Application Report

TIDA014–June 2014

Sensorless Trapezoidal Control of BLDC Motors using BEMF Integration (InstaSPINTM-BLDC) Application Report

ABSTRACT

This application note presents a solution for a sensorless control of Brushless DC motors. This solution uses Texas Instruments TMS320F2803x microcontrollers. TMS320F280x devices are part of the C2000 microcontrollers family. These devices enable cost-effective design of intelligent controllers for three-phase motors by reducing the number of system components and increasing the system efficiency. Using these devices, it is possible to realize precise control algorithms.

This application note covers the following:

- Incremental build levels based on modular software blocks
- Experimental results

Contents

List of Figures

List of Tables

1 System Overview

This document describes the “C” real-time control framework that is used to show the trapezoidal control of BLDC motors. The “C” framework is designed to run on TMS320C2803x-based controllers in Code Composer Studio. The framework uses the following modules:

Table 1. Framework Modules

Macro Names[(1)]	Explanation
BLDCPWM / PWMDAC	PWM and PWMDAC Drives
InstaSPIN-BLDC	InstaSPIN-BLDC Library Functions
PID_GRANDO	PID Regulators
RC	Ramp Controller (slew rate limiter)
RC3	Ramp Down Modules
SPEED_PR	Speed Measurement (based on sensor signal period)
IMPULSE	Impulse Generator
MOD6_CNT_DIR	Mod 6 Counter with direction control

(1) Please refer to documents in motor control folder that explain the details and theoretical background of each macro.

In this system, you experiment with sensorless trapezoidal control of BLDC motors, and you explore the performance of the speed controller. The BLDC motor is driven by a DRV830x device, a Three-Phase PWM Motor Driver. The TMS320F2803x control card generates three pulse-width modulation (PWM) signals. An integrated power module drives the motor by using BLDC-specific PWM techniques. Phase voltages and DC-bus return current (Ifb Ret) are measured and sent to the TMS320x2803x control card through analog-to-digital converters (ADCs).

The InstaSPIN_BLDC project has the following properties:

Table 2. C Framework

System Name	Program Memory Use 2803x	Data Memory Use 2803x[(1)]
InstaSPIN_BLDC	4597 words[(2)]	1200 words

(1) Excluding the stack size

(2) Excluding "IQmath" Look-Up Tables

Table 3. System Features

Development /Emulation	Code Composer Studio v4.1 (or above) with Real Time debugging
Target Controller	TMS320F2803x
PWM Frequency	20kHz PWM (Default), 60kHz PWMDAC
PWM Mode	Symmetrical with 4 quadrant switching and programmable dead-band.
Interrupts	ADCINT1 EOC
Peripherals Used	PWM 1 / 2 / 3 for motor control
	PWM 5A, 6A, 6B & 4A for DAC outputs
	ADC A2 for low side DC bus return current sensing, B7, A7 and B4 for
	Bemf sensing
	SPI-B for communication and configuration of the DRV8301 (DRV8302 uses discrete digital and analog I/O for configuration)

Table 4. CPU Use of Trapezoidal BLDC Control (Sensorless)

Name of Modules (1)	Number of Cycles
BLDCPWM	105
InstaSPINTM-BLDC Library	277
PID	91
RC	29
RC3	26 (2)
SPEED_PR	42
IMPULSE	17 (2)
MOD6_CNT_DIR	9
Contxt Save, Virtual Timer etc.	153
Pwm Dac (optional)	
DataLog (optional)	
Total Number of Cycles	**749** (3)
CPU Utilization @ 60 Mhz	25%
CPU Utilization @ 40 Mhz	37.40%

(1) The modules are defined in the header files as "macros"
(2) Not included in the speed loop
(3) At 20kHz ISR frequency

Figure 1 shows the overall system, which implements a 3-ph sensorless BLDC control.

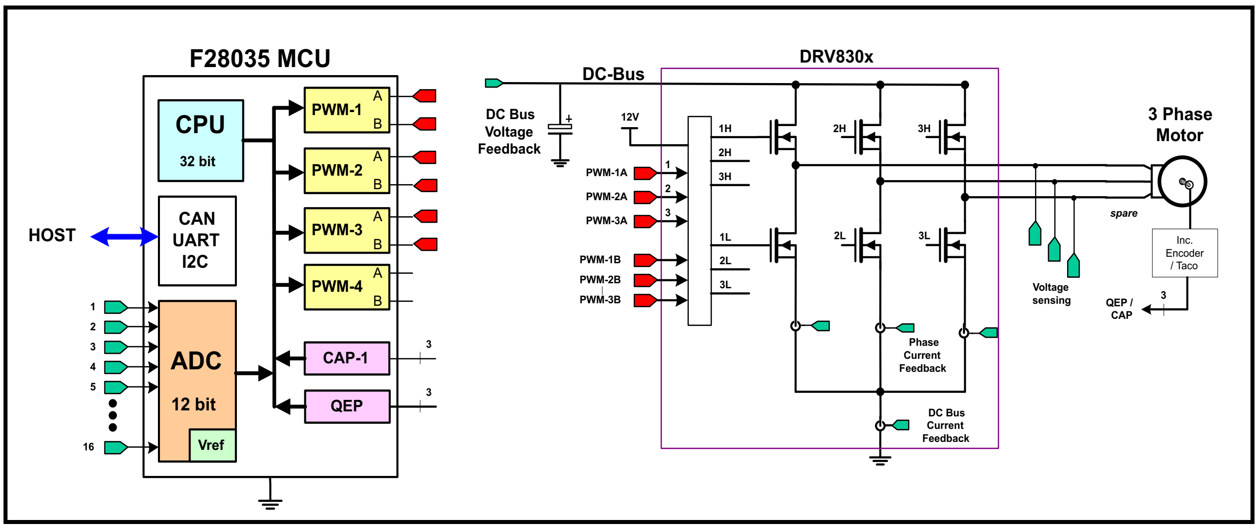


Figure 1. 3-ph BLDC Drive Implementation

Figure 2 illustrated the software flow.

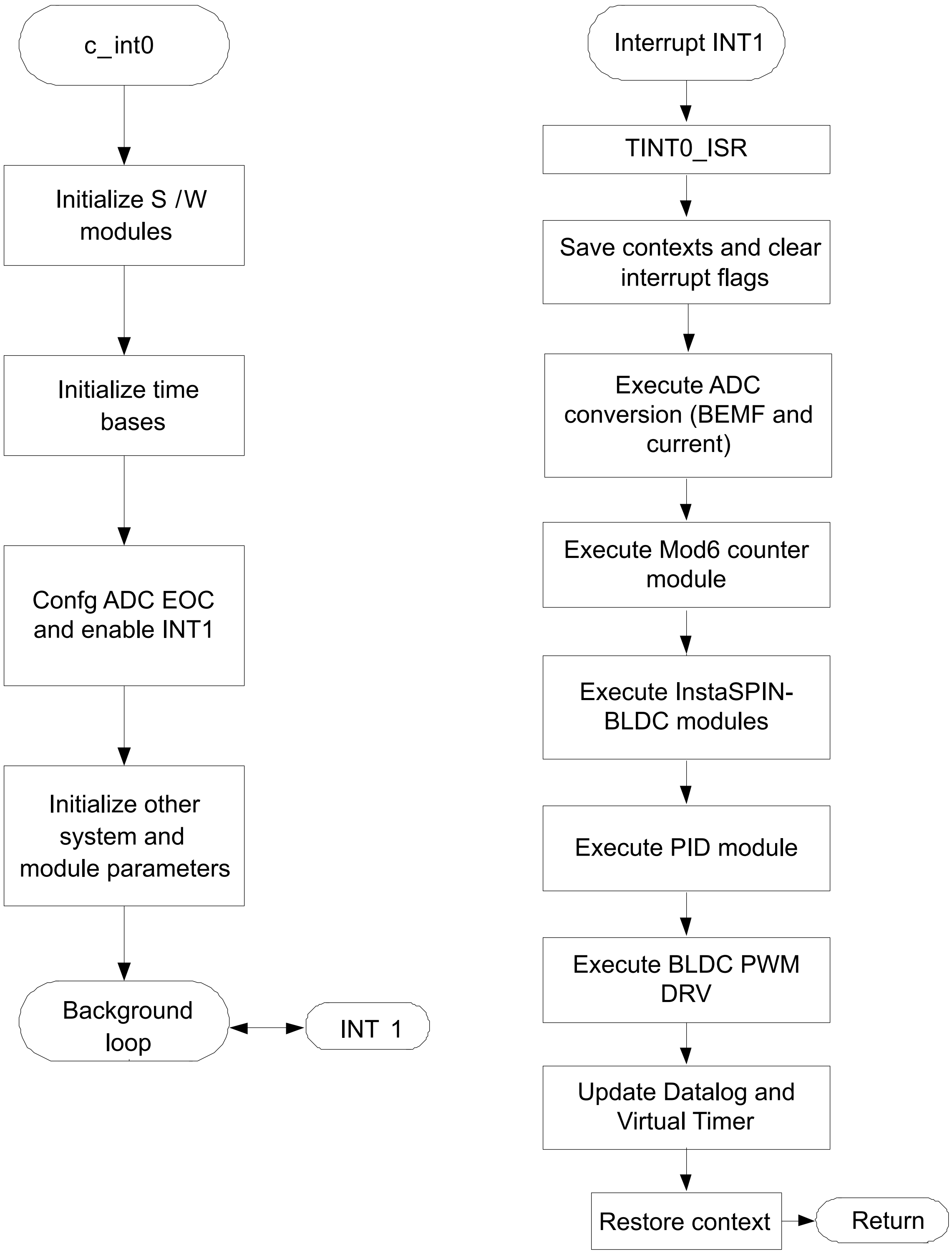


Figure 2. Software Flowchart

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

For an overview of the kit hardware and the steps for kit setup, refer to the *DRV830x-HC-EVM How to Run Guide* and *Hardware Reference Guide* in the following directory:

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

Some of the hardware setup instructions can be found in the following sections, for quick reference.

2.1 Hardware Setup Instructions

1. Unpack the DIMM style controlCARD and verify that the DIP switch settings match Figure 3.

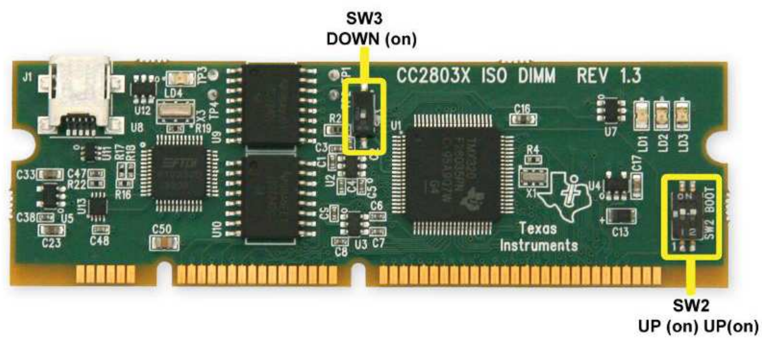


Figure 3. controlCARD DIP Switch Settings

2. Place the controlCARD in the connector slot of J1. Push vertically down using even pressure from both ends of the card until the clips snap and lock. To remove the card simply spread open the retaining clip with thumbs.
3. Connect a USB cable to connector J1 on the controlCARD. This connection will enable isolated JTAG emulation to the C2000 device. LD4 should turn on.

   If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will install automatically. If a windows installation window appears, try to automatically install the drivers from those already on your computer.

   The emulation drivers can be found at http://www.ftdichip.com/Drivers/D2XX.htm. The correct driver is the one listed to support the FT2232.
4. Connect a power supply (60V max) to the PVDD and GND terminals of the DRV830x-HC-EVM. LED1 and LED3 should turn on. Notice that the control card LED also lights up, indicating that the control card is receiving power from the board.
5. After completing the first incremental build step, 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, Figure 4 shows the jumper and connectors that need to be connected for this lab.

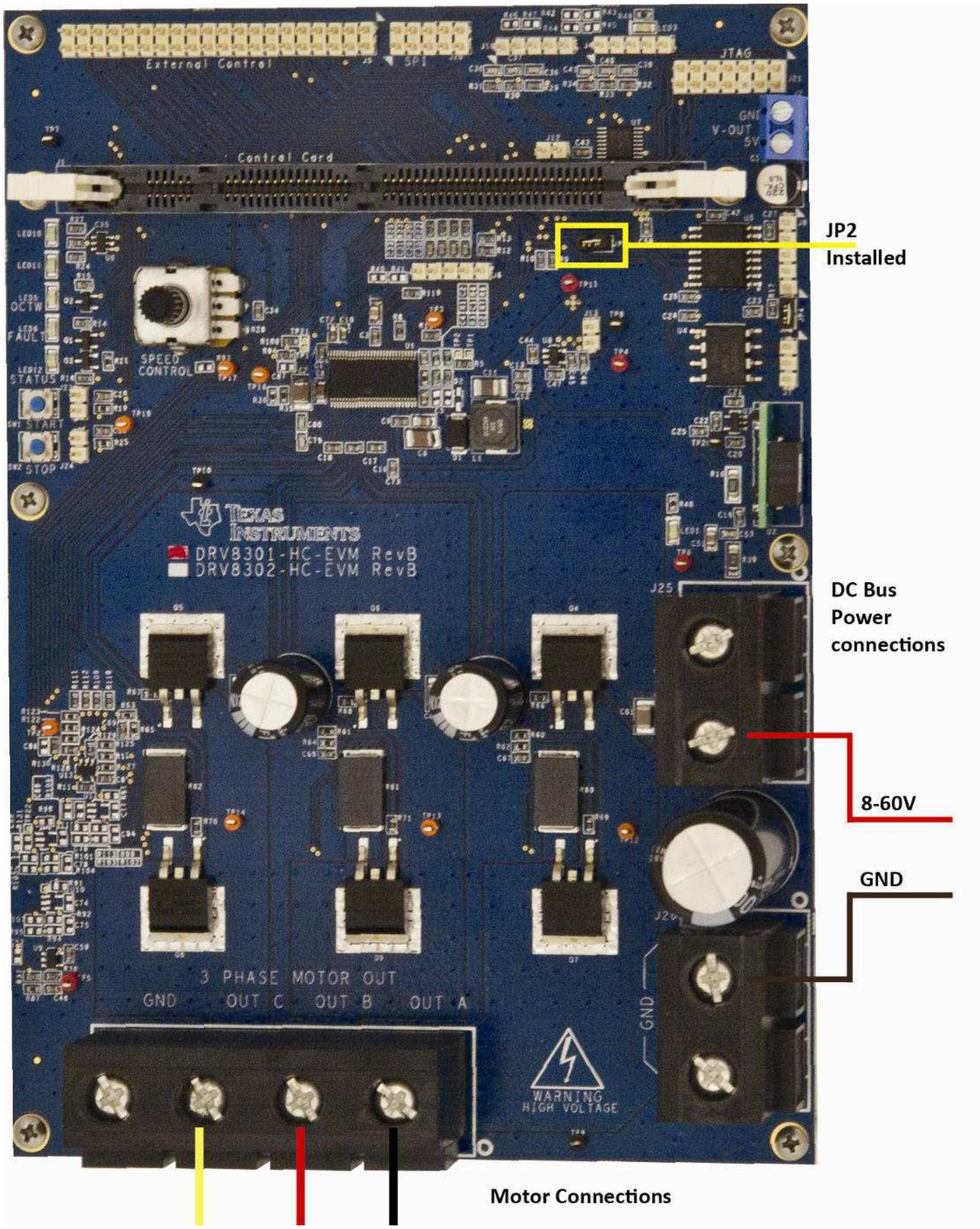


Figure 4. DRV830x-HC-EVM Connections and Settings

WARNING

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

2.2 Software Setup Instructions to Run InstaSPIN_BLDC Project

Please refer to the *Generic Steps for Software Setup for DRV830x-HC-C2-KIT Projects* section in the DRV830x-HC-EVM *How To Run Guide*

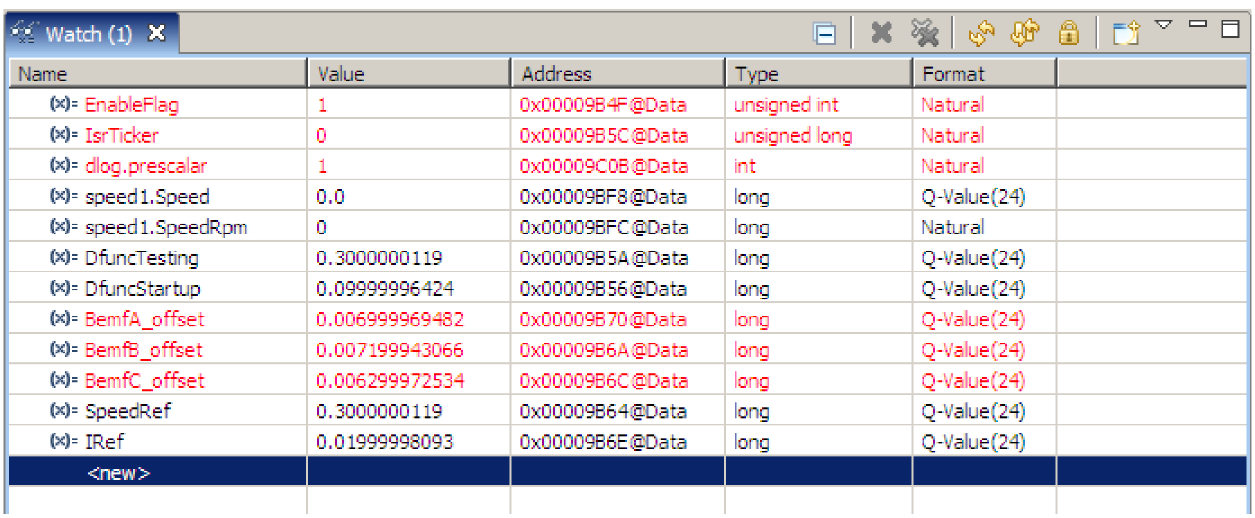
`C:\TI\controlSUITE\development_kits\DRV830x-HC-C2-KITv*\~Docs`

This section shows how to install Code Composer Studio (CCS) and how to set it up to run with this project.

The remainder of this application note discusses the hardware configuration for this project, which consists of the DRV8301-HC-EVM with an installed TMS320F2803x controlCARD. The process for other configurations, such as a DRV8302-HC-EVM, would be similar except the corresponding build configuration would need to be chosen in CCS.

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

1. Select the InstaSPIN_BLDC as the active project.
2. Verify that the build level is set to 1.
3. Right-click on the project name and select “Rebuild Project”.
4. Once build completes, launch a debug session to load the code into the controller.
5. Open a watch window and add the variables shown in Figure 5, and select the appropriate Q format for them.


Watch (1)

Name	Value	Address	Type	Format
(x)= EnableFlag	1	0x00009B4F@Data	unsigned int	Natural
(x)= IsrTicker	0	0x00009B5C@Data	unsigned long	Natural
(x)= dlog.prescalar	1	0x00009C0B@Data	int	Natural
(x)= speed1.Speed	0.0	0x00009BF8@Data	long	Q-Value(24)
(x)= speed1.SpeedRpm	0	0x00009BFC@Data	long	Natural
(x)= DfuncTesting	0.3000000119	0x00009B5A@Data	long	Q-Value(24)
(x)= DfuncStartup	0.09999996424	0x00009B56@Data	long	Q-Value(24)
(x)= BemfA_offset	0.006999969482	0x00009B70@Data	long	Q-Value(24)
(x)= BemfB_offset	0.007199943066	0x00009B6A@Data	long	Q-Value(24)
(x)= BemfC_offset	0.006299972534	0x00009B6C@Data	long	Q-Value(24)
(x)= SpeedRef	0.3000000119	0x00009B64@Data	long	Q-Value(24)
(x)= IRef	0.01999998093	0x00009B6E@Data	long	Q-Value(24)
<new>				

Figure 5. Watch Window Setup

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

2.3 Incremental System Build for InstaSPIN™-BLDC project

For the final system to be confidently operated, we gradually build the system over eight phases. The eight phases (builds) of the incremental system build are designed to test and verify the correct operation of the major software modules used in the system. Table 5 summarizes the modules tested and used in each incremental system build.

Table 5. Testing Modules in Incremental System Builds

Software Module	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8
PWMDAC_MACRO	(1)	(1)		(2)	(2)	(2)	(2)	(2)
RC3_MACRO	(1)	(2)		(2)	(2)	(2)	(2)	(2)
MOD6_CNT_DIR_MACRO	(1)	(2)		(2)	(2)	(2)	(2)	(2)
IMPULSE_MACRO	(1)	(2)		(2)	(2)	(2)	(2)	(2)
BLDCPWM_MACRO	(1)	(2)		(2)	(2)	(2)	(2)	(2)
ADC Offset Calibration			(1)	(2)	(2)	(2)	(2)	(2)
InstaSPINTM-BLDC Lib				(1)	(2)	(2)	(2)	(2)
SPEED_PR_MACRO				(1)	(2)	(2)	(2)	(2)
PID_MACRO (IDC)					(1)	(2)		(2)
RC_MACRO					(2)			
PID_MACRO (SPD)							(1)	(2)

(1) This module is being tested in this phase
(2) This module is used in this phase

2.3.1 Level 1 Incremental Build

Assuming the load and build steps described in the *DRV830x-HC-C2-KIT How To Run Guide* completed successfully, this section describes the steps for a “minimum” system check-out which confirms operation of system interrupts, some peripheral- and target-independent modules, and one peripheral-dependent module.

1. Open *BLDC_Int-Settings.h*
2. Select level 1 incremental build option by setting the BUILDLEVEL to LEVEL1 (#define BUILDLEVEL LEVEL1).
3. Right-Click on the project name and click Rebuild Project.
4. Once the build is complete, click on the debug button
5. Reset the CPU and restart
6. Enable real-time mode and run.
7. 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:

- **RampDelay (Q0 format):** for changing the ramping time.
- **CmtnPeriodTarget (Q0 format):** for changing the targeted commutation interval.
- **CmtnPeriodSetpt (Q0 format):** for changing the initial startup commutation interval.
- **DfuncStartup:** for changing the PWM duty cycle in per-unit

Explanations of some modules and variables are given below:

- The **RMP3CNTL** module controls the start-up and the initial speed up of the BLDC motor. This module generates a ramp-down function. This ramp-down function of the module allows the speed-up of the BLDC motor from a stand-still in an open-loop configuration, similar to a stepper motor.

  The system variable, *CmtnPeriodTarget*, provides one of the input variables, *DesiredInput*, to the **RMP3CNTL** module. The *DesiredInput* variable determines the final speed at the end of the motor speed-up phase. Initialize the *CmtnPeriodTarget* system variable with an appropriate value, which depends on the type of BLDC motor used.

  The second input to **RMP3CNTL** module is *rmp3_dly*, which the user initializes by using the system variable, *RampDelay*. The *RampDelay* variable determines the rate at which the motor speeds up.

  The output of the **RMP3CNTL** module is *Out*, which provides a variable time period that gradually decreases in time. The Out terminal is initialized by using the system variable, *CmtnPeriodSetpt*, which sets the initial startup speed of the motor.

  The *CmtnPeriodTarget* and *CmtnPeriodSetpt* system variables are both initialized by using the *#defines* for RAMP_END_RATE and RAMP_START_RATE, respectively. These *#defines* are located in *BLDC_Int-Settings.h* and they set the initial and final speed of the startup ramp. The *#defines* allow these quantities to be entered in RPM units.

  The second output of the **RMP3CNTL** module is *Ramp3DoneFlag*, which, when set to 0x7FFF, indicates the end of the ramp-down or the motor speed-up phase.

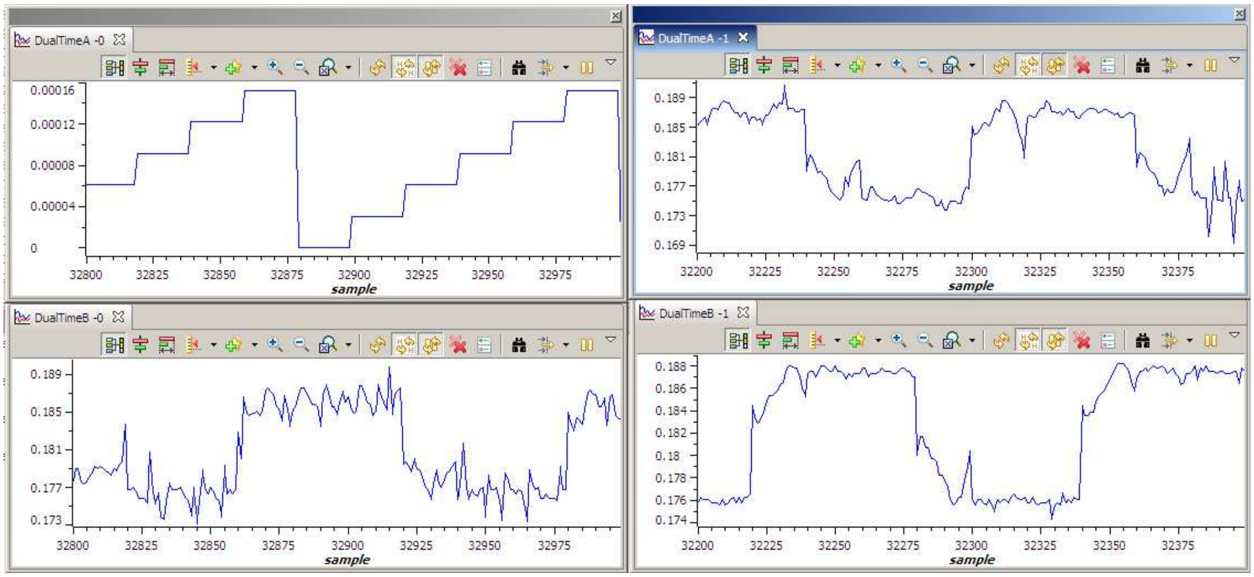
- The *Out* output variable provides the input Period for the **IMPULSE** module. This module generates periodic impulses, with a period specified by its input Period.
- The **DATALOG** module is used to view the output variables of the modules. The initialization required for this is done in the level 1 incremental build initialization routine. During this initialization, one of the inputs of the **DATALOG** module is configured to point to *mod1.Counter*, so the *Out* signal is shown in the CCS graph.
- The periodic impulse output, *Out*, is applied to the input, *TrigInput*, of the **MOD6_CNT** module. The output of the **MOD6_CNT** module is the variable, *Counter*, which can assume one of 6 possible values: 0, 1, 2, 3, 4 or 5. When a trigger pulse is applied to the input, the *Counter* variable changes from one state to the next.

  The *Counter* output is used as the pointer input, *CmtnPointer*, for the **BLDC_3PWM_DRV** module. These 6 values (0-5) of the pointer variable, *CmtnPointer* , are used to generate the 6 commutation states of the power inverter that drives the BLDC motor. During the motor speed-up phase, the input variable, *DfuncStartup*, determines the duty cycle of the generated PWM outputs, according to the 6 commutation states.

The key steps are given below:

1. Compile, load, and run the program with real-time mode
2. Set *EnableFlag* to 1 in the watch window. Initially when **RMP3CNTL** ramps down, Period (the period of *Out*) will also gradually go down. At the end of ramp period (when *Out* equals *DesiredInput*) Period will become constant and Ramp3DoneFlag will set to 0x7FFF.
3. Check the output variable, *Counter*, of the **MOD6_CNT_DIR** module in the watch window and in the graph window. The *Counter* variable will vary between 0 and 5.
4. Use a scope to check the PWM outputs that are controlled by the peripheral-dependent module, **BLDC_3PWM_DRV**.
5. The output states of all the 6 PWM outputs will be such that together they generate the 6 commutation states of the power inverter that drives the BLDC motor.
6. After verifying the PWM output states, disable real-time mode and reset the processor .
7. Terminate the debug session .

While running this level, the Graph windows should look similar to Figure 6.



A mod6 counter

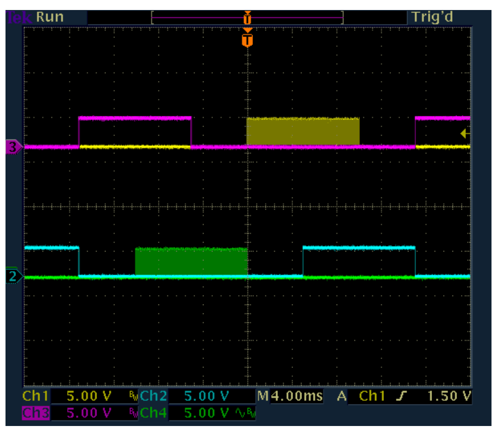
B BemfA

C BemfB

D BemfC

Figure 6. Graph Windows for Build Level 1

While running this level, the PWM outputs should appear as in Figure 7.



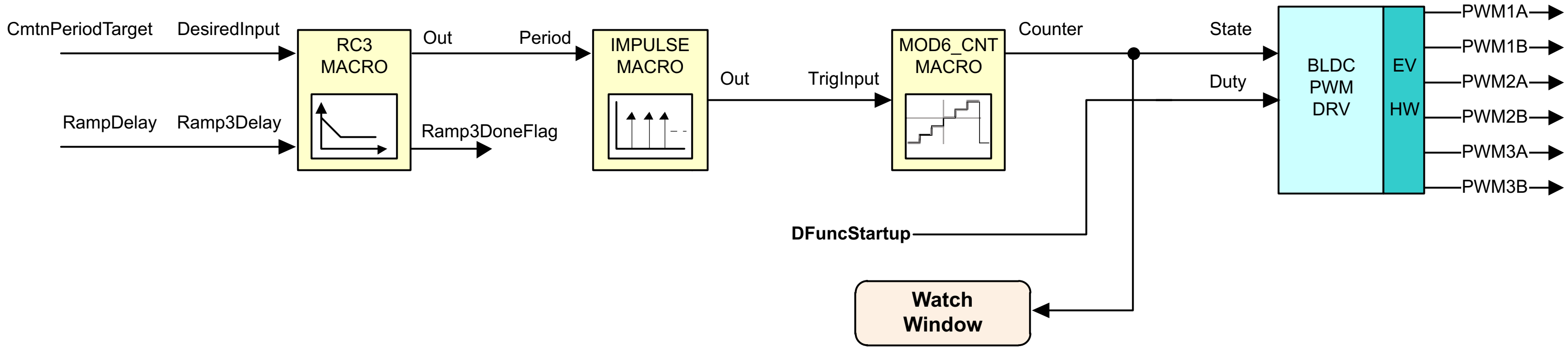
(1) Yellow = PWM 1

(2) Pink = PWM 2

(3) Green = PWM 5

(4) Blue = PWM 6

Figure 7. Level 1 PWM Outputs



(1) Level 1 describes the steps for a “minimum” system check-out which confirms operation of system interrupts, some peripheral- and target-independent modules and one peripheral-dependent module.

Figure 8. Level 1 Incremental System Build Block Diagram

2.3.2 Level 2 Incremental Build

Assuming the previous section is completed successfully, this section verifies the open-loop motor operation and current measurement.

1. Open *BLDC_Int-Settings.h*
2. Select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL LEVEL2).
3. Right-Click on the project name and click Rebuild Project.
4. Once the build is complete, click on debug button
5. Reset CPU and restart
6. Enable real-time mode and run.
7. 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.

- **RampDelay (Q0 format):** for changing the ramping time.
- **CmtnPeriodTarget (Q0 format):** for changing the targeted commutation interval.
- **CmtnPeriodSetpt (Q0 format):** for changing the initial startup commutation interval.
- **DfuncStartup:** for changing the PWM duty cycle in per-unit.

The key steps are explained in the following sections.

2.3.2.1 Open-Loop Test

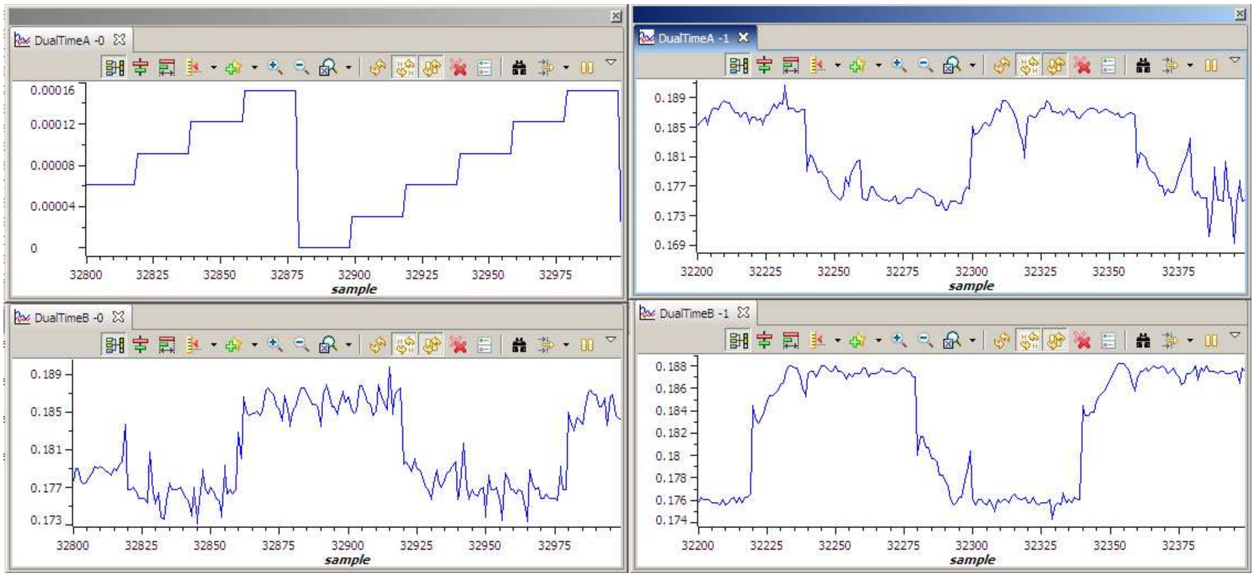
1. Compile, load, and run program with real-time mode enabled. Now, the motor is running with the default *DFuncStartup* value. If the open-loop commutation parameters are chosen properly, the motor will gradually speed up and finally run at a constant speed in open-loop commutation mode.
2. The final speed of the motor will depend on the *CmtnPeriodTarget* parameter. The lower the value for the *CmtnPeriodTarget* variable, the higher the final speed of the motor will be. Since the motor, **Bemf**, depends on its speed, the value chosen for *CmtnPeriodTarget* will also determine the generated **Bemf**.
3. The average applied voltage to the motor during startup will depend on the *DfuncStartup* parameter. The parameters *DfuncStartup* and *CmtnPeriodTarget* should be such that, at the end of the motor speed-up phase, the generated **Bemf** is lower than the average voltage being applied to the motor winding. This will prevent the motor from stalling or vibrating. The motor speed-up time will depend on *RampDelay*, the time period of the main sampling loop, and the difference between *CmtnPeriodTarget* and *CmtnPeriodSetpt*.

   Note: This step is not meant for wide speed and torque range operation; instead, the overall system is tested and calibrated before closing the loops at a certain speed under no load.
4. Bring the system to a safe stop, as described below by setting *EnableFlag* to 0, taking the controller out of real-time mode and reset.

WARNING

After verifying the system is at a safe stop, set *EnableFlag* to 0, take the controller out of real time mode (disable), reset the processor (see *DRV830x-HC-C2-KIT How To Run Guide* **for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, so caution needs to be taken.**

While running this level, the waveforms in the CCS graphs should appear as shown in Figure 9:



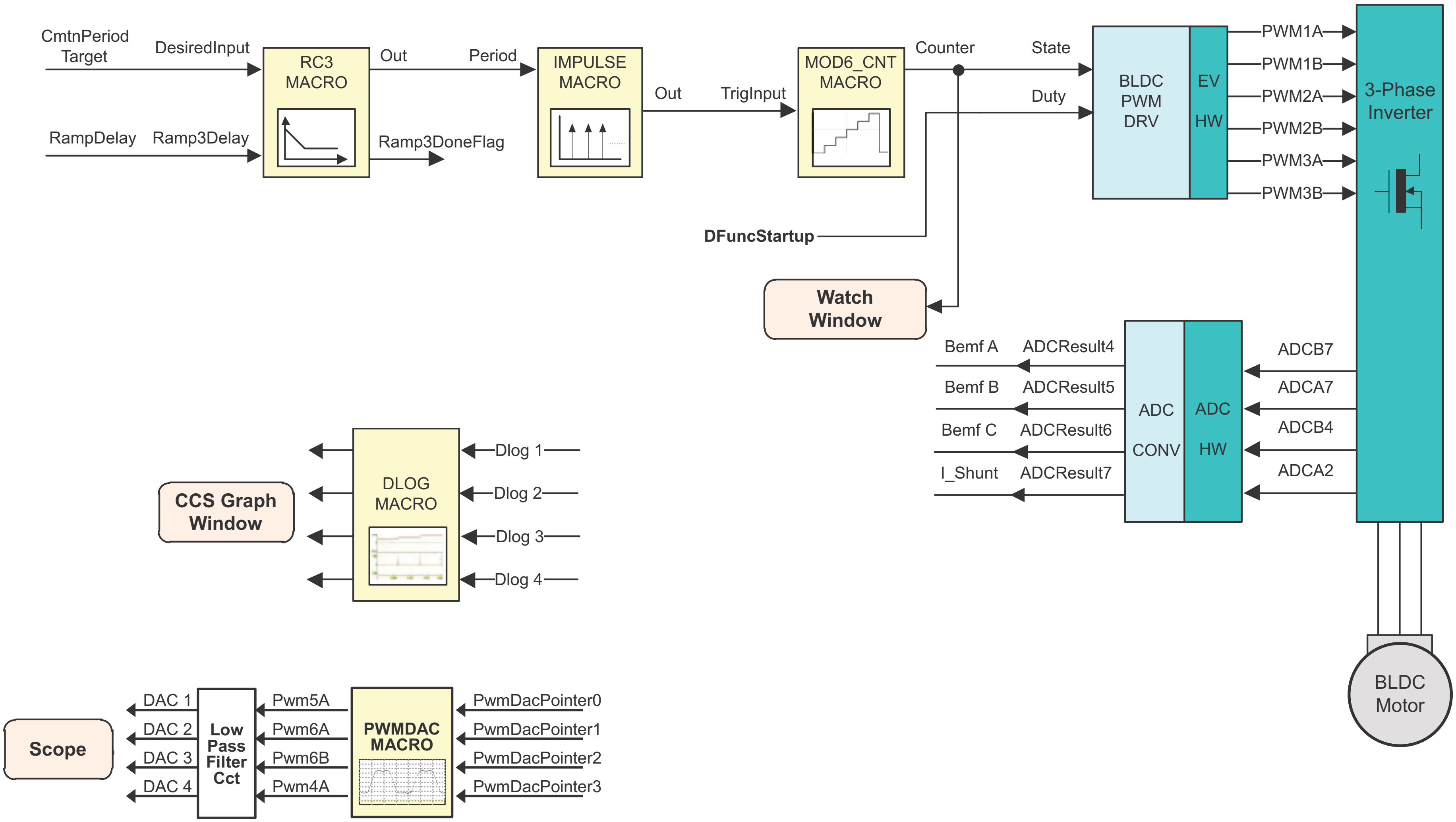
A mod6 counter

B BemfA

C BemfB

D BemfC

Figure 9. Graph Windows for Build Level 2



(1) Level 2 verifies the open loop motor operation and current measurement.

Figure 10. Level 2 Incremental System Build Block Diagram

2.3.3 Level 3 Incremental Build

Assuming the previous section is completed successfully, this section performs automatic calibration of the current sensor offsets.

1. Open *BLDC_Int-Settings.h*
2. Select level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3)
3. Right-click on the project name and click Rebuild Project.
4. Once the build is complete, click on the debug button
5. Reset the CPU and restart
6. Enable real-time mode and run.
7. 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:

- **IDC_offset:** for changing the DC Bus current sensor offset in per-unit.
- **BemfA_offset:** for changing the Phase A BEMF offset.
- **BemfB_offset:** for changing the Phase B BEMF offset
- **BemfC_offset:** for changing the Phase C BEMF offset

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

1. Initialize *IDC_offset* to 0.5 in the code
2. Initialize the three BEMF offsets to 0,0, then recompile and run the system
3. Watch the offset values from the watch window. Ideally the measured phase current offsets should be 0.5 and the BEMF offsets should be 0.0. Note the value of the offsets in the watch window and change their values in the code by going to the code snippet below:

   ```
   _iq BemfA_offset = _IQ15(0.0);
   _iq BemfB_offset = _IQ15(0.0);
   _iq BemfC_offset = _IQ15(0.0);
   _iq IDC_offset = _IQ15(0.5000);
   ```

4. Change the `IQ15(0.5000)` offset value (e.g. `IQ15(0.5087)` or `IQ15(0.4988)`, depending on the value observed in the watch window). Try to enter an offset with 4 significant digits. These offset values will now be used for the remaining build levels.

   Note: Piccolo devices have 12- and 16-bit ADC registers. The **AdcResult.ADCRESULT** registers are right-justified for Piccolo devices, so the measured phase-current value is first 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.

5. Bring the system to a safe stop, by setting *EnableFlag* to 0, taking the controller out of real-time mode, and reset.

2.3.4 Level 4 Incremental Build

Assuming the previous section completed successfully, this section verifies the peripheral independent InstaSPINTM-BLDC library functions.

1. Open *BLDC_Int-Settings.h*
2. Select level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4).
3. Right-click on the project name and click Rebuild Project.
4. Once the build is complete click on debug button
5. Reset the CPU and restart
6. Enable real-time mode and run.
7. 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:

- **RampDelay (Q0 format):** for changing the ramping time.
- **CmtnPeriodTarget (Q0 format):** for changing the targeted commutation interval.
- **CmtnPeriodSetpt (Q0 format):** for changing the initial startup commutation interval.
- **DfuncStartup:** for changing the PWM duty cycle in per-unit.

The key steps can be explained as follows:

- Compile, load, and run program in real-time mode. Now, the motor will gradually speed up and finally run at a constant speed in open-loop commutation mode with the default *DFuncTesting* value.
- View the **MOD6_CNT_DIR** output as well as the **InstaSPINTM-BLDC** output variables (*Sense*, *Vphase* and *V_int*) from either the graphs window or the scope.

  The *Sense* variable indicates which of the three motor phases is inactive.

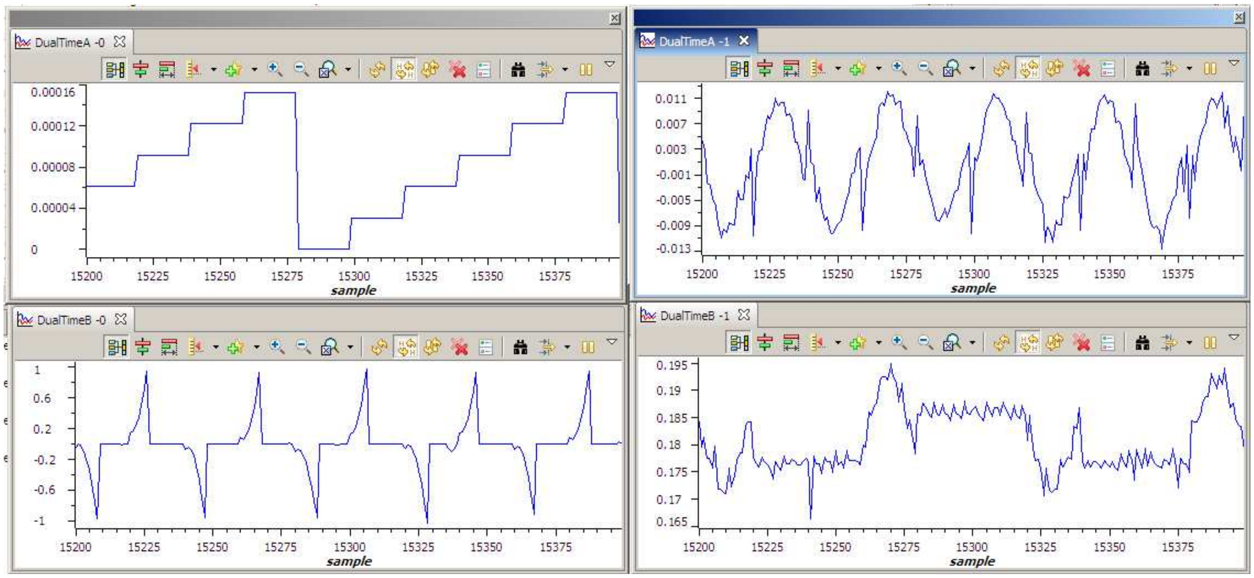
  The BEMF of this phase is pointed to by the *Vphase* variable.

  The *V_int* variable shows the integrated BEMF that used for commutation in the next build-levels.
- Bring the system to a safe stop as described below by setting *EnableFlag* to 0, taking the controller out of real-time mode, and reset.

WARNING

After verifying the system is at a safe stop, set *EnableFlag* to 0, take the controller out of real time mode (disable), reset the processor (see *DRV830x-HC-C2-KIT How To Run Guide* **for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, so caution needs to be taken.**

During this level, the graph waveforms should look similar to Figure 11.



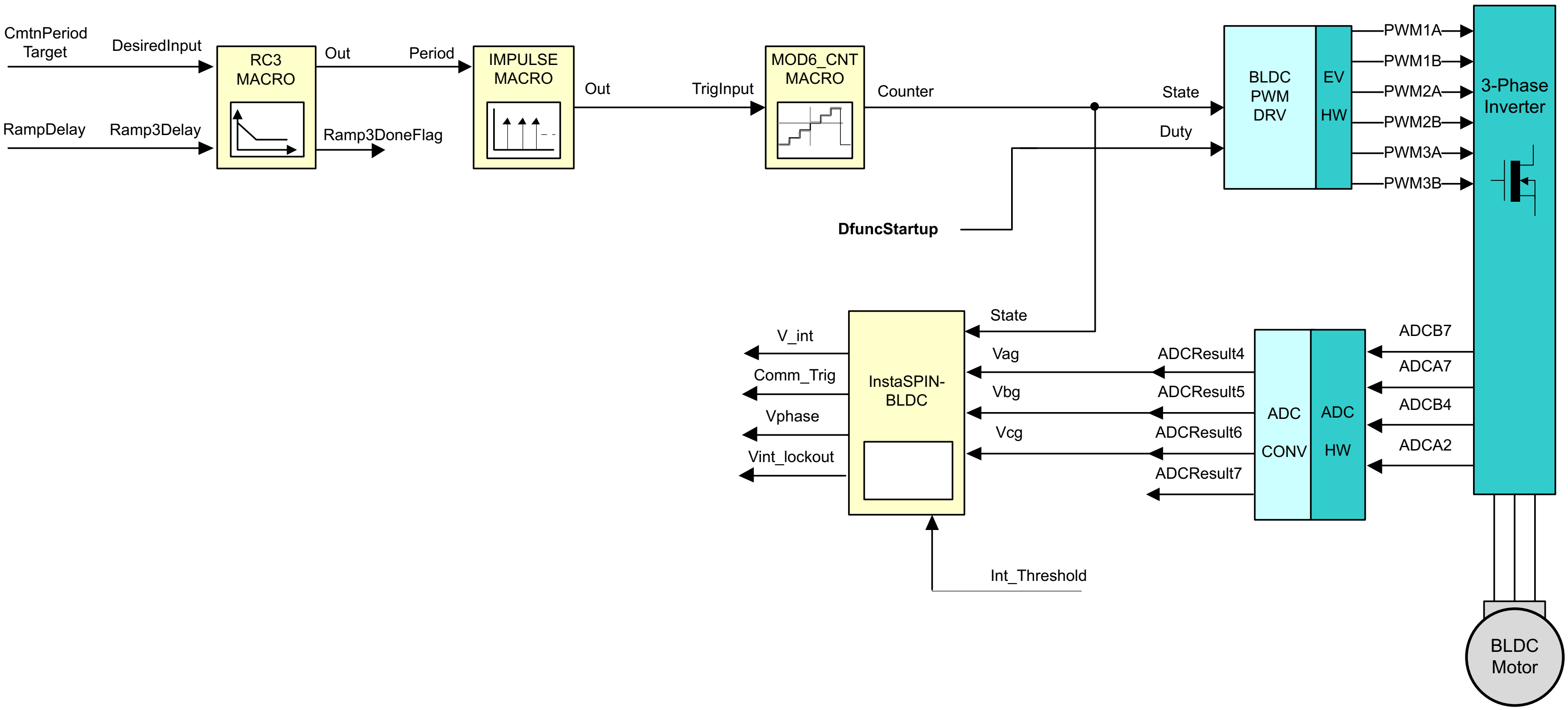
A mod6 counter

B V_int

C Vphase

D Vag

Figure 11. Graph Windows for Build Level 4



(1) Level 4 verifies the peripheral independent InstaSPIN-BLDC library functions

Figure 12. Level 4 Incremental System Build Block Diagram

2.3.5 Level 5 Incremental Build

Assuming the previous section is completed successfully, this section verifies the sensorless motor commutation based on **InstaSPINTM-BLDC**.

1. Open *BLDC_Int-Settings.h*
2. Select level 5 incremental build option by setting the BUILDLEVEL to LEVEL5 (#define BUILDLEVEL LEVEL5)
3. Save the file.
4. Right-Click on the project name and click Rebuild Project.
5. Once the build is complete click on debug button
6. Reset the CPU and restart
7. Enable real-time mode and run.
8. 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 is working properly.

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

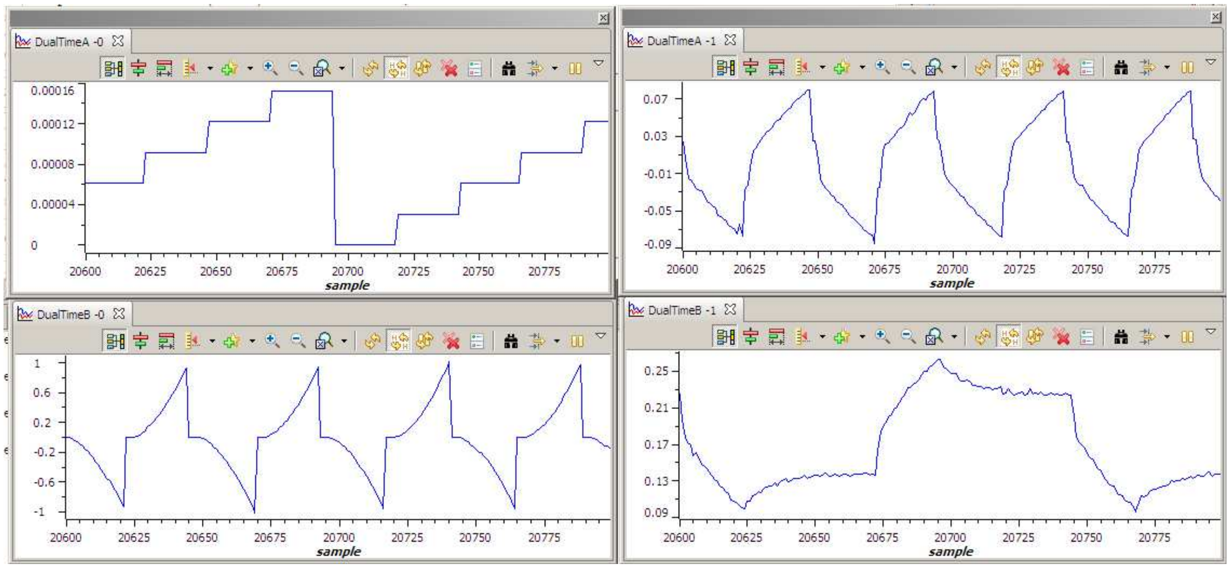
- **RampDelay (Q0 format):** for changing the ramping time.
- **CmtnPeriodTarget (Q0 format):** for changing the targeted commutation interval.
- **CmtnPeriodSetpt (Q0 format):** for changing the initial startup commutation interval.
- **DfuncStartup:** for changing the startup PWM duty cycle in per-unit.
- **DFuncTesting:** changing the PWM duty function in per-unit.
- **InstaSPIN_BLDC1.Int_Threshold:** for changing the BEMF integration threshold in per-unit

The key steps can be explained as follows:

- Compile, load, and run the program in real-time mode.
- The motor will gradually speed up and finally enter closed-loop commutation mode.
- The motor enters closed-loop commutation mode from open-loop commutation mode when *Ramp3DoneFlag* is set to 0x7FFFFFFF, which indicates the end of the motor speed-up phase. Until this switch between commutation modes occurs, the **MOD6_CNT** module is triggered by the output of the **IMPULSE** module. After the switch between modes, the **MOD6_CNT** module is triggered by the output of the **InstaSPINTM-BLDC** module.
- When the speed-up phase is over, change the motor speed by changing the value of *DFuncTesting*. This varies the power that is delivered to the motor, which also alters the motor speed.
- Adjust *InstaSPIN_BLDC1.Int_Threshold* to achieve the desired commutation.
- Bring the system to a safe stop as described below by setting *EnableFlag* to 0, taking the controller out of real-time mode and reset.

WARNING

After verifying the system is at a safe stop, set *EnableFlag* to 0, take the controller out of real time mode (disable), reset the processor (see *DRV830x-HC-C2-KIT How To Run Guide* for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, so caution needs to be taken.



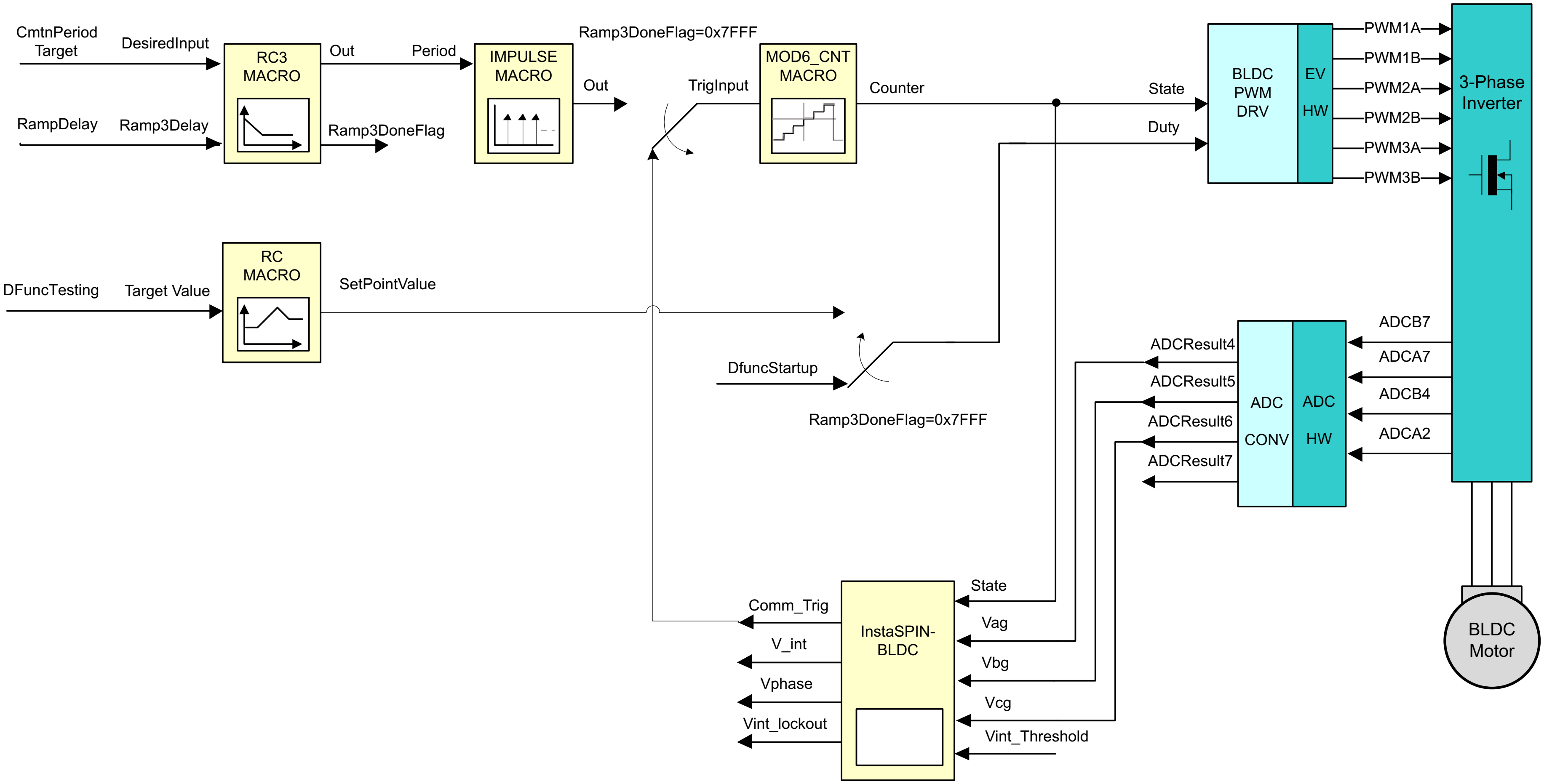
A mod6 counter

B V_int

C Vphase

D Vag

Figure 13. Graph Windows for Build Level 5



(1) Level 5 verifies the closed-loop motor operation, based on instaSPIN-BLDC and the resulting commutation trigger points.

Figure 14. Level 5 Incremental System Build Block Diagram

2.3.6 Level 6 Incremental Build

Assuming the previous section is completed successfully, this section verifies the closed current loop and current PI controller.

1. Open *BLDC_Int-Settings.h*
2. Select level 6 incremental build option by setting the BUILDLEVEL to LEVEL6 (#define BUILDLEVEL LEVEL6).
3. Right-Click on the project name and click Rebuild Project.
4. Once the build is complete click on debug button
5. Reset the CPU and restart
6. Enable real-time mode and run.
7. 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.

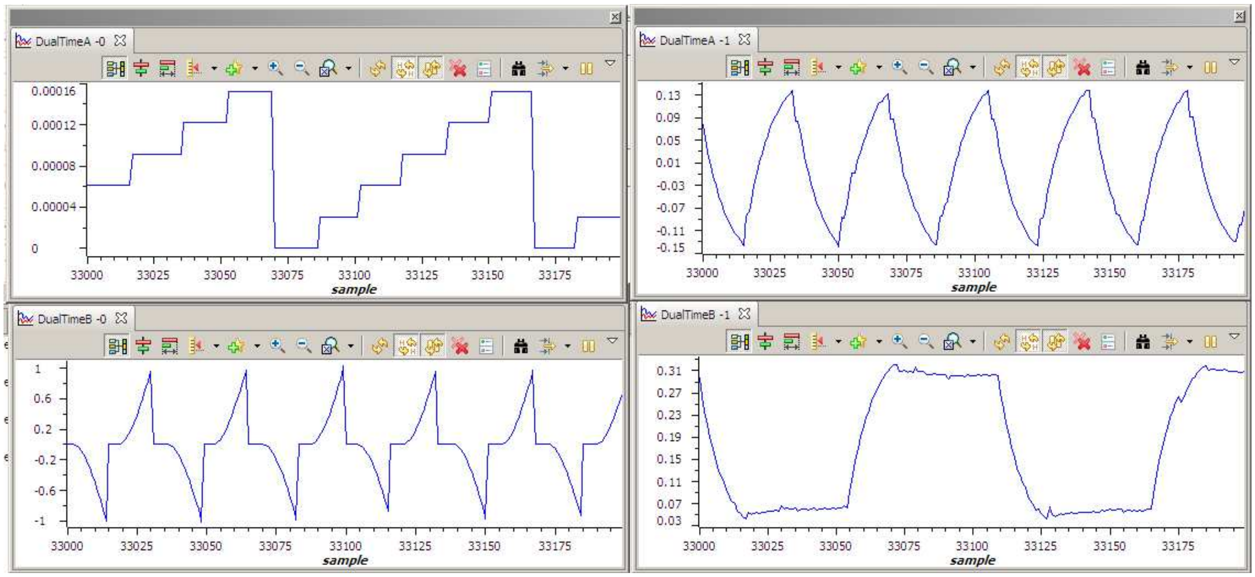
In the software, the key variables to be adjusted are summarized below.

- **RampDelay (Q0 format):** for changing the ramping time.
- **CmtnPeriodTarget (Q0 format):** for changing the targeted commutation interval.
- **CmtnPeriodSetpt (Q0 format):** for changing the initial startup commutation interval.
- **CurrentStartup:** for changing the startup current in per-unit.
- **IRef:** changing the running current in per-unit.
- **InstaSPIN_BLDC1.Int_Threshold:** for changing the BEMF integration threshold in per-unit

The steps are explained as follows:

1. Compile, load, and run program with real-time mode.
2. The motor will gradually speed up and finally enter closed-loop commutation mode.
3. Now use the variable, *IRef*, to specify the reference current for the PI controller. The PI controller begins to regulate the DC bus current, and thereby also regulating the motor current. Gradually increase or decrease the command current (the *IRef* value) to change the torque command and to adjust the PI gains. Note that the speed is not controlled in this step and that a non-zero torque reference will keep increasing the motor speed. Therefore, the motor should be loaded using a brake or generator (or manually if the motor is small enough) after closing the loop. Initially, apply a relatively light load and then gradually increase the amount of the load. If the applied load is higher than the torque reference, the motor can not handle the load and stops immediately after closing the current loop.
4. Verify that the motor speed (both pu and rpm) calculated by *SPEED_PR* is correct by viewing the following variables in the Watch Window.
   - *speed1.Speed* (pu)
   - *speed1.SpeedRpm (rpm)*
5. Bring the system to a safe stop as described below by setting *EnableFlag* to 0, taking the controller out of real-time mode and reset.

> **WARNING**
>
> **After verifying the system is at a safe stop, set *EnableFlag* to 0, take the controller out of real time mode (disable), reset the processor (see *DRV830x-HC-C2-KIT How To Run Guide* for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, so caution needs to be taken.**



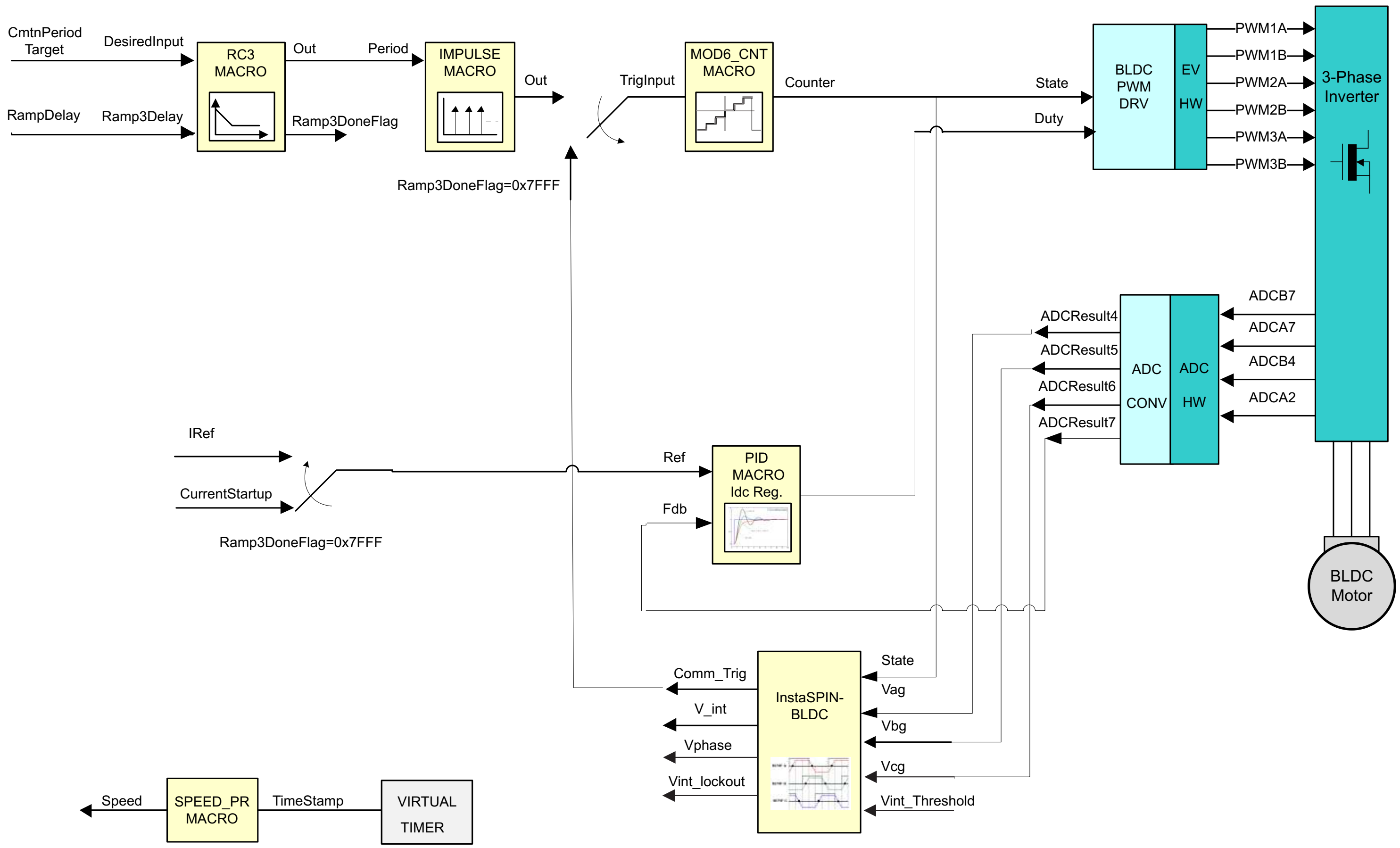
A mod6 counter

B V_int

C Vphase

D Vag

Figure 15. Graph Windows for Build Level 6



(1) Level 6 verifies the closed-current loop and current PI controller.

Figure 16. Level 6 Incremental System Build Block Diagram

2.3.7 Level 7 Incremental Build

Assuming the previous section is completed successfully, this section verifies the closed-speed loop and the speed PI controller.

1. Open *BLDC_Int-Settings.h*
2. Select level 7 incremental build option by setting the BUILDLEVEL to LEVEL7 (#define BUILDLEVEL LEVEL7).
3. Right-Click on the project name and click Rebuild Project.
4. Once the build is complete click on debug button
5. Reset the CPU and restart
6. Enable real-time mode and run.
7. 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.

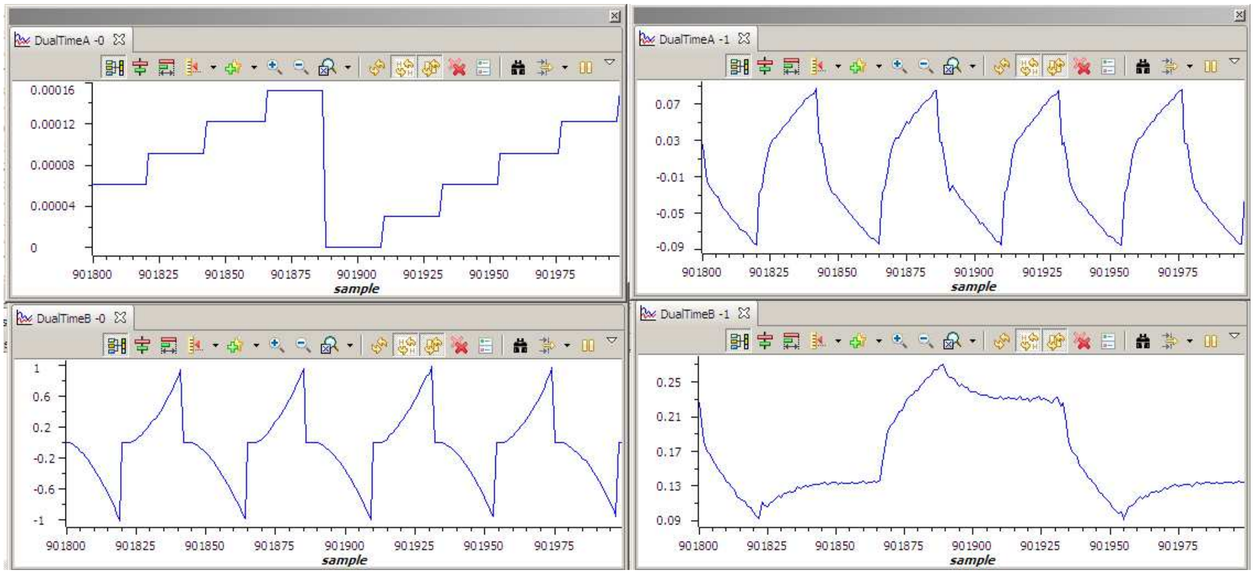
- **SpeedRef (GLOBAL_Q format):** for changing the reference Speed in per-unit

The steps are explained as follows:

- Compile, load, and run the program in real-time mode.
- The motor will gradually speed up and finally enter closed-loop commutation mode.
- Once in closed-loop commutation mode, use the variable, *SpeedRef*, to specify the reference speed for the PI controller, PID_REG3. The *SpeedLoopFlag* is automatically activated when the PI reference is ramped up from zero speed to SpeedRef. Once this is done, the PI controller begins regulating the motor speed. Gradually increase the command speed (the *SpeedRef* 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 *EnableFlag* to 0, taking the controller out of real-time mode and reset.

WARNING

After verifying the system is at a safe stop, set *EnableFlag* to 0, take the controller out of real time mode (disable), reset the processor (see *DRV830x-HC-C2-KIT How To Run Guide* for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, so caution needs to be taken.



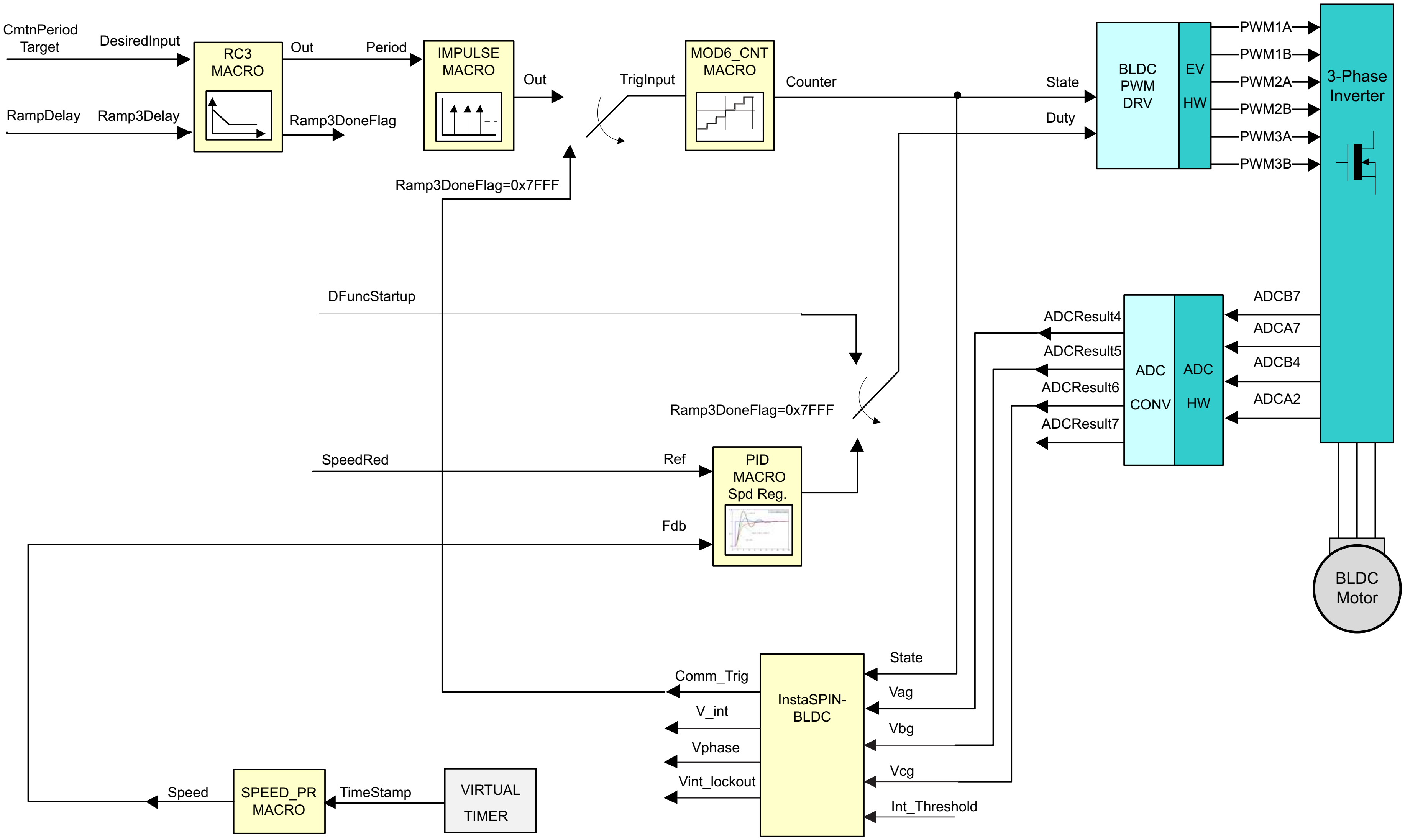
A mod6 counter

B V_int

C Vphase

D Vag

Figure 17. Graph Windows for Build Level 7



(1) Level 7 verifies the closed-speed loop and speed PI controller

Figure 18. Level 7 Incremental System Build Block Diagram

2.3.8 Level 8 Incremental Build

Assuming the previous section is completed successfully, this section verifies the cascaded closed speed and current loops.

1. Open *BLDC_Int-Settings.h*
2. Select level 8 incremental build option by setting the BUILDLEVEL to LEVEL8 (#define BUILDLEVEL LEVEL8).
3. Right-click on the project name and click Rebuild Project.
4. Once the build is complete, click on debug button
5. Reset the CPU and restart
6. Enable real-time mode and run.
7. 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:

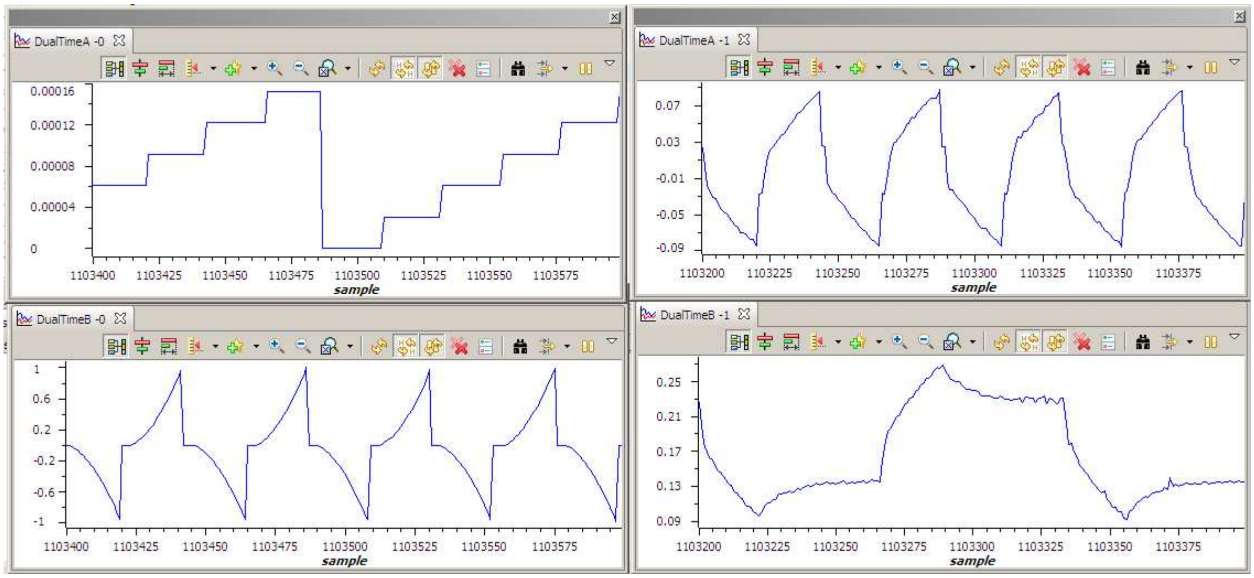
- **SpeedRef (GLOBAL_Q format):** for changing the reference Speed in per-unit.

The steps are explained as follows:

- Compile, load, and run the program in real-time mode.
- The motor will gradually speed up and finally enter closed-loop commutation mode.
- Once in closed-loop commutation mode, use the variable, *SpeedRef*, to specify the reference speed for the PI controller, PID_REG3. The *SpeedLoopFlag* is automatically activated when the PI reference is ramped up from zero speed to *SpeedRef*. Once this is done, the PI controller begins regulating the motor speed. Gradually increase the command speed (the *SpeedRef* 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 *EnableFlag* to 0, taking the controller out of real-time mode and reset.

WARNING

After verifying the system is at a safe stop, set *EnableFlag* to 0, take the controller out of real time mode (disable), reset the processor (see *DRV830x-HC-C2-KIT How To Run Guide* for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, so caution needs to be taken.



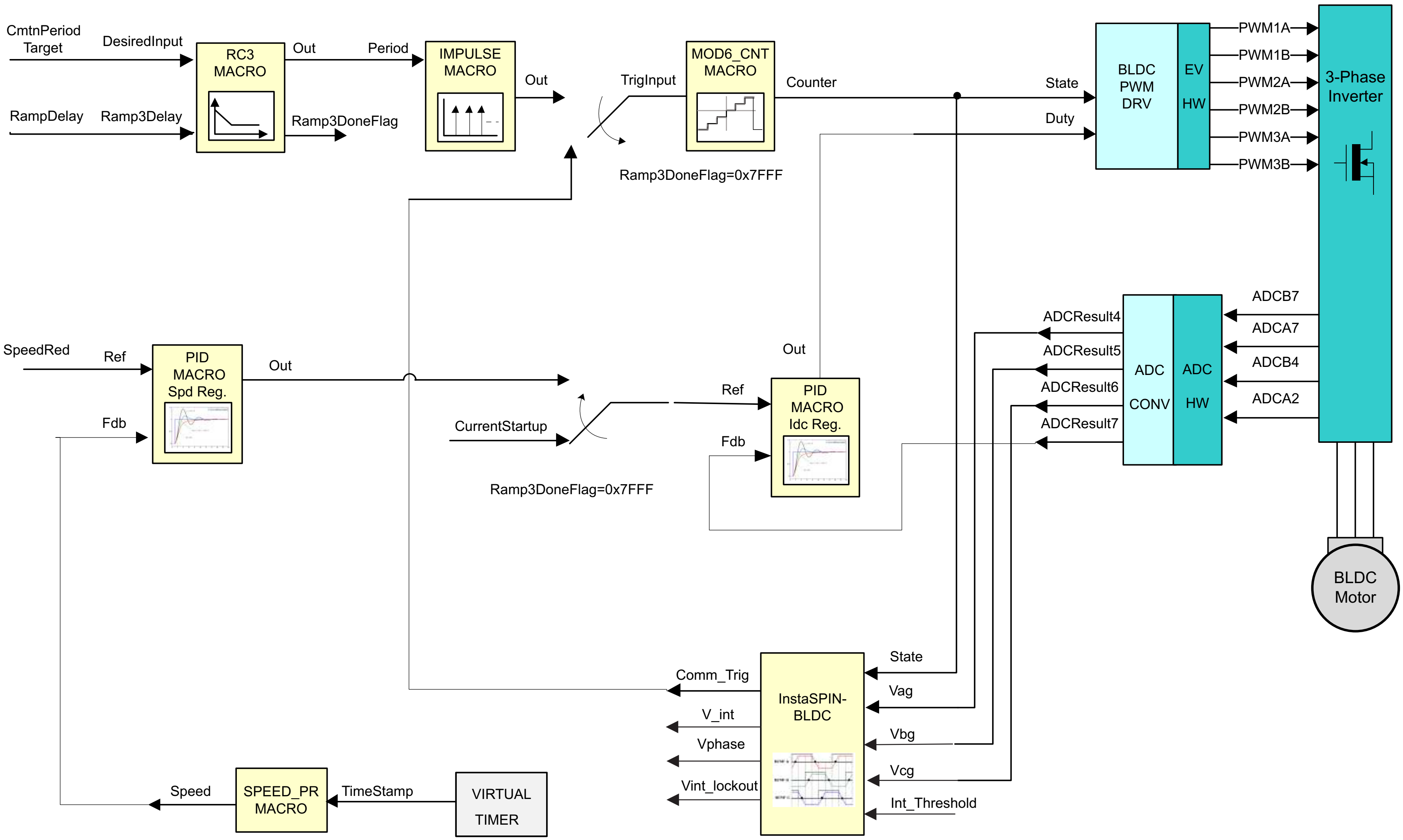
A mod6 counter

B V_int

C Vphase

D Vag

Figure 19. Graph Windows for Build Level 8



(1) Level 8 verifies the cascaded, closed-speed and closed-current loops.

Figure 20. Level 8 Incremental System Build Block Diagram

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265