h` and select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (`#define BUILDLEVEL LEVEL2`) and save the file. 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 be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

- SpeedRef (Q24): for changing the rotor speed in per-unit.
- VdTesting(Q24): for changing the d-qxis voltage in per-unit.
- VqTesting(Q24): for changing the q-axis voltage in per-unit.

Phase 2A – Testing the Clarke module

In this part the Clarke module will be tested. Now, gradually increase the DC bus voltage. 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° at the same magnitude.

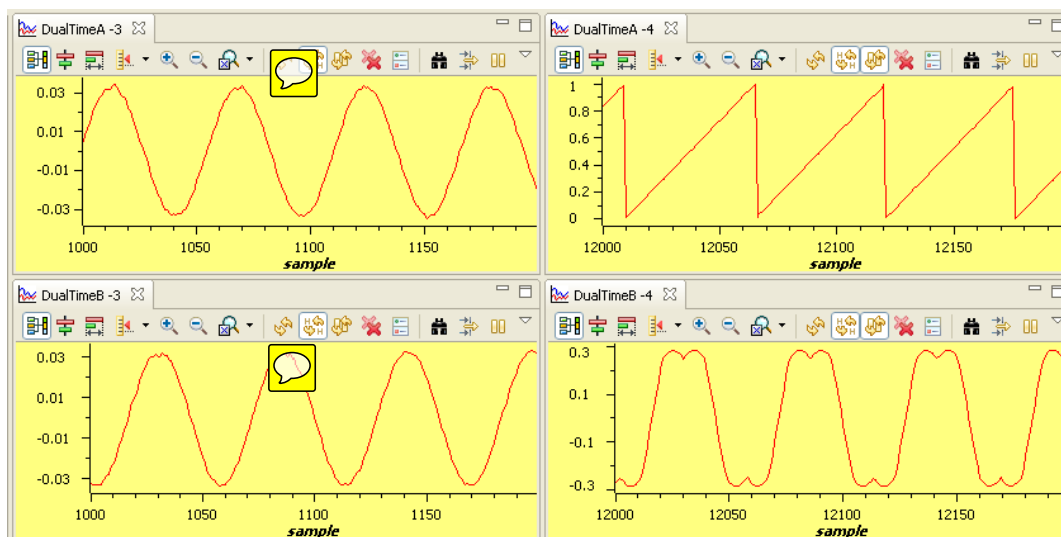


Fig 16 The waveforms of Phase A&B current, rg1.Out and svgen_dq1.Ta (duty cycle)

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.

Since the low side current measurement technique is used employing shunt resistors on inverter phase legs, the phase current waveforms observed from current test points ([M5]-Ifb-U, and [M5]-Ifb-V) are composed of pulses as shown in Fig 17

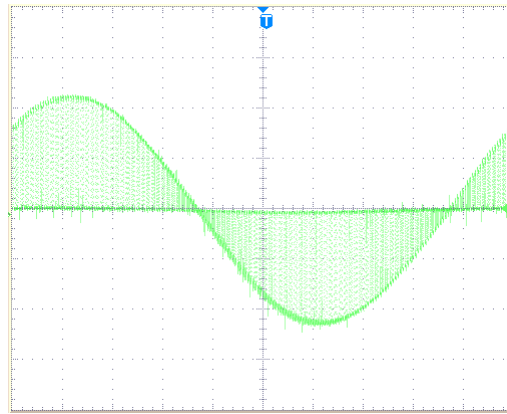


Fig.17 Amplified Phase A current

Level 2B – Calibrating the Phase Current Offset

Note that especially the low power motors draw low amplitude current after closing the speed loop under no-load. The performance of the sensed 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.

Set VdTesting, VqTesting and SpeedRef to zero in the code, recompile and run the system and watch the clarke1.As & clarke1.Bs from watch window. Make sure that the clarke1.As & clarke1.Bs values are less than 0.001 or minimum possible. If not, adjust the offset value in the code by going to :

```
clarke1.As = _IQ15toIQ( (AdcResult.ADCRESULT0<<3) - _IQ15(0.50) )<<1
```

and changing IQ15(0.50) offset value (e.g. IQ15(0.5087) or IQ15(0.4988) depending on the sign and amount of the offset)

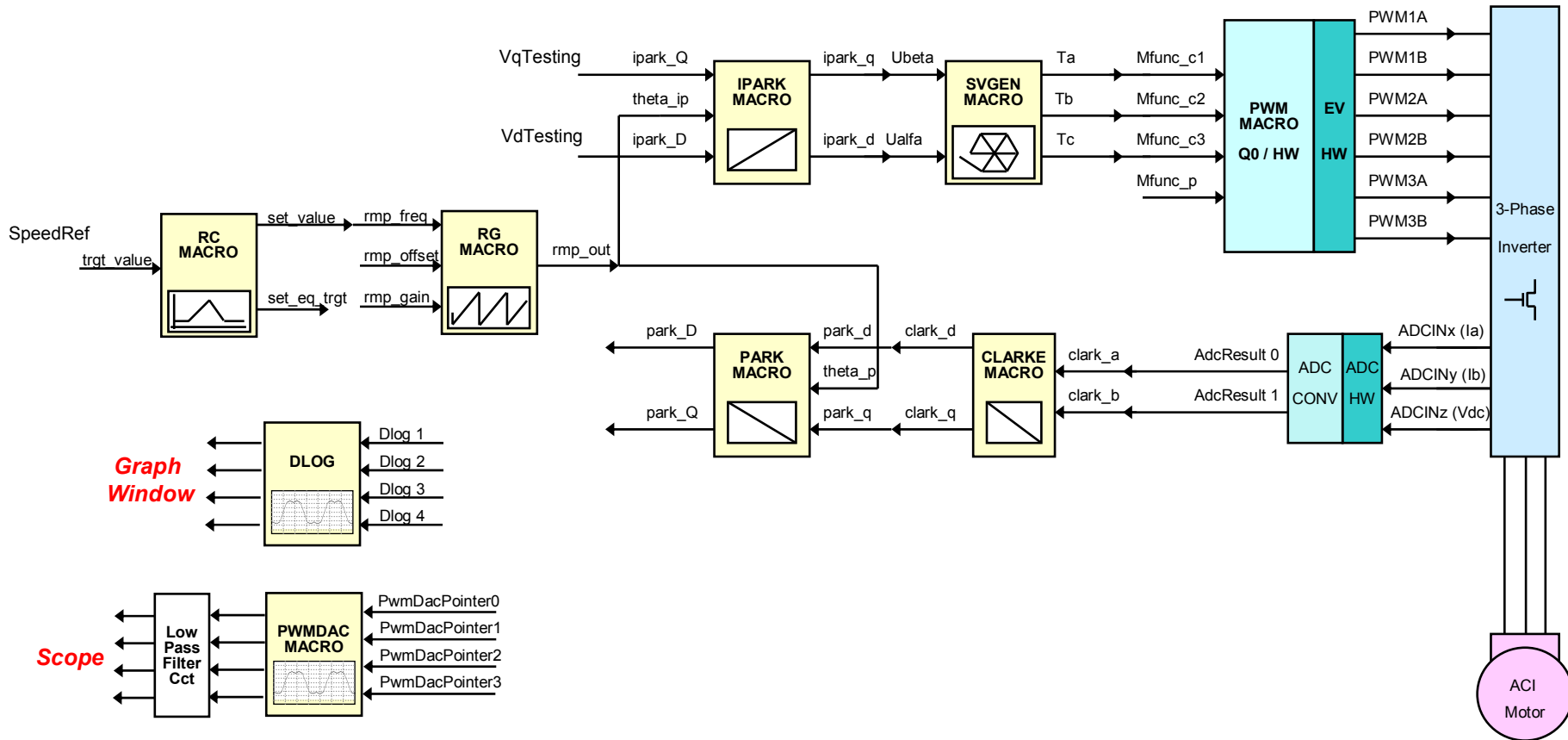
Hint: If the value of clarke1.As is greater than zero, then increase the offset term by the half of the clarke1.As value in the watch window. If the value of clarke1.As is less than zero, then decrease the offset term by the half of the clarke1.As in the watch window.

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.

Rebuild the project and then repeat the calibration procedure again until the clarke1.As and clarke1.Bs offset values are minimum.

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 PID_REG3 modules and speed measurement modules. To confirm the operation of current regulation, the gains of these two PID controllers are necessarily tuned for proper operation.




Open HVAC1_Sensored-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 be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

- SpeedRef (Q24): for changing the rotor speed in per-unit.
- IdRef(Q24): for changing the d-qxis voltage in per-unit. 
- IqRef(Q24): 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_REG3 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 a certain value to generate rated flux. 
- Gradually increase voltage at variac / dc power supply to get an appropriate DC-bus voltage.
- 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_REG3 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_REG3 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 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:

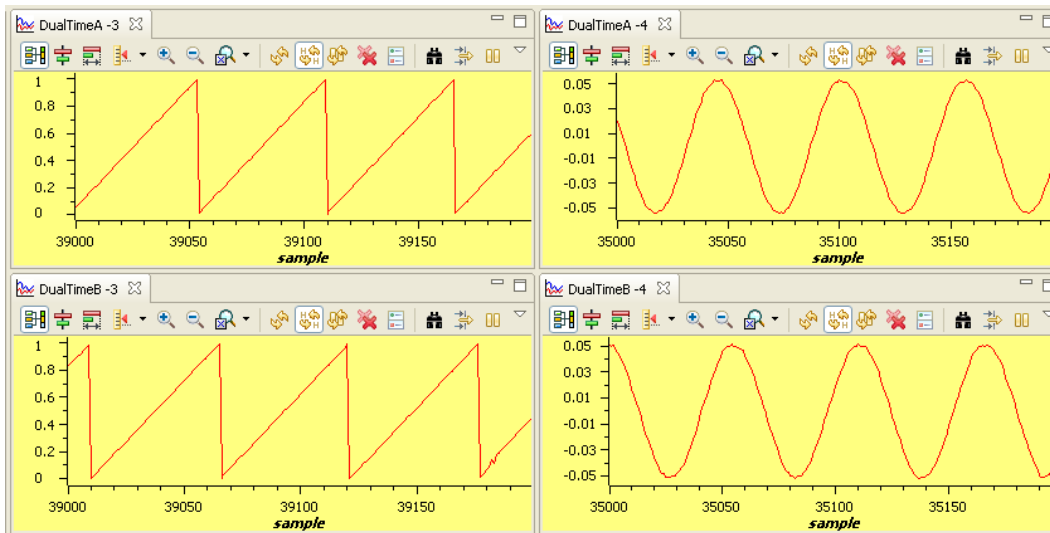


Fig.18 Measured theta, rg1.Out, Phase A & B current waveforms.

Level 3B – QEP / SPEED_FR test (for incremental encoder)

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 [Main]-J10 and QEP/SPEED_FR macros are initialized properly in the HVACI_Sensored.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 0.3 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. 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 less than SpeedRef a little bit due to a slip of motor.
- To confirm these modules, try different values of SpeedRef to test the Speed.
- Use oscilloscope to view the electrical angle output, ElecTheta, from QEP_MACRO module and the emulated rotor angle, Out, from RG_MACRO at PWMDAC outputs with external low-pass filters.
- Check that both ElecTheta and 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 (TB3 for HVDMC kit).
- 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 reducing the bus voltage, taking the controller out of realtime mode and reset.

Level 3C – CAP / SPEED_PR test (for tacho or sprocket)

In this case, the CAP1 input is chosen to detect the edge. If available, make sure that the sensor output is connected to [Main]-J15 (first pin) and CAP/SPEED_PR macros are initialized properly in the HVAC1_Sensored.c file depending on the features of the speed sensor. Typically the capture is used to measure speed when a simple low cost speed sensing system is available. The sensor generates pulses when detecting the teeth of a sprocket or gear and the capture drive provides the instantaneous value of the selected time base (GP Timer) captured on the occurrence of an event. 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 0.3 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. Now the motor is running reference speed.
- Check the “speed2.Speed” in the watch windows with continuous refresh feature whether or not they should be less than SpeedRef a little bit due to a slip of motor.
- To confirm these modules, try different values of SpeedRef to test the Speed.
- Reduce voltage at variac / dc power supply to zero, halt program and stop real time mode. Now the motor is stopping.

An alternative to verify these two software modules without running the motor can be done by using a function generator. The key steps can be explained as follows:

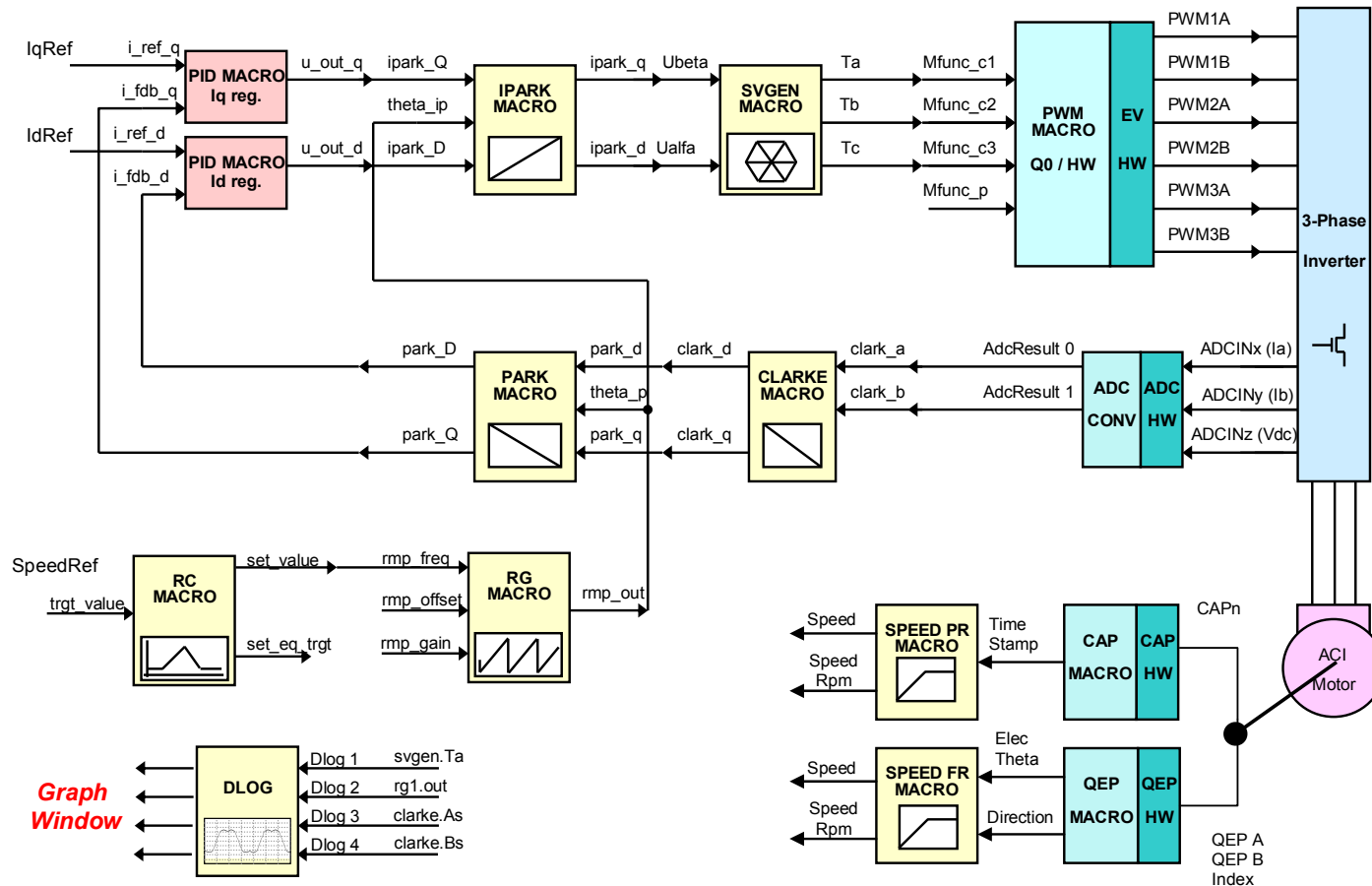
- Use a function generator to generate the 3.3V (DC) square-wave with the desired frequency corresponding to the number of teeth in sprocket and the wanted speed in rpm. Then, connect only the pulse signal and ground wires from the function generator to HVDMC board. The desired frequency of the square-wave produced by function generator can be formulated as:

$$f_{square_wave} = \frac{RPM \times TEETH}{60} \text{ Hz}$$

where RPM is the wanted speed in rpm, and TEETH is the number of teeth in sprocket.

- Compile/load/run program with real time mode and then increase voltage at variac to get the appropriate DC-bus voltage. Now the motor is running. Note that the SpeedRef could be set to any number.
- Check the speed2.Speed and speed2.SpeedRpm in the watch windows with continuous refresh feature whether or not they should be corresponding to the wanted speed that is chosen before.
- To confirm these modules, change different frequencies of square-wave produced by function generator with corresponding wanted (known) speed to check the Speed and SpeedRpm.

Level 3 - Incremental System Build Block Diagram



Level 3 verifies the dq-axis current regulation performed by pid_id, pid_iq and speed measurement modules

Level 4 Incremental Build

Assuming the previous section is completed successfully; this section verifies the current model (CUR_MOD).

Open HVACI_Sensored-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 be incrementally increased as seen in watch windows to confirm the interrupt working properly.

- SpeedRef (Q24): for changing the rotor speed in per-unit.
- IdRef (Q24): for changing the d-qxis voltage in per-unit.
- IqRef (Q24): for changing the q-axis voltage in per-unit.

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. Now the motor is running close to reference speed.
- Compare Curmod1.Theta with rg1.Out via PWMDAC or CCS graph window. They should be identical with a small phase shift.
- When the measured speed is correct and current regulations are performed well, the Theta should give the ramp waveform with the same frequency as one from RG module.
- To confirm this current model, 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. Now terminate the debug session.

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

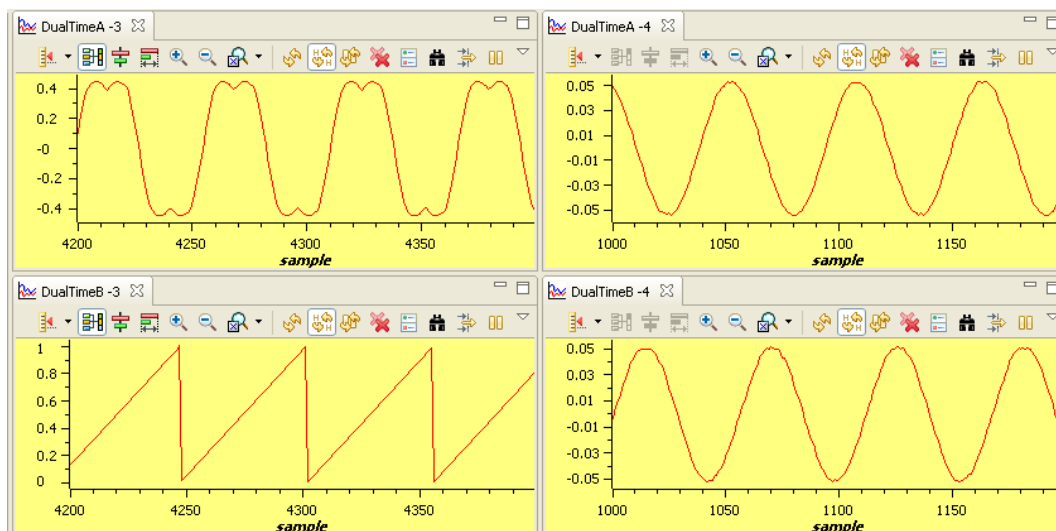
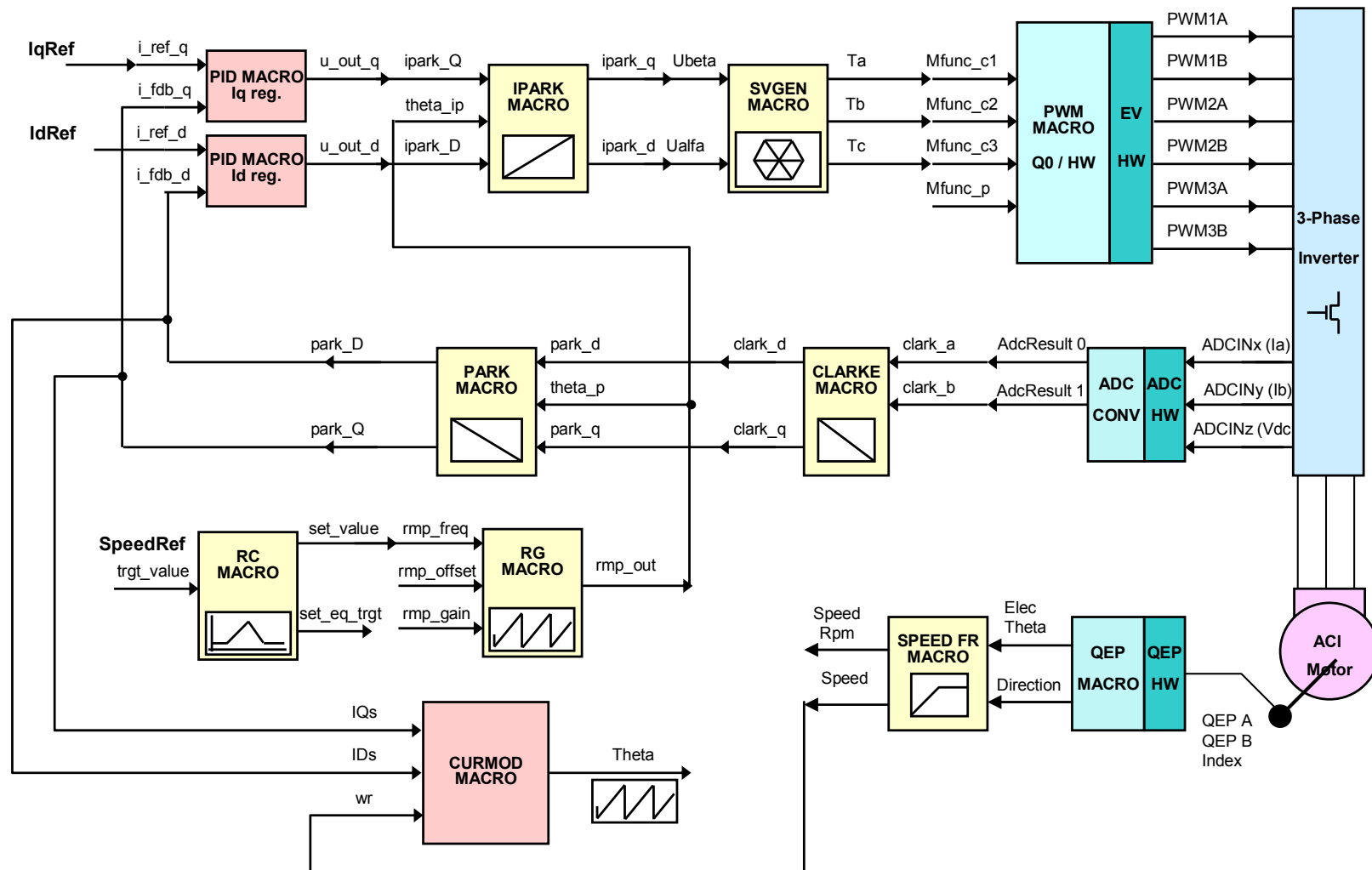


Fig.18 Svgen_dq1.Ta,Curmod theta, and Phase A&B current waveforms

Level 4 - Incremental System Build Block Diagram



Level 4 verifies the current module .

Level 5 Incremental Build

Assuming the previous section is completed successfully, this section verifies the speed regulator performed by PID_REG3 module. The system speed loop is closed by using the measured speed as a feedback.

Open HVACI_Sensored-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 be incrementally increased as seen in watch windows to confirm the interrupt working properly.

- SpeedRef (Q24): for changing the rotor speed in per-unit.
- IdRef (Q24): for changing the d-qxis voltage in per-unit.

The speed loop is closed by using measured speed. It should be emphasized that the motor can spin only one direction when the measured speed (from capture driver) does not give information about the direction like QEP based speed measurement. Therefore, if the speed sensor is not an incremental encoder, the SpeedRef is required to be positive. 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).
- Gradually increase voltage at variac / dc power supply to get an appropriate DC-bus voltage and now the motor is running around the reference speed (0.3 pu).
- 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 (positive only for tachometer).
- For speed PID controller, the proportional, integral, derivative and integral correction gains may be retuned to have the satisfied responses.
- At very low speed range, the performance of speed response relies heavily on the good rotor fluxangle computed by current model.
- 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 that the IdRef is set to be constant at a certain value that is not too much for driving the motor. Practically, it may be calculated from the rated flux condition.

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

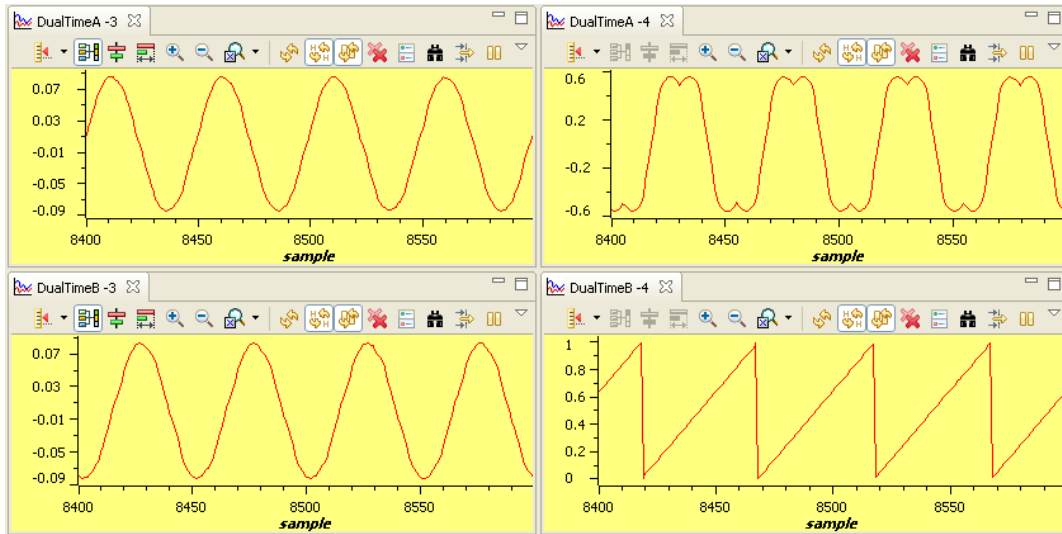


Fig.19 Phase A&B currents, $Sygen_dq1.Ta$, and Curmod theta waveforms under 0.5 pu load, 0.3 pu speed

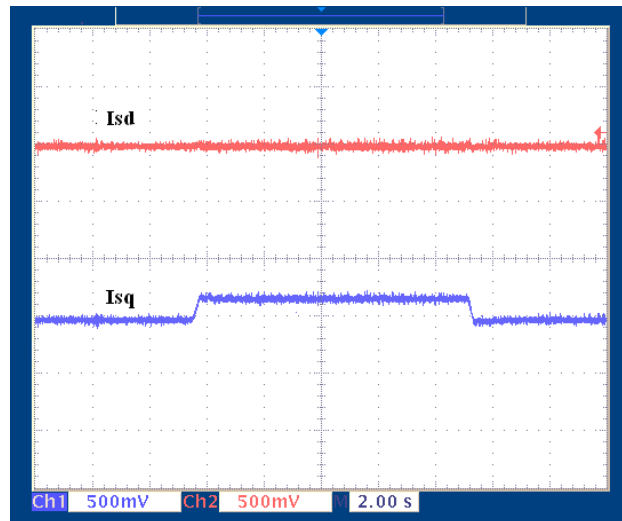
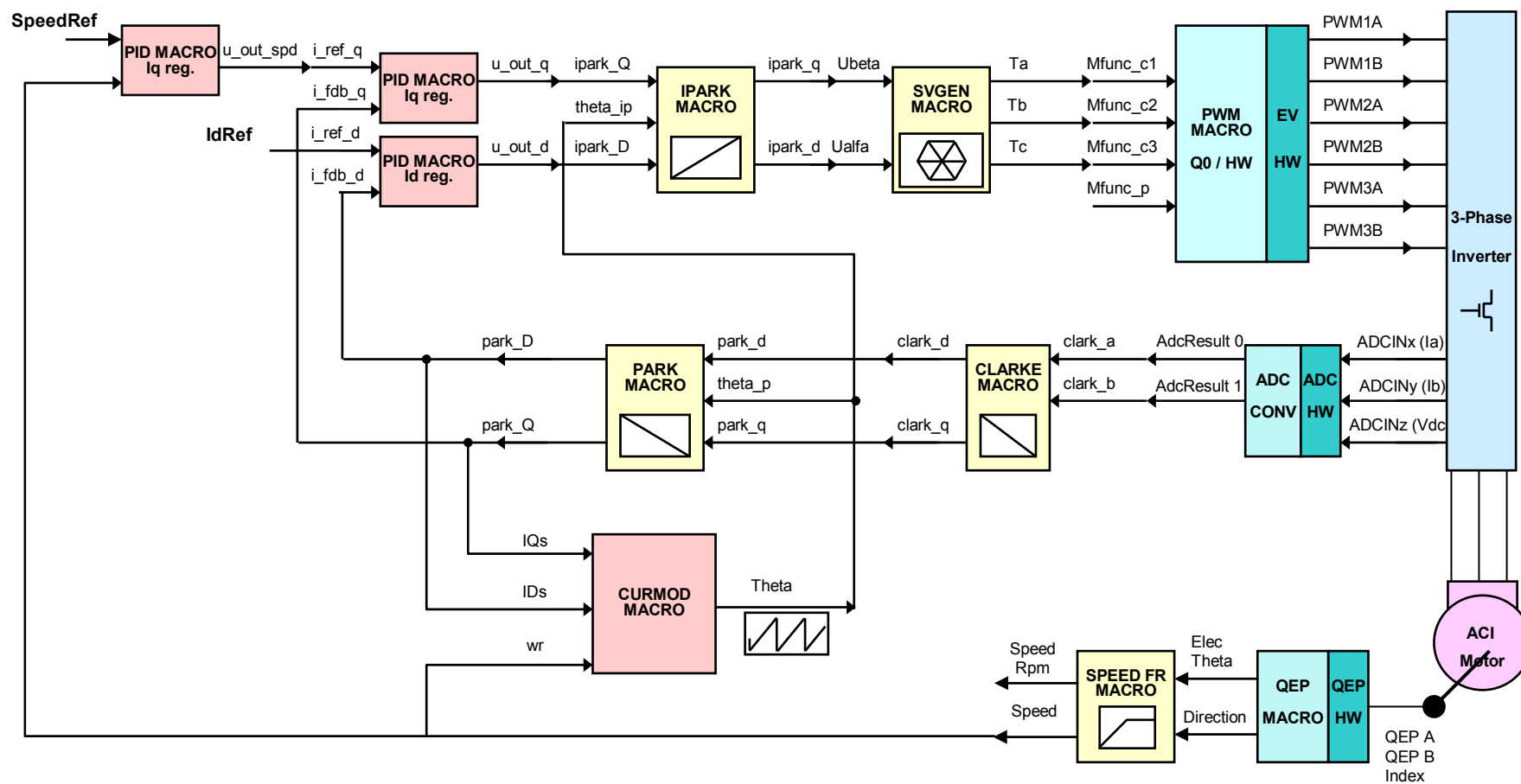


Fig21 Flux and torque components of the stator current in the synchronous reference frame under 1.0 pu step- load and 0.3 pu speed monitored from PWMDAC outputs

Level 5 - Incremental System Build Block Diagram



Level 5 verifies the speed loop .

