Texas Instruments, Inc.
C2000 Systems and Applications

Sensorless Trapezoidal Control of BLDC Motors

Authors: Bilal Akin, Manish Bhardwaj

Contents

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 increase efficiency. Using these devices, it is possible to realize precise control algorithms. 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 trapezoidal BLDC motor control principle.
- Incremental build levels based on modular software blocks.
- Experimental results

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 apply 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.

BLDC Motors

The BLDC motor is an AC synchronous motor with permanent magnets on the rotor (moving part) and windings on the stator (fixed part). Permanent magnets create the rotor flux and the energized stator windings create electromagnet poles. The rotor (equivalent to a bar magnet) is attracted by the energized stator phase. By using the appropriate sequence to supply the stator phases, a rotating field on the stator is created and maintained. This action of the rotor - chasing after the electromagnet poles on the stator - is the fundamental action used in synchronous permanent magnet motors. The lead between the rotor and the rotating field must be controlled to produce torque and this synchronization implies knowledge of the rotor position.

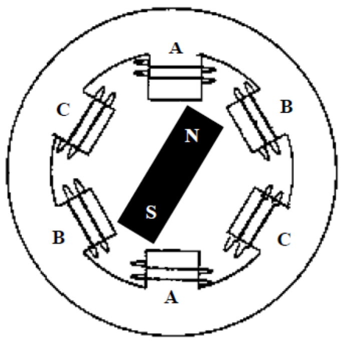


Fig.1 A three-phase synchronous motor with a one permanent magnet pair pole rotor

On the stator side, three phase motors are the most common. These offer a good compromise between precise control and the number of power electronic devices required to control the stator currents. For the rotor, a greater number of poles usually create a greater torque for the same level of current. On the other hand, by adding more magnets, a point is reached where, because of the space needed between magnets, the torque no longer increases. The manufacturing cost also increases with the number of poles. As a consequence, the number of poles is a compromise between cost, torque and volume.

Permanent magnet synchronous motors can be classified in many ways, one of these that is of particular interest to us is that depending on back-emf profiles: Brushless Direct Current Motor (BLDC) and Permanent Magnet Synchronous Motor (PMSM). This terminology defines the shape of the back-emf of the synchronous motor. Both BLDC and PMSM motors have permanent magnets on the rotor but differ in the flux distributions and back-emf profiles. To get the best performance out of the synchronous motor, it is important to identify the type of motor in order to apply the most appropriate type of control as described in the next chapters.

Comparison of BLDC and PMSM motors	
BLDC	**PMSM**
Synchronous machine	Synchronous machine
Fed with direct currents	Fed with sinusoidal currents
Trapezoidal Bemf	Sinusoidal Bemf
Stator Flux position commutation each 60 degrees	Continuous stator flux position variation
Only two phases ON at the same time	Possible to have three phases ON at the same time
Torque ripple at commutations	No torque ripple at commutations
Low order current harmonics in the audible range	Less harmonics due to sinusoidal excitation
Higher core losses due to harmonic content	Lower core loss
Less switching losses	Higher switching losses at the same switching freq.
Control algorithms are relatively simple	Control algorithms are mathematically intensive

Table 1. Comparison of BLDC and PMSM motors

- Both motor types are synchronous machines. The only difference between them is the shape of the induced voltage, resulting from two different manners of wiring the stator coils. The back-emf is trapezoidal in the BLDC motor case, and sinusoidal in the PMSM motor case.

- BLDC machines can be driven with sinusoidal currents and PMSM with direct currents, but for better performance, PMSM motors should be excited by sinusoidal currents and BLDC machines by direct currents.

- The control structure (hardware and software) of a sinusoidal motor required several current sensors and sinusoidal phase currents were hard to achieve with analog techniques. Therefore many motors (sinusoidal like trapezoidal) were driven with direct current for cost and simplicity reasons (low resolution position sensors and single low cost current sensor), compromising efficiency and dynamic behavior.

- Digital techniques addressed by the C2000 DSP controller make it possible to choose the right control technique for each motor type: Processing power is used to extract the best performance from the machine and reduce system costs. Possible options are using sensorless techniques to reduce the sensor cost, or even eliminate it, and also complex algorithms can help simplify the mechanical drive train design, lowering the system cost.

BLDC Motor Control

The key to effective torque and speed control of a BLDC motor is based on relatively simple torque and Back EMF equations, which are similar to those of the DC motor. The Back EMF magnitude can be written as:

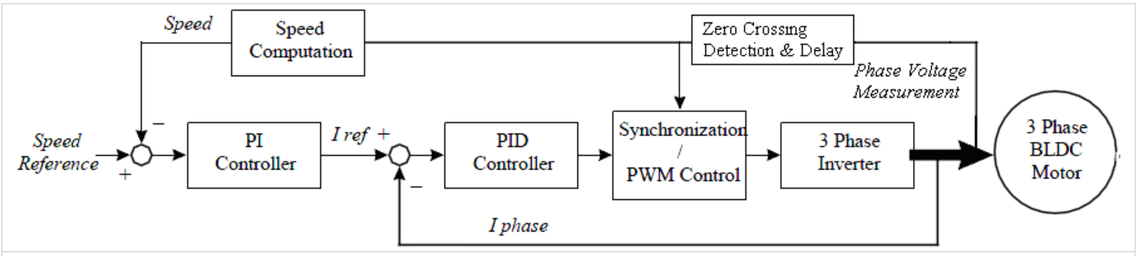
$$E = 2NlrBw$$

and the torque term as:

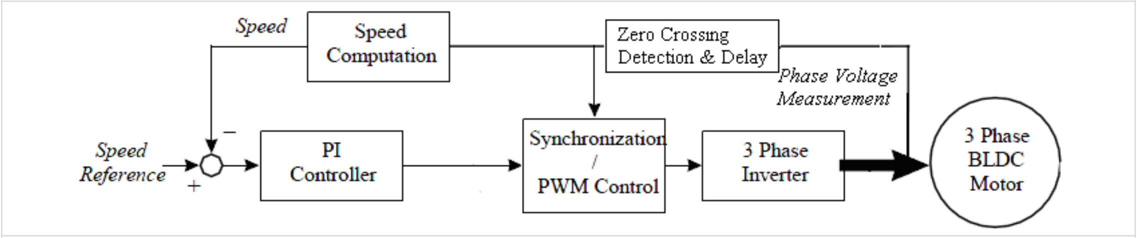
$$T = \left(\frac{1}{2}i^2\frac{dL}{d\theta}\right) - \left(\frac{1}{2}B^2\frac{dR}{d\theta}\right) + \left(\frac{4N}{\pi}Brl\pi i\right)$$

where *N* is the number of winding turns per phase, *l* is the length of the rotor, *r* is the internal radius of the rotor, *B* is the rotor magnet flux density, *w* is the motor's angular velocity, *i* is the phase current, *L* is the phase inductance, *θ* is the rotor position, *R* is the phase resistance.

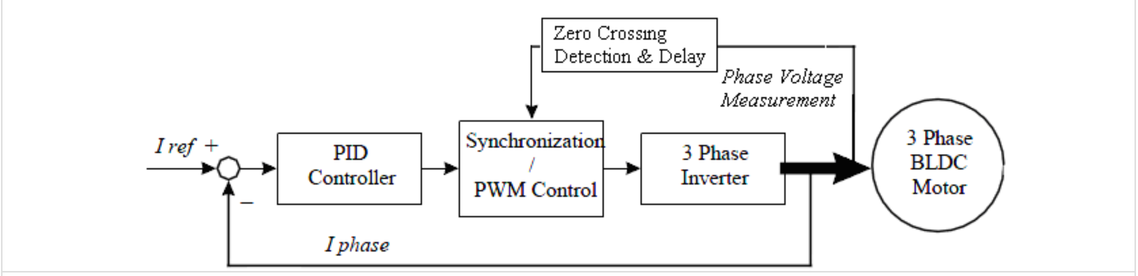
The first two terms in the torque expression are parasitic reluctance torque components. The third term produces mutual torque, which is the torque production mechanism used in the case of BLDC motors. To sum up, the Back EMF is directly proportional to the motor speed and the torque production is almost directly proportional to the phase current. These factors lead to the following BLDC motor speed control schemes:



(a)



(b)


Zero Crossing Detection & Delay
Phase Voltage Measurement
I ref +
−
PID Controller
Synchronization / PWM Control
3 Phase Inverter
3 Phase BLDC Motor
I phase

(c)

Fig.2 Speed and Current Control Loop Configurations for a BLDC Motor

The BLDC motor is characterized by a two phase ON operation to control the inverter. In this control scheme, torque production follows the principle that current should flow in only two of the three phases at a time and that there should be no torque production in the region of Back EMF zero crossings. The following figure describes the electrical wave forms in the BLDC motor in the two phases ON operation.

This control structure has several advantages:

- Only one current at a time needs to be controlled.
- Only one current sensor is necessary (or none for speed loop only, as detailed in the next sections).
- The positioning of the current sensor allows the use of low cost sensors as a shunt.

We have seen that the principle of the BLDC motor is, at all times, to energize the phase pair which can produce the highest torque. To optimize this effect the Back EMF shape is trapezoidal. The combination of a DC current with a trapezoidal Back EMF makes it theoretically possible to produce a constant torque. In practice, the current cannot be established instantaneously in a motor phase; as a consequence the torque ripple is present at each 60 degree phase commutation.

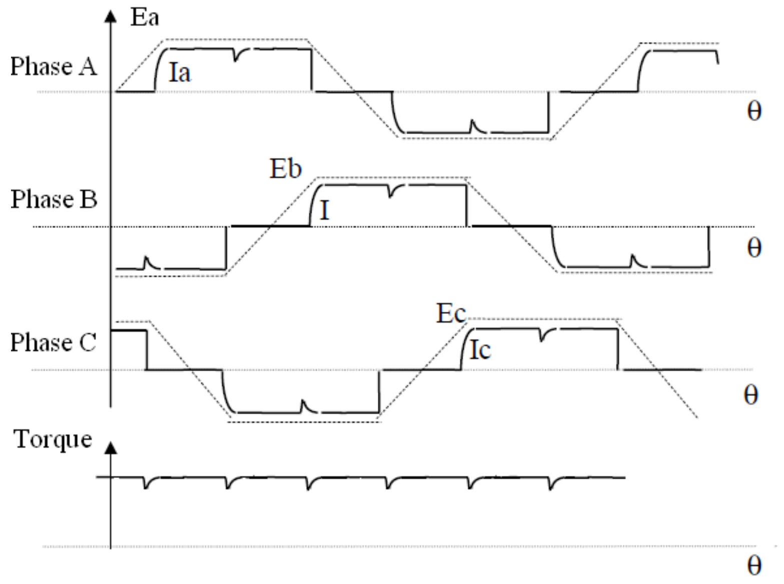


Fig 3. Electrical Waveforms in the Two Phase ON Operationand Torque Ripple

If the motor used has a sinusoidal Back EMF shape, this control can be applied but the produced torque is:

- Firstly, not constant but made up from portions of a sine wave. This is due to its being the combination of a trapezoidal current control strategy and of a sinusoidal Back EMF. Bear in mind that a sinusoidal Back EMF shape motor controlled with a sine wave strategy (three phase ON) produces a constant torque.
- Secondly, the torque value produced is weaker.

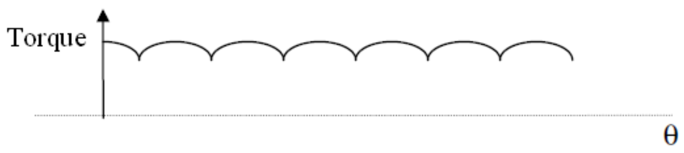


Fig.4 Torque Ripple in a Sinusoidal Motor Controlled as a BLDC

System Topology

Three Phase Inverter

The BLDC motor control consists of generating DC currents in the motor phases. This control is subdivided into two independent operations: stator and rotor flux synchronization and control of the current value. Both operations are realized through the three phase inverter depicted in the following scheme.

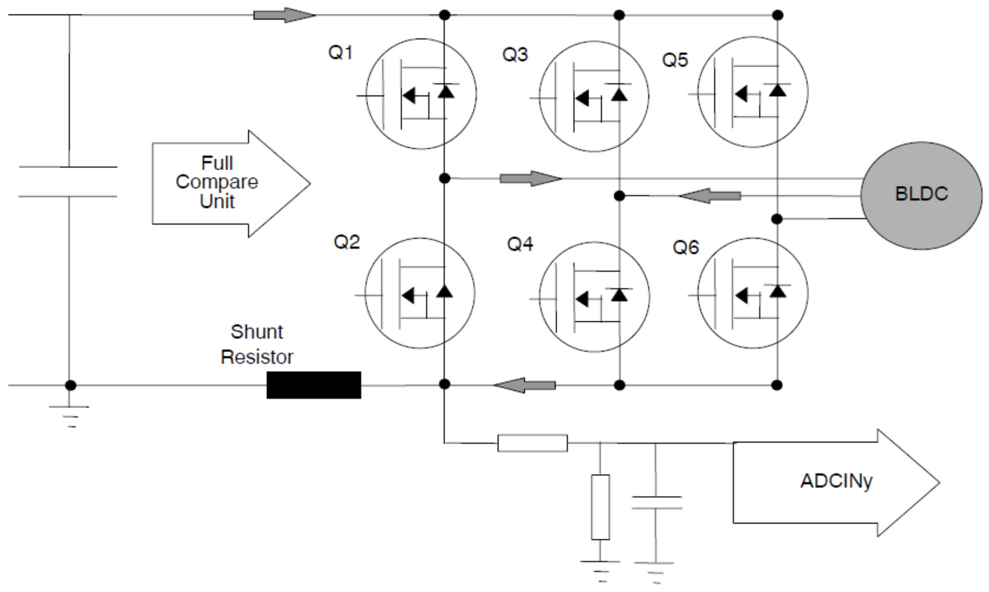


Fig.5 Three Phase Inverter

The flux synchronization is derived from the position information coming from sensors, or from sensorless techniques. From the position, the controller determines the appropriate pair of transistors (Q1 to Q6) which must be driven. The regulation of the current to a fixed 60 degrees reference can be realized in either of the two different modes:

1. The Pulse Width Modulation (PWM) Mode

The supply voltage is chopped at a fixed frequency with a duty cycle depending on the current error. Therefore both the current and the rate of change of current can be controlled. The two phase supply duration is limited by the two phase commutation angles. The main advantage of the PWM strategy is that the chopping frequency is a fixed parameter; hence, acoustic and electromagnetic noises are relatively easy to filter.

There are also two ways of handling the drive current switching: hard chopping and soft chopping.
In the hard chopping technique both phase transistors are driven by the same pulsed signal: the two transistors are switched-on and switched-off at the same time. The power electronics board is then easier to design and is also cheaper as it handles only three pulsed signals. A disadvantage of the hard chopping operation is that it increases the current ripple by a large factor in comparison with the soft chopping approach.

The soft chopping approach allows not only a control of the current and of the rate of change of the current but a minimization of the current ripple as well. In this soft chopping mode the low side transistor is left ON during the phase supply and the high side transistor switches according to the pulsed signal. In this case, the power electronics board has to handle six PWM signals.

2. The Hysteresis Mode

In the hysteresis-type current regulator, the power transistors are switched off and on according to whether the current is greater or less than a reference current. The error is used directly to control the states of the power transistors. The hysteresis controller is used to limit the phase current within a preset hysteresis band. As the supply voltage is fixed, the result is that the switching frequency varies as the current error varies. The current chopping operation is thus not a fixed chopping frequency PWM technique. This method is more commonly implemented in drives where motor speed and load do not vary too much, so that the variation in switching frequency is small. Here again, both hard and soft chopping schemes are possible. Since the width of the tolerance band is a design parameter, this mode allows current control to be as precise as desired, but acoustic and electromagnetic noise are difficult to filter because of the varying switching frequency.

Current Sensing

A characteristic of the BLDC control is to have only one current at a time in the motor (two phases ON). Consequently, it is not necessary to put a current sensor on each phase of the motor; one sensor placed in the line inverter input makes it possible to control the current of each phase. Moreover, using this sensor on the ground line, insulated systems are not necessary, and a low cost resistor can be used. Its value is set such that it activates the integrated over-current protection when the maximum current permitted by the power board has been reached.

Each current measurement leads to a new PWM duty cycle loaded at the beginning of a PWM cycle. Note that, during Turn OFF, the shunt resistor does not have this current to sense, regardless of whether the inverter is driven in hard chopping or in soft chopping mode. The figure below depicts the shunt current in soft chopping mode and shows that in the Turn OFF operation the decreasing current flows through the M2 free wheeling diode and through the maintained closed M4 (so there is no current observable in the shunt in this chopping mode during Turn OFF). This implies that it is necessary to start a current conversion in the middle of the PWM duty cycle.

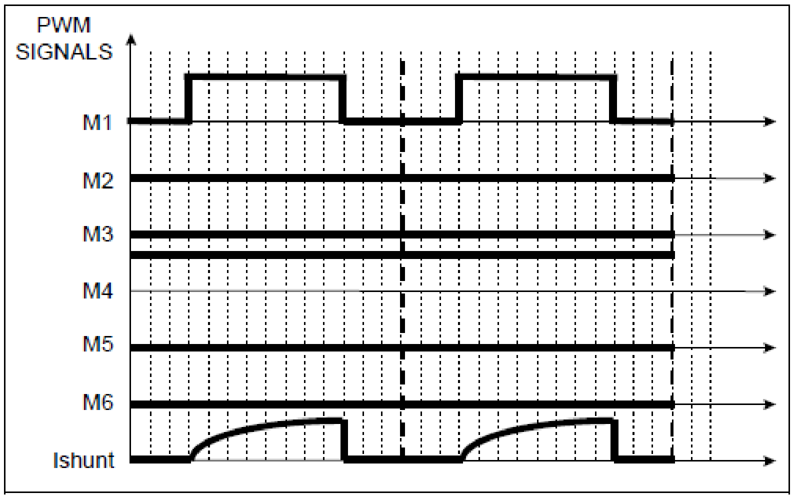


Fig. 6 Shunt Resistor Voltage Drop according to PWM DutyCycles (Soft Chopping)

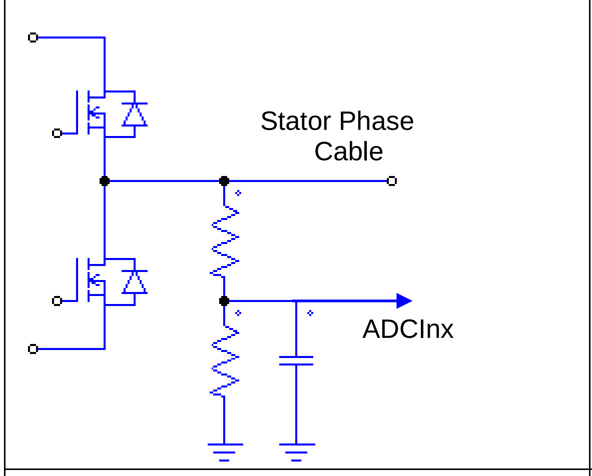
In the hard chopping mode during the Turn OFF neither M1 nor M4 drive current, so that the decreasing phase current flows from ground through the shunt resistor via M2 and M3 free wheeling diodes and back to ground via the capacitor. In this chopping mode it is possible to see the exponentially decreasing phase current across the shunt as a negative shunt voltage drop appears. Assuming that neither the power board nor the control board support negative voltages, this necessitates that the current be sensed in the middle of the Turn ON.

Achieving a BLDC speed control requires three control layers to be performed. The innermost one is to get the rotor position in order to correctly commutate the stator flux. Once the rotor position is known, the magnitude of the stator flux has to be generated and controlled. Assuming that the stator flux is proportional to the current flowing in the stator coils, the control of the stator flux magnitude is equivalent to the control of the input current. The outermost control loop is the speed regulation loop.

Bemf Zero Crossing Point Computation

The resistor divider circuit is specified such that the maximum output from this voltage sensing circuit utilizes the full ADC conversion range. The filtering capacitor should filter the chopping frequency, so only very small values are necessary (in the range of nF or even less). The sensorless algorithm is based only on the three motor terminal voltage measurements and thus requires only four ADC input lines.

In the sensored control structure, the phases are commutated once every 60º mechanical rotation of the rotor. This implies that only six commutation signals are sufficient to drive a BLDC motor. Furthermore, an efficient control implies synchronization between the phase Bemf and the phase supply so that the Bemf crosses zero once during the non-fed 60º sector. The next paragraph shows how it is possible to get the three Bemfs and their zero crossings. The figure below depicts the motor terminal model, where L is the phase inductance, R is the phase resistance, E is the back electromotive force, Vn is the star connection voltage referenced to ground and Vx is the phase voltage referenced to ground. Vx voltages are measured by means of the ADC Unit and via the resistance bridge depicted in Fig. 8.



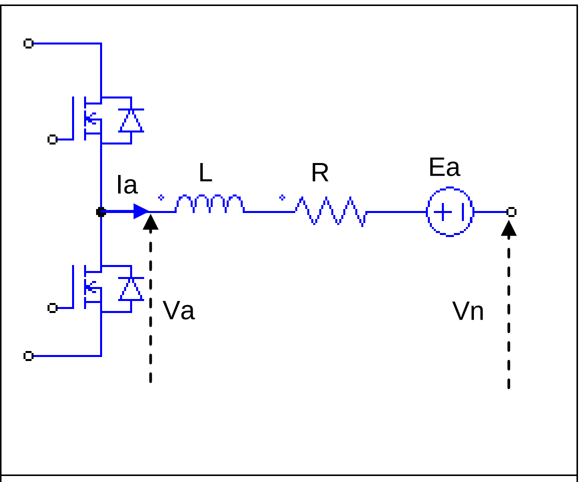


Fig. 7 Backemf sensing circuit on HDMC kit

Fig. 8 Stator Terminal Electrical Model

Assuming that phase C is the non-fed phase it is possible to write the following equations for the three terminal voltages:

$$Va = RIa + L\frac{dIa}{dt} + Ea + Vn$$

$$Vb = RIb + L\frac{dIb}{dt} + Eb + Vn$$

$$Vc = Ec + Vn$$

As only two currents flow in the stator windings at any one time, two phase currents are equal and opposite. Therefore,

$$Ia = -Ib$$

Thus, by adding the three terminal voltage equations we have,

$$Va + Vb + Vc = Ea + Eb + Ec + 3Vn$$

The instantaneous Bemf waveforms of the BLDC motor are shown in Figure 9. From this figure it is evident that at the Bemf zero crossing points the sum of the three Bemfs is equal to zero. Therefore the last equation reduces to,

$$Va + Vb + Vc = 3Vn$$

This equation is implemented in the code to compute the neutral voltage. In the code, the quantity 3Vn is represented by the variable called *neutral*.

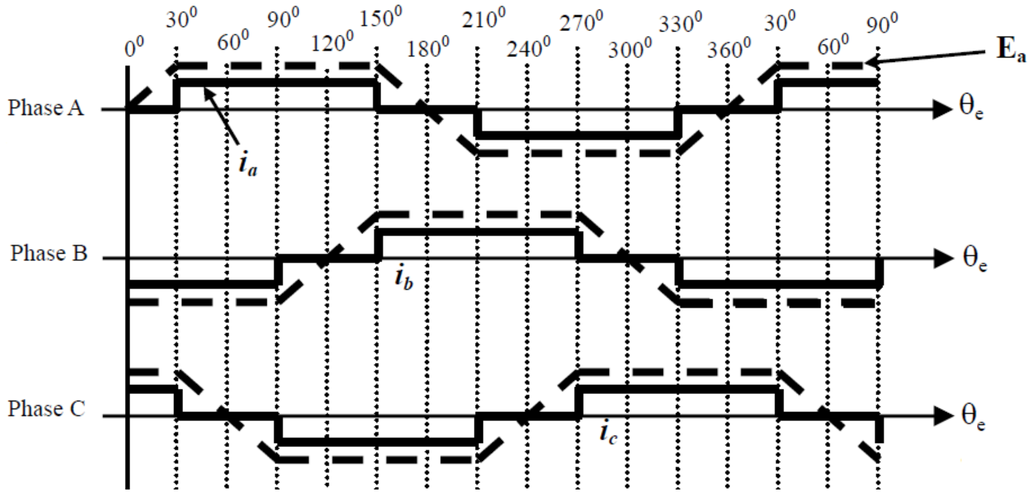


Fig. 9 Typical current and BEMF waveform sin BLDC motor drive

For the non-fed phase (zero current flowing), the stator terminal voltage can be rewritten as follows:

$$3Ec = 3Vc - 3Vn$$

This equation is used in the code to calculate the Bemf zero crossing point of the non-fed phase C. Similar equations are used to calculate the Bemf zero crossing points of other Bemf voltages Ea and Eb. As we are interested in the zero crossing of the Bemf it is possible to check only for the Bemf sign change; this assumes that the Bemf scanning loop period is much shorter than the mechanical time constant. This function is computed after the three terminal voltage samples, typically once every 50us.

Electrical Behavior at Commutation Points

At the instants of phase commutation, high dV/dt and dI/dt glitches may occur due to the direct current level or to the parasitic inductance and capacitance of the power board. This can lead to a misreading of the computed neutral voltage. This is overcome by discarding the first few scans of the Bemf once a new phase commutation occurs. In the code this is implemented by the function named 'NOISE_WIN'. The duration depends on the power switches, the power board design, the phase inductance and the driven direct current. This parameter is system-dependent and is set to a large value in the low speed range of the motor. As the speed increases, the s/w gradually lowers this duration since the Bemf zero crossings also get closer at higher speed.

Commutation Instants Computation

In an efficient sensored control the Bemf zero crossing points are displaced 30° from the instants of phase commutation. So before running the sensorless BLDC motor with help of the six zero crossing events it is necessary to compute the time delay corresponding to this 30° delay angle for exact commutation points. This is achieved by implementing a position interpolation function. In this software it is implemented as follows: let T be the time that the rotor spent to complete the previous revolution and α be the desired delay angle. By dividing α by 360° and multiplying the result by T we obtain the time duration to be spent before commutating the next phase pair. In the code this delay angle is fixed to 30°. The corresponding time delay is represented in terms of the number of sampling time periods and is stored in the variable *cmtn_delay*. Therefore,

Time delay = *cmtn_delay* **.**Ts = T(α /360) = *v_timer***.**Ts(α /360) = *v_timer* **.** Ts/12

Where, Ts is the sampling time period and *v_timer* is a timer that counts the number of sampling cycles during the previous revolution of the rotor.

The above equation is further simplified as,

cmtn_delay = v_timer /12

This equation is implemented in the code in order to calculate the time delay corresponding to the 30° commutation delay angle.

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

C2000 family of devices posses the desired computation power to execute complex control algorithms along with the right mix of peripherals to interface with the various components of the DMC hardware like the ADC, ePWM, QEP, eCAP etc. These peripherals have all the necessary hooks for implementing systems which meet safety requirements, like the trip zones for PWMs and comparators. Along with this the C2000 ecosystem of software (libraries and application software) and hardware (application kits) help in reducing the time and effort needed to develop a Digital Motor Control solution. The DMC Library provides configurable blocks that can be reused to implement new control strategies. IQMath Library enables easy migration from floating point algorithms to fixed point thus accelerating the development cycle.

Thus, with C2000 family of devices it is easy and quick to implement complex control algorithms (sensored and sensorless) for motor control. The use of C2000 devices and advanced control schemes provides the following system improvements:

- Favors system cost reduction by an efficient control in all speed range implying right dimensioning of power device circuits
- Use of advanced control algorithms it is possible to reduce torque ripple, thus resulting in lower vibration and longer life time of the motor
- Advanced control algorithms reduce harmonics generated by the inverter thus reducing filter cost.
- Use of sensorless algorithms eliminates the need for speed or position sensor.
- Decreases the number of look-up tables which reduces the amount of memory required
- The Real-time generation of smooth near-optimal reference profiles and move trajectories, results in better-performance
- Generation of high resolution PWM's is possible with the use of ePWM peripheral for controlling the power switching inverters
- Provides single chip control system

For advanced controls, C2000 controllers can also perform the following:

- Enables control of multi-variable and complex systems using modern intelligent methods such as neural networks and fuzzy logic.
- Performs adaptive control. C2000 controllers have the speed capabilities to concurrently monitor the system and control it. A dynamic control algorithm adapts itself in real time to variations in system behaviour.
- Performs parameter identification for sensorless control algorithms, self commissioning, online parameter estimation update.
- Performs advanced torque ripple and acoustic noise reduction.
- Provides diagnostic monitoring with spectrum analysis. By observing the frequency spectrum of mechanical vibrations, failure modes can be predicted in early stages.
- Produces sharp-cut-off notch filters that eliminate narrow-band mechanical resonance. Notch filters remove energy that would otherwise excite resonant modes and possibly make the system unstable.

TI Literature and DMC Library

The Digital Motor Control (DMC) library is composed of functions represented as blocks. These blocks are categorized as Transforms & Estimators (Clarke, Park, Sliding Mode Observer, Phase Voltage Calculation, and Resolver, Flux, and Speed Calculators and Estimators), Control (Signal Generation, PID, BEMF Commutation, Space Vector Generation), and Peripheral Drivers (PWM abstraction for multiple topologies and techniques, ADC drivers, and motor sensor interfaces). Each block is a modular software macro is separately documented with source code, use, and technical theory. Check the folders below for the source codes and explanations of macro blocks:

- C:\TI\controlSUITE\libs\app_libs\motor_control\math_blocks\v4.0
- C:\TI\controlSUITE\libs\app_libs\motor_control\drivers\f2803x_v2.0

These modules allow users to quickly build, or customize, their own systems. The Library supports the three motor types: ACI, BLDC, PMSM, and comprises both peripheral dependent (software drivers) and target dependent modules.

The DMC Library components have been used by TI to provide system examples. At initialization all DMC Library variables are defined and inter-connected. At run-time the macro functions are called in order. Each system is built using an incremental build approach, which allows some sections of the code to be built at a time, so that the developer can verify each section of their application one step at a time. This is critical in real-time control applications where so many different variables can affect the system and many different motor parameters need to be tuned.

Note: TI DMC modules are written in form of macros for optimization purposes (refer to application note *SPRAAK2* for more details at TI website). The macros are defined in the header files. The user can open the respective header file and change the macro definition, if needed. In the macro definitions, there should be a backslash "\" at the end of each line as shown below which means that the code continue in the next line. <u>Any character including invisible ones like "space" after the backslash will cause compilation error.</u> Therefore, make sure that the backslash is the last character in the line. In terms of code development, the macros are almost identical to C function, and the user can easily convert the macro definition to a C functions.

```
#define PARK_MACRO(v)                                                    \
                                                                         \
    v.Ds = _IQmpy(v.Alpha,v.Cosine) + _IQmpy(v.Beta,v.Sine);             \
    v.Qs = _IQmpy(v.Beta,v.Cosine)  - _IQmpy(v.Alphv.Sine);
```

A typical DMC macro definition

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
COM_TRIG	Commutation Trigger Generator Module
PID	PID Regulators
RC	Ramp Controller (slew rate limiter)
RC2	Ramp up and Ramp down Module
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 conventional voltage-source inverter. 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 (I fb Ret) is measured and sent to the TMS320x2803x via analog-to-digital converters (ADCs).

HVBLDC_Sensorless project has the following properties:

C Framework		
System Name	**Program Memory Usage 2803x**	**Data Memory Usage[1] 2803x**
HVBLDC_Sensorless	4098 words[2]	2012 words

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

CPU Utilization – BLDC Sensorless	
Total Number of Cycles	**431***
CPU Utilization @ 60 Mhz	**14.4%**
CPU Utilization @ 40 Mhz	**21.6%**

* At 20 kHz ISR freq. Debug macros excluded.

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	Asymmetrical with no dead band
Interrupts	CPU Timer 0 – Implements 40 kHz ISR execution rate
Peripherals Used	PWM 1 / 2 / 3 for motor control PWM 6A, 6B, 7A & 7B for DAC outputs ADC A2 for low side DC bus return current sensing, B1-3 for Bemf sensing

The overall system implementing a 3-ph sensorless BLDC control is depicted in Fig.6 and 7.

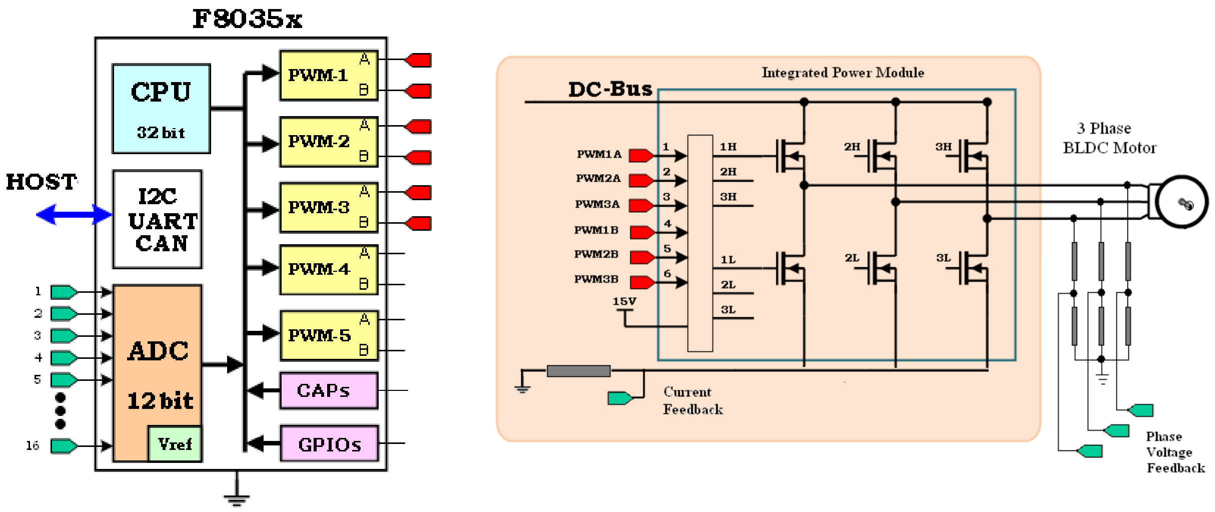


Fig 6 A 3-ph BLDC drive implementation

*Note that the dcbus return current is obtained through the summation of three phase currents in R1.1

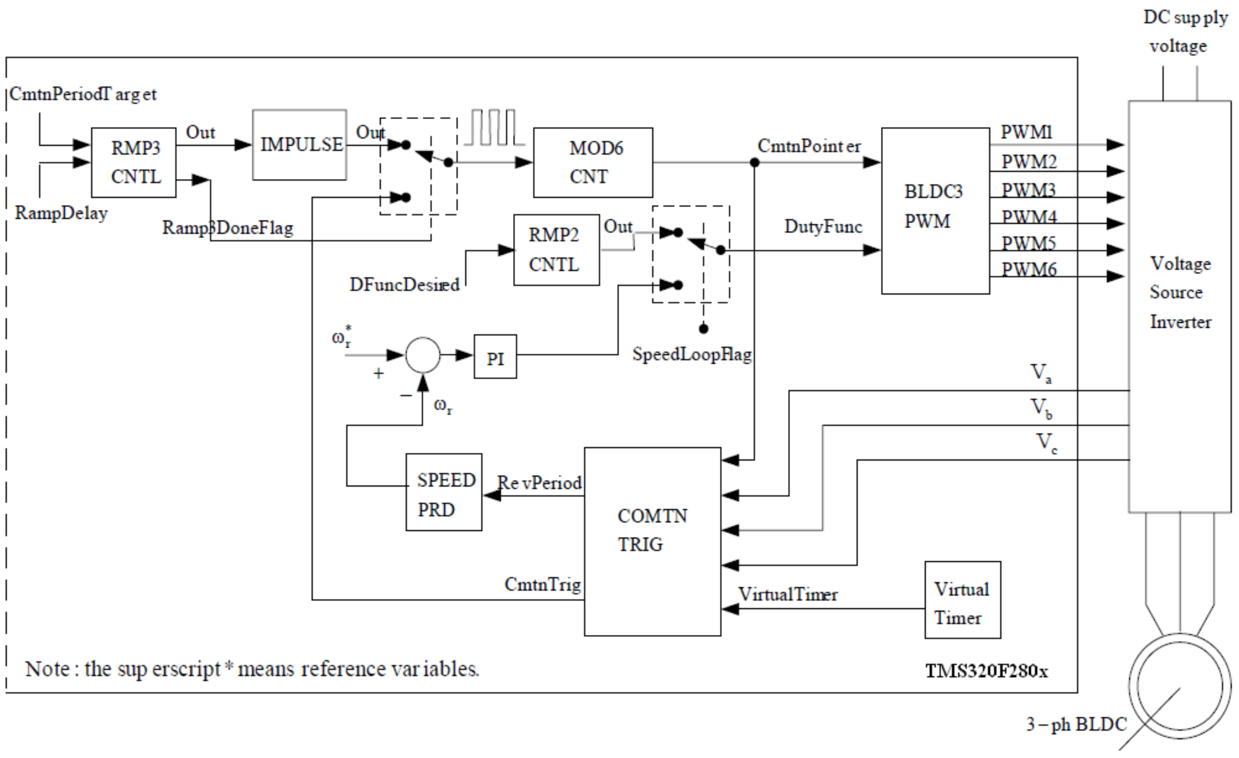
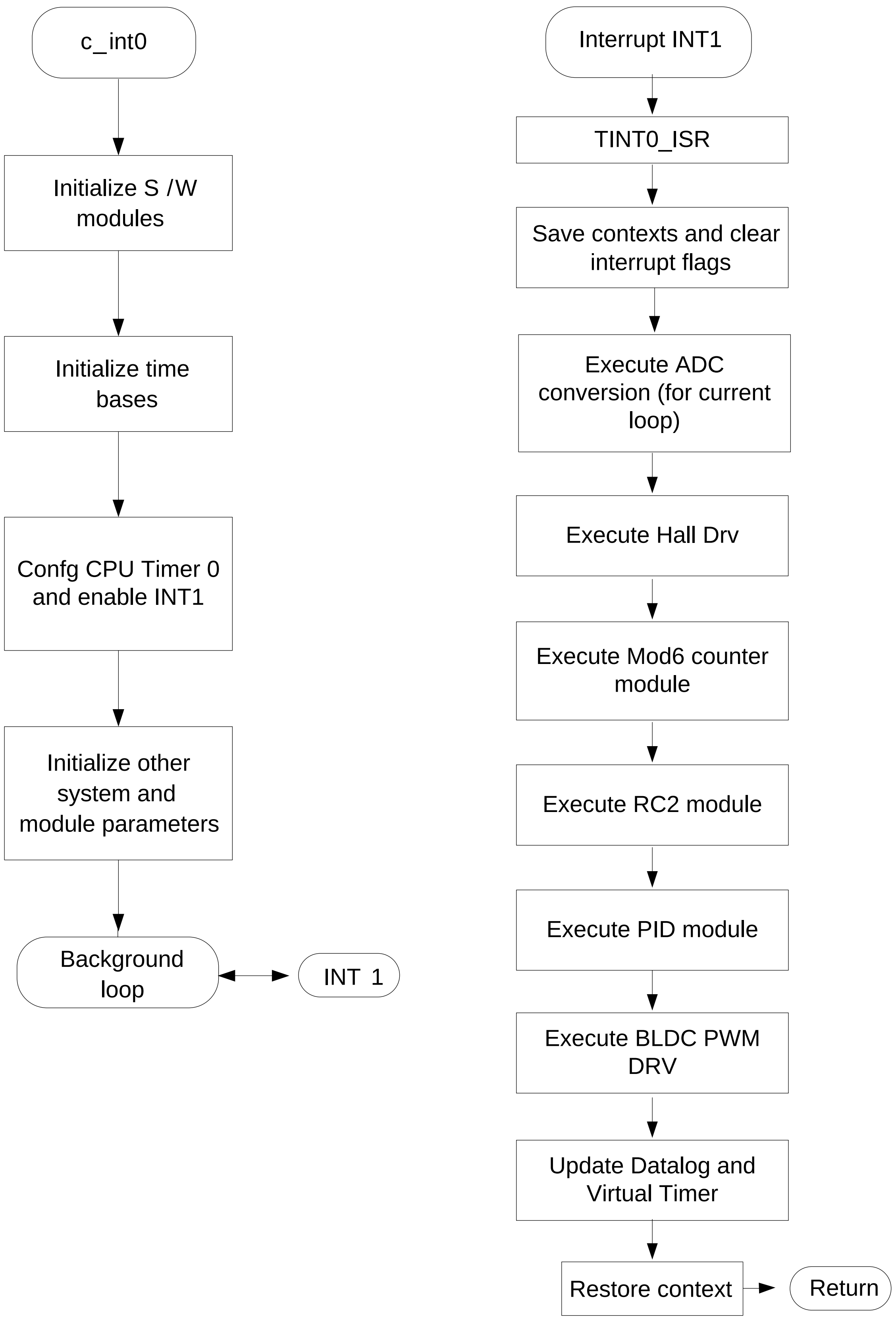


Fig. 7 Overall block diagram of sensorless control of BLDC

The software flow is described below:


c_int0
Initialize S /W modules
Initialize time bases
Confg CPU Timer 0 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 (for current loop)
Execute Hall Drv
Execute Mod6 counter module
Execute RC2 module
Execute PID module
Execute BLDC PWM DRV
Update Datalog and Virtual Timer
Restore context
Return

Hardware Configuration (HVDMC Kit R1.1)

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

C:\TI\controlSUITE\development_kits\HVMotorCtrl+PfcKit_v2.0\~Docs

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

HW Setup Instructions

1. Open the Lid of the HV Kit
2. Install the Jumpers [Main]-J3, J4 and J5, J9 for 3.3V, 5V and 15V power rails and JTAG reset line.
3. Unpack the DIMM style controlCARD and place it in the connector slot of [Main]-J1. Push vertically down using even pressure from both ends of the card until the clips snap and lock. (to remove the card simply spread open the retaining clip with thumbs)
4. Connect a USB cable to connector [M3]-JP1. This will enable isolated JTAG emulation to the C2000 device. [M3]-LD1 should turn on. Make sure [M3]-J5 is not populated. If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will automatically be installed. If a windows installation window appears try to automatically install drivers from those already on your computer. The emulation drivers are found at http://www.ftdichip.com/Drivers/D2XX.htm. The correct driver is the one listed to support the FT2232.
5. If a third party JTAG emulator is used, connect the JTAG header to [M3]-J2 and additionally [M3]-J5 needs to be populated to put the onboard JTAG chip in reset.
6. Ensure that [M6]-SW1 is in the "Off" position. Connect 15V DC power supply to [M6]-JP1.
7. Turn on [M6]-SW1. Now [M6]-LD1 should turn on. Notice the control card LED would light up as well indicating the control card is receiving power from the board.
8. Note that the motor should be connected to the [M5]-TB3 terminals after you finish with the first incremental build step.
9. Note the DC Bus power should only be applied during incremental build levels when instructed to do so. The two options to get DC Bus power are discussed below,

(i) To use DC power supply, set the power supply output to zero and connect [Main]-BS5 and BS6 to DC power supply and ground respectively.

(ii) To use AC Mains Power, Connect [Main]-BS1 and BS5 to each other using banana plug cord. Now connect one end of the AC power cord to [Main]-P1. The other end needs to be connected to output of a variac. Make sure that the variac output is set to zero and it is connected to the wall supply through an isolator.

Note: Phase voltage sensing caps (C21, 22, 23) are optimized for AC motor control. Use lower value capacitors (i.e. <=2.2nF) to reach peak torque. Also note that, dc bus current feedback is obtained as sum of all three phases instead of a shunt resistor on the dc bus return path.

For reference the pictures below show the jumper and connectors that need to be connected for this lab. Note that the order of motor cable colors connected to inverter output should be as shown below. Please check the motor datasheet for further details.

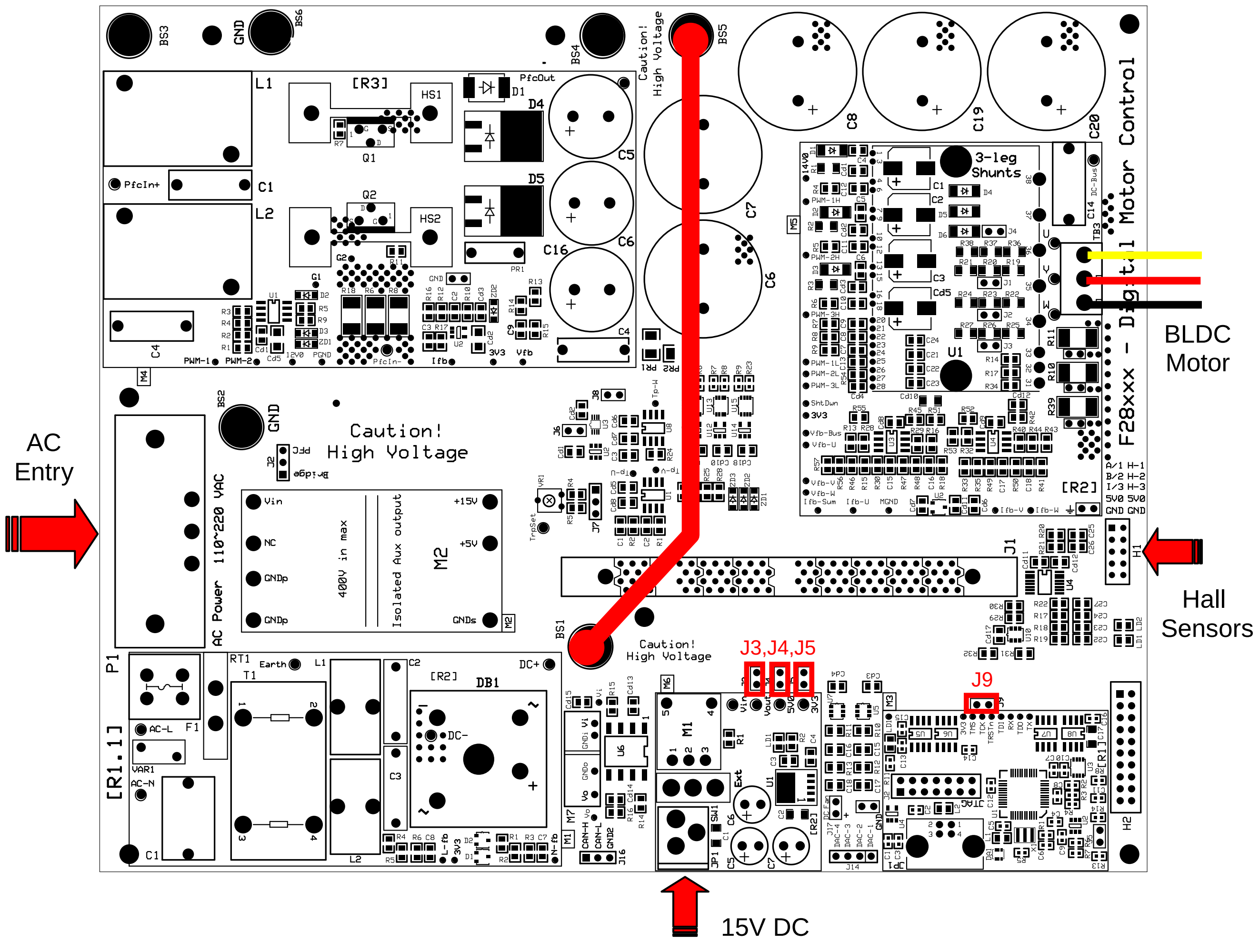


Fig. 8 Using AC Power to generate DC Bus Power

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

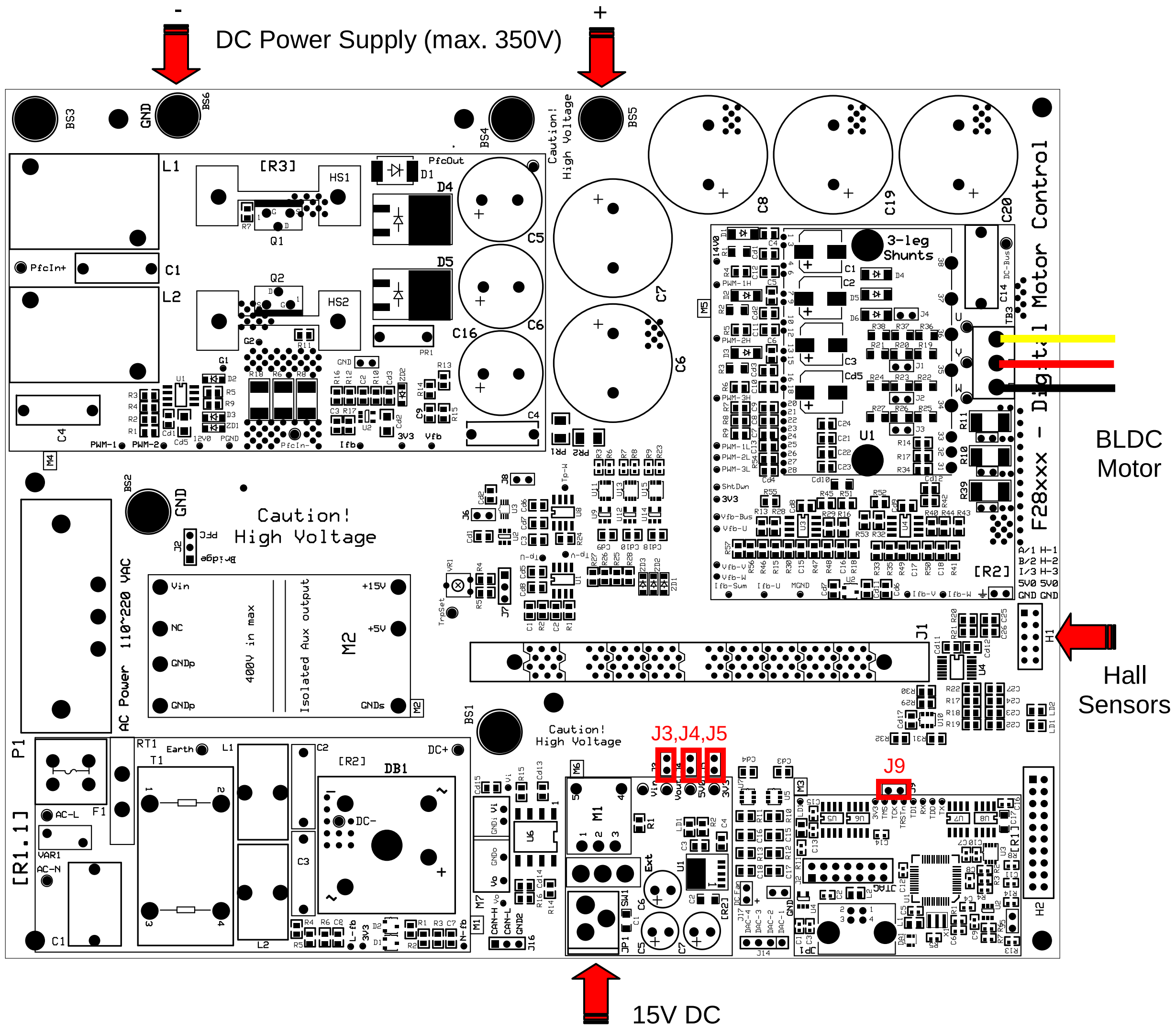


Fig.9 Using External DC power supply to generate DC-Bus for the inverter

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 HVBLDC_Sensorless Project

Please refer to the "Generic Steps for Software Setup for HVMotorCtrl+PFC Kit Projects" section in the HVMotorCtrl+PFC Kit How To Run Guide

C:\TI\controlSUITE\development_kits\HVMotorCtrl+PfcKit_v2.0\~Docs

This section goes over how to install CCS and set it up to run with this project.
Select the HVBLDC_Sensorless 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 open a watch window and add the variables shown in the table below and select the appropriate Q format for them.

Watch Window Variables	
Variable Name	**Viewed as**
EnableFlag	Unsigned Integer
IsrTicker	Unsigned Integer
SpeedRef	Q24
Dlog.prescalar	Integer
SpeedLoopFlag	Unsigned Integer
ILoopFlag	Unsigned Integer
CmtnPeriodTarget	Unsigned Integer
DFuncDesired	Unsigned Integer
ClosedFlag	Unsigned Integer
pid1_spd.Out	Q24
pid1_idc.Out	Q24
pid1_spd.Kp	Q24
speed1.Speed	Q24

Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location C:\TI\ControlSUITE\developement_kits\HVMotorCtrl+PfcKit_v2.0\HVBLDC_Sensorless 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 Sensorless BLDC project

The system is gradually built up in order for the final system can be confidently operated. Six 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**
PWMDAC_MACRO		√√	√	√	√
RC3_MACRO	√√	√√	√	√	√
MOD6_CNT_MACRO	√√	√√	√	√	√
IMPULSE_MACRO	√√	√√	√	√	√
BLDCPWM_MACRO	√√	√√	√	√	√
RC2_MACRO			√√	√	√
HALL3 _READ_MACRO			√√	√	√
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 "HVMotorCtrl+PFC Kit How To Run Guide" completed successfully, this section describes the steps for a "minimum" system check-out which confirms operation of system interrupts, some peripheral & target independent modules and one peripheral dependent module. Open HVBLDC_Sensorless-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.

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). Note that the
- 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 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 DfuncTesting.

- 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. Enter a new lower value for CmtnperiodTarget (DesiredInput). Then Period will gradually reduce to the new value.
- Check MOD6_CNT 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 odd numbered PWM outputs (PWM1, PWM3, and PWM5) will either generate PWM pulses or remain OFF. The even numbered PWM outputs (PWM2, PWM4, and PWM6) will either remain ON or OFF.
- 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.

During running this level, the PWM outputs should be appeared as follow:

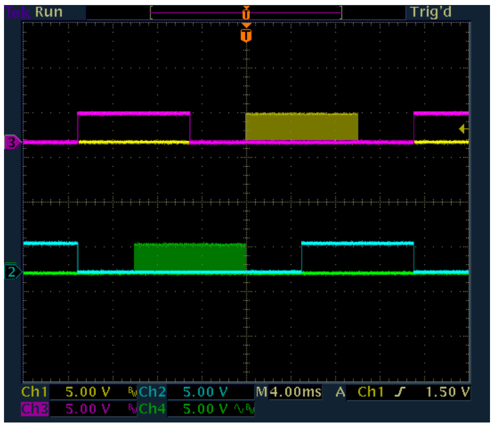
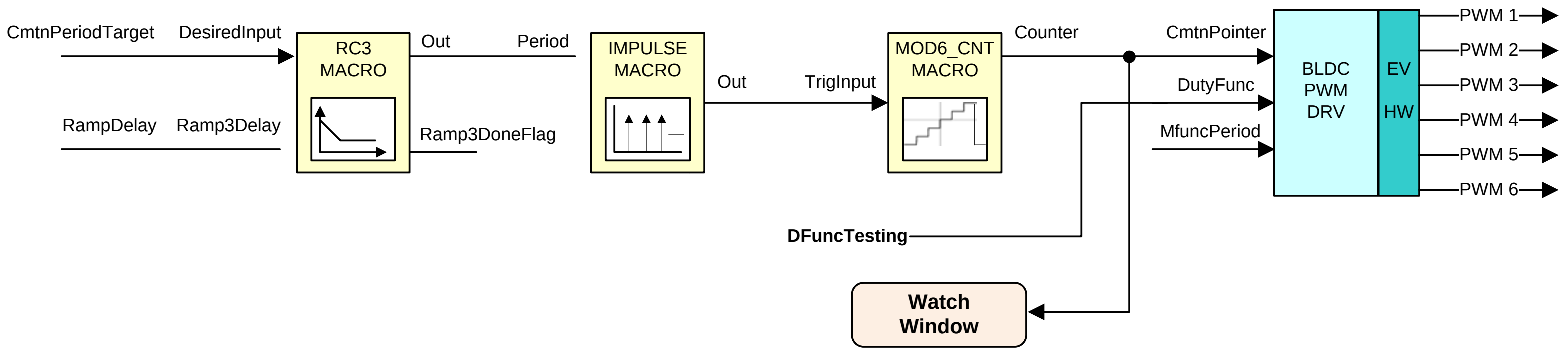


Fig. 10 The PWM outputs , PWM 1 (Yellow) , PWM 2 (Pink) and PWM 5 (Green), PWM 6 (Blue)

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 HVBLDC_Sensorless-Settings.h and select level 1 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.

The key steps can be explained as follows:

Level 2A Open Loop Test

- Compile/load/run program with real time mode and then increase voltage at variac / dc power supply to get the appropriate DC-bus voltage. Now the motor is running with default DFuncTesting 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 it’s 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 DfuncTesting. The parameters DfuncTesting 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 default DfuncTesting and CmtnPeriodTarget values in the initialization section is selected for the motor in HVDMC kit. When a different motor is tested, these values need to be tuned to prevent possible vibration and startup the motor properly. Both DfuncTesting and CmtnPeriodTarget should be adjusted accordingly in the watch windw to increase the motor speed.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 reducing the bus voltage, taking the controller out of realtime mode and reset.

⚠ After verifying this, reduce the DC Bus voltage, take the controller out of real time mode (disable), reset the processor (see “HVMotorCtrl+PFC Kit How To Run Guide” for details). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, hence caution needs to be taken while doing these be taken while doing these experiments.

Level 2B ADC Verification and Offset Calibration

- Verify ADC operation by monitoring dc bus return current and all three back emfs (optional).
- Turn off the power supply and compile/load/run program with real time mode. When the dc bus voltage is zero, the displayed current on the watch window (DCbus_current) should be zero. If not, adjust the offset value in the code by going to :

```
DCbus_current = _IQ12toIQ(AdcResult.ADCRESULT4)-_IQ(0.5);
```

and change IQ15(0.50) offset value (e.g. IQ15(0.5087) or IQ15(0.4988) depending on the sign and amount of the offset. Once this step is completed, turn on the power supply and set the output value to zero.

During level 2, the BLDC Hall Effect sensors' output (if applicable) and PWMDAC outputs should be appeared as follows:

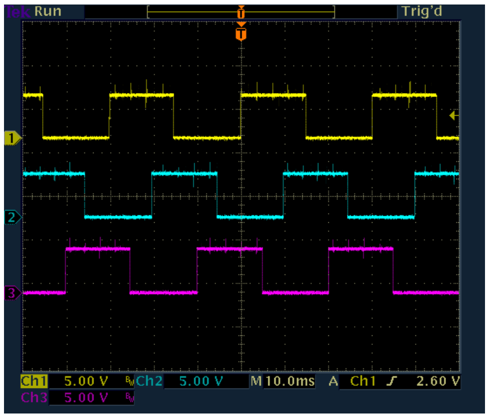


Fig. 11 The outputs of Hall Effect sensors, Hall A, B and C

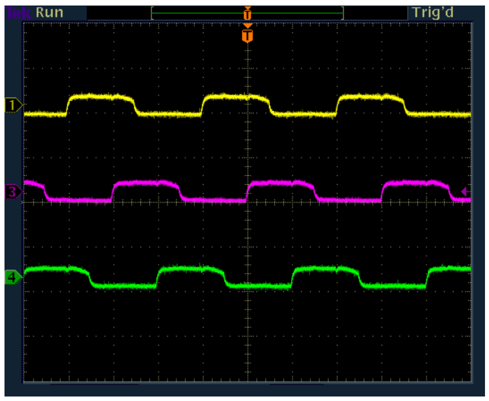


Fig. 12 PWMDAC outputs BemfA, BemfB and BemfC, (VdcBus=160V)

During running this level, the waveforms in the CCS graphs should be appeared as follow:

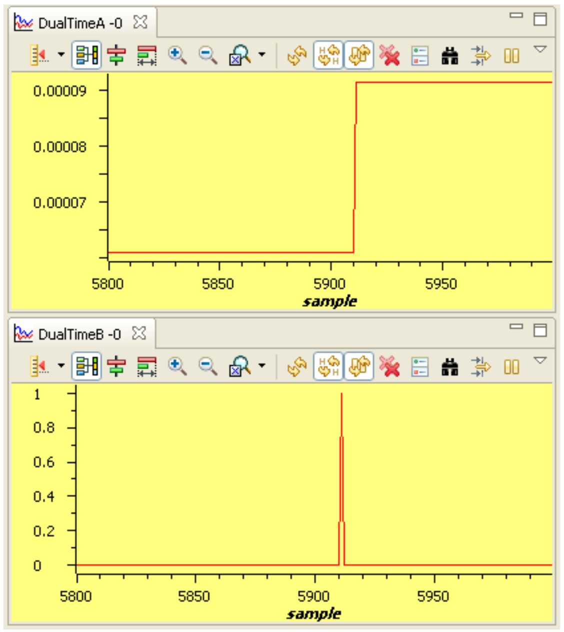


Fig. 13 (a) mod6 counter (b) impulse output (dlog.prescalar=3)

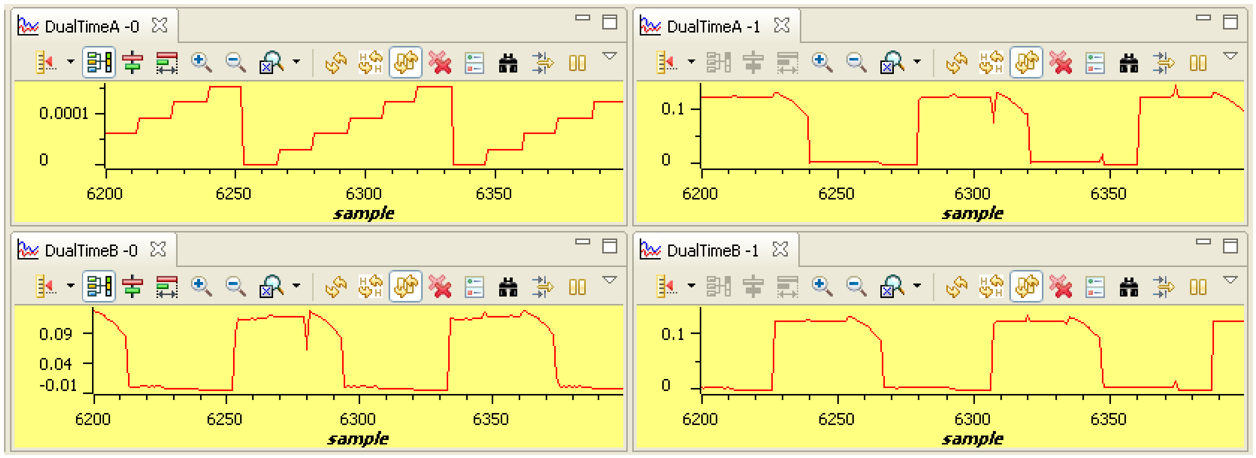
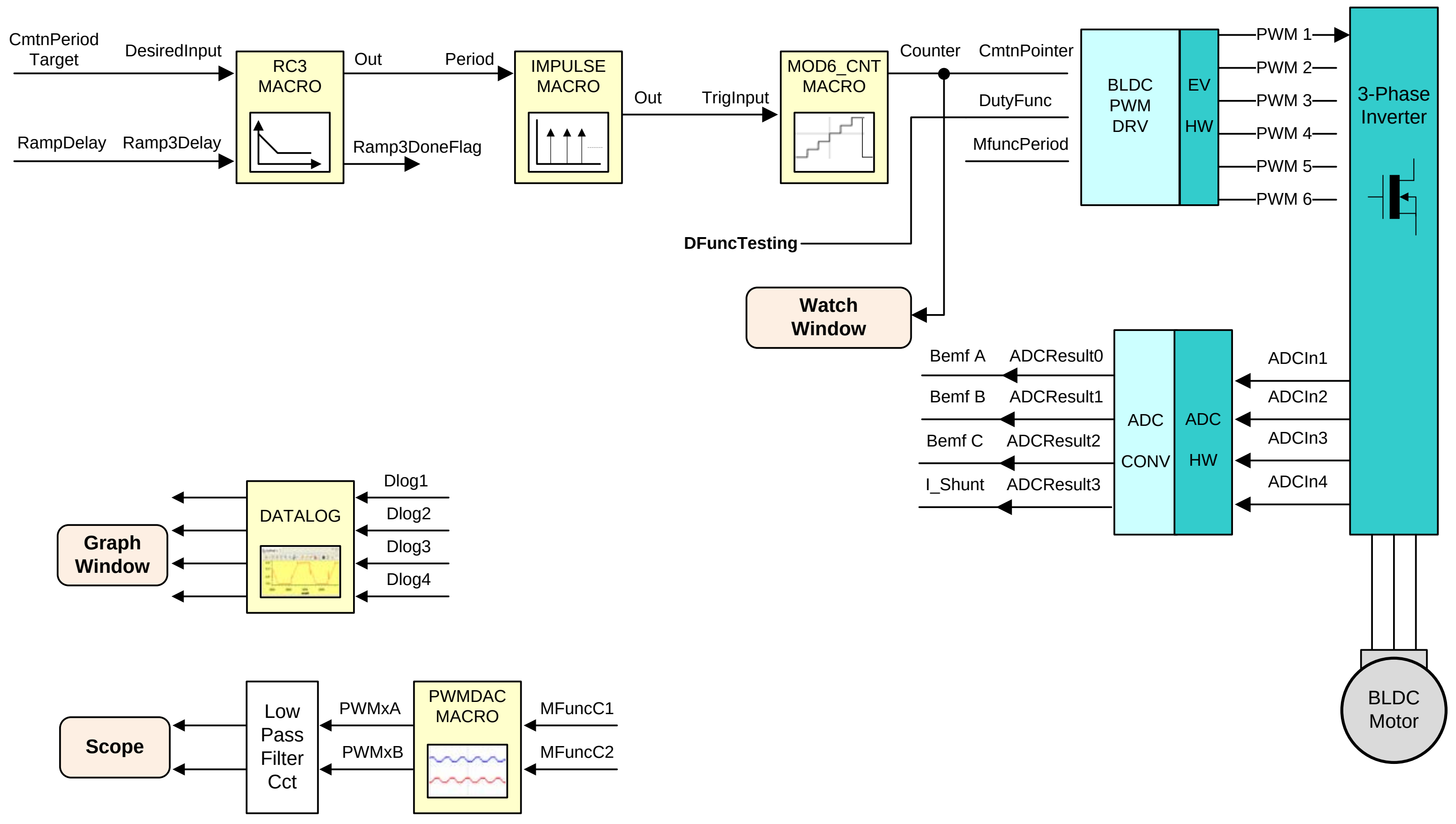


Fig. 14 (a) mod6 counter, (b) BemfA, (c) BemfB and (d)BemfC, (dlog.prescalar = 25 & VdcBus =160V)

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 completed successfully, this section verifies the peripheral independent module COMTN_TRIG. Open HVBLDC_Sensorless-Settings.h and select level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

- RampDelay (Q0 format): for changing the ramping time.
- CmtnPeriodTarget (Q0 format): for changing the targeted commutation interval.

The key steps can be explained as follows:

- Compile/load/run program with real time mode and then increase voltage at variac / dc power supply to get the appropriate DC-bus voltage. Now the motor will gradually speed up and finally run at a constant speed in open loop commutation mode with default DFuncDesired value.

- View COMTN_TRIG module output variables CmtnTrig, Neutral, ZcTrig and DebugBemf from either from graphs window or scope.

- Verify that *Neutral* has been correctly re-constructed and *DebugBemf* shows the 6 "non-energized" Bemf ramps. Zero crossing triggers (*ZcTrig*) should be correctly aligned with zero's of Bemf ramps.

- Commutation triggers (CmtnTrig) should appear 30^o after ZcTrig and be aligned with commutation points, i.e. direction changes of Bemf ramps.

- Bring the system to a safe stop as described below by reducing the bus voltage, taking the controller out of realtime mode and reset.

During running this level, the waveforms at the PWM outputs should appear as follows:

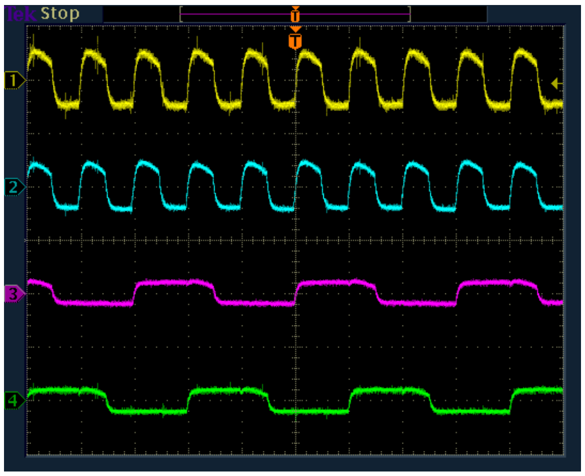
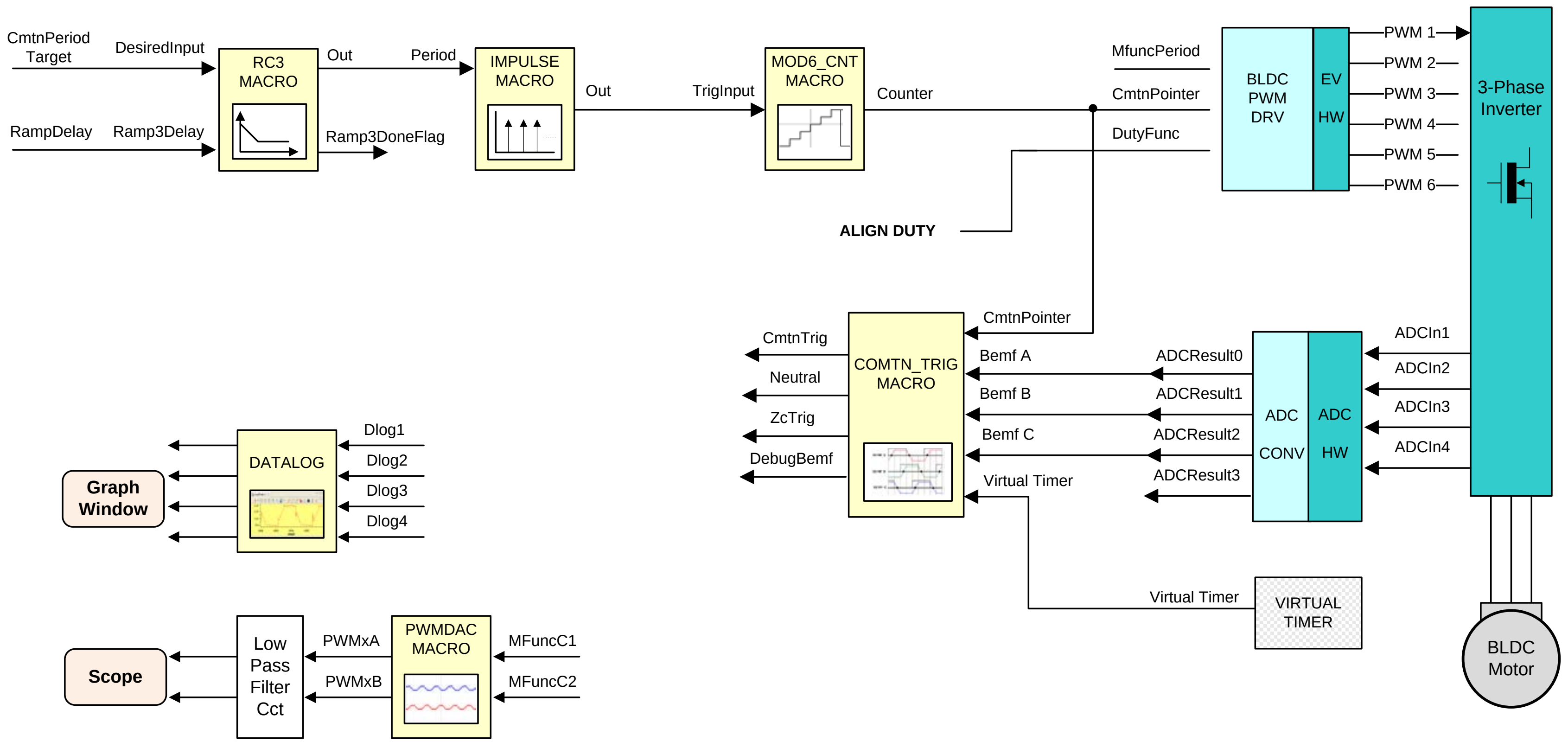


Fig. 15 (a) Neutral, (b) DebugBemf, (c) BemfA and (d)BemfB (VdcBus =160V)

Level 3 Incremental System Build Block Diagram



Level 3 verifies the peripheral independent module COMTN_TRIG

Level 4 Incremental Build

Assuming the previous section is completed successfully, this section verifies the closed loop motor operation based on the computed Bemf zero crossings and the resulting commutation trigger points. Open HVBLDC_Sensorless-Settings.h and select level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4) 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.
- DFuncDesired (Q15 format): changing the PWM duty function in per-unit.

The key steps can be explained as follows:

- Compile/load/run program with real time mode and then increase voltage at variac / dc power supply to get the appropriate DC-bus voltage.
- 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 COMTN_TRIG module.
- When the speed up phase is over, vary the motor speed by changing DFuncDesired. This parameter is used as input to a second ramp control module RMP2CNTL. The output of this module is Out, which controls the duty cycle parameter DutyFunc. This varies the power delivered to the motor and hence it’s speed. The second input to the RMP2CNTL module is Ramp2Delay. This controls the rate at which the ramp value changes. This variable is initialized by the user from the HVBLDC_Sensorless.c file.
- Bring the system to a safe stop as described below by reducing the bus voltage, taking the controller out of realtime mode and reset.

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

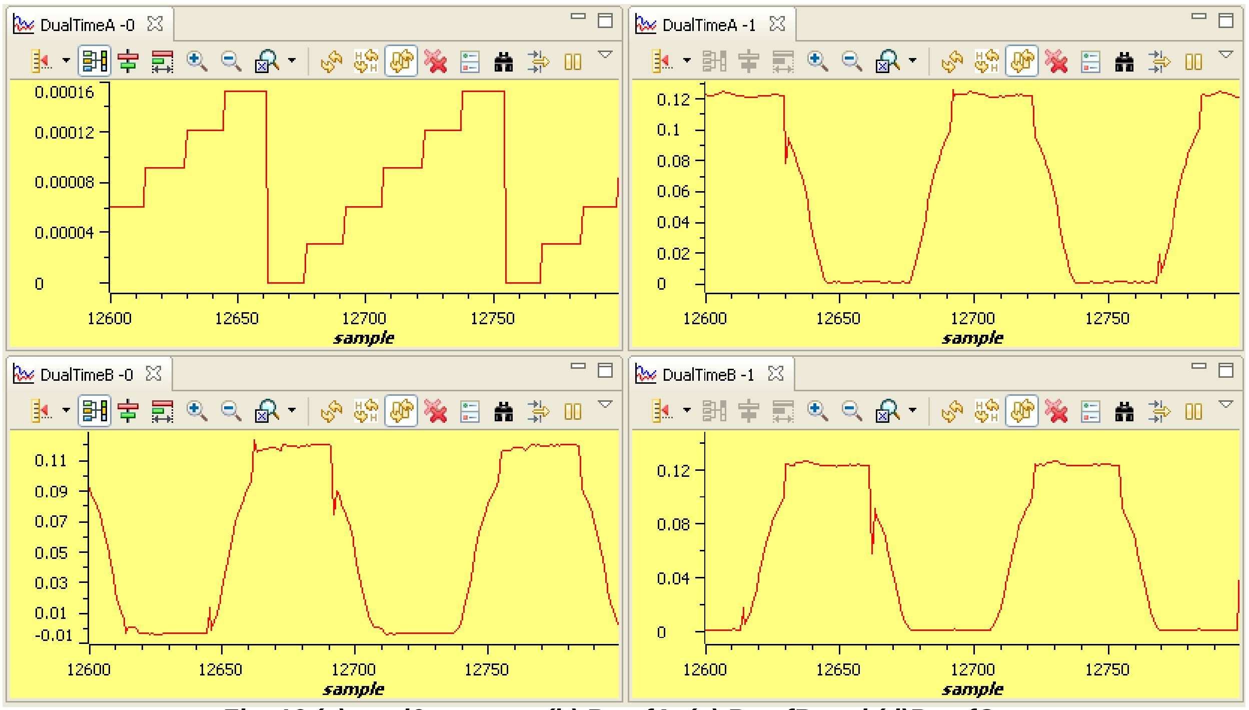


Fig. 16 (a) mod6 counter, (b) BemfA, (c) BemfB and (d)BemfC (dlog.prescalar=25 &Vdcbus=160V)

PWMDAC outputs should appear as follows on the scope:

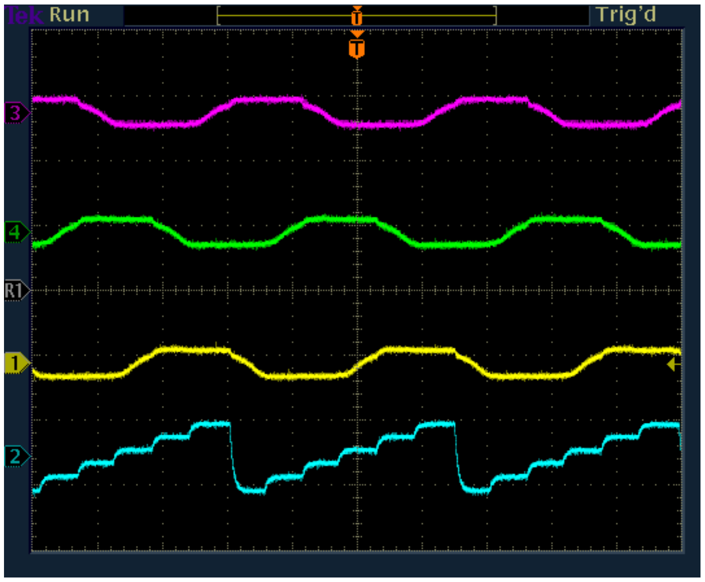
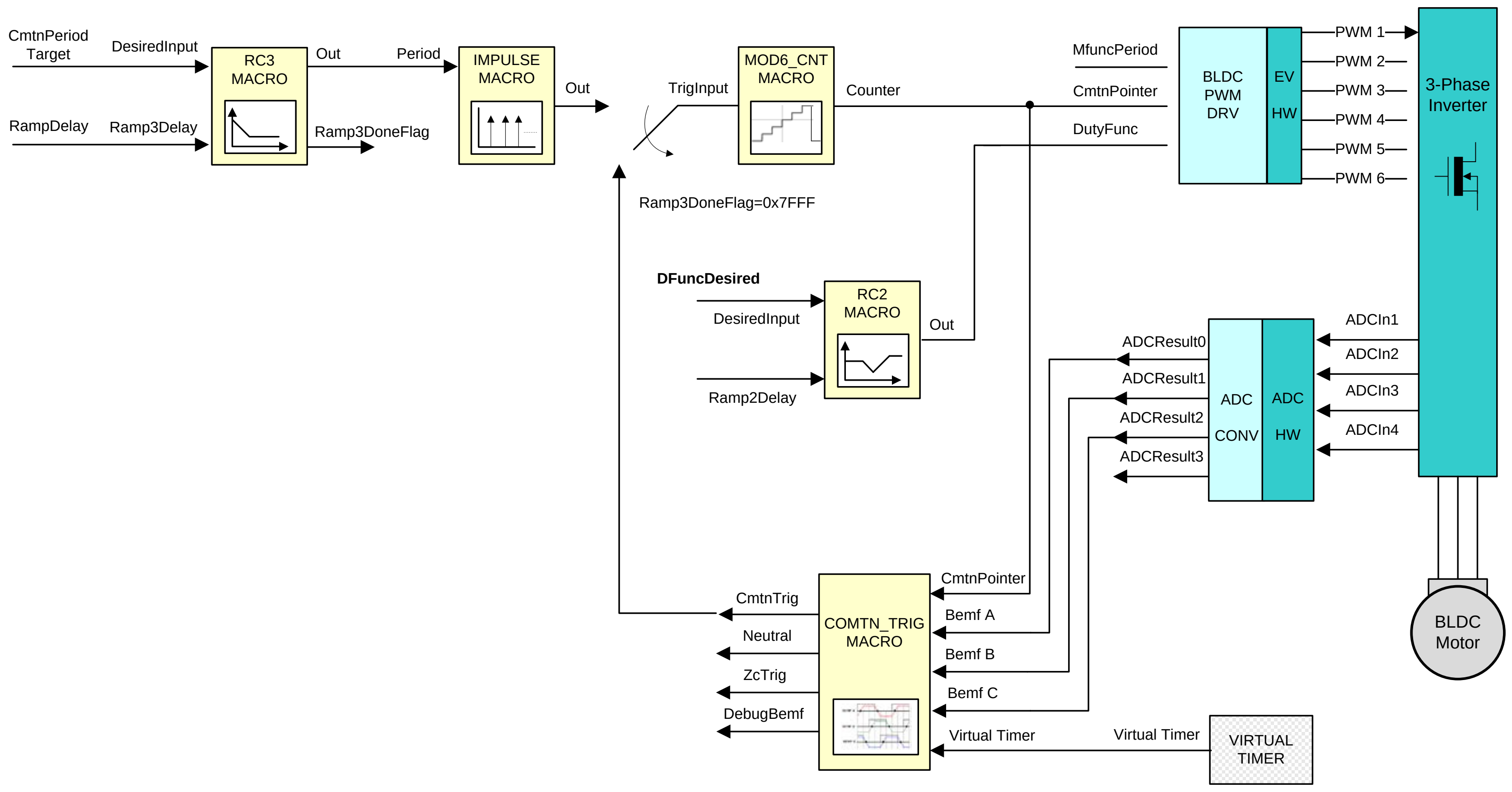


Fig. 17 PWMDAC outputs a) BemfA, (b) BemfB, (c)BemfC and (d) mod6 counter (Vdcbus=160V)

Level 4 Incremental System Build Block Diagram



Level 4 verifies the closed loop motor operation based on the computed Bemf zero crossings and the resulting commutation trigger points.

Level 5 Incremental Build

Assuming the previous section is completed successfully, this section verifies the closed current loop and current PI controller. Open HVBLDC_Sensorless-Settings.h and select level 5 incremental build option by setting the BUILDLEVEL to LEVEL5 (#define BUILDLEVEL LEVEL5). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

- DFuncDesired (Q15 format): for changing the PWM duty cycle in per-unit.
- CurrentSet (GLOBAL_Q format): for changing the reference DC-bus current in per-unit.
- ILoopFlag (Q0 format): for switching between fixed duty-cycle and controlled Idc duty-cycle.

The steps are explained as follows:

- Compile/load/run program with real time mode and then increase voltage at variac / dc power supply to get the appropriate DC-bus voltage.

- The motor will gradually speed up and finally switch to closed loop commutation mode.

- Increase/decrease the motor speed by changing DFuncDesired.

- Now use the variable CurrentSet to specify the reference current for the PI controller PID_REG3. Once the ClosedFlag set to 1 in the code, change ILoopflag to 1 to activate the current loop PI controller. Once this is done, the PI controller will start to regulate the DC bus current and hence the motor current. Gradually increase/decrease the command current (CurrentSet 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

- Bring the system to a safe stop as described at the end of build 1 by reducing the bus voltage, taking the controller out of realtime mode and reset.

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

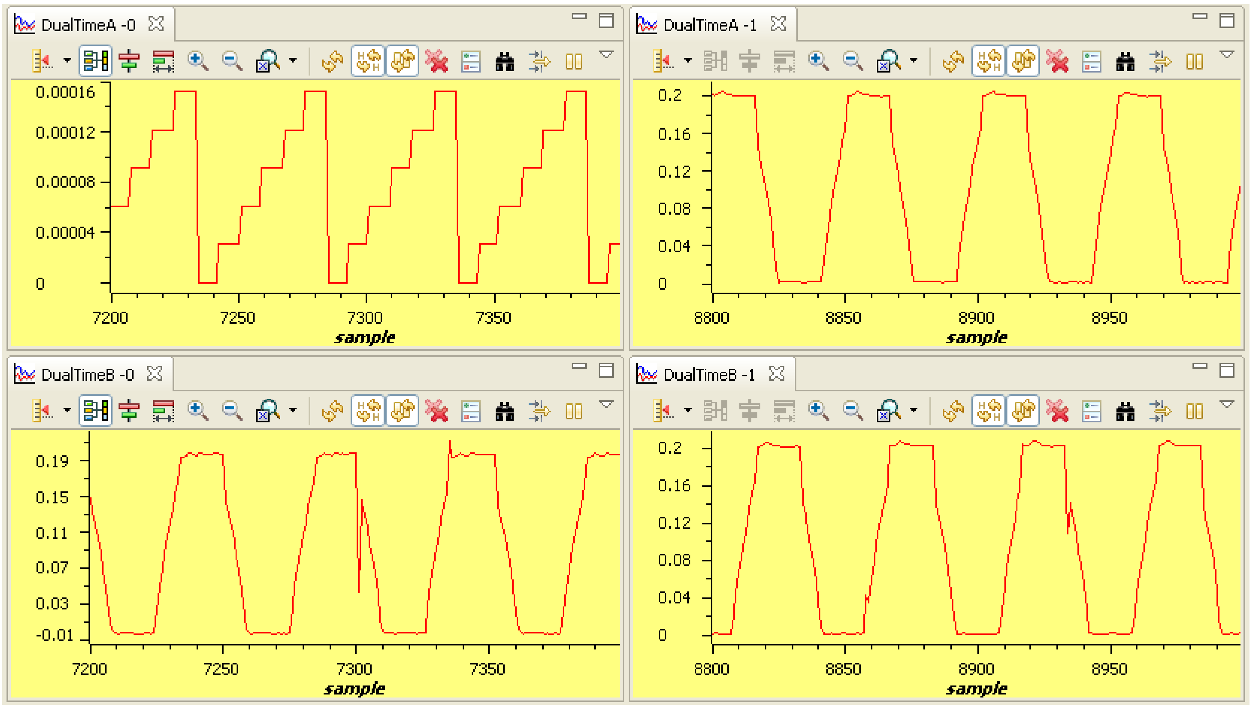


Fig. 18 a) mod 6 counter (b) BemfA, (c) BemfB, (d)BemfC (dlog.presalar = 25 & Vdcbus=160V)

PWMDAC outputs should appear as follows on the scope:

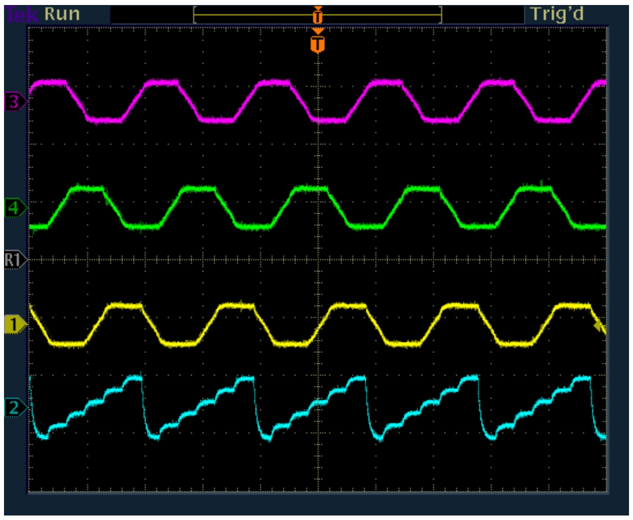
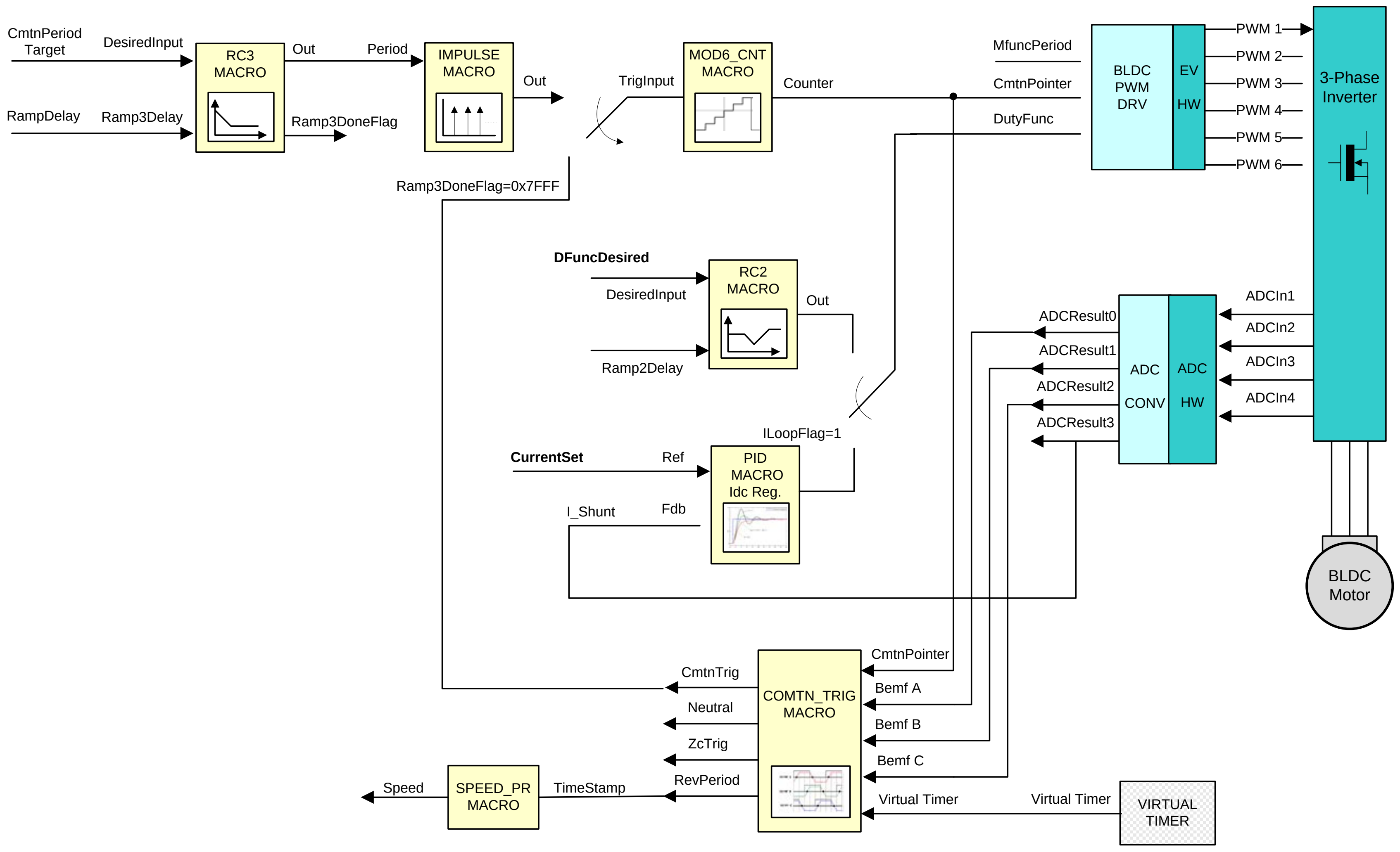


Fig. 19 PWMDAC outputs a) BemfA, (b) BemfB, (c)BemfC and (d) mod6 counter (Vdcbus=160V)

Level 5 Incremental System Build Block Diagram



Level 5 verifies the closed current loop and current PI controller.

Level 6 Incremental Build

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

- SpeedRef (GLOBAL_Q format): for changing the reference DC-bus current in per-unit.

The steps are explained as follows:

- Compile/load/run program with real time mode and then increase voltage at variac / dc power supply to get the appropriate DC-bus voltage.

- 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 at the end of build 1 by reducing the bus voltage, taking the controller out of realtime mode and reset.

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

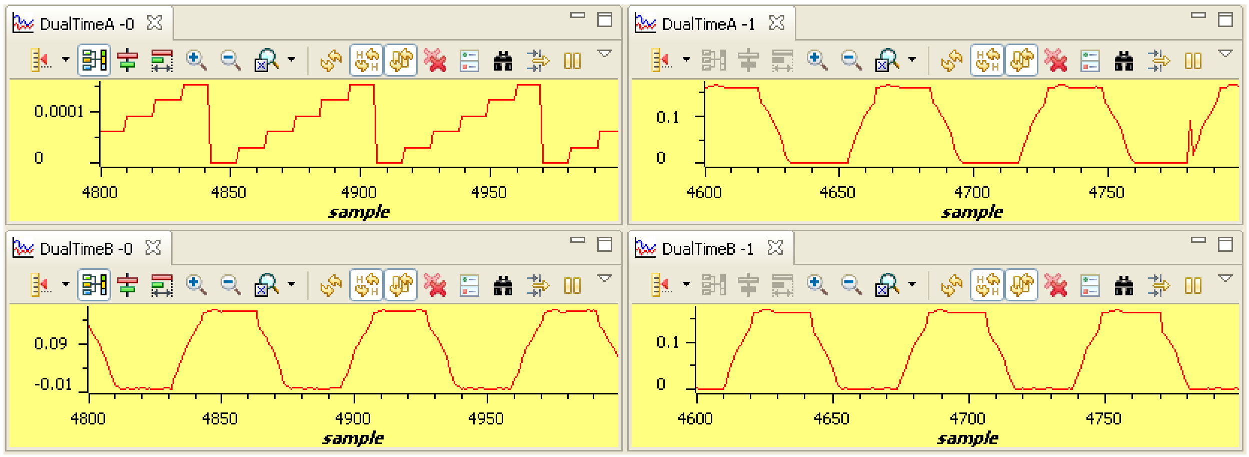


Fig. 20 a) mod 6 counter (b) BemfA, (c) BemfB, (d)BemfC
(at 0.25 pu speed under no-load, Vdcbus=160V)

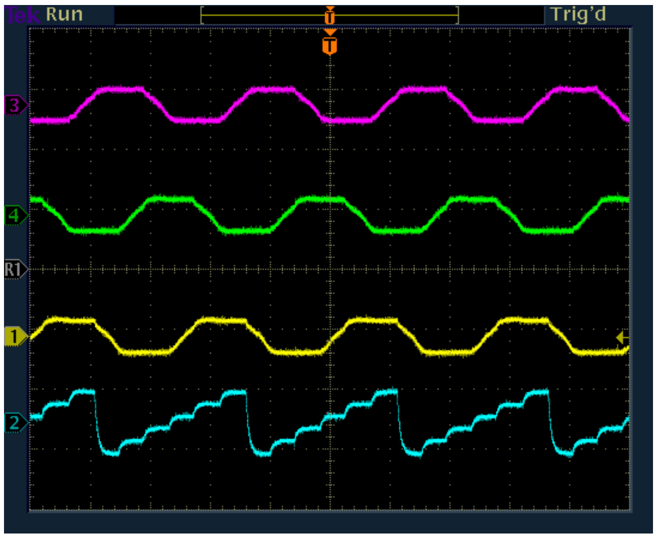
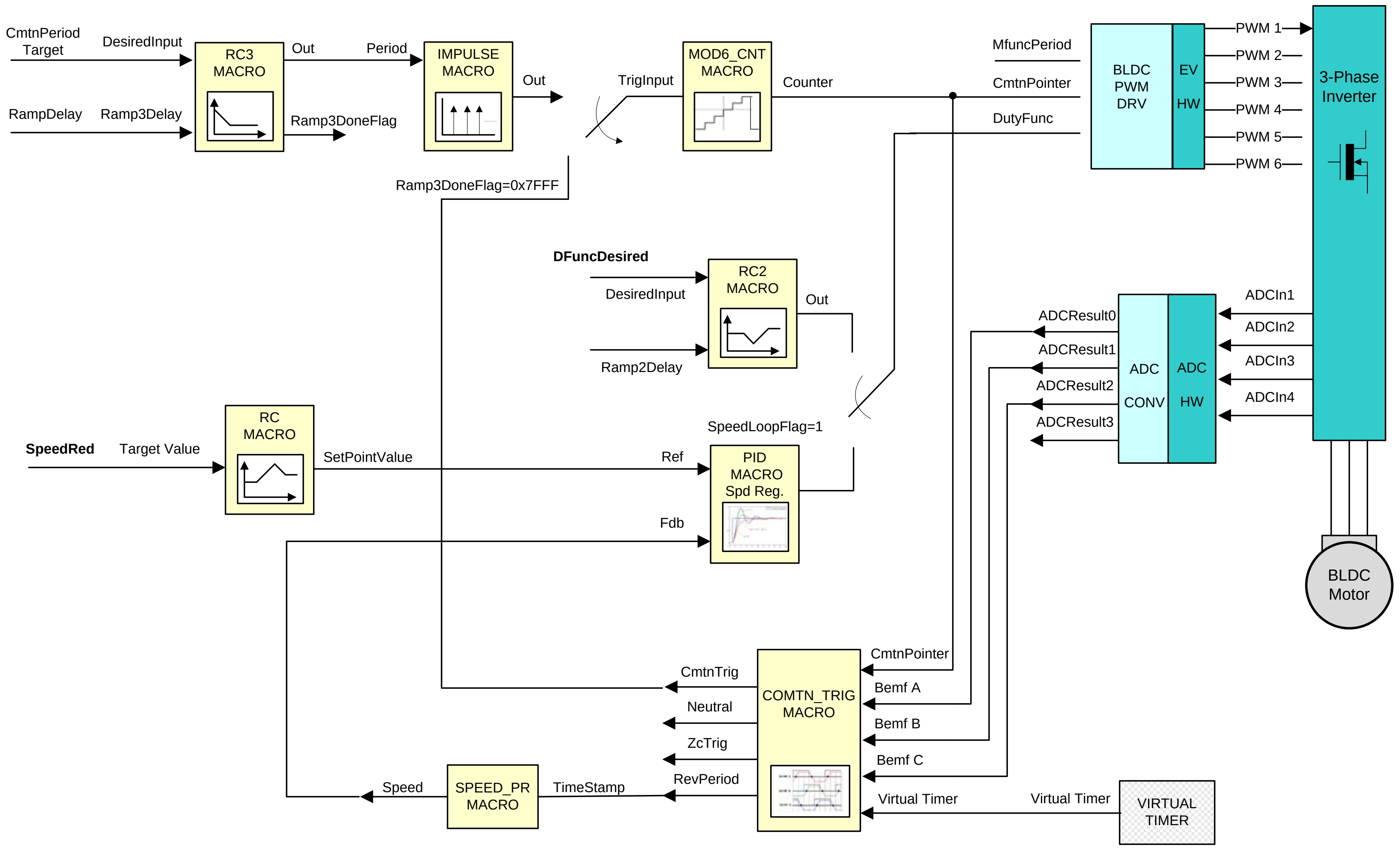


Fig. 21 PWMDAC outputs a) BemfA, (b) BemfB, (c)BemfC and (d) mod6 counter
(at 0.25 pu speed under no-load, Vdcbus=160V)

Level 6 Incremental System Build Block Diagram



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