Sensorless Trapezoidal Control of BLDC Motors using BEMF Integration (InstaSPIN^{TM}-BLDC)

Jon Warriner D3 Engineering

Abstract

This application note presents a solution for sensorless control of Brushless DC motors using the TMS320F2803x microcontrollers. TMS320F280x devices are part of the family of C2000 microcontrollers which enable cost-effective design of intelligent controllers for three phase motors by reducing the system components and increasing efficiency. 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

Table of Contents

Version 1.0 – Aug 2011 - SPRABZ1

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 TMS320C2803x based controllers on Code Composer Studio. The framework uses the following modules[1]:

Macro Names	**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 Module
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 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 DRV8312 Three Phase PWM Motor Driver. The TMS320F2803x 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 (Ifb Ret) is measured and sent to the TMS320x2803x via analog-to-digital converters (ADCs).

InstaSPIN_BLDC project has the following properties:

C Framework		
System Name	**Program Memory Usage 2803x**	**Data Memory Usage[1] 2803x**
InstaSPIN_BLDC	4597 words[2]	1200 words

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

CPU Utilization of Trapezoidal BLDC Control (Sensorless)	
Name of Modules *	**Number of Cycles**
BLDCPWM	105
InstaSPIN^{TM}-BLDC Library	277
PID	91
RC	29
RC3	26 √
SPEED_PR	42
IMPULSE	17 √
MOD6_CNT_DIR	9
Contxt Save, Virtual Timer etc.	153
Pwm Dac (optional)	
DataLog (optional)	
Total Number of Cycles	**749 ****
CPU Utilization @ 60 Mhz	**25%**
CPU Utilization @ 40 Mhz	**37.4%**

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

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 for DAC outputs ADC A2 for low side DC bus return current sensing, B7, A7 and B4 for Bemf sensing

The overall system implementing a 3-ph sensorless BLDC control is depicted in Figure 1

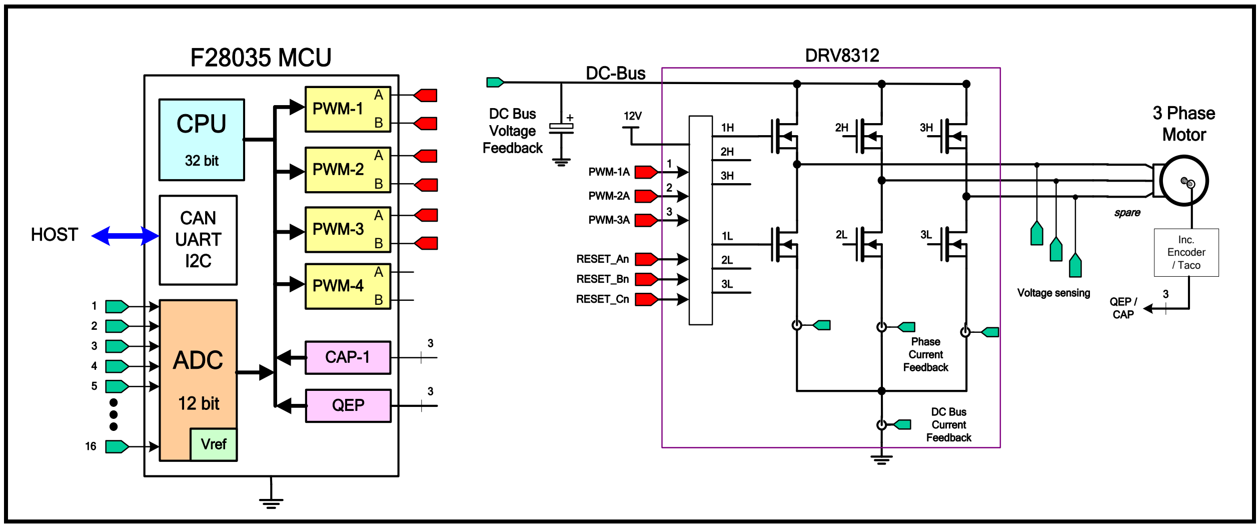


Figure 1 A 3-ph BLDC drive implementation

The software flow is described in the Figure 2 below.

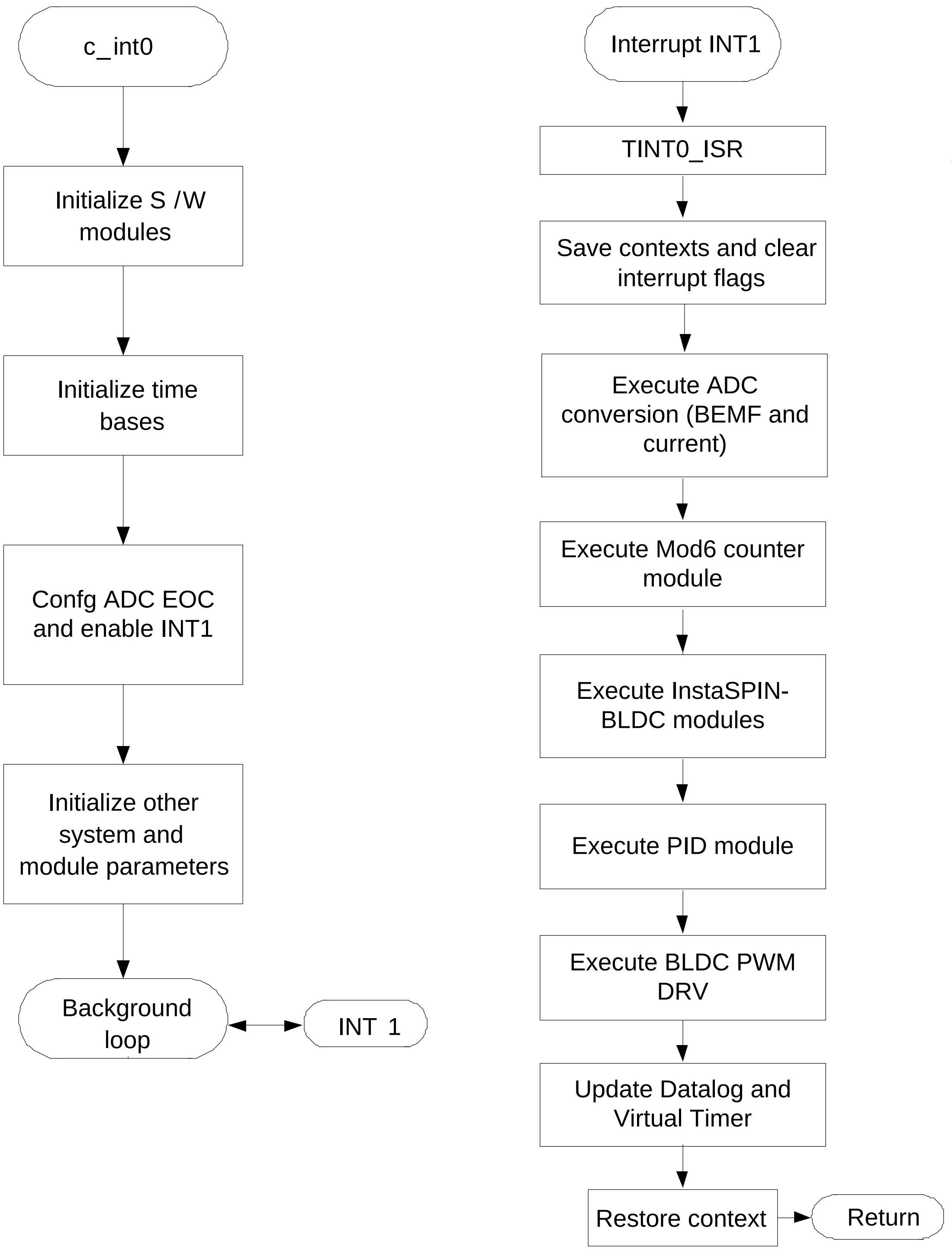

c_int0
Initialize S /W modules
Initialize time bases
Confg ADC EOC and enable INT1
Initialize other system and module parameters
Background loop
INT 1
Interrupt INT1
TINT0_ISR
Save contexts and clear interrupt flags
Execute ADC conversion (BEMF and current)
Execute Mod6 counter module
Execute InstaSPIN-BLDC modules
Execute PID module
Execute BLDC PWM DRV
Update Datalog and Virtual Timer
Restore context
Return

Figure 2 Software Flowchart

Hardware Configuration (DRV8312-C2-KIT)

Please refer to the DRV8312 How to Run Guide and HW Reference Guide found:

C:\TI\controlSUITE\development_kits\DRV8312-C2-KIT_v*\~Docs

for an overview of the kit's hardware and steps on how to setup this kit. Some of the hardware setup instructions are captured below for quick reference.

HW Setup Instructions

1. Unpack the DIMM style controlCARD and verify that the DIP switch settings match Figure 13

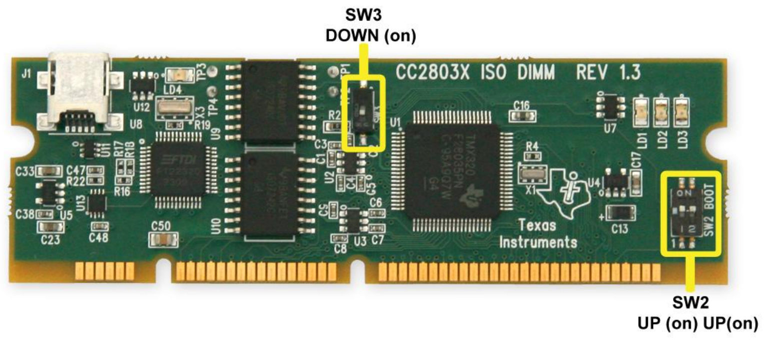


Figure 33 controlCARD DIP Switch Settings

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. Make sure DRV8312 mode jumpers and +12V source jumper are set according to **Error! Reference source not found.**.
4. Connect a USB cable to connector J1 on the controlCARD. This will enable isolated JTAG emulation to the C2000 device. LD4 should turn on. If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will automatically be installed. If a windows installation window appears try to automatically install drivers from those already on your computer. The emulation drivers are found at http://www.ftdichip.com/Drivers/D2XX.htm. The correct driver is the one listed to support the FT2232.
5. Connect a 24V power supply to J9 of the DRV8312-EVM. Now LED1, LED2 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.
6. Note that the motor should be connected to the MOA, MOB and MOC terminals after you finish with the first incremental build step. For more details on motor wiring please refer to the datasheet provided with your motor.

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

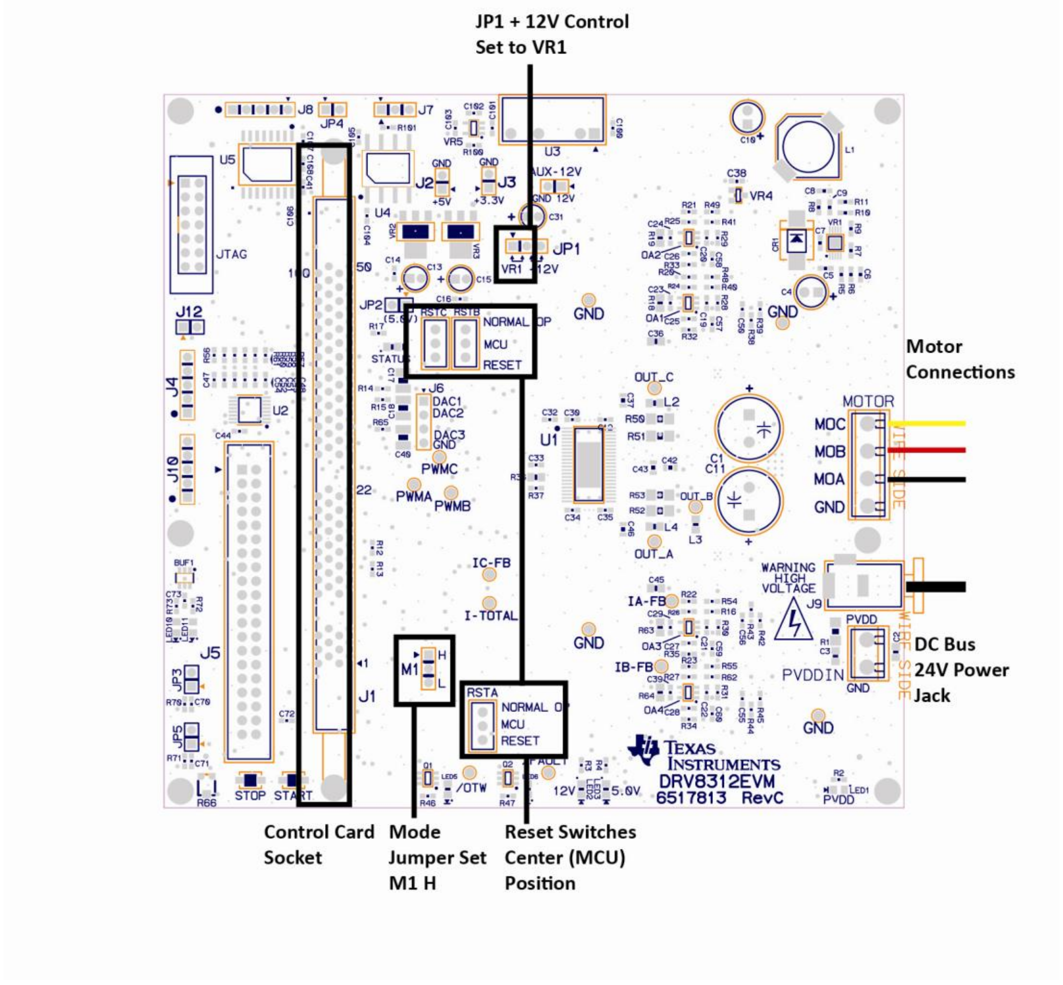


Fig. 14 DRV8312-EVM Connections and Settings

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

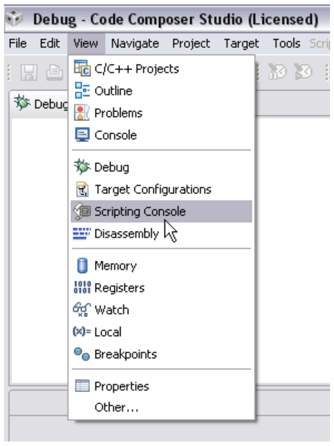
Software Setup Instructions to Run InstaSPIN_BLDC Project

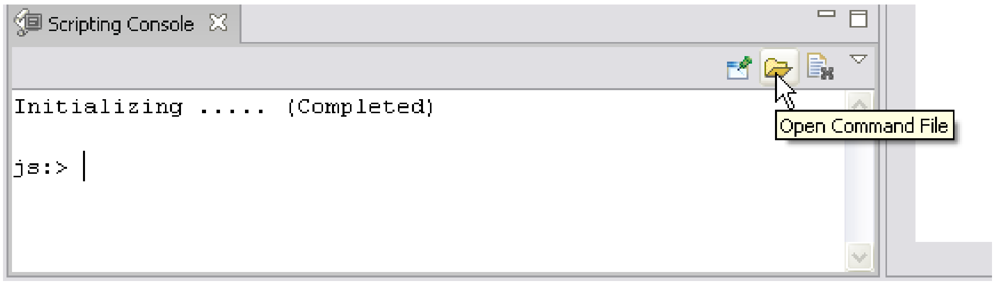
Please refer to the “Generic Steps for Software Setup for DRV8312-C2-KIT Projects” section in the DRV8312-C2-KIT How To Run Guide

C:\TI\controlSUITE\development_kits\DRV8312-C2-KIT_v*\~Docs

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

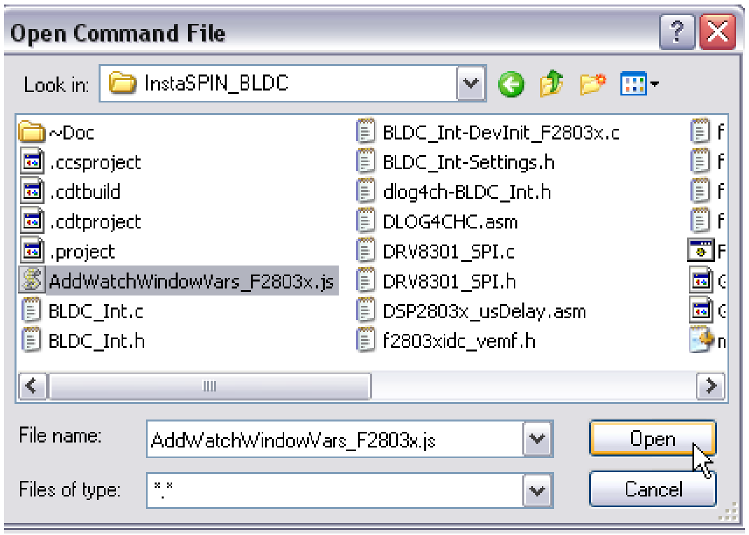
Select the InstaSPIN_BLDC as the active project. Verify that the build level is set to 1, and then right click on the project name and select “Rebuild Project”. Once build completes, launch a debug session to load the code into the controller. Now add the variables to a watch window by using the scripting tool:

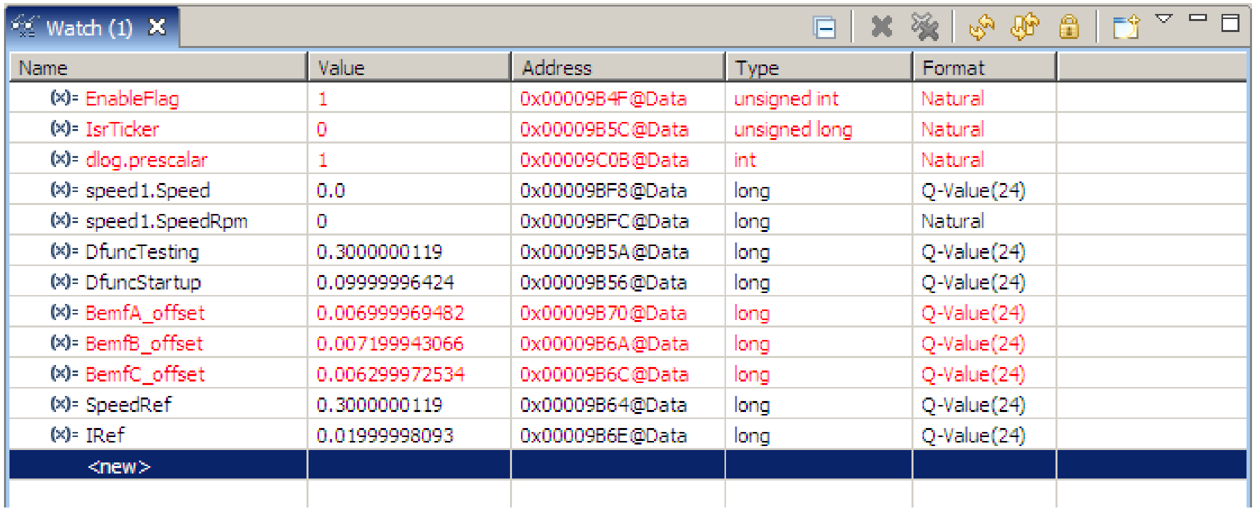




```
Initializing ..... (Completed)

js:>
```





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 4 Watch Window Setup

Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location C:\TI\ControlSUITE\development_kits\DRV8312-C2-KIT_v*\InstaSPIN_BLDC Click on Continuous Refresh button on the top left corner of the graph tab to enable periodic capture of data from the microcontroller.

Incremental System Build for InstaSPIN^{TM}-BLDC project

The system is gradually built up in order so that the final system can be confidently operated. Eight phases of the incremental system build are designed to verify the major software modules used in the system. The table below summarizes the modules testing and using in each incremental system build.

Testing modules in each incremental system build								
Software Module	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**	**Phase 5**	**Phase 6**	**Phase 7**	**Phase 8**
PWMDAC_MACRO	√√	√√		√	√	√	√	√
RC3_MACRO	√√	√		√	√	√	√	√
MOD6_CNT_DIR_MACRO	√√	√		√	√	√	√	√
IMPULSE_MACRO	√√	√		√	√	√	√	√
BLDCPWM_MACRO	√√	√		√	√	√	√	√
ADC Offset Calibration			√√	√	√	√	√	√
InstaSPIN^{TM}-BLDC Lib				√√	√	√	√	√
SPEED_PR_MACRO				√√	√	√	√	√
PID_MACRO (IDC)					√√	√		√
RC_MACRO					√			
PID_MACRO (SPD)							√√	√
Note: the symbol √ means this module is using and the symbol √√ means this module is testing in this phase.								

Level 1 Incremental Build

Assuming the load and build steps described in the “DRV8312-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 & target independent modules and one peripheral dependent module. Open BLDC_Int-Settings.h and select level 1 incremental build option by setting the BUILDLEVEL to LEVEL1 (#define BUILDLEVEL LEVEL1). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will 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 explanations and steps are given as follows:

- The start-up and the initial speed up of the BLDC motor is controlled by the RMP3CNTL module. This module generates a ramp down function. This ramp down feature of RMP3CNTL module allows speed up of the BLDC motor from stand still in an open loop configuration (like a stepper motor).
- One of the inputs to RMP3CNTL module, DesiredInput, determines the final speed at the end of the motor speed up phase. This input is provided from the system using the system variable CmtnPeriodTarget. User initializes this system variable with appropriate value depending on the type of the BLDC motor. The second input to RMP3CNTL module is rmp3_dly, which is also user initialized by using the system variable RampDelay. This determines the rate at which the motor speeds up. The output of RMP3CNTL module is Out, which provides a variable time period gradually decreasing in time. The Out terminal is initialized by the system variable CmtnPeriodSetpt which sets the initial startup speed of the motor. CmtnPeriodTarget and CmtnPeriodSetpt are both initialized by the #defnes for RAMP_END_RATE and RAMP_START_RATE respectively. These #defines are located in BLDC_Int-Settings.h and set the initial and final speed of the startup ramp. The #defines allow these quantities to be entered in units of RPM. The second output of RMP3CNTL module is Ramp3DoneFlag, which, when set to 0x7FFF, indicates the end of the ramp down (or motor speed up) phase.
- Out is used to provide the input Period for the IMPULSE module. This module generates periodic impulses with period specified by its input Period.
- The DATALOG module is used to view the output variables of the modules. The initialization required to perform this, is done in the level 1 incremental build initialization routine. During this initialization, one of the inputs of DATALOG module is configured to point to mod1.Counter. Thus Out signal is shown in the graph in CCS.
- The periodic impulse output, Out, is applied to the input TrigInput of the MOD6_CNT module. The output of this module is Counter, which can assume one of the 6 possible values 0, 1, 2, 3, 4 or 5. This output changes from one state to the next when a trigger pulse is applied to the input. This Counter is finally used as the pointer input, CmtnPointer, for the module BLDC_3PWM_DRV. These 6 values of the pointer variable, CmtnPointer , are used to generate the 6 commutation states of the power inverter driving the BLDC motor. The duty cycle of the generated PWM outputs (according to the 6 commutation states) during the motor speed up phase are determined by the input DfuncStartup.

- Now, compile/load/run program with real time mode and 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.
- Check MOD6_CNT_DIR output variable Counter in the watch window and graph window. This will vary between 0 and 5.
- Use a scope to check the PWM outputs controlled by the peripheral dependent module BLDC_3PWM_DRV.
- The output states of all the 6 PWM outputs will be such that together they generate the 6 commutation states of the power inverter driving the BLDC motor.
- After verifying this take the controller out of real time mode (disable), reset the processor and then terminate the debug session.

While running this level, the Graph windows should look similar to Figure 5

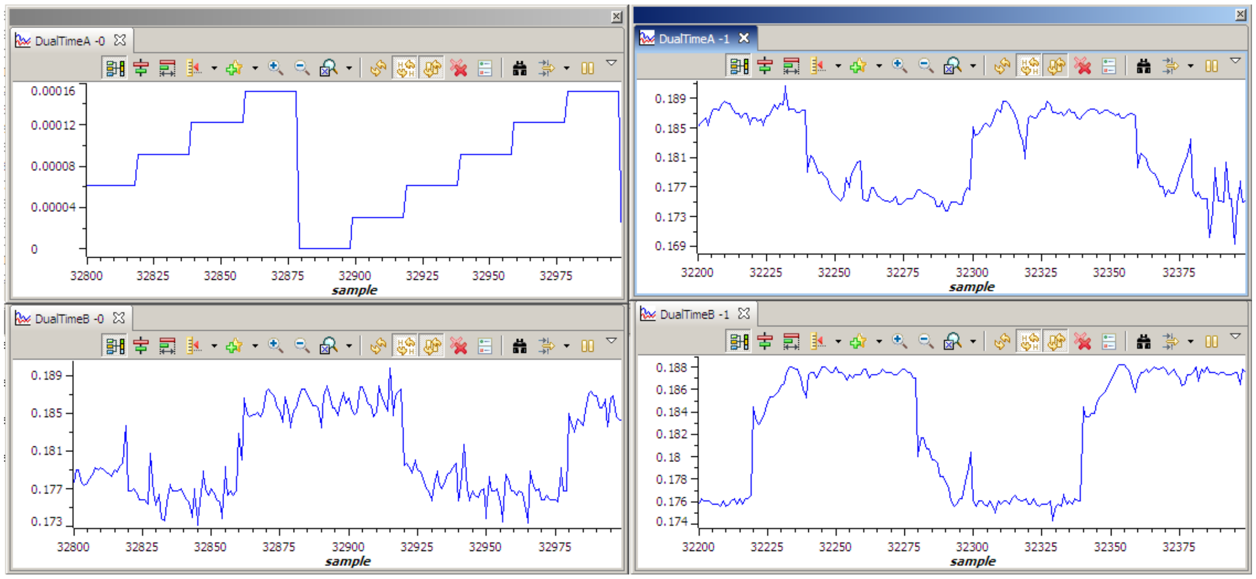


Figure 5 Graph Windows for Build Level 1 (a) mod6 counter, (b) BemfA, (c) BemfB and (d)BemfC

While running this level, the PWM outputs should appear as in Figure 6

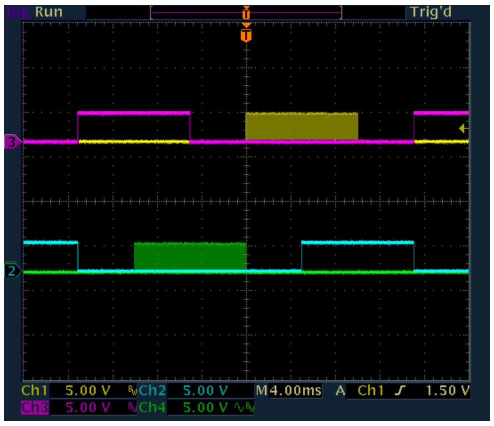
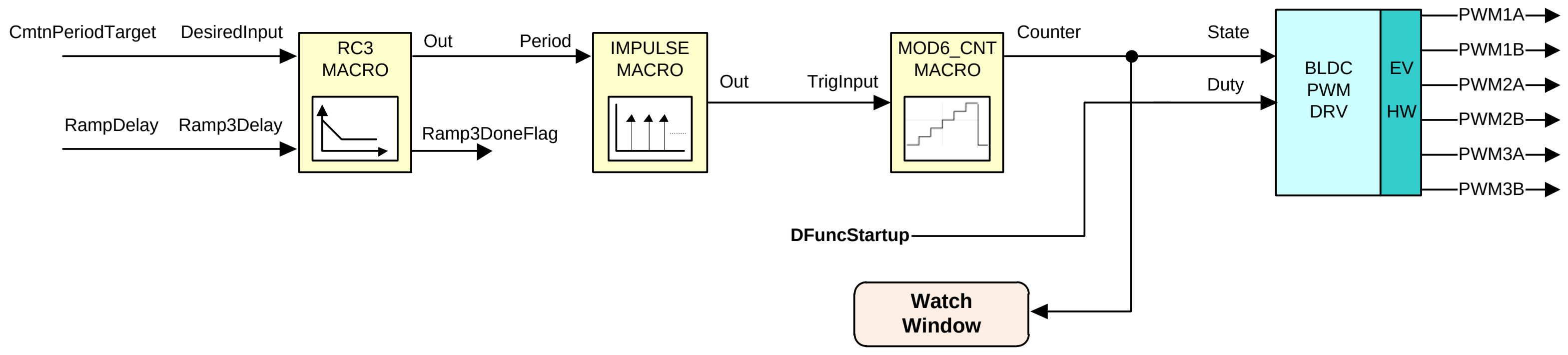


Figure 6 The PWM outputs , PWM 1 (Yellow) , PWM 2 (Pink) and PWM 5 (Green), PWM 6 (Blue)

Level 1 Incremental System Build Block Diagram



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

Level 2 Incremental Build

Assuming the previous section is completed successfully, this section verifies the open loop motor operation and current measurement. Open BLDC_Int-Settings.h and select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL LEVEL2). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will 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:

Open Loop Test

- Compile/load/run program with real time mode. Now the motor is running with default DFuncStartup value.

- If the open loop commutation parameters are chosen properly then the motor will gradually speed up and finally run at a constant speed in open loop commutation mode.

- The final speed of the motor will depend on the parameter CmtnPeriodTarget. The lower the value for this variable the higher will be the motor final speed. Since the motor Bemf depends on its speed, the value chosen for CmtnPeriodTarget will also determine the generated Bemf.

- The average applied voltage to the motor during startup will depend on the parameter DfuncStartup. The parameters DfuncStartup and CmtnPeriodTarget should be such that, at the end of motor speed up phase, the generated Bemf is lower than the average voltage applied to 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.

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

After verifying this, set EnableFlag to 0, take the controller out of real time mode (disable), reset the processor (see "DRV8312-C2-KIT How To Run Guide" for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, hence caution needs to be taken.

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

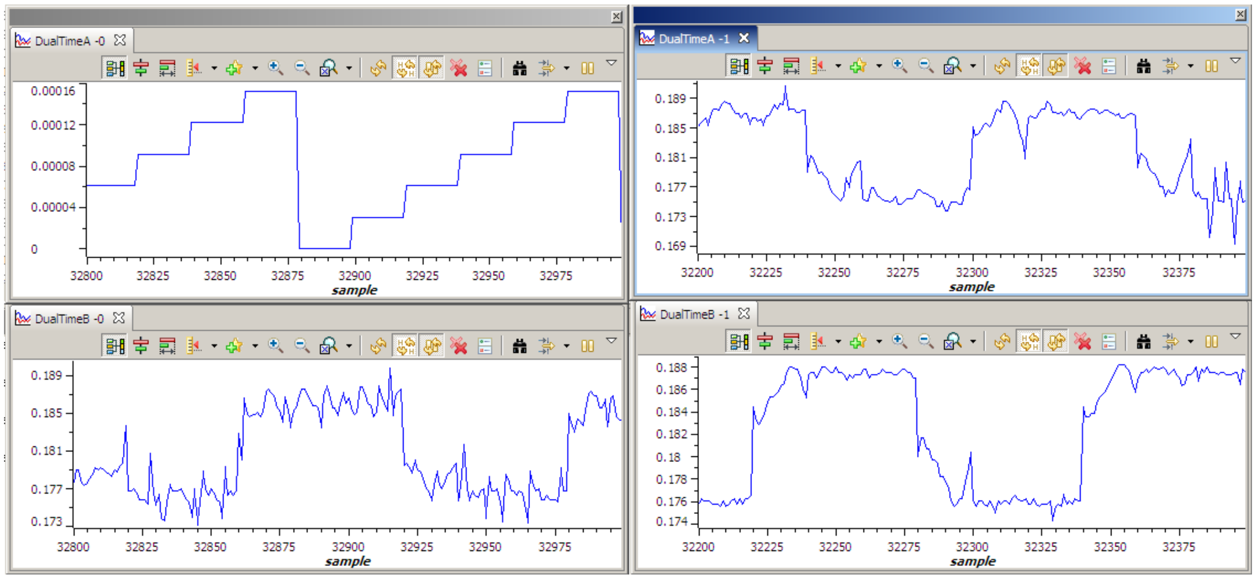
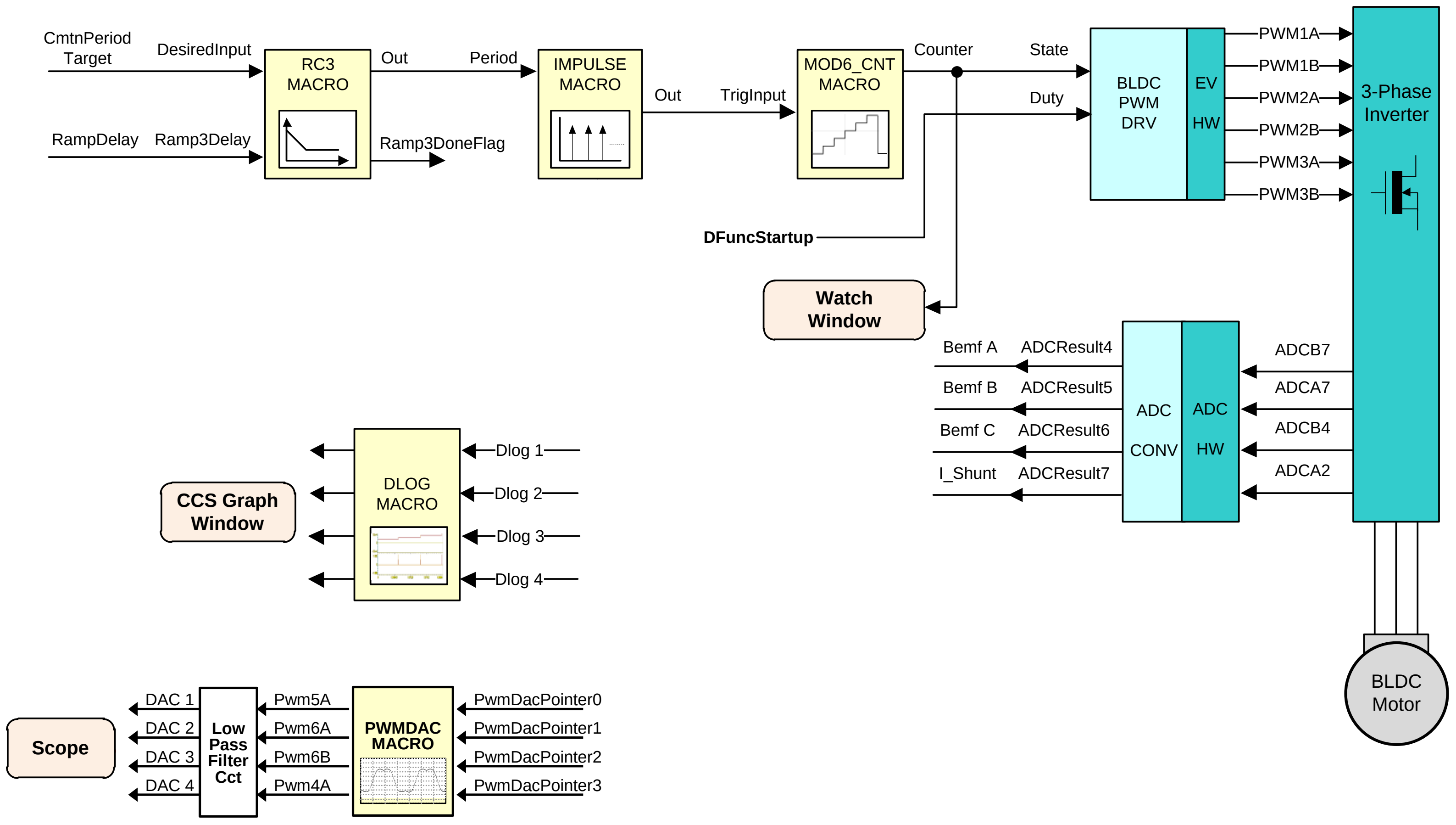


Figure 7 Graph Windows for Build Level 2 (a) mod6 counter, (b) BemfA, (c) BemfB and (d)BemfC

Level 2 Incremental System Build Block Diagram



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

Level 3 Incremental Build

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

Open BLDC_Int-Settings.h and select level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3). Now right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will now keep on increasing, confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

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

- 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 the low power motors draw 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 automatically calculated by passing the measured currents through a low-pass filter to obtain the average value when zero current is flowing through the sensors.

Initialize IDC_offset to 0.5 in the code and initialize the three BEMF offsets to 0,0, recompile and run the system and 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 Ithe offsets in the watch window and change their values in the code by going to:

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

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

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

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

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

Level 4 Incremental Build

Assuming the previous section completed successfully, this section verifies the peripheral independent InstaSPIN^{TM}-BLDC library functions. Open BLDC_Int-Settings.h and select level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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/run program with real time mode. Now the motor will gradually speed up and finally run at a constant speed in open loop commutation mode with default DFuncTesting value.

- View the MOD6_CNT_DIR output as well as the InstaSPIN^{TM}-BLDC output variables Sense, Vphase and V_int from either from graphs window or scope.

- The Sense variable will indicate which of the three motor phases inactive. This BEMF of this phase is pointed to by the Vphase variable. The V_int variable shows the integrated BEMF that will be 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 realtime mode and reset.

During this level, the graph waveforms should look similar to Figure 8

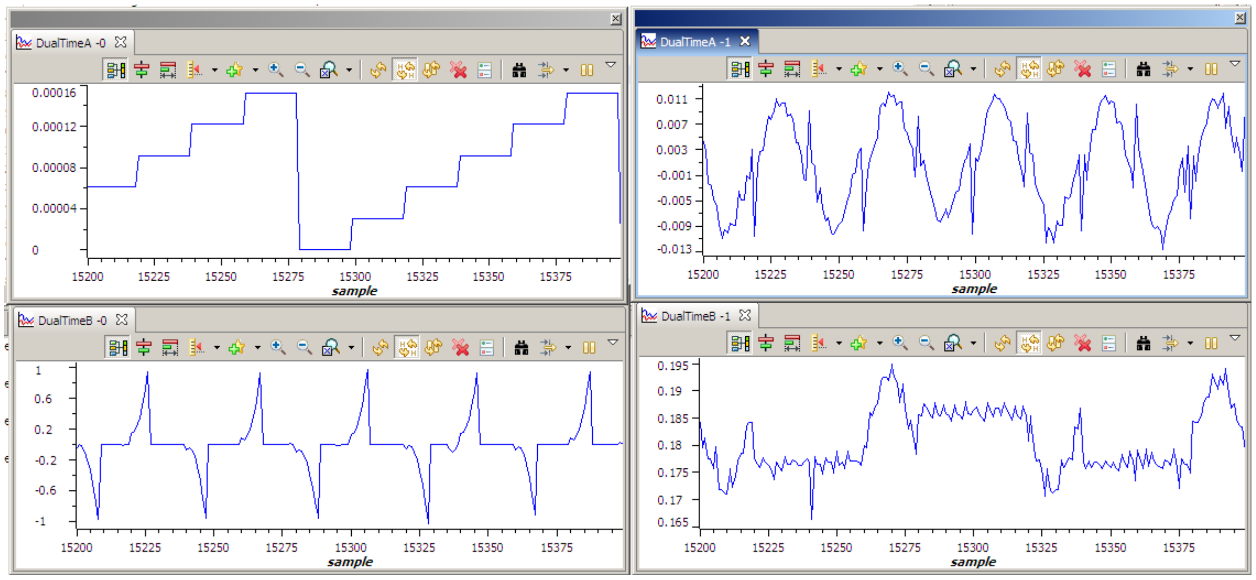
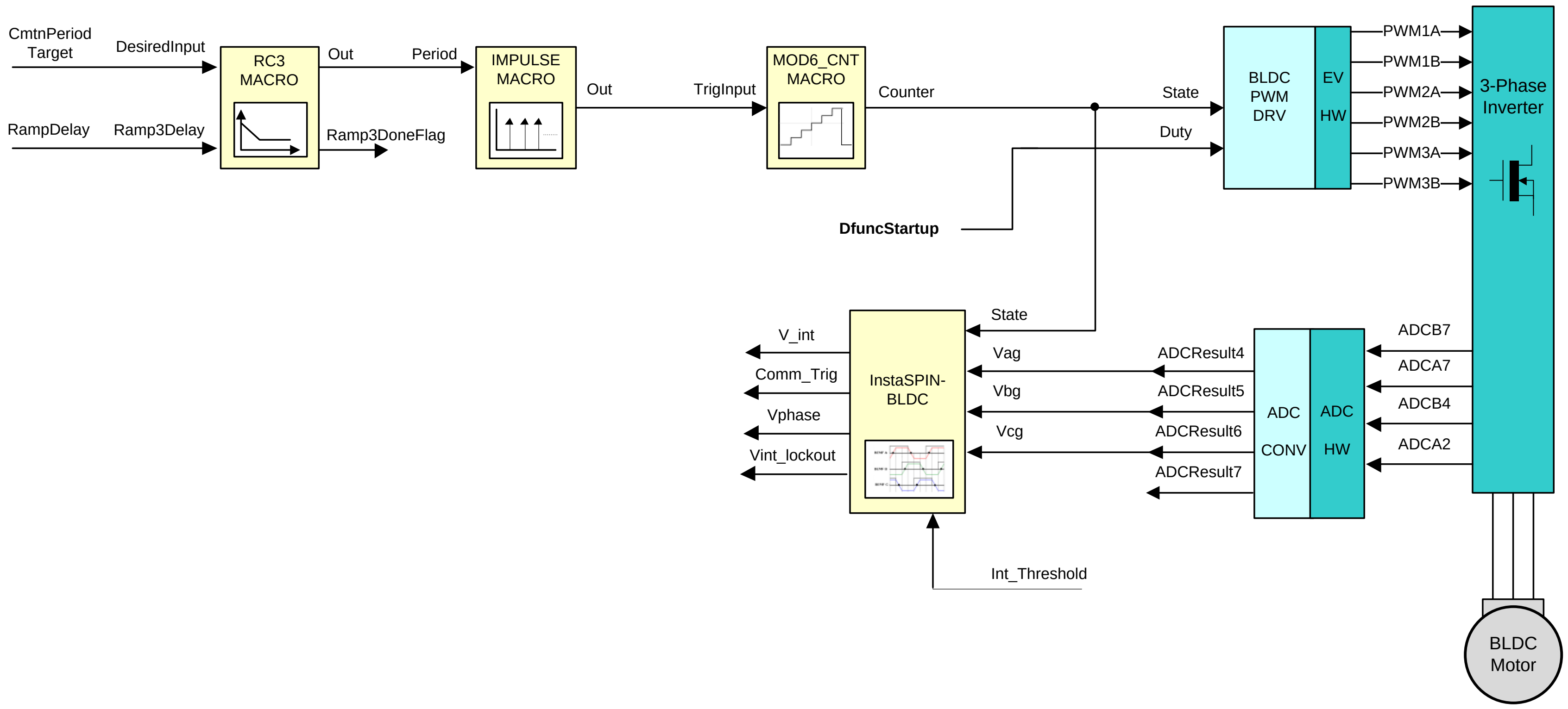


Figure 8 Graph Windows for Build Level 4 (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag

Level 4 Incremental System Build Block Diagram



Level 4 verifies the peripheral independent InstaSPIN-BLDC library functions

Level 5 Incremental Build

Assuming the previous section is completed successfully, this section verifies the sensorless motor commutation based on InstaSPIN^{TM}-BLDC. Open BLDC_Int-Settings.h and select level 5 incremental build option by setting the BUILDLEVEL to LEVEL5 (#define BUILDLEVEL LEVEL5) and save the file. Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

- 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/run program with real time mode.
- 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 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^{TM}-BLDC module.
- When the speed up phase is over, vary the motor speed by changing DFuncTesting. This varies the power delivered to the motor and hence it’s 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 realtime mode and reset.

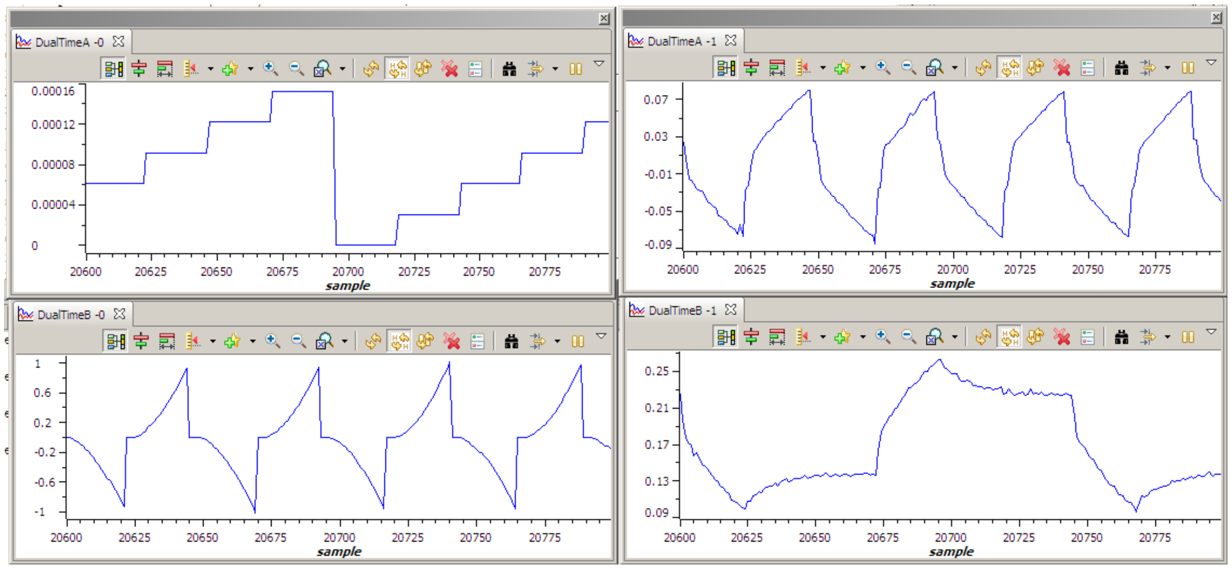
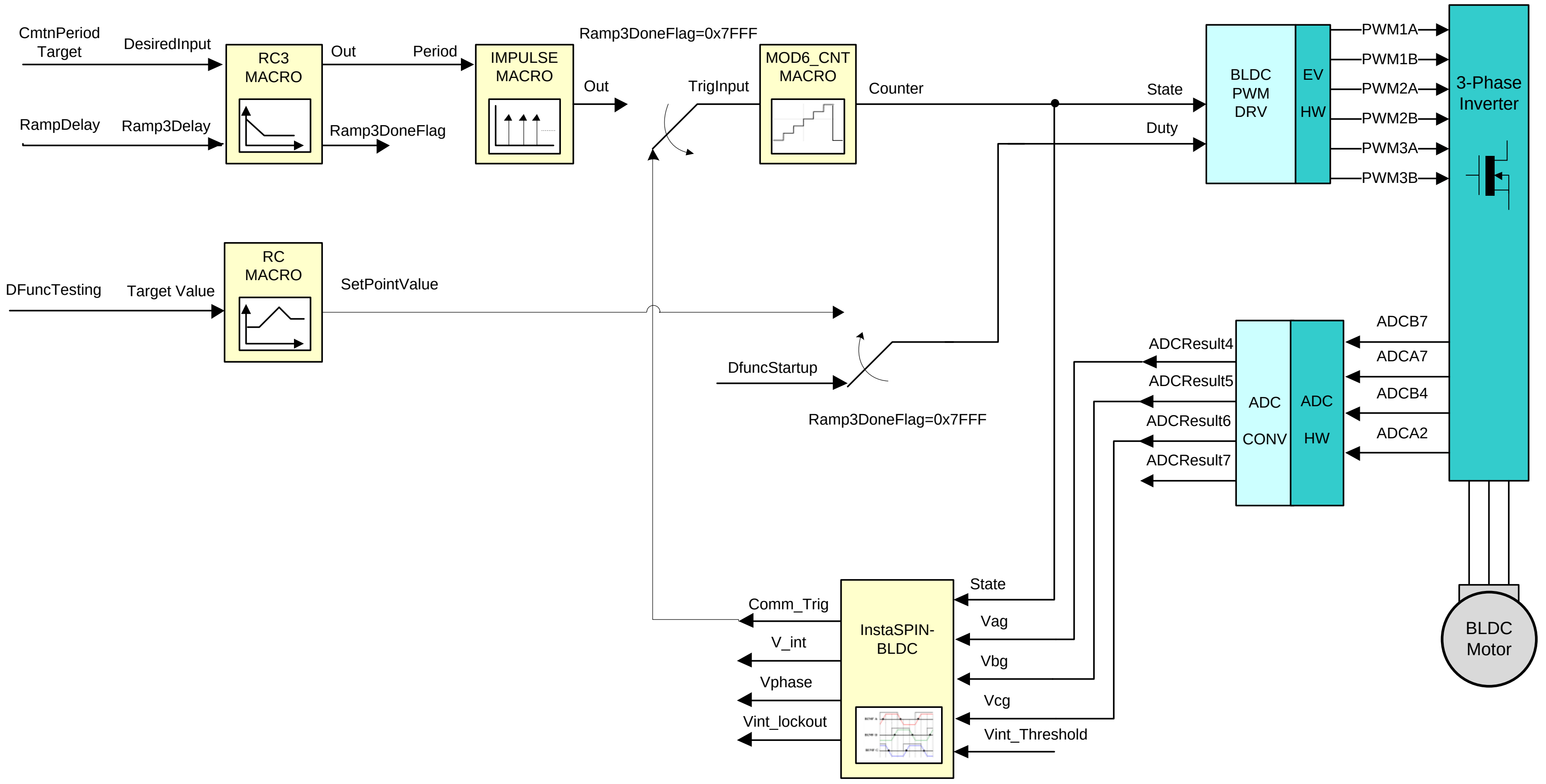


Figure 9 Graph Windows for Build Level 5 (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag

Level 5 Incremental System Build Block Diagram



Level 5 verifies the closed loop motor operation based on InstaSPIN-BLDC and the resulting commutation trigger points.

Level 6 Incremental Build

Assuming the previous section is completed successfully, this section verifies the closed current loop and current PI controller. Open BLDC_Int-Settings.h and select level 6 incremental build option by setting the BUILDLEVEL to LEVEL6 (#define BUILDLEVEL LEVEL6). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will 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:

- 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 IRef 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 (IRef 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. View the following variables in the Watch Window.

    - speed1.Speed (pu)

    - speed1.SpeedRpm (rpm)

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

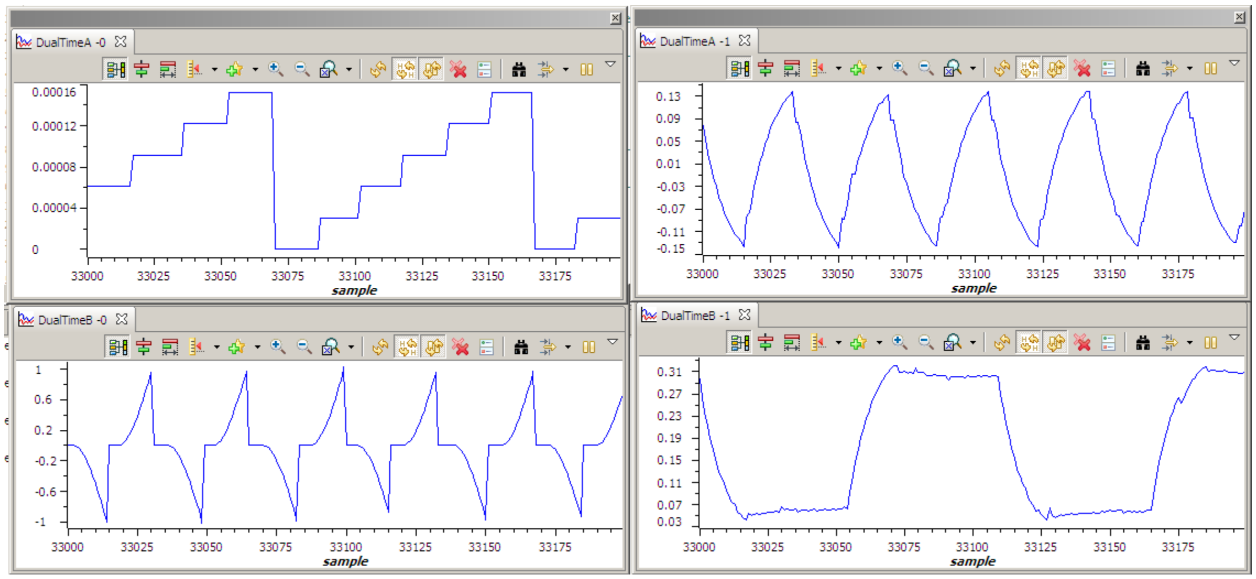
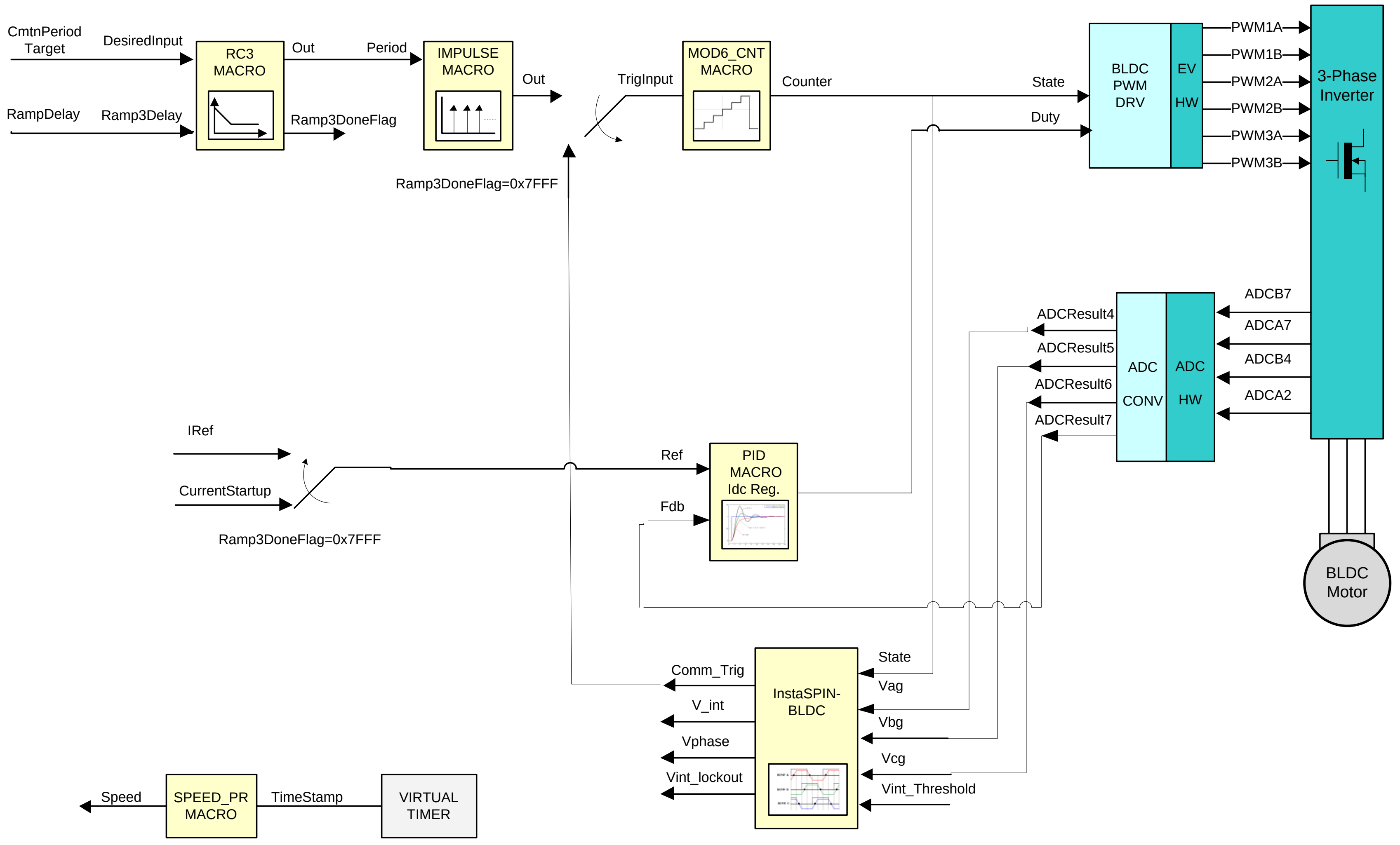


Figure 10 Graph Windows for Build Level 6 (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag

Level 6 Incremental System Build Block Diagram



Level 6 verifies the closed current loop and current PI controller.

Level 7 Incremental Build

Assuming the previous section is completed successfully, this section verifies the closed speed loop and speed PI controller. Open BLDC_Int-Settings.h and select level 7 incremental build option by setting the BUILDLEVEL to LEVEL7 (#define BUILDLEVEL LEVEL7). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will 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/run program with real time mode.
- The motor will gradually speed up and finally switch to closed loop commutation mode.
- Now 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 will start to regulate the motor speed. Gradually increase the command speed (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 realtime mode and reset.

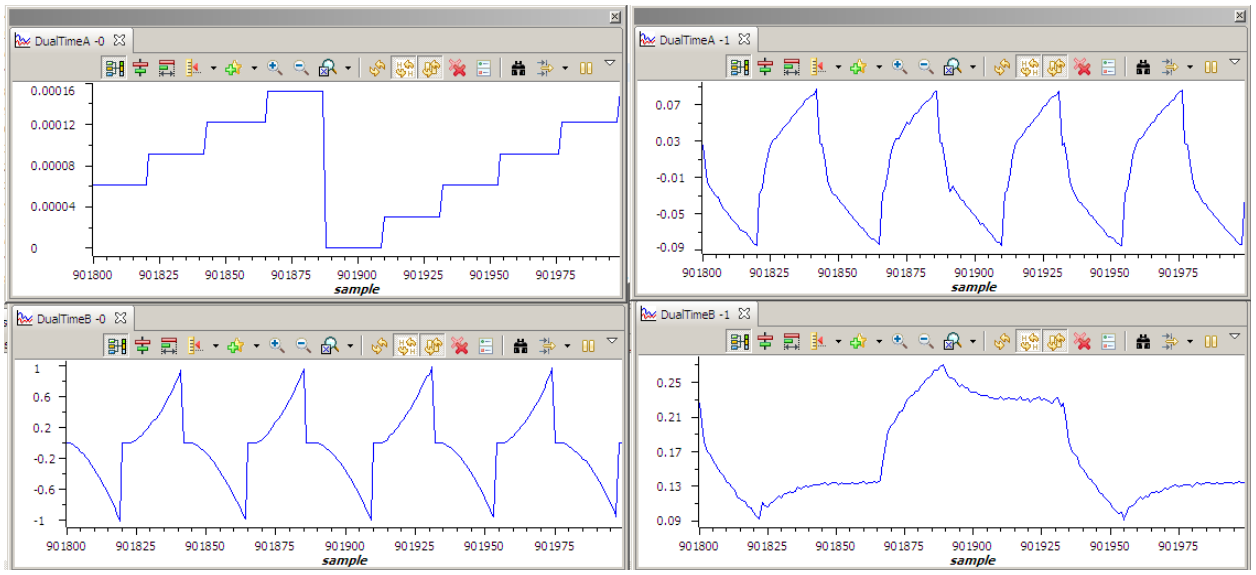
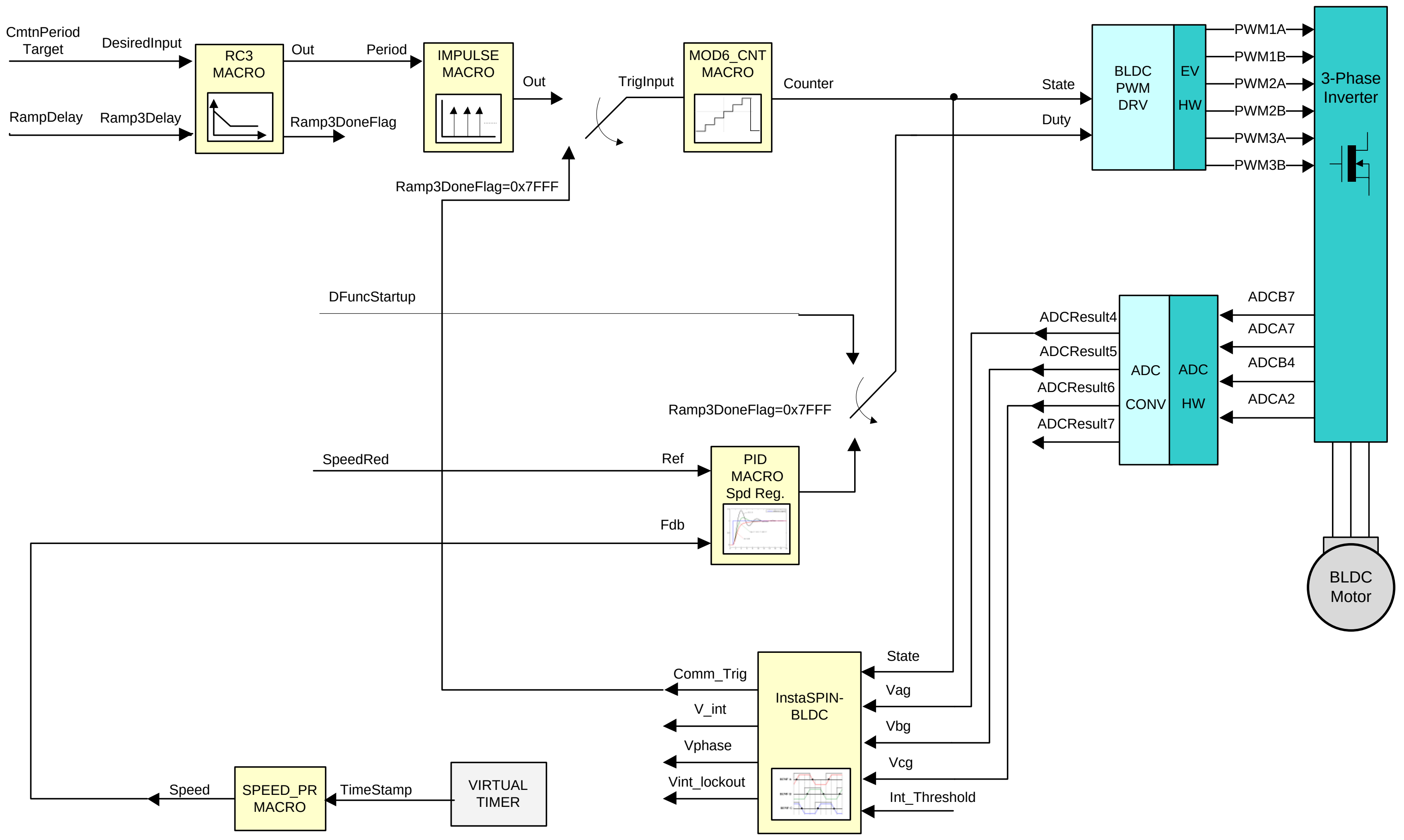


Figure 11 Graph Windows for Build Level 7 (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag

Level 7 Incremental System Build Block Diagram



Level 7 verifies the closed speed loop and speed PI controller.

Level 8 Incremental Build

Assuming the previous section is completed successfully, this section verifies the cascaded closed speed and current loops. Open BLDC_Int-Settings.h and select level 8 incremental build option by setting the BUILDLEVEL to LEVEL8 (#define BUILDLEVEL LEVEL8). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

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

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 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 will start to regulate the motor speed. Gradually increase the command speed (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 realtime mode and reset.

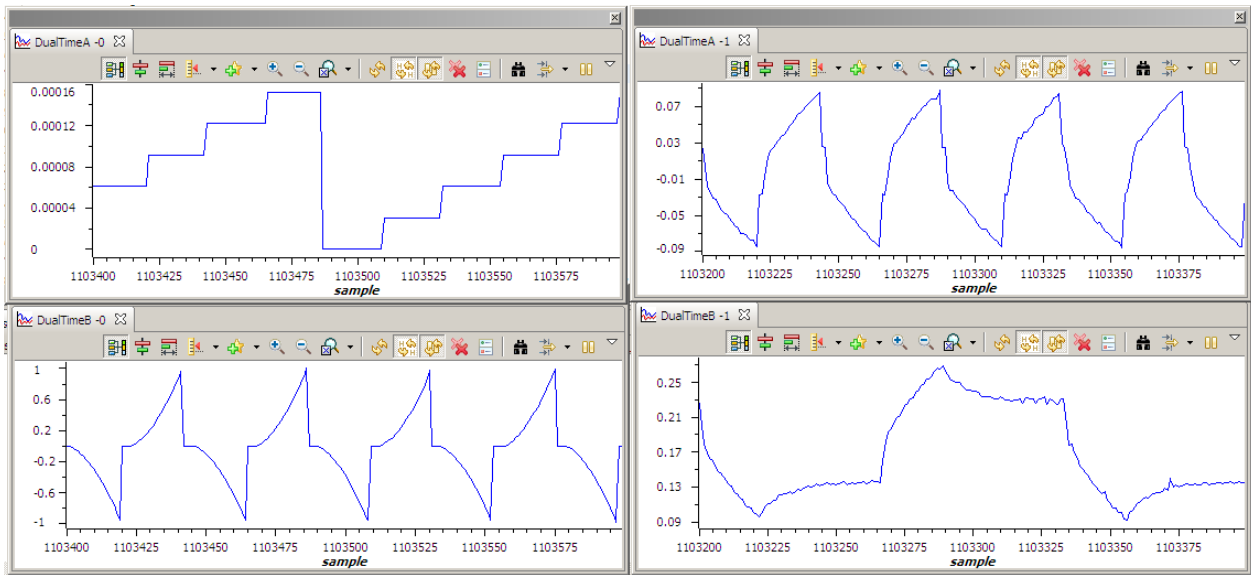
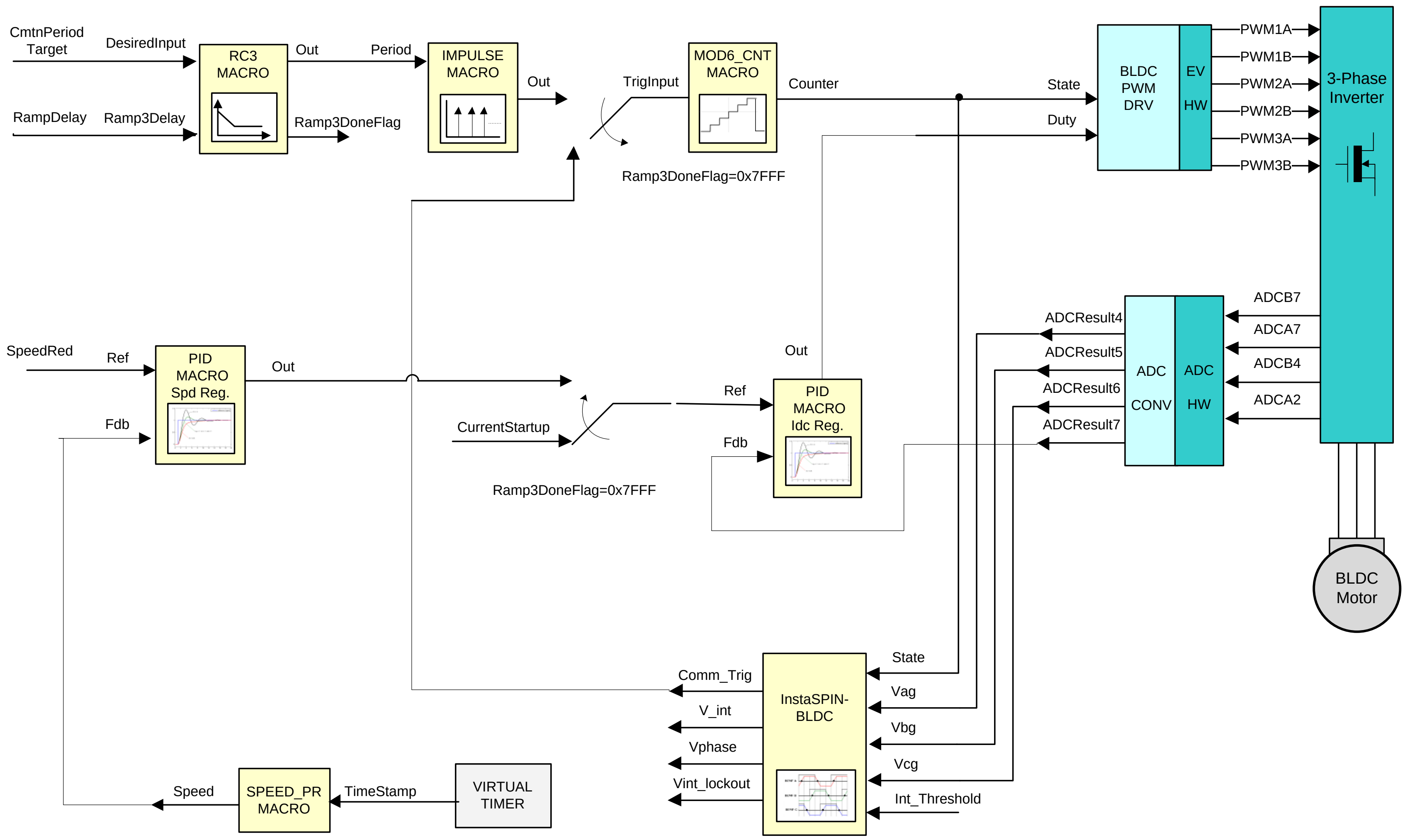


Figure 12 Graph Windows for Build Level 8 (a) mod6 counter, (b) V_int, (c) Vphase and (d)Vag

Level 8 Incremental System Build Block Diagram



Level 8 verifies the cascaded closed speed and current loops.

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		