TI Spins Motors

# InstaSPIN Control for the RM46x

Version 1.0.0 *Motor Solutions*


## Abstract

This application note presents a solution for sensorless control of Brushless DC motors using a RM46x microcontroller. The RM46x devices is part of the family of Cortex-R4F microcontrollers which enable cost-effective safety critical design of intelligent controllers for three phase motors by reducing the system components and increasing efficiency. Using these devices, it is possible to realize precise control algorithms for safety critical applications. A complete solution proposal is presented below: control structures, power hardware topology, control hardware and remarks on energy conversion efficiency can be found in this document.

This application note covers the following:

- A theoretical background on InstaSPIN BLDC motor control principle.
- A discussion of the BLDC drive imperfection handling the operating system


## Table of Contents


Introduction ..... 2
Directory Overview ..... 2
System Overview ..... 3
Hardware Configuration ..... 8
Software Setup Instructions to Run InstaSPIN_BLDC Project ..... 10
Running/Modifying the Project ..... 12
System Build ..... 18

## 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 time, 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 paper describes the use of a Brushless DC Motor (BLDC). Although the brushless characteristic can be applied to several kinds of motors – AC synchronous motors, stepper motors, switched reluctance motors, AC induction motors - the BLDC motor is conventionally defined as a permanent magnet synchronous motor with a trapezoidal Back EMF waveform shape. Permanent magnet synchronous machines with trapezoidal Back-EMF and (120 electrical degrees wide) rectangular stator currents are widely used as they offer the following advantages first, assuming the motor has pure trapezoidal Back EMF and that the stator phases commutation process is accurate, the mechanical torque developed by the motor is constant; secondly, the Brushless DC drives show a very high mechanical power density.

## Directory Overview

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

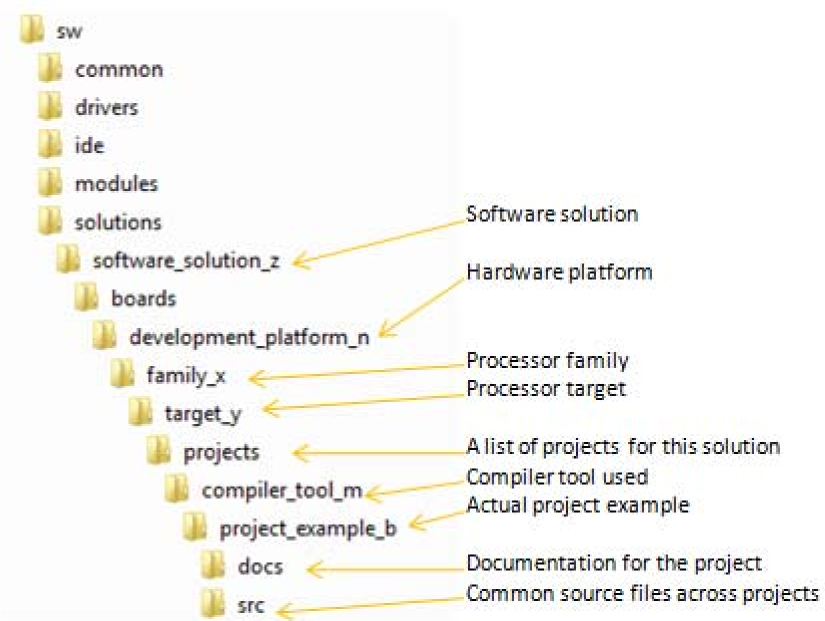


*Fig 1 MotorWare Directory Structure*

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

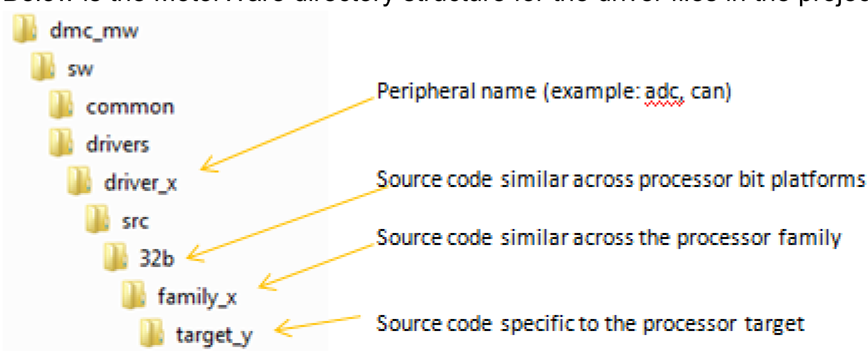


***Fig 2 Driver Files Directory Structure***

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

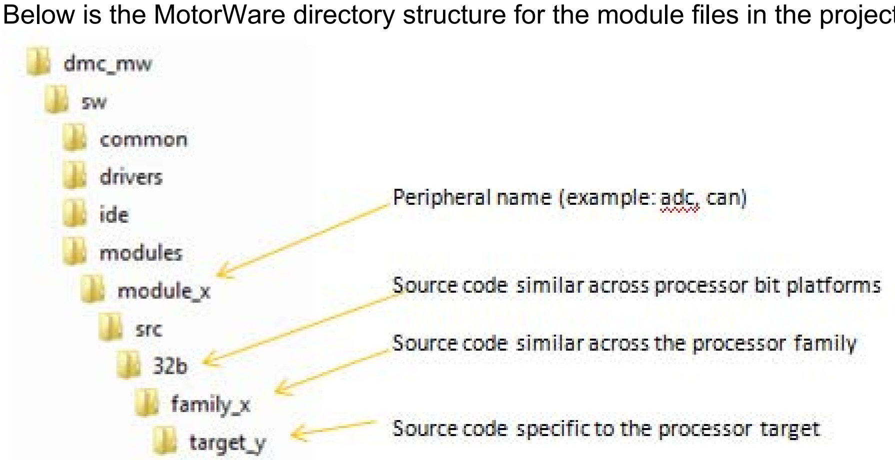


***Fig 3 Module Files Directory Structure***

## System Overview

This document describes the “C” real-time control framework used to demonstrate the trapezoidal control of BLDC motors. The “C” framework is designed to run on RM46 based controllers on Code Composer Studio.

The framework uses the following modules[1]:

| Macro Names | Explanation |
|---|---|
| BLDCPWM | PWM Drivers |
| InstaSPINTM-BLDC | InstaSPIN-BLDC Library Functions |
| PID | PID Regulators |
| RC | Ramp Controller (slew rate limiter) |
| RC3 | Ramp down Module |
| SPEED_PR | Speed Measurement (based on sensor signal period) |
| IMPULSE | Impulse Generator |
| MOD6_CNT | Mod 6 Counter |
| [1] Please refer to pdf documents in motor control folder explaining the details and theoretical background of each macro | |

In this system, the sensorless trapezoidal control of BLDC motors will be experimented with and will explore the performance of the speed controller. The BLDC motor is driven by a DRV8301 Three Phase PWM Motor Driver. The RM46x control card is used to generate three pulse width modulation (PWM) signals. The motor is driven by an integrated power module by means of BLDC specific PWM technique. Phase voltages and DC bus return current (I fb Ret) is measured and sent to the RM46x via analog-to-digital converters (ADCs).

InstaSPIN_BLDC project has the following properties:

| C Framework | | |
|---|---|---|
| **System Name** | **Program Memory Usage** RM46x | **Data Memory Usage**[1] RM46x |
| InstaSPIN_BLDC | 30744 bytes | 5300 bytes |

[1] Excluding the stack size

| CPU Utilization of Trapezoidal BLDC Control (Sensorless) | |
|---|---|
| **Name of Modules*** | **Number of Cycles** |
| BLDCPWM | 225 |
| InstaSPINTM-BLDC Library | 81 |
| PID | 268** |
| RC | 35 |
| RC3 | 30√ |
| SPEED_PR | 44 |
| IMPULSE | 23√ |
| MOD6_CNT | 20 |
| Data Update | 47 |
| | |
| **Total Number of Cycles** | **1015**** |
| CPU Utilization @ 160 Mhz | **13%***** |

* The modules are defined in the header files as “inline functions”
** Two Instances of PID in Cascade mode
*** At **20 kHz** ISR freq.
√ Not included in the speed loop

| System Features | |
|---|---|
| **Development /Emulation** | Code Composer Studio v5.2.1 (or later) with Real Time debugging |
| **Target Controller** | RM46x |
| **PWM Frequency** | 20kHz PWM (Default) |
| **PWM Mode** | Symmetrical with 4 quadrant switching and programmable dead-band. |
| **Interrupts** | adcNotification() |
| **Peripherals Used** | ePWM – PWM Generation<br>eQEP – Quadrature Encoder Read<br>SPI – Gate Driver and Control Card Power Supply Configuration<br>GIO – Fault diagnostics, Gate driver enable/disable<br>RTI – Time base for delays and profiling<br>ADC Channel A9 and A21 for low side current sensing, Channels 23, 15, and 20 for Bemf sensing (A, B, and C, respectively), and channel 18 for DC bus voltage sensing. |

The overall system implementing a 3-ph InstaSPIN BLDC control is depicted in Fig.4.

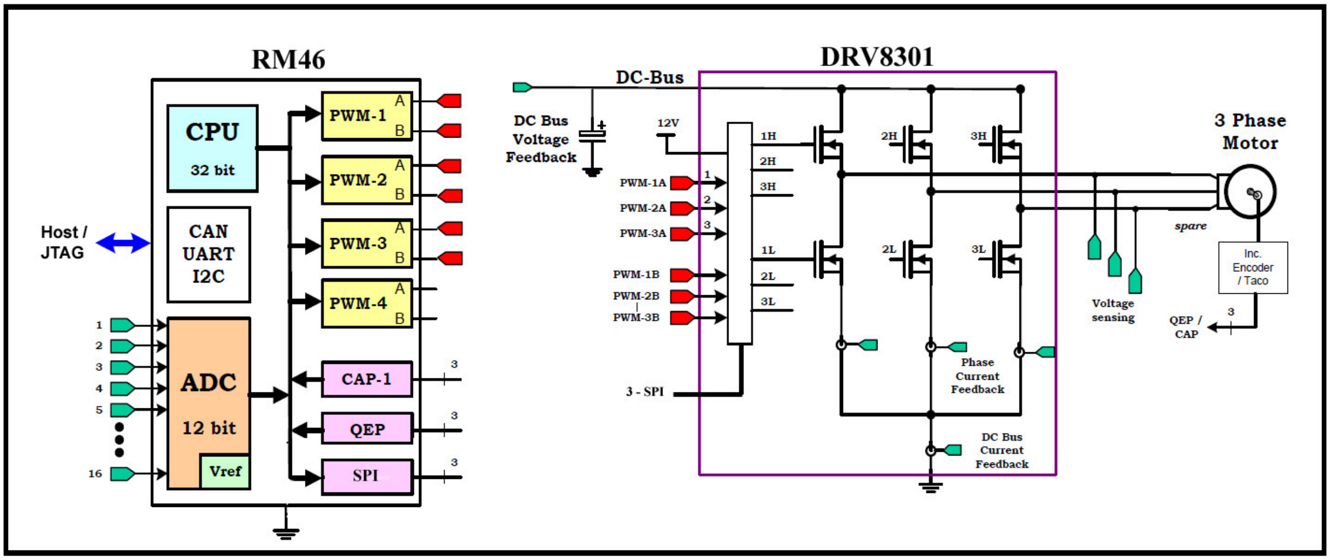


*Fig 4 A 3-ph BLDC drive implementation*

**Special Setup Notes:**

**The DRV8301 driver chip has configurable PWM and protection modes that are configured with SPI at startup. The start-up sequence with the RM46 is as follows:**

1) **RM46 Boots and Initializes.**
2) **The EN_GATE signal on the DRV8301 is driven high with a GPIO.**
3) **The RM46 waits for 50ms while the DRV8301 powers up.**
4) **The RM46 executes write SPI commands to the DRV8301 for setup.**
   a. **For the InstaSPIN project:**
      i. **The Gate Driver Peak Current is set to 1.7A.**
      ii. **The Gate Driver Reset is Normal.**
      iii. **The PWM Mode is Independent.**
      iv. **The OC mode is Latched Shut down (to reset the fault a power cycle needs to occur).**
      v. **The Gain of the internal Amplifiers is set to 80V/V.**
5) **The RM46 starts PWMs and peripheral interrupts.**

The software flow is described in the Figure 5 below.

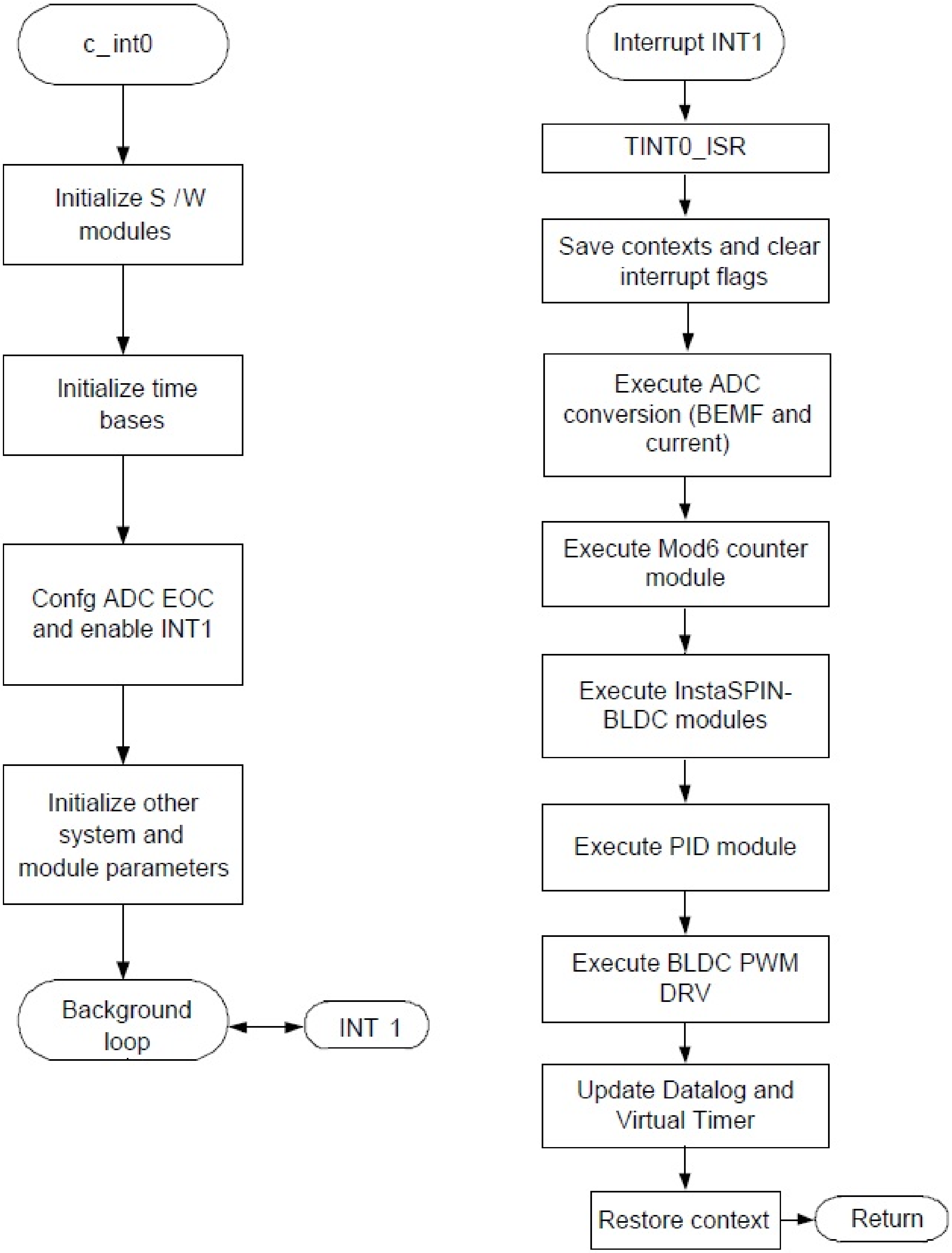


***Fig 5 Software Flow Chart***

## Hardware Configuration (DRV8301-EVM)

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

sw/boards/drv8301kit_revB_revC/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 following jumpers & connector settings are vaild

   (i) JP2 is installed
4. Connect a USB cable to connector J101 on the RM46. This will enable isolated JTAG emulation to the Hercules RM4x device. The power (POW) LED should turn on. If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will automatically be installed. If this is your first time connecting to the control card set all positions of DIP switch SW1 to off. This will disable the TPS65381 watch dog.
5. Connect a USB cable from the computer to the USB connector on the control card.
6. Connect a 24V power supply to the DRV8301-EVM with positive connected to J25 and ground connected to J26. 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.

   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.
7. Note that the motor should be connected to the OUTA, OUTB and OUTC terminals. For more details on motor wiring please refer to the datasheet provided with your motor.

For reference the pictures below show the connectors on the DRV8301-EVM

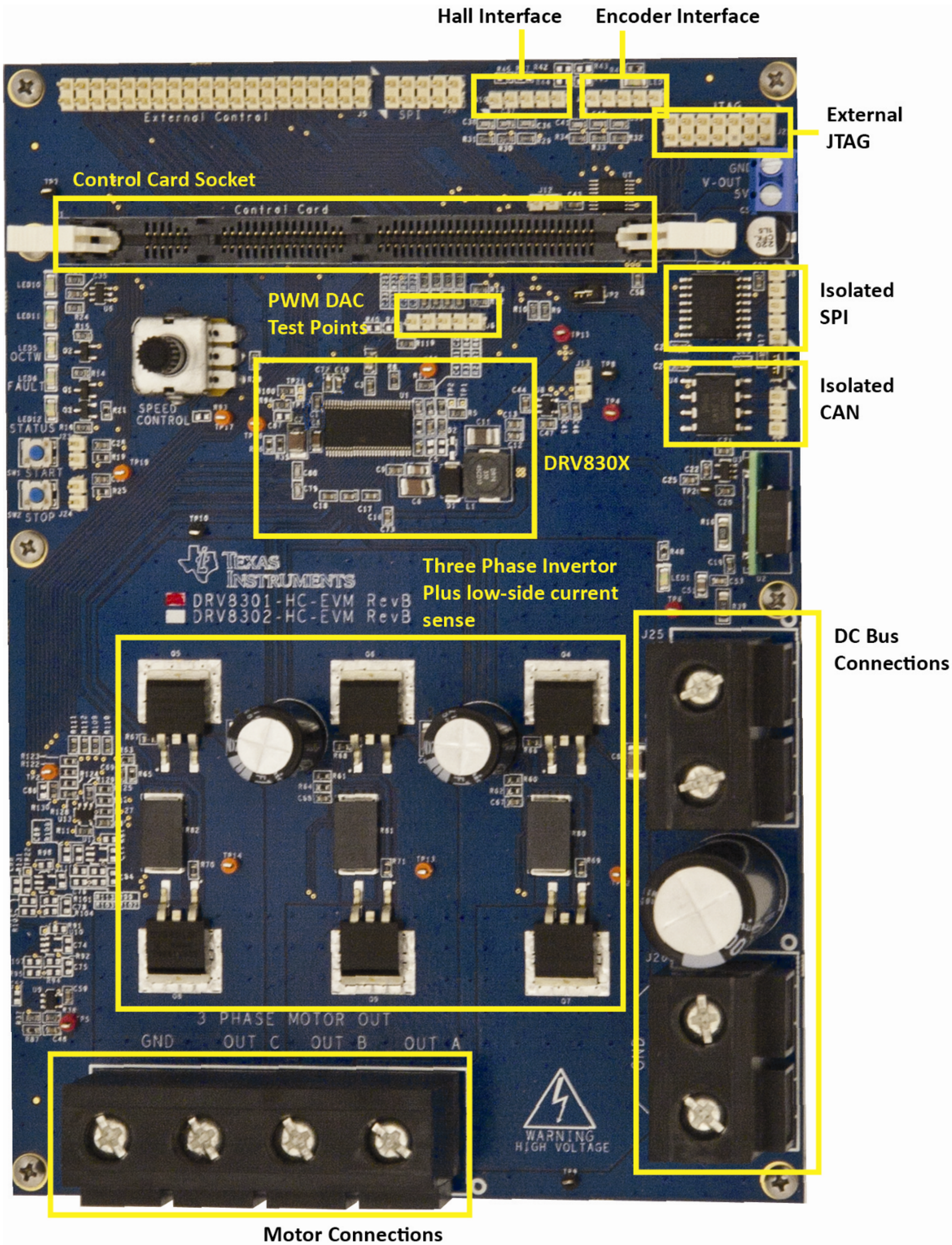


*Fig. 6 DRV8301-EVM Connections*

**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 InstaSPIN_BLDC 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 will help you download, install, and run Code Composer Studio v5.

CCS v5.2.1.00018 or later needs to be used with ARM compiler 5.0.0 or later installed for the software to build correctly.

When CCSv5 has been downloaded, installed, and is running, the InstaSPIN project needs to be imported. In CCS, go to Project->Import Existing CCS/CCE Eclipse Project. Choose to browse and go to: **sw/solutions/instaspin_bldc/boards/drv8301kit_revD/rm46l852/projects/ccs5/project01** in your MotorWare directory. The InstaSPIN project should be discovered. Make sure the checkbox next to the project is checked. Click on Finish to import the project into your workspace.

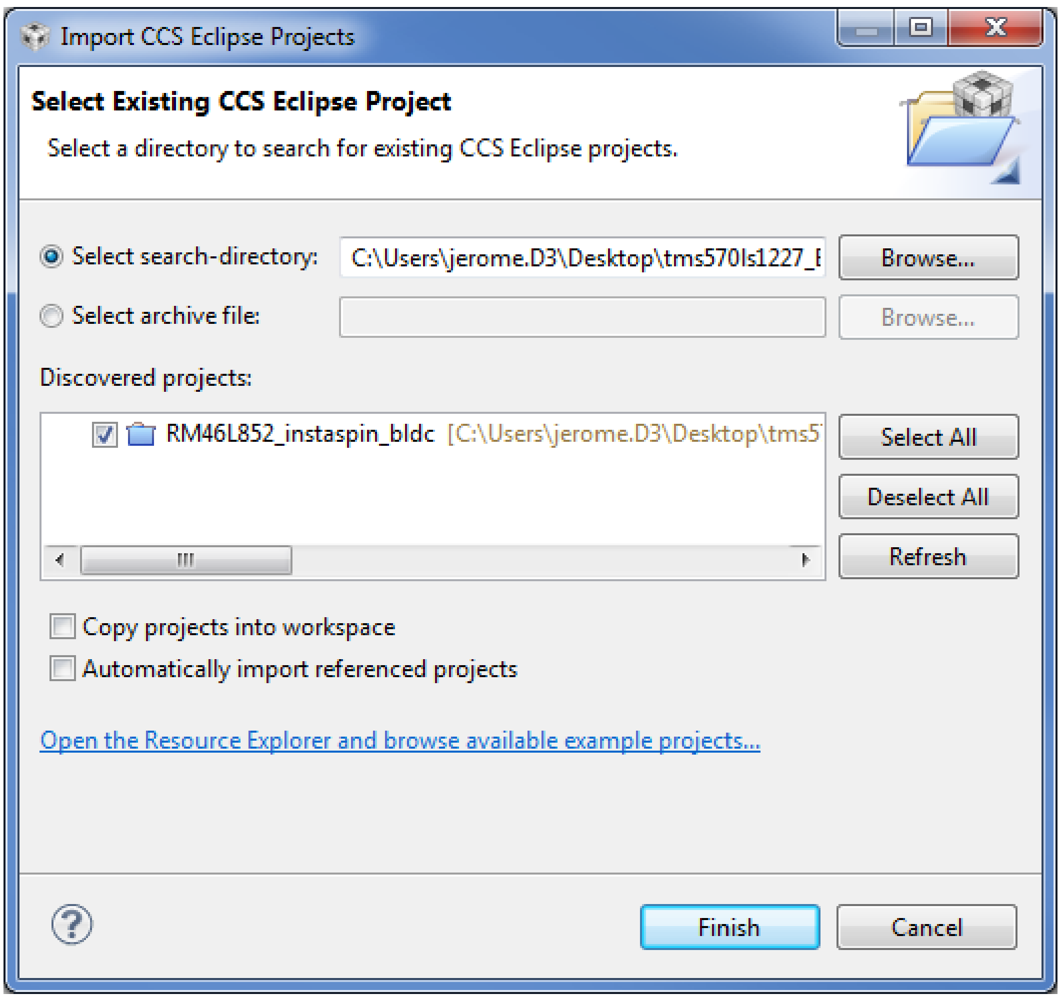


***Fig 7 Importing a Project***

Right click on the project name and select “Clean Project” then “Build Project”. Once the build completes, launch a debug session to load the code into the controller. Make sure you are connected to the RM46x CortexR4 device (Run->Connect Target) and load the program (Run->Load->Load Program). Press the Run button.

Next, create a Watch Window and add the control variables by running a script. To run the script, open the Scripting Console by going to View->Scripting Console. In the Scripting Console, click on the folder icon to choose a script to run. Go to: **sw/solutions/instaspin_bldc/boards/drv8301kit_revD/rm46l852/projects/ccs5/project01**, and choose InstaSPINWatch.js. The script should run. You should now have a watch window that looks like Figure 8.

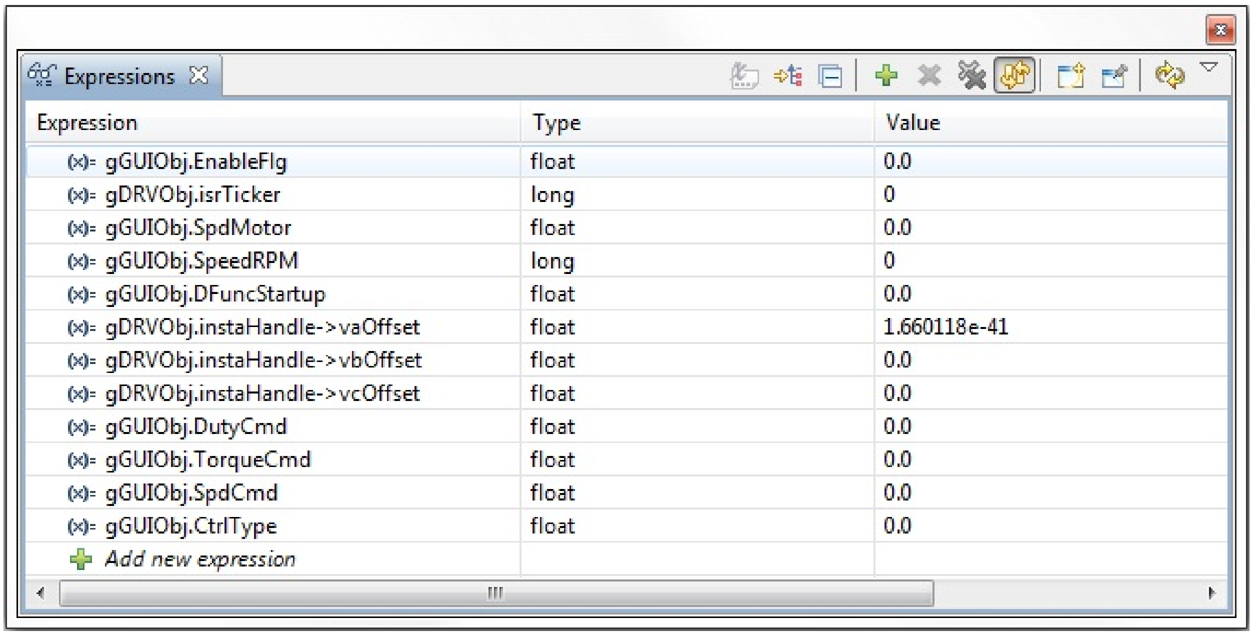


*Fig 8 Watch Window Setup*

Make sure you are connected to the RM46x CortexR4 device and load the program. Press the Run button.

Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp (Tools->Graph->Dual Time: Import Button) from the following location:

**sw/solutions/instaspin_bldc/boards/drv8301kit_revD/rm46l852/projects/ccs5/project01**. Click on Continuous Refresh button on the top left corner of the graph tab to enable periodic capture of data from the microcontroller.

**Running/Modifying the Project**

**TPS65381 Safety Critical Power Supply**

The TPS65381 regulator powers the RM46 control card. The first time you connect to the processor make sure all positions of the SW1 DIP switch are set to the off position to disable the watchdog on the TPS65381. When the TPS65381 watch dog is used the watch dog must be disabled within 600ms of power on or it will have to be serviced. If the TPS65381 is enabled and the software flashed on the control card does not disable the watch dog within 600ms you will not be able to connect to the control card. The provided software project needs the TPS65381 to be enabled to run correctly, after connecting to the control card for the first time, load the compiled software, disconnect from the target in CCS, power down the system, place positions 1 and 4 of DIP switch SW1 into the on position and re-power the system.

When the TPS65381 is enabled the supplied software will detect its presence and automatically disable the watchdog within the first 600ms of power up. The software will then wait for user input. Because the TPS65381 controls the enable signal of the DRV8301 gate driver the system will wait for the user to enable the gGUIObj.TPSFlag in the watch window. Enabling this Flag allows the software to utilize the SPI port to enable the TPS65381 watchdog, this will activate the DRV8301 powering up the motor inverter. The motor inverter cannot be powered without enabling the watchdog. It should be noted that once the watchdog is enabled it has to be serviced over the SPI regularly or power to the processor will be disabled. The software automatically handles the servicing, but if you halt the code execution in CCS or set a break point the TPS65381 will power cycle and you will lose your connection to the processor. If you would like to be able to set break points or halt the processor disable the TPS65381 by setting all positions on DIP switch SW1 to OFF and commenting out lines 89-98 in drv.c.

At any point in time, you can modify the settings in the Watch Window to control how the motor runs. While the motor is stopped (by setting “gGUIObj.EnableFlg” to 0), you may change which mode the motor runs in.

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 a simply Duty Cycle (sensorless) commutation mode. The possible values of CtrlType are described below:

| Value | Mode |
|---:|---|
| 0 | Duty Cycle |
| 1 | Current |
| 2 | Velocity |
| 3 | Current + Velocity |

**Duty Cycle (Sensorless) Commutation Mode**

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

- gDRVObj.rmpCntlHandle->targetValue: for changing the targeted commutation interval.
- gDRVObj.rmpCntlHandle->setpointValue: for changing the initial startup commutation interval.
- gGUIObj.DFuncStartup: for changing the startup PWM duty cycle in per-unit.
- gGUIObj.DutyCmd: for changing the speed of the motor
- gGUIObj.CommThreshold: for changing the BEMF integration threshold in per-unit

To run in Duty Cycle 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 Duty Cycle Commutation Mode.
3. Set “gGUIObj.EnableFlg” to 1.
4. At this point, the variable “gDRVObj.isrTicker should be increasing. This confirms that the interrupt is running properly. The motor should start moving.

The key steps can be explained as follows:

- The motor will gradually speed up and finally switch to closed loop commutation mode.
- The switch over from open loop commutation to closed loop commutation occurs when gDRVObj.rmp3Handle->ramp3DoneFlag is set to 0x7FFFFFFF indicating the end of motor speed up phase. Until this switch over occurs MOD6_CNT module is triggered by the output of IMPULSE module. After the switch over, MOD6_CNT module is triggered by the output of the InstaSPIN™-BLDC module.
- When the speed up phase is over, vary the motor speed by changing gGUIObj.DutyCmd. This varies the power delivered to the motor and hence it’s speed.
- Adjust gGUIObj.CommThreshold to achieve the desired commutation.
- Bring the system to a safe stop as described below by setting gGUIObj.EnableFlg to 0, taking the controller out of realtime mode and reset.

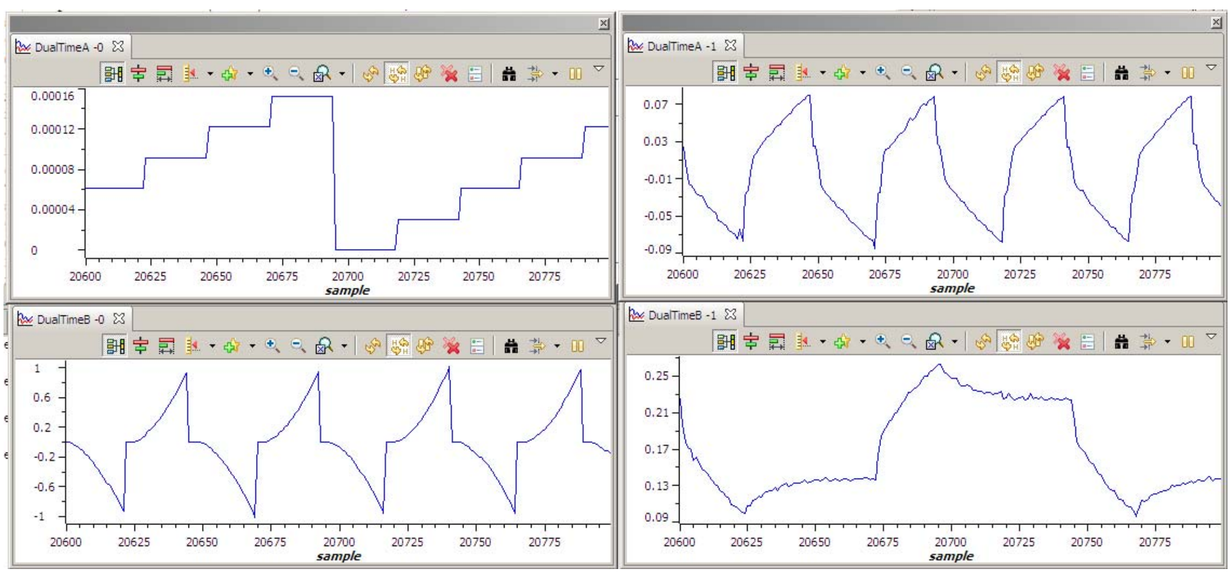


*Fig 1 Graph Windows for Duty Cycle Commutation Mode (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag*

### Closed Current Loop Mode

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

- gDRVObj.rmpCntlHandle->targetValue: for changing the targeted commutation interval.
- gDRVObj.rmpCntlHandle->setpointValue: for changing the initial startup commutation interval.
- gGUIObj.CurrentStartup: for changing the startup current in per-unit.
- gGUIObj.TorqueCmd: changing the running current in per-unit.

- gGUIObj.CommThreshold: for changing the BEMF integration threshold in per-unit

To run in Closed Current Loop 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 1, for Closed Current Loop Mode.
3. Set “gGUIObj.EnableFlg” to 1.
4. At this point, the variable “gDRVObj.isrTicker should be increasing. This confirms that the interrupt is running properly. The motor should start moving.

The steps are explained as follows:

- The motor will gradually speed up and finally switch to closed loop commutation mode.
- Now use the variable gGUIObj.TorqueCmd to specify the reference current for the PI controller. The PI controller will start to regulate the DC bus current and hence the motor current. Gradually increase/decrease the command current (gGUIObj.TorqueCmd value) to change the torque command and adjust PI gains. 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. Initially apply relatively light load and then gradually increase the amount of the load. If the applied load is higher than the torque reference, the motor cannot handle the load and stops immediately after closing the current loop.
- Verify the motor speed (both pu and rpm) calculated by SPEED_PR module. View the following variables in the Watch Window.
    - gGUIObj.SpdMotor (pu)
    - gGUIObj.SpeedRpm (rpm)
- Bring the system to a safe stop as described below by setting gGUIObj.EnableFlg to 0, taking the controller out of realtime mode and reset.

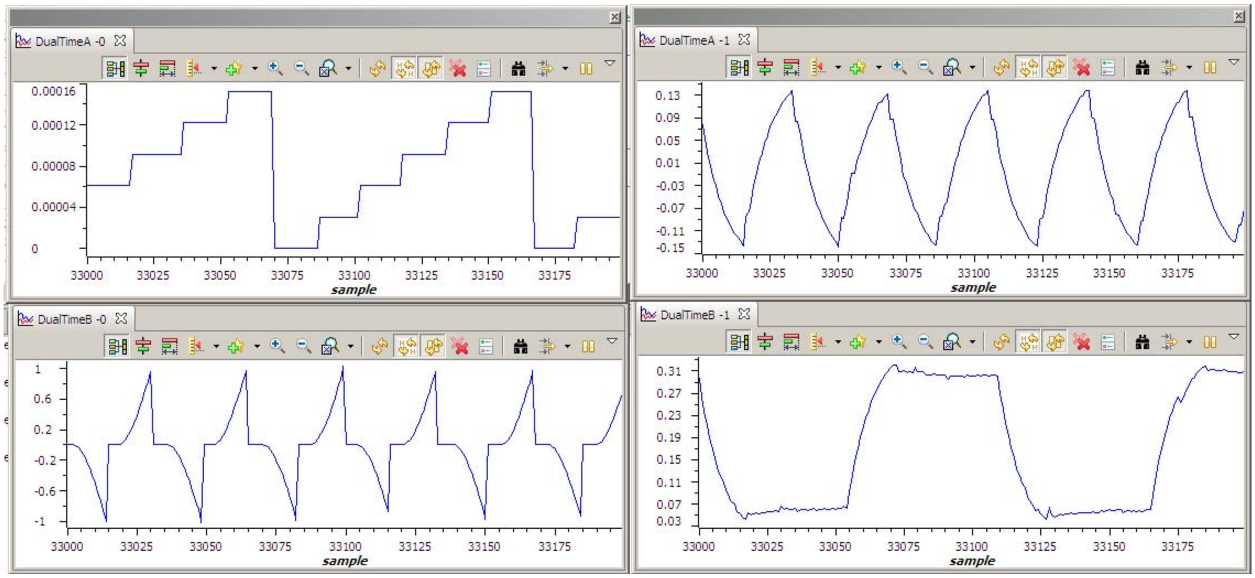


*Fig 2 Graph Windows for Closed Current Loop Mode (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag*

### Velocity (Closed Loop Speed and Speed PI) Mode

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

- gGUIObj.SpeedCmd: for changing the reference Speed in per-unit.

To run in Velocity Loop 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 2, for Velocity Mode.
3. Set “gGUIObj.EnableFlg” to 1.
4. At this point, the variable “gDRVObj.isrTicker should be increasing. This confirms that the interrupt is running properly. The motor should start moving.

The steps are explained as follows:

- Compile/load/run program with real time mode.
- The motor will gradually speed up and finally switch to closed loop commutation mode.
- Now use the variable gGUObj.SpeedCmd to specify the reference speed for the PI controller PID_REG3. The gDRVObj.speedLoopFlag is automatically activated when the PI reference is ramped up from zero speed to gGUObj.SpeedCmd. Once this is done, the PI controller will start to regulate the motor speed. Gradually increase the command speed (gGUObj.SpeedCmd value) to increase the motor speed.

- Adjust speed PI gains to obtain the satisfied speed responses, if needed.
- Bring the system to a safe stop as described below by setting gGUIObj.EnableFlg to 0, taking the controller out of realtime mode and reset.

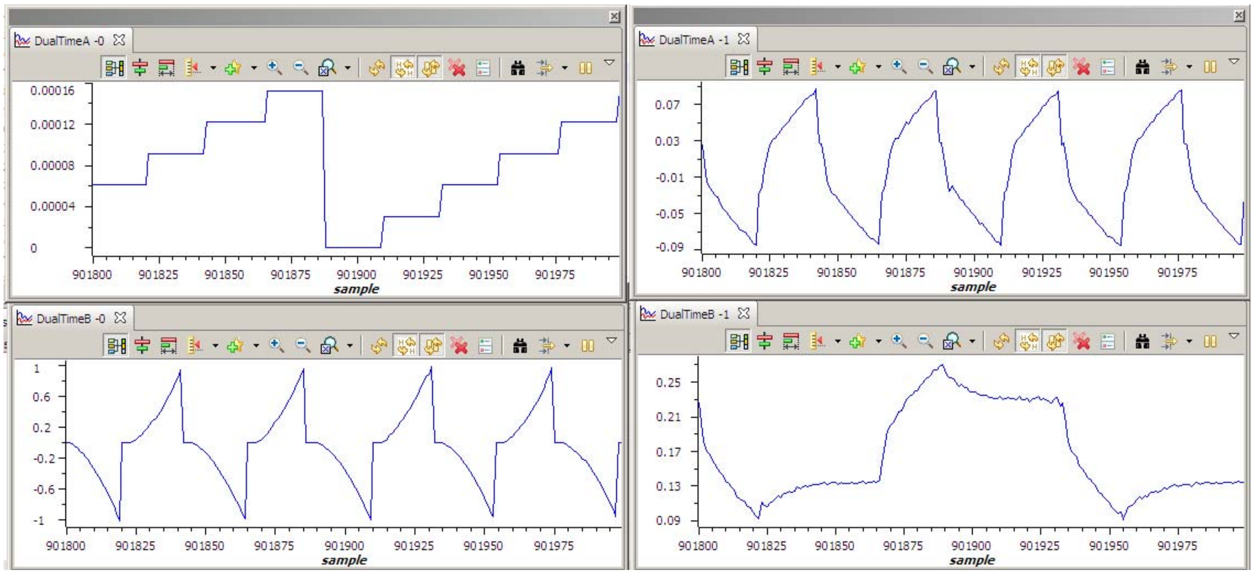


*Fig 3 Graph Windows for Velocity Mode (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag*

**Cascaded Velocity + Current Loop Mode**

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

- gGUIObj.SpeedCmd: for changing the reference Speed in per-unit.

To run in Cascaded Velocity + Current Loop 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 3, for Cascaded Velocity + Current Loop Mode.
3. Set “gGUIObj.EnableFlg” to 1.
4. At this point, the variable “gDRVObj.isrTicker should be increasing. This confirms that the interrupt is running properly. The motor should start moving.

The steps are explained as follows:

- The motor will gradually speed up and finally switch to closed loop commutation mode.
- Now use the variable gGUIObj.SpeedCmd to specify the reference speed for the PI controller PID_REG3. The gDRVObj.speedLoopFlag is automatically activated when the PI reference is ramped

up from zero speed to gGUIObj.SpeedCmd. Once this is done, the PI controller will start to regulate the motor speed. Gradually increase the command speed (gGUIObj.SpeedCmd value) to increase the motor speed.

- Adjust speed PI gains to obtain the satisfied speed responses, if needed.

- Bring the system to a safe stop as described below by setting gGUIObj.EnableFlg to 0, taking the controller out of realtime mode and reset.

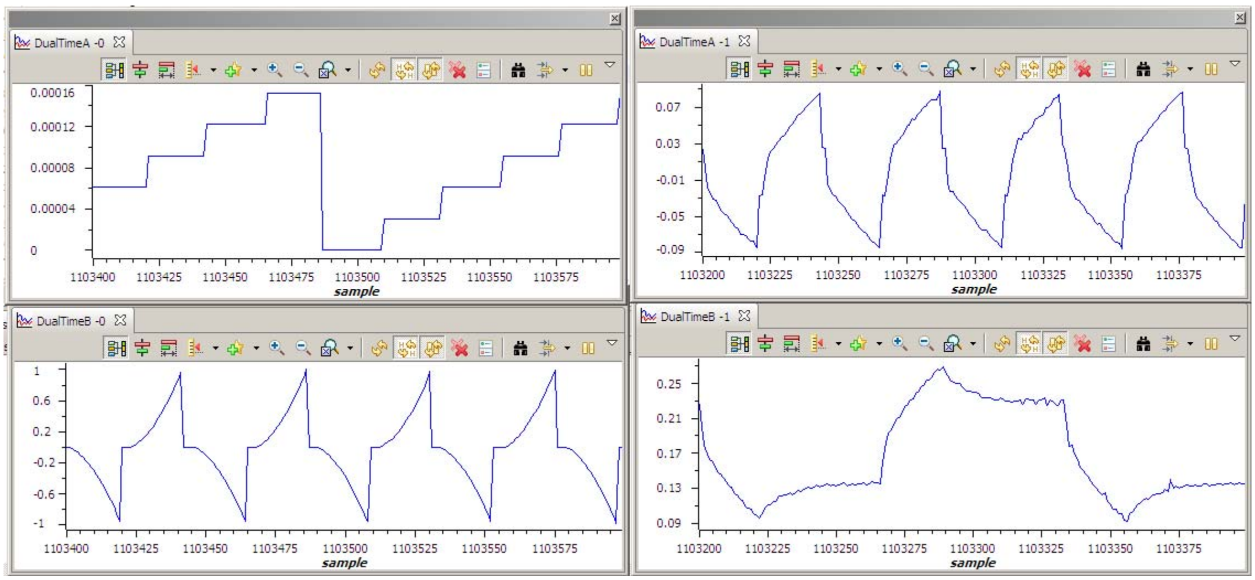


***Fig 4 Graph Windows for Cascaded Velocity/Current Loop Mode (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag***

# Hercules InstaSPIN BLDC Block Diagram