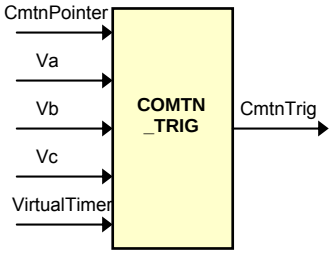
COMTN_TRIG	*Commutation Trigger Generator for BLDC Drive*

Description

This module determines the Bemf zero crossing points of a 3-ph BLDC motor based on motor phase voltage measurements and then generates the commutation trigger points for the 3-ph power inverter switches.



Availability

This IQ module is available in one interface format:

1) The C interface version

Module Properties

Type: Target Independent, Application Independent

Target Devices: x281x or x280x

C Version File Names: com_trig.c, com_trig.h

IQmath library files for C: IQmathLib.h, IQmath.lib

Item	C version	Comments
Code Size□ (x281x/x280x)	396/396 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" CMTN structure consumes 42 words in the data memory

□ Code size mentioned here is the size of the ***calc()*** function

C Interface

Object Definition

The structure of CMTN object is defined by following structure definition

```
typedef  struct { Uint32 CmtnTrig;            // Output: Commutation trigger output (0 or 00007FFF)
                  _iq Va;                     // Input: Motor phase a voltages referenced to GND
                  _iq Vb;                     // Input: Motor phase b voltages referenced to GND
                  _iq Vc;                     // Input: Motor phase c voltages referenced to GND
                  _iq Neutral;                // Variable: 3*Motor netural voltage
                  Uint32 RevPeriod;           // Variable: revolution time counter (Q0)
                  Uint32 ZcTrig;              // Variable: Zero-Crossing trig flag (0 or 00007FFF)
                  Uint32 CmtnPointer;         // Input: Commutation state pointer input (Q0)
                  _iq DebugBemf;              // Variable: 3*Back EMF = 3*(vx=vn), x=a,b,c
                  Uint32 NoiseWindowCounter; // Variable: Noise windows counter (Q0)
                  Uint32 Delay30DoneFlag;     // Variable: 30 Deg delay flag (0 or 0000000F)
                  Uint32 NewTimeStamp;        // Variable: Time stamp (Q0)
                  Uint32 OldTimeStamp;        // History: Previous time stamp (Q0)
                  Uint32 VirtualTimer;        // Input: Virtual timer (Q0)
                  Uint32 CmtnDelay;           // Variable: Time delay (Q0)
                  Uint32 DelayTaskPointer;  // Variable: Delay task pointer, see note below (0 or 1)
                  Uint32 NoiseWindowMax;  // Variable: Maximum noise windows counter (Q0)
                  Uint32 CmtnDelayCounter;   // Variable: Time delay counter (Q0)
                  Uint32 NWDelta;             // Variable: Noise windows delta (Q0)
                  Uint32 NWDelayThres;        // Variable: Noise windows dynamic threshold (Q0)
                  void (*calc)();             // Pointer to calculation function
                } CMTN;

typedef CMTN *CMTN_handle;
```

Item	Name	Description	Format*	Range(Hex)
Inputs	CmtnPointer	Commutation state pointer input. This is used for Bemf zero crossing point calculation for the appropriate motor phase.	Q0	0 - 5
	Va	Motor phase-a voltages referenced to GND.	GLOBAL_Q	00000000-7FFFFFFF
	Vb	Motor phase-b voltages referenced to GND.	GLOBAL_Q	00000000-7FFFFFFF
	Vc	Motor phase-c voltages referenced to GND.	GLOBAL_Q	00000000-7FFFFFFF
	VirtualTimer	A virtual timer used for commutation delay angle calculation.	Q0	80000000-7FFFFFFF
Output	CmtnTrig	Commutation trigger output.	Q0	0 or 00007FFF
Internal	Neutral	3*Motor netural voltage	GLOBAL_Q	80000000-7FFFFFFF
	RevPeriod	revolution time counter	Q0	00000000-7FFFFFFF
	ZcTrig	Zero-Crossing trig flag	Q0	0 or 00007FFF
	DebugBemf	3*Back EMF	GLOBAL_Q	80000000-7FFFFFFF
	NoiseWindowCounter	Noise windows counter	Q0	80000000-7FFFFFFF
	Delay30DoneFlag	30 Deg delay flag	Q0	0 or 0000000F
	NewTimeStamp	Time stamp	Q0	00000000-7FFFFFFF
	OldTimeStamp	Previous time stamp	Q0	00000000-7FFFFFFF
	CmtnDelay	Time delay in terms of number of sampling time periods	Q0	00000000-7FFFFFFF
	DelayTaskPointer	Delay task pointer	Q0	0 or 1
	NoiseWindowMax	Maximum noise windows counter	Q0	80000000-7FFFFFFF
	CmtnDelayCounter	Time delay counter	Q0	80000000-7FFFFFFF
	NWDelta	Noise windows delta	Q0	80000000-7FFFFFFF
	NWDelayThres	Noise windows dynamic threshold	Q0	80000000-7FFFFFFF

* GLOBAL_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

Special Constants and Data types

CMTN

The module definition is created as a data type. This makes it convenient to instance an interface to ramp generator. To create multiple instances of the module simply declare variables of type CMTN.

CMTN_handle

User defined Data type of pointer to CMTN module

CMTN_DEFAULTS

Structure symbolic constant to initialize CMTN module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void cmtn_trig_calc(CMTN_handle);

This definition implements one method viz., the commutation trigger generator function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two CMTN objects
CMTN cm_trig1, cm_trig2;

Initialization
To Instance pre-initialized objects
CMTN cm_trig1 = CMTN_DEFAULTS;
CMTN cm_trig2 = CMTN_DEFAULTS;

Invoking the computation function
cm_trig1.calc(&cm_trig1);
cm_trig2.calc(&cm_trig2);

Example

The following pseudo code provides the information about the module usage.

```
main()
{

}

void interrupt periodic_interrupt_isr()
{
        cm_trig1.CmtnPointer = input11;         // Pass inputs to cm_trig1
        cm_trig1.Va = input12;                  // Pass inputs to cm_trig1
        cm_trig1.Vb = input13;                  // Pass inputs to cm_trig1
        cm_trig1.Vc = input14;                  // Pass inputs to cm_trig1
        cm_trig1.VirtualTimer = input15;        // Pass inputs to cm_trig1

        cm_trig2.CmtnPointer = input21;         // Pass inputs to cm_trig2
        cm_trig2.Va = input22;                  // Pass inputs to cm_trig2
        cm_trig2.Vb = input23;                  // Pass inputs to cm_trig2
        cm_trig2.Vc = input24;                  // Pass inputs to cm_trig2
        cm_trig2.VirtualTimer = input25;        // Pass inputs to cm_trig2

        cm_trig1.calc(&cm_trig1);               // Call compute function for cm_trig1
        cm_trig2.calc(&cm_trig2);               // Call compute function for cm_trig2

        out1 = cm_trig1.CmtnTrig;               // Access the outputs of cm_trig1
        out2 = cm_trig2.CmtnTrig;               // Access the outputs of cm_trig2
}
```

Technical Background

Figure 1 shows the 3-phase power inverter topology used to drive a 3-phase BLDC motor. In this arrangement, the motor and inverter operation is characterized by a two phase ON operation. This means that two of the three phases are always energized, while the third phase is turned off.

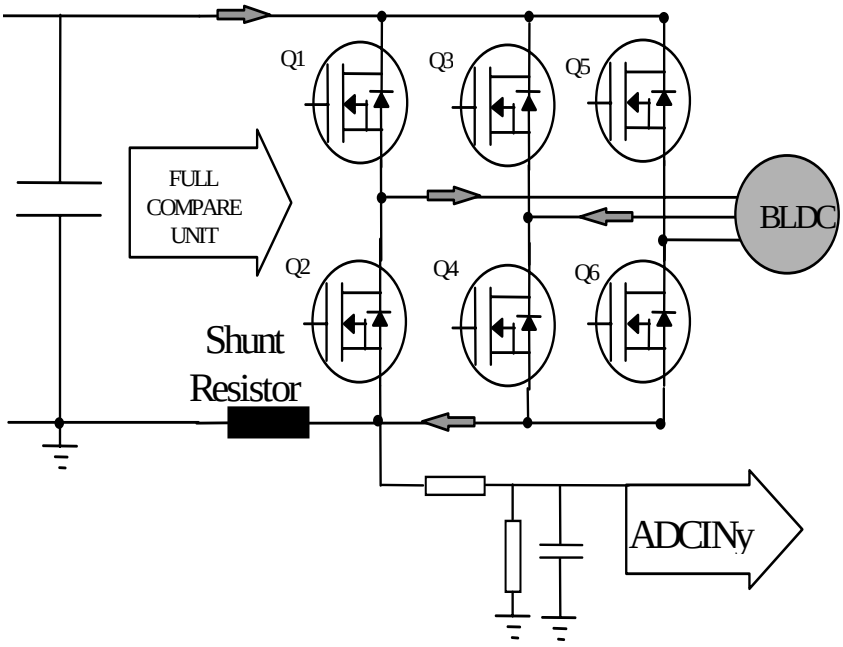


Figure 1: Three Phase Power Inverter for a BLDC Motor Drive

The bold arrows on the wires indicate the Direct Current flowing through two motor stator phases. For sensorless control of BLDC drives it is necessary to determine the zero crossing points of the three Bemf voltages and then generate the commutation trigger points for the associated 3-ph power inverter switches.

The figure below shows the basic hardware necessary to perform these tasks.

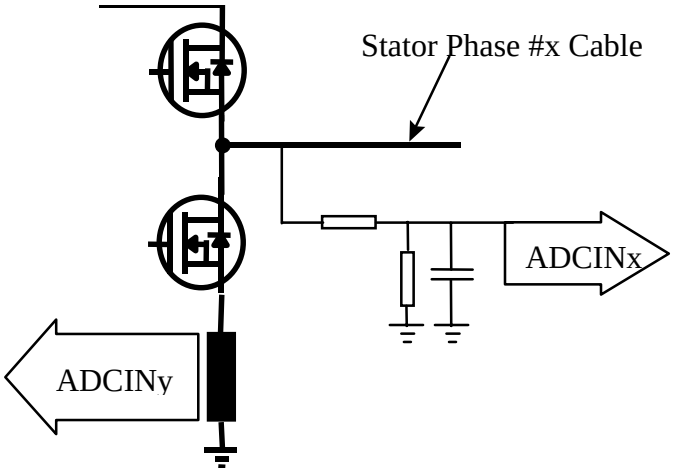


Figure 2: Basic Sensorless Additional Hardware

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). The sensorless algorithm is based only on the three motor terminal voltage measurements and thus requires only four ADC input lines.

Figure 3 shows the motor terminal model for phase A, where L is the phase inductance, R is the phase resistance, Ea is the back electromotive force, Vn is the star connection voltage referenced to ground and Va is the phase voltage referenced to ground. Va voltages are measured by means of the DSP controller ADC Unit and via the voltage sense circuit shown in Figure 2.

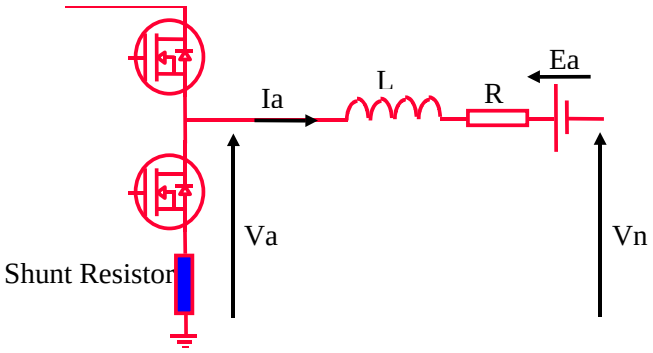


Figure 3: Stator Terminal Electrical Model

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 4. 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*.

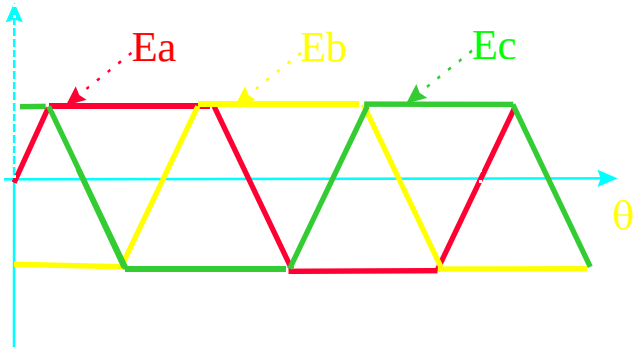


Figure 4: Instantaneous Bemf Wave-forms

.

Bemf Zero Crossing Point Computation

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, e.g., once every 16.7μs (60kHz sampling loop).

Electrical Behaviour 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 overcomed 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 30º 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 30º 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 360º 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 30º. The corresponding time delay is represented in terms of the number of sampling time periods and is stored in the variable *CmtnDelay*. Therefore,

Time delay = *CmtnDelay* **.**Ts = T(α/360) = *VirtualTimer***.**Ts(α/360) = *VirtualTimer* **.** Ts/12

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

The above equation is further simplified as,

CmtnDelay = *VirtualTimer* /12

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