TI Spins Motors

Sensored FOC Control with SMO for the Hercules RM48x

Version 1.0.2 *Motor Solutions*

Abstract

This application note presents a solution for sensored FOC with SMO control of Brushless permanent magnet motors using a Hercules RM48x microcontroller. The RM48 family of MCUs are designed for safety critical applications in the transportation, medical and industrial sectors. They meet safety standards such as IEC61508/SIL3 and are well suited for high performance real-time, computationally intensive and cost constrained control applications.

A complete 3 phase motor control solution is presented below: control structures, power hardware topology, control hardware and remarks on performance efficiency can be found in this document.

This application note covers the following:

- Hardware and software setup of the sensored FOC control with SMO project.
- Step by step instructions to get a brushless PM motor running in different control and commutation modes.

Table of Contents

Introduction

The economic constraints and new standards legislated by governments place increasingly stringent requirements on electrical systems. New generations of equipment must have higher performance parameters such as better efficiency and reduced electromagnetic interference. System flexibility must be high to facilitate market modifications and to reduce development time. All these improvements must be achieved while, at the same stime, decreasing system cost.

Brushless motor technology makes it possible to achieve these specifications. Such motors combine high reliability with high efficiency, and for a lower cost in comparison with brush motors. This application note describes the use of a Brushless Permanent Magnet motor. The control algorithm presented here in, allows for high performance speed and torque control of motors with a sinusoidal Back EMF waveform shape. Permanent magnet synchronous machines with sinusoidal Back-EMF and sinusoidal stator currents are widely used due their smooth torque output, low inertia and high energy efficiency characteristics.

Directory Overview

Below is the MotorWare directory structure for the main files in the project:

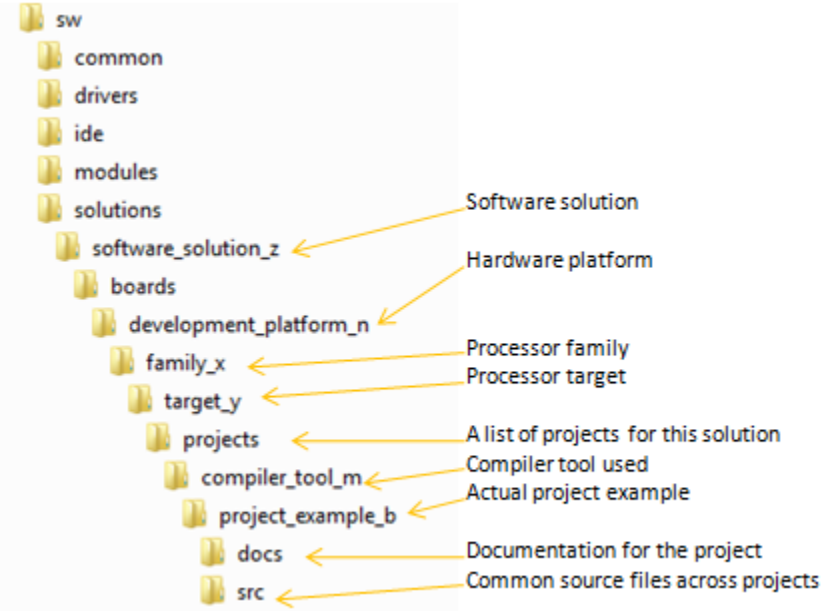

Fig 1 MotorWare Directory Structure

Below is the MotorWare directory structure for the driver files in the project:

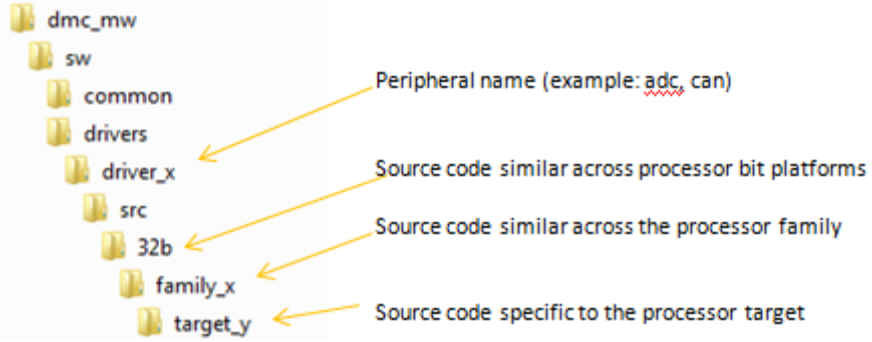

Fig 2 Driver Files Directory Structure

Below is the MotorWare directory structure for the module files in the project:

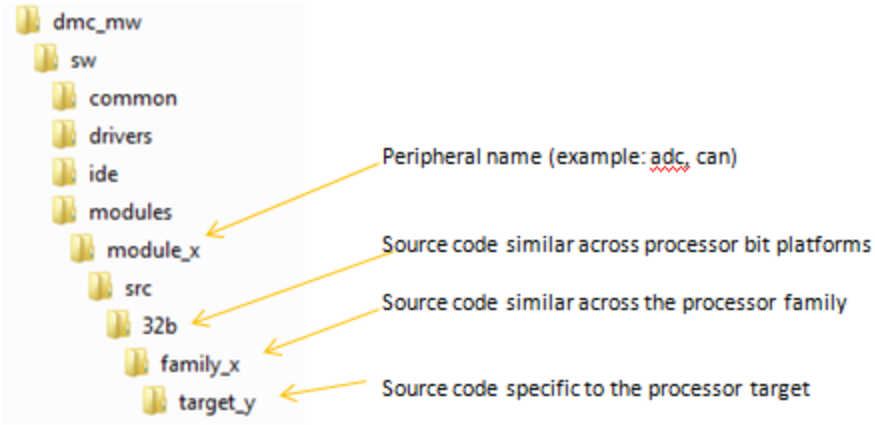

Fig 3 Module Files Directory Structure

System Overview

This document describes the “C” real-time control framework used to demonstrate Field Oriented Control (FOC) of brushless permanent magnet motors. The “C” framework is designed to run on the Hercules RM48 based controllers in Code Composer Studio.

The framework uses the following modules[1]:

Macro Names	Explanation
RAMPGEN	Ramp Generator
QEP	Encoder Position Measurement (sensor based electrical angle)
SPEED_FR	Speed Measurement (based on encoder electrical angle)
CLARKE	Clarke Transform
PARK	Park Transform
RAMP_CNTL	Ramp Controller (speed command slew rate limiter)
PID	PID Controller
IPARK	Inverse Park Transform
SVGEN	Space Vector Generator
VOLT_CALC	Voltage Estimator (phase voltage applied to motor)
SMOPOS	Sliding Mode Observer Position Estimator (for sensorless electrical angle)
SPEED_EST	Speed Measurement (based on SMO electrical angle)
DLOG	Data Logger

[1] Please refer to pdf documents in motor control folder explaining the details and theoretical background of each module

In this system, sensored and sensorless Field Oriented Control (FOC) of brushless permanent magnet motors will be experimented with and the performance of the speed controller will examined. The motor is driven by using the DRV8301-EVM board with the DRV8301 gate driven IC and external bridge stage. The Hercules RM48x control card is used to generate three pulse width modulation (PWM) signals for high side gate drive. The motor phase currents and DC bus voltage are measured and sent to the RM48x via analog-to-digital converters (ADCs).

The sensored_speed_smo project has the following properties:

C Framework		
System Name	**Flash Memory Usage RM48x**	**RAM Memory Usage RM48x**
RM48L952_sensored_speed_smo	26212 bytes	5836 bytes

CPU Utilization of Sensored FOC Control with SMO	
Name of Modules *	**Number of Cycles (RTI Counter Clock)**
RAMPGEN	25√
QEP	78
SPEED_FR	69
CLARKE	11
PARK	31
RAMP_CNTL	22
PID (x3)	88 (x3)
IPARK	31
SVGEN	48
VOLT_CALC	34
SMOPOS	105
SPEED_EST	66
DLOG	27
Total Number of Cycles (RTI Counter Clock @ 40 MHz)	786
Total Number of Cycles CPU	**3144 ** **
CPU Utilization @ 160 MHz	**39.3%**

* The modules are defined in the header files as inline functions.
** At **20 kHz** ISR freq.
√ Not included in the speed loop.

System Features	
Development /Emulation	Code Composer Studio v5.2.1 (or above) with Real Time debugging
Target Controller	Hercules RM48x
PWM Frequency	20kHz PWM (Default)
PWM Mode	PWM control of high side gates only (i.e. three independent signals). Low side is complementary to high side gates with programmable dead-band.
Interrupts	adc1Group1Interrupt() adc1Group2Interrupt()
Peripherals Used	NHET – PWM Generation, QEP readin, SPI config of gate driver GIO – Fault diagnostics, Gate driver enable/disable RTI – Time base for delays and profiling ADC – Motor phase currents and DC bus voltage

The overall system implementation of brushless 3-ph permanent magnet motor control using sensored FOC with SMO is depicted in Fig.4.

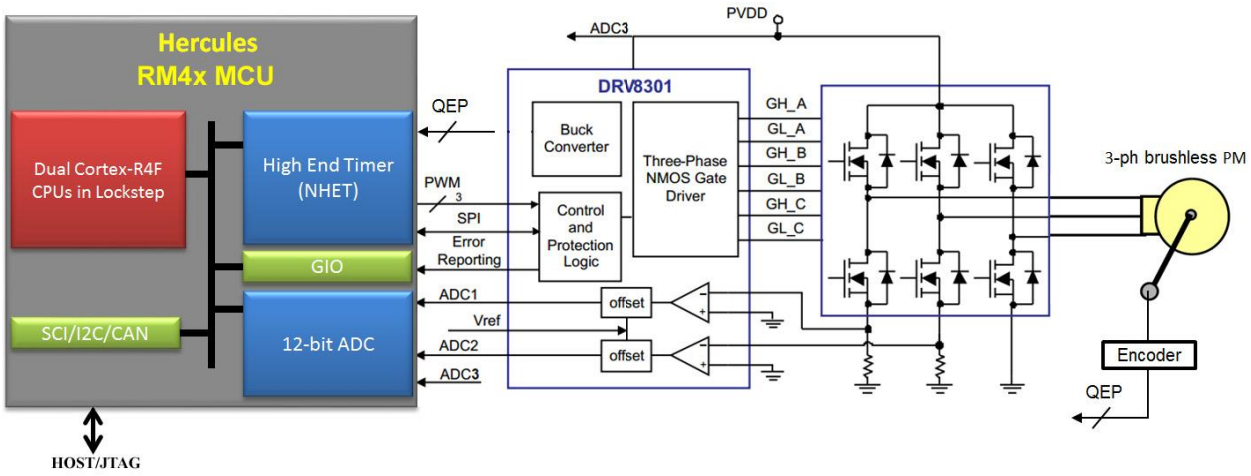

Fig 4 A 3-ph Brushless permanent magnet drive implementation

The software flow is described in the Figure 5 below.

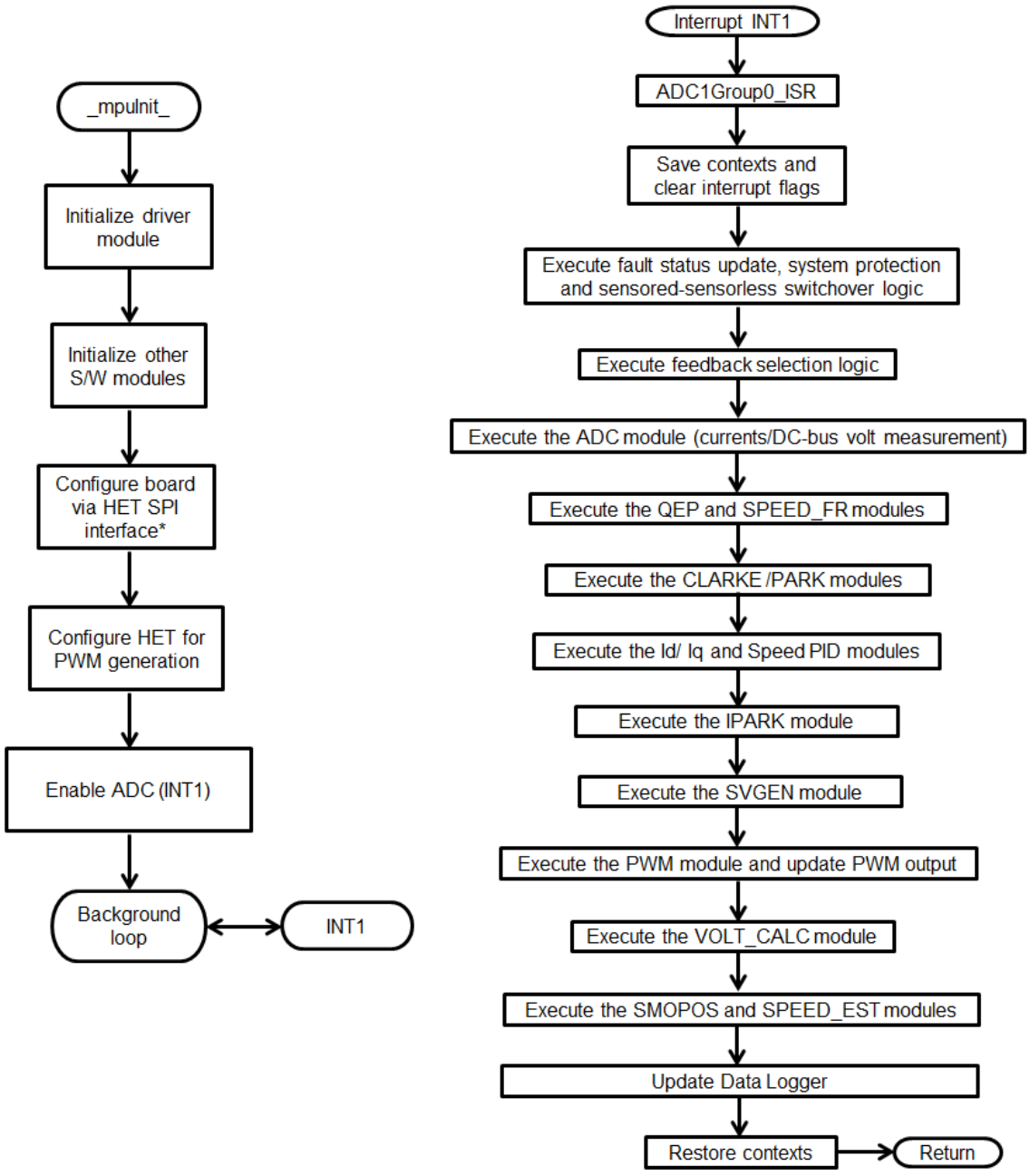

Fig 5 Software Flow Chart

*The DRV8301 gate driver IC settings such as current amplifier gains, PWM mode and diagnostic reporting are configured via SPI interface. During the driver module initialization, the High Efficiency Timer (HET) module is initialized. The RTI timer module is used to count down a 50 mS delay for HET initialization. At this point the gate driver enable signal (EN_GATE) of the DRV8301 IC is asserted. Once again, the RTI timer module is used to count down a 50 mS delay for the gate driver power up (i.e. charge pumps, internal regulator, current amplifiers). After initialization of the remaining driver and software modules, the HET perepheral is used to issue SPI commands to configure the DRV8301 IC.

Hardware Configuration (DRV8301-EVM)

Please refer to the DRV8301-EVM How to Run Guide and HW Reference Guide found at:

sw\boards\drv8301kit_revD\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.
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. Make sure the DRV8301 control card +5V source jumper (JP2) is installed. Refer to the DRV8301 Setup Guide.
4. **RM48 RevC control Card:** Connect an XDS100v2 emulator to the JTAG port J21 on the DRV8301 board. Install jumpers (J101) on the revC control card to route JTAG signals to the DIMM connector. Refer to the revC control card schematic. If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will automatically be installed.
5. Connect the encoder feedback cable to connector J4.
6. The motor phases R, S, T should be connected to the terminals OUTA, OUTB and OUTC respectively. The shield (drain) wire should get connected to the ground terminal. For more details on motor wiring please refer to the datasheet provided with your motor.
7. Connect a 24V power supply to J25(+ terminal) and J26(- terminal) of the DRV8301-EVM (refer to Figure 6). Now LED1 and LED3 should turn on. Notice the control card LEDs would light up as well indicating the control card is receiving power from the board.

   The included power supplies are only capable of 2.5A maximum and have been shown to drop voltage and may not provide enough current during rapid speed or load changes, especially with motors that can draw more than 2.5A When doing full performance evaluation please use a power supply that more than meets the demands of the motor and application use.

TI Spins Motors

WARNING

This EVM is meant to be operated in a lab environment only and is not considered by TI to be a finished end-product fit for general consumer use

This EVM must be used only by qualified engineers and technicians familiar with risks associated with handling high voltage electrical and mechanical components, systems and subsystems.

This equipment operates at voltages and currents that can result in electrical shock, fire hazard and/or personal injury if not properly handled or applied. Equipment must be used with necessary caution and appropriate safeguards employed to avoid personal injury or property damage.

It is the user's responsibility to confirm that the voltages and isolation requirements are identified and understood, prior to energizing the board and or simulation. When energized, the EVM or components connected to the EVM should not be touched.

TI Spins Motors

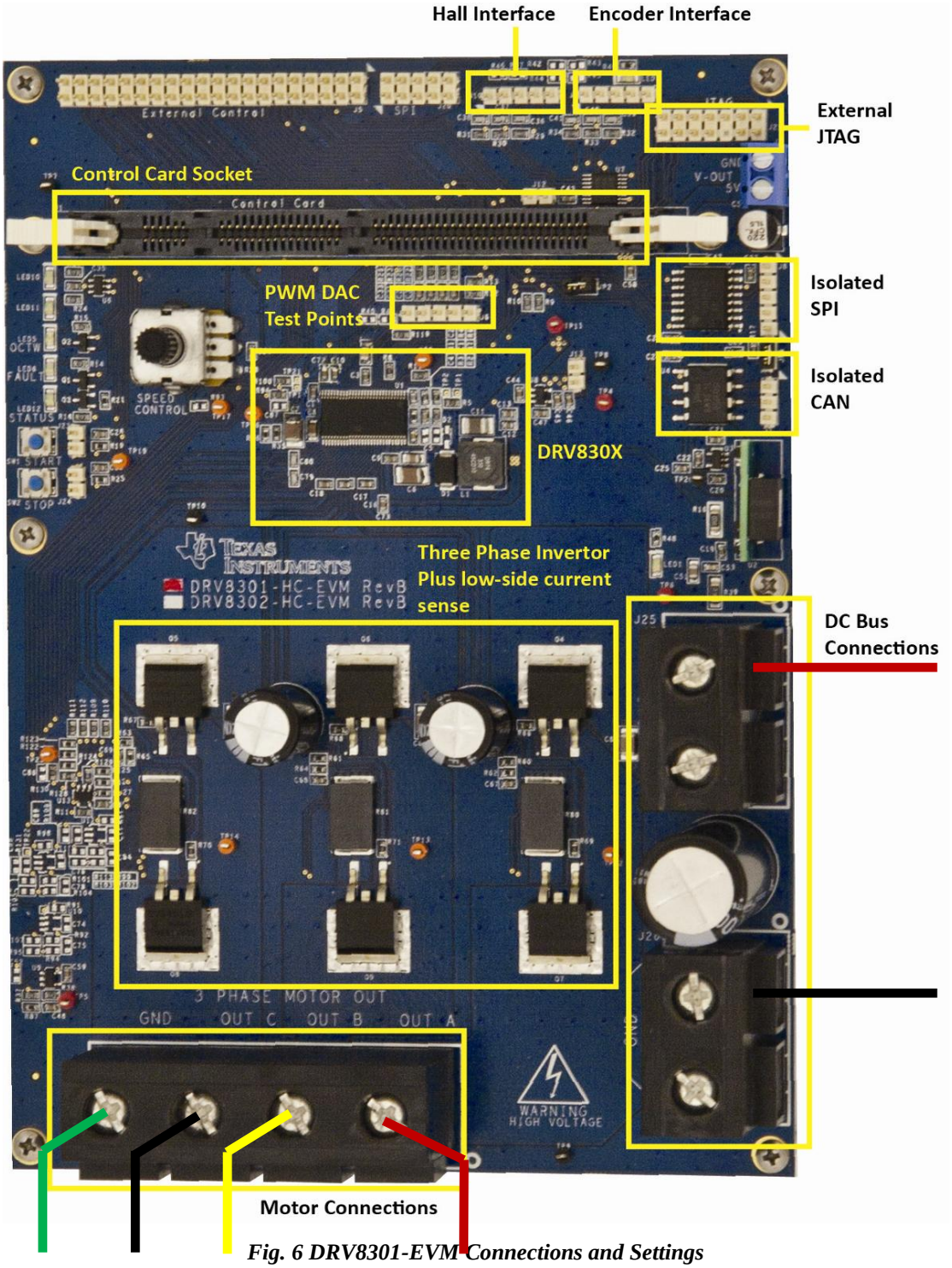

Fig. 6 DRV8301-EVM Connections and Settings

Software Setup Instructions to Run RM48L952_sensored_speed_smo Project

For information on getting started with Code Composer Studio v5, use the CCSv5 Getting Started Guide online at http://processors.wiki.ti.com/index.php/CCSv5_Getting_Started_Guide. The Getting Started Guide details the process to download, install, and run Code Composer Studio v5. At the TI wiki downloads page, ensure to select CCSv5 build 5.2.1.00018 or later for installation (refer to Figure 7 below).

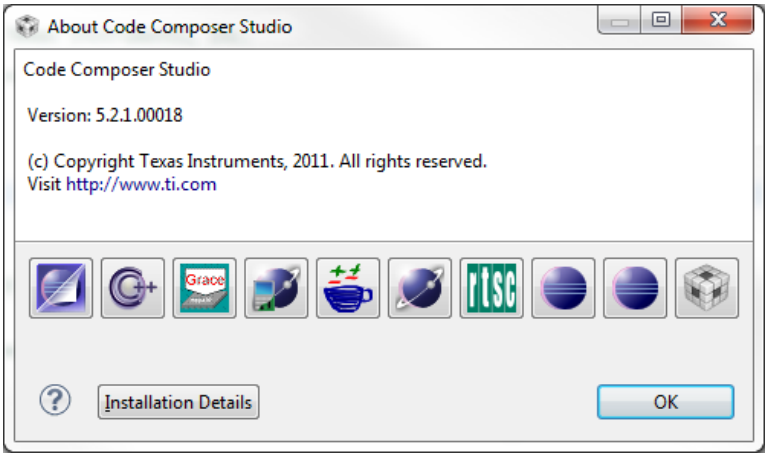

Fig 7 CCSv5 release and build information

When CCSv5 has been downloaded, installed, and is running, the sensored_speed_smo project can be imported into the desired workspace. In CCS, go to Project->Import Existing CCS/CCE Eclipse Project. Browse to the project path below: **sw\solutions\sensored_speed_smo\boards\drv8301kit_revD\hercules\rm48l952\projects\ccs5\project01** in your MotorWare directory. The sensored_speed_smo project should be discovered. Make sure the checkbox next to the project title is selected (refer to Figure 8 below). Click on Finish to import the project into your workspace.

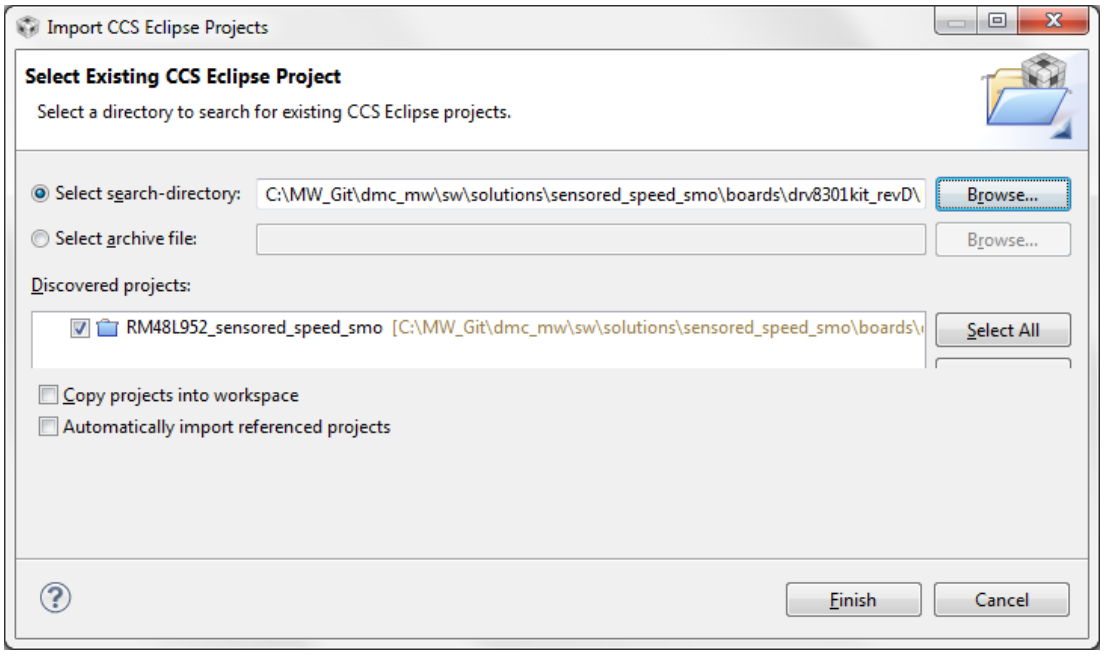

Fig 8 Importing a Project

Right click on the project name and select “Properties”. Select the “General” tab in the left pane of the properties window (refer to Figure 9 below). Confirm that the compiler version selected is 5.0.0 or later. Click “ok” to close the properties window.

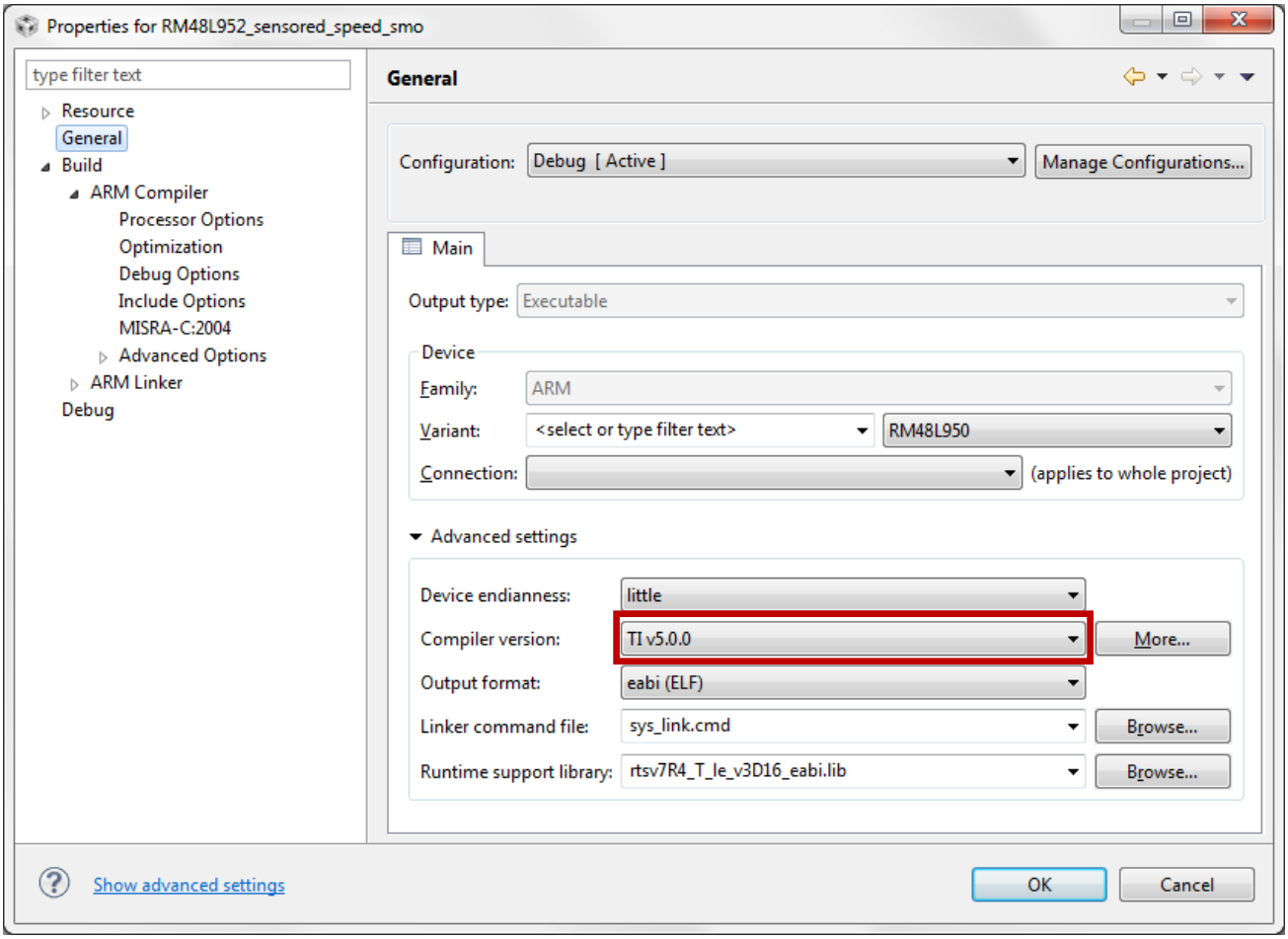

Fig 9 Compiler version selection

In the project explorer window, right click on the project name and select “Clean Project” then “Build Project”. Once the build completes, launch a debug session (Run->Debug) to load the code into the MCU. Make sure you are connected to the Hercules RM48x device (Run->Connect Target) and load the program (Run->Load->Load Program). Press the Run button to start code execution on the target.

While in Debug mode, click on View->Expressions. This should bring up the expressions tab of the CCS Watch Window (if not already open). Right Click inside the expressions window and select “Import”. Navigate to the path: **sw\solutions\sensored_speed_smo\boards\drv8301kit_revD\hercules\rm48l952\projects\ccs5\project01** and choose "RM48_DRV8301_GUI_Watch_Window_Variables.txt". You should now have a watch window that looks like Figure 10. Click on the Continuous Refresh button on the top right corner of the watch window. Note that Figure 10 only shows a partial list of the variables within the gui object (gGUIObj).

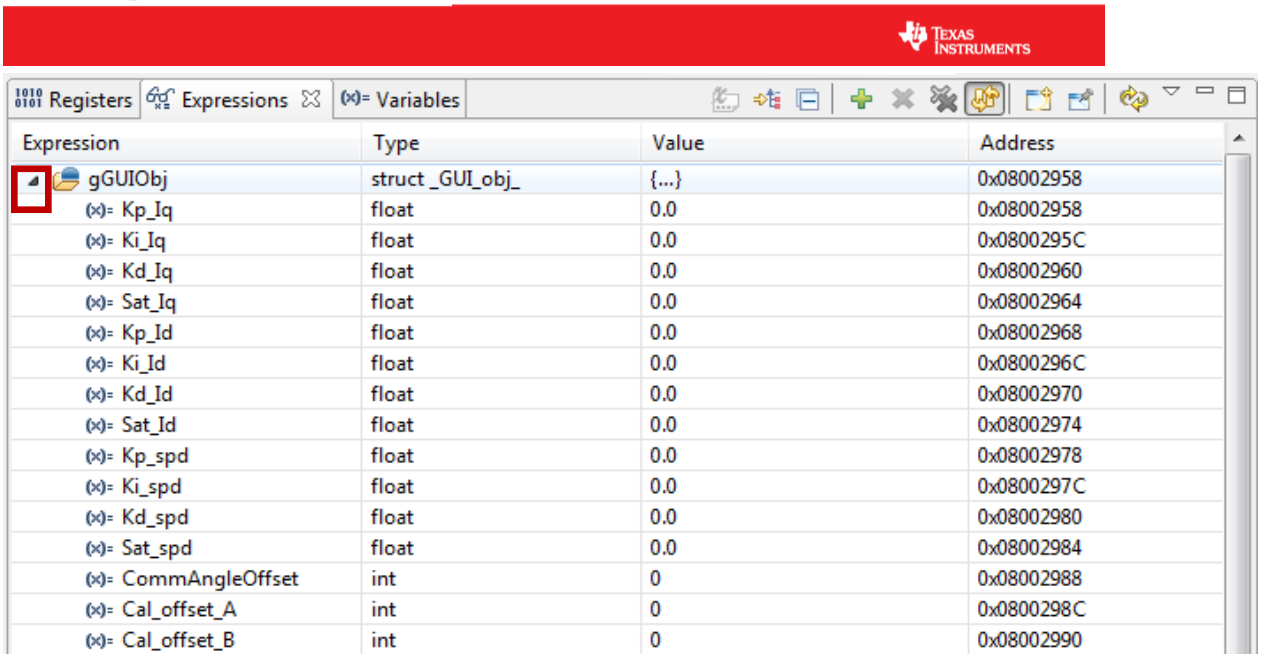

Fig 10 Watch Window Setup

Make sure you are connected to the RM48x device and that run button has been pressed.

Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location: **sw\solutions\sensored_speed_smo\boards\drv8301kit_revD\hercules\rm48l952\projects\ccs5\project01**. Click on Continuous Refresh button on the top right corner of the graph tab to enable periodic capture of data from the microcontroller.

Running/Modifying the Project

System Calibration Sequence

Once the run button  has been pressed after following the steps in the preceding section, the system is ready for user commands. During run time, the motor can be enabled or disabled using the gGUIObj.EnableFlg variable (0 = Disabled, 1 = Enabled). The default state of the variable is 0 (i.e. disabled).

The calibration routine calculates the offset (in encoder ticks) between the encoder index position and motor zero pole pair position. This is a necessary step, in order to align the direct and quadrature current vectors being applied to the motor. In the software, the key variables to be adjusted/ observed for system calibration are summarized below.

- gGUIObj.EnableFlg -> for enabling/ disabling the motor.
- gGUIObj.calibrateEnable-> for enabling/disabling the motor to encoder index calibration routine.
- gGUIObj.CommAngleOffset-> for observing the calculated encoder index angle offset.
- sdrv.commutationMode-> the motor commutation mode.

To run the calibration routine, complete the following:

1. Make sure the program is currently running on the device. If so, also assure that “gGUIObj.EnableFlg” and “gGUIObj.CommAngleOffset” are set to 0.
2. Set “gGUIObj.EnableFlg” to 1.
3. Set “gGUIObj.calibrateEnable” to 1. The motor should start rotating.
4. Once the motor has stopped rotating. Set “gGUIObj.calibrateEnable” to 0. This concludes the calibration routine. gGUIObj.CommAngleOffset should now have a value in the range of160 -180.

The key steps can be explained as follows:

- Setting “gGUIObj.EnableFlg” to 1 starts the main interrupt service routine for FOC control, ADC sampling of motor phase currents and DC bus voltage as well as generation of PWM output. Shown in Figure 11, is the expected Scope output (PWM signals) seen on pins 25, 26 and 28 of connector J5. If no change is observed, check to make sure continuous refresh is pressed.
- Upon setting “gGUIObj.calibrateEnable” to 1, the motor starts ramp commutating to determine the index position of the encoder. When the index is discovered, the rotor is locked in position to read in the offset between the encoder index and zero pole pair position. At this point “drv.commutationMode” should be equal to “LOCK_ROTOR_MODE”.
- The offset observed is stored into the controller when “gGUIObj.calibrateEnable” is set to 0. The updated value is reflected in “gGUIObj.CommAngleOffset”. The system is now ready for motor control commands.

The calibration routine needs to be executed only once during motor bring up. If the commutation angle offset is known, it can be used to directly update “drv.calibrateAngle”. In this case calibration is no longer necessary.

Fig 10 PWM output on pins 25,26 and 28 of connector J5

Motor Control Modes

To change the mode that the motor is running, you will have to modify the "gGUIObj.CtrlType" variable in the Watch Window. The default value is 0, which represents disabled mode. The possible values of CtrlType are described below:

Value	Mode
0	Disabled
1	Closed loop Torque Control
2	Closed loop Speed Control

It is important that the motor be in disabled mode first ("gGUIObj.CtrlType" equal to 0) before switching to any other control mode. When the control mode changes from disabled to torque control or speed control mode, the motor ramp commutates to find the encoder index position. After this index find routine, the system accepts torque or speed commands from the watch window.

Torque Control Mode

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

- gGUIObj.CtrlType: for changing motor control mode.
- gGUIObj.IqCmd: for changing the torque output of the motor.
- drv.commutationMode-> the motor commutation mode.
- gGUIObj.CommAngleOffset-> the calculated/ applied encoder index angle offset.
- gGUIObj.GraphInput-> to modify graphing variables fed to data logger module.

To run in torque control mode, complete the following:

1. Make sure the program is currently running on the device. If so, also assure that “gGUIObj.EnableFlg” is set to 0.
2. Verify that “gGUIObj.CtrlType” is set to 0, for disabled Mode.
3. Set “gGUIObj.EnableFlg” to 1.
4. Check to make sure “gGUIObj.CommAngleOffset” is set to the correct value. If going through startup, complete the system calibration routine described above. Alternatively, “drv.calibrateAngle” can be set to a known value manually.
5. Verify that “gGUIObj.IqCmd” is set to 0.0, for zero output torque request and that “drv.commutationMode” is set to “ENCODER_COMMUTATION_MODE”.
6. Set “gGUIObj.CtrlType” to 1 for torque control mode.
7. Set “gGUIObj.IqCmd” (quadrature current vector) to 0.06. At this point the motor should start spinning. Adjust “gGUIObj.IqCmd” for desired motor toque output. Note that *“gGUIObj.IdCmd”* (i.e. the direct current vector) should always be set to 0.0.

The key steps can be explained as follows:

- After startup, when the control mode is changed from disabled to torque control, the motor ramp commutates to find the encoder index. The duration of ramp commutation depends on the position of the rotor.
- At this point, “gGUIObj.IqCmd” can be used to specify the reference current command for the quadrature component (i.e. torque component) PI controller. Note that the speed is not controlled in this step and a non-zero torque reference will keep increasing the motor speed. Therefore, the motor should be loaded using a brake/generator (or manually if the motor is small enough) after closing the loop.
- Set “gGUIObj.GraphInput” to 2 to enable graphing of current control mode variables. Verify the DC bus voltage (real world value, unit Volt) and PWM output (per unit) waveforms. Refer to Figure 11.
- Bring the system to a safe stop by setting “gGUIObj.IqCmd” and “gGUIObj.CtrlType” to 0. Take the controller out of Continuous Refresh mode  and close all graph windows before attempting to reset.
- If a fault condition is encountered, “gGUIObj.CtrlType” and “gGUIObj.IqCmd” are automatically reset to 0. The gate driver fault indication LED6 on the DRV8301 will be asserted. The variable “gGUIObj.RstFault” can be used to reset the fault (by setting to a value of 1). The shipped motor has a maximum operating voltage range greater than 50VDC and a back EMF rating of 4.6 V/krpm. When using a 24 Volt brick power supply, the maximum speed of the motor is physically limited. Operating the motor unloaded in torque control mode may therefore cause a fault condition to occur.

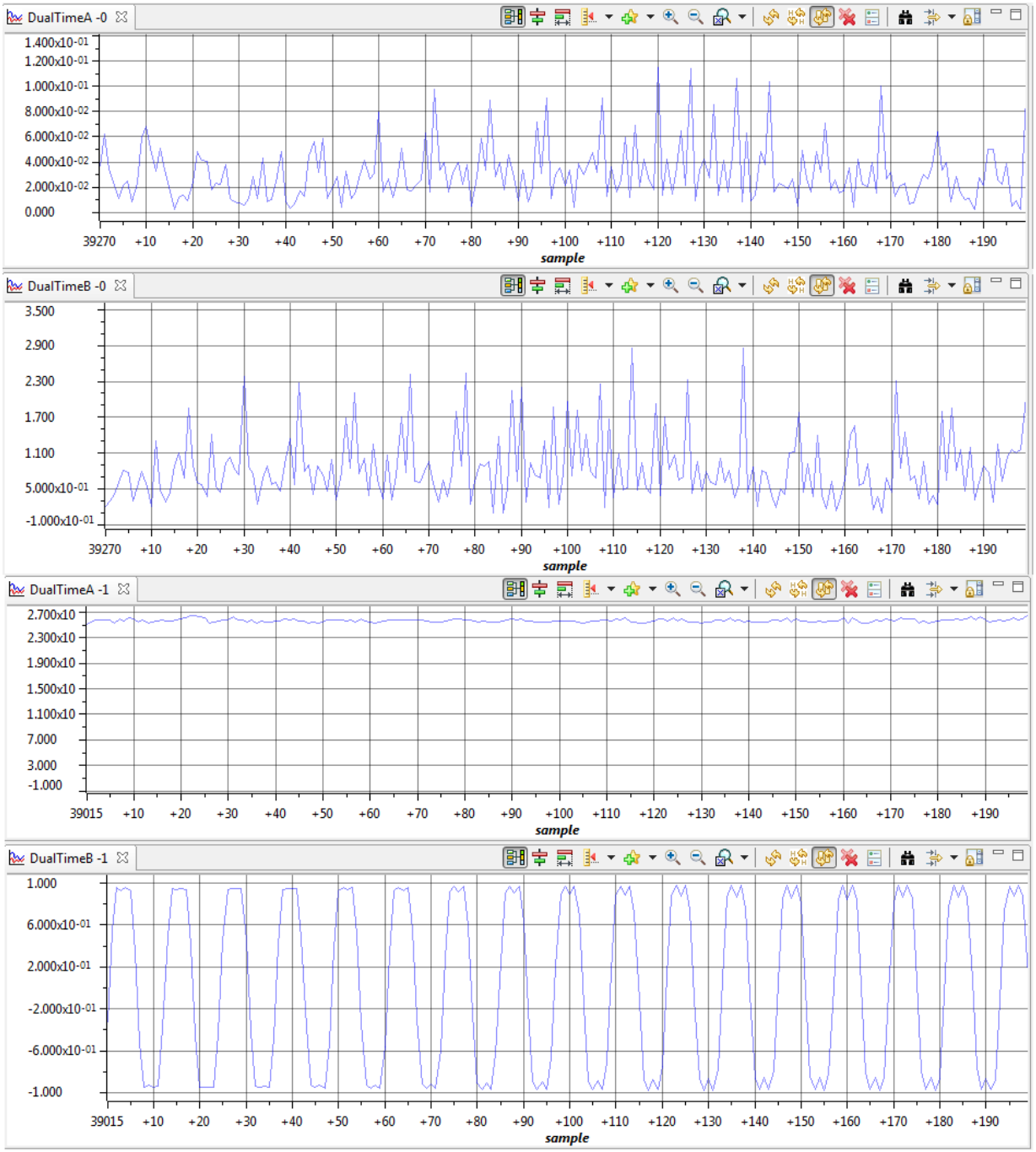

Fig 11 Graph Windows for Torque Control Mode (a) Motor torque output, (b) Total motor Current, (c) DC bus measurement (Volt) and (d) PWM1 output (pu)

Speed Control Mode

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

- gGUIObj.CtrlType: for changing motor control mode.
- gGUIObj.SpdCmd: for changing the speed output of the motor.
- drv.commutationMode-> the motor commutation mode.

- gGUIObj.CommAngleOffset-> the calculated/ applied encoder index angle offset.
- gGUIObj.GraphInput-> to modify graphing variables fed to data logger module.
- drv.currentSpd-> per unit measured speed of motor.
- gGUIObj.SwitchOverSpdFwd -> increasing speed encoder/smo switchover threshold
- gGUIObj.SwitchOverSpdRev -> decreasing speed encoder/smo switchover threshold
- gGUIObj.SpeedEncoder-> encoder based speed measurement in RPM
- gGUIObj.SMO-> sliding mode observer based speed measurement in RPM
- gGUIObj.FaultEncoder-> encoder fault status

To run in speed control mode, complete the following:

1. Make sure the program is currently running on the device. If so, also assure that "gGUIObj.EnableFlg" is set to 0.
2. Verify that "gGUIObj.CtrlType" is set to 0, for disabled Mode.
3. Set "gGUIObj.EnableFlg" to 1.
4. Check to make sure "gGUIObj.CommAngleOffset" is set to the correct value. If going through startup, complete the system calibration routine described above. Alternatively, "drv.calibrateAngle" can be set to a known value manually.
5. Verify that "gGUIObj.SpdCmd" is set to 0.0, for zero output speed request and that "drv.commutationMode" is set to "ENCODER_COMMUTATION_MODE".
6. Set "gGUIObj.CtrlType" to 2 for speed control mode.
7. Set "gGUIObj.SpdCmd" to 0.5. At this point the motor should start spinning. Adjust "gGUIObj.SpdCmd" for desired motor speed output. Verify that "drv.commutationMode" is set to "SENSORLESS_COMMUTATION_MODE".

The key steps can be explained as follows:

- After startup, when the control mode is changed from disabled to speed control, the motor ramp commutates to find the encoder index. The duration of ramp commutation depends on the position of the rotor.
- At this point, "gGUIObj.SpdCmd" can be used to specify the reference speed command for the speed PI controller. If the measured speed (pu) of the motor "drv.currentSpd" is greater than the forward speed threshold "gGUIObj.SwitchOverSpdFwd", the commutation mode will automatically be switched to sensorless commutation mode. If "drv.currentSpd" is less than the reverse speed threshold "gGUIObj.SwitchOverSpdRev" the commutation mode is changed back to encoder commutation mode.
- Set "gGUIObj.GraphInput" to 1 to enable graphing of speed control mode variables. Verify the motor speed feedback (pu) and SMO based electrical angle (pu) waveforms (Refer to Figure 11).
- While "drv.commutationMode" is set to "SENSORLESS_COMMUTATION_MODE", the encoder feedback cable can be unplugged from connector J4. The variable "gGUIObj.FaultEncoder" will register an encoder fault. The motor will continue to operate until the speed measurement falls below the reverse direction speed threshold ("gGUIObj.SwitchOverSpdRev"). When this happens the motor is disabled. If "gGUIObj.CtrlType" is now set to speed control mode, the encoder index pulse will not be detected. Once again the motor is put back in disabled mode. Normal speed control operation resumes when the encoder is plugged back in.

- Bring the system to a safe stop by setting “gGUIObj.SpdCmd” and “gGUIObj.CtrlType” to 0. Take the controller out of Continuous Refresh mode and close all graph windows before attempting to reset.

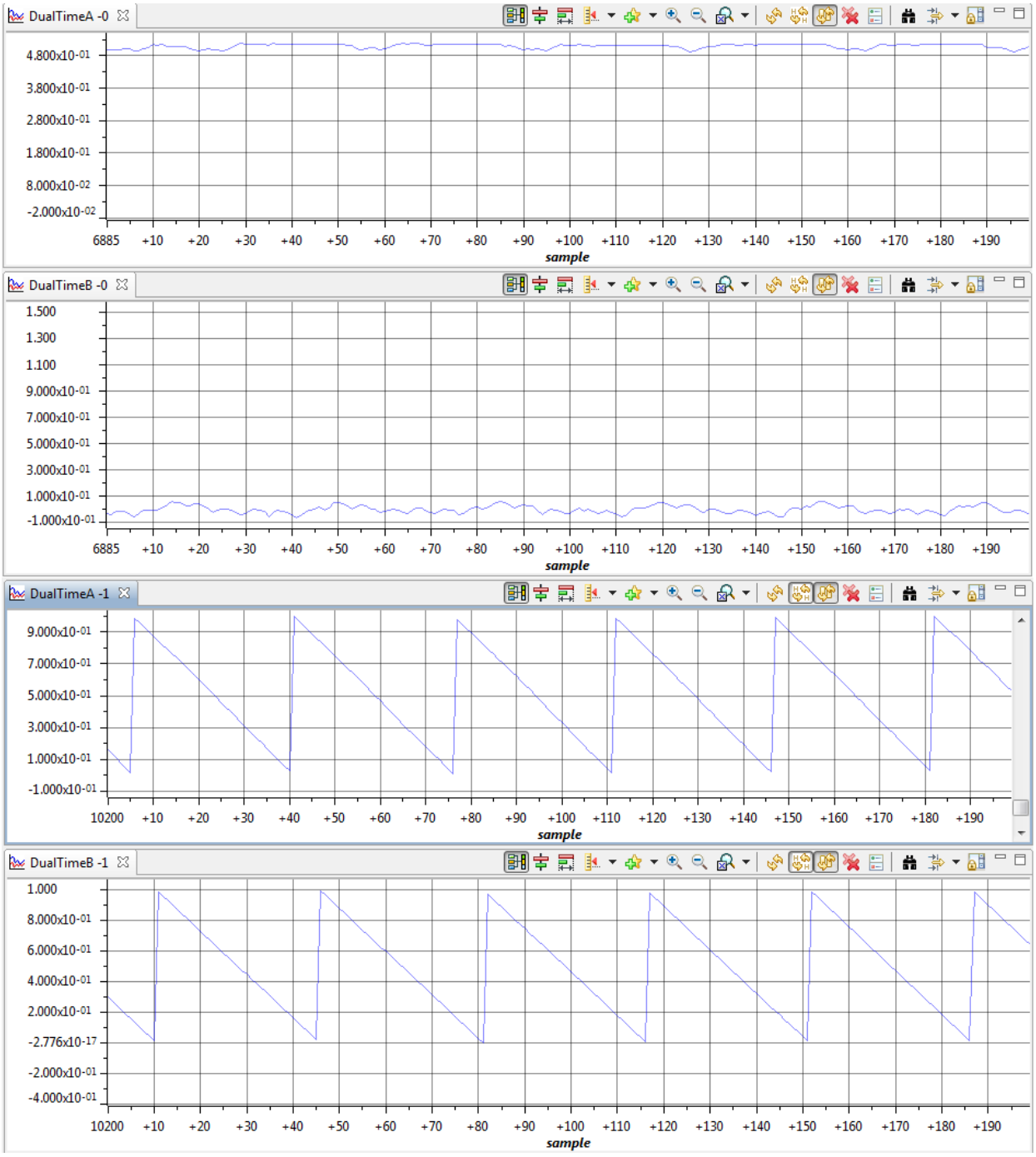

Fig 11 Graph Windows for Speed Control Mode (a) Motor speed feedback (pu), (b) Phase A current (pu), (c) SMO based electrical angle and (d) Encoder based electrical Angle

TI Spins Motors

Hercules Sensored FOC with SMO Block Diagram