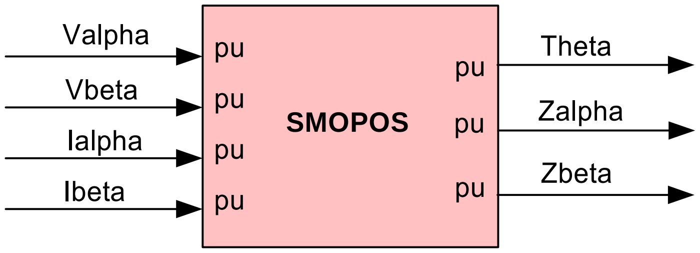
SMOPOS — *Sliding-Mode Rotor Position Observer of PMSM*

Description

This software module implements a rotor position estimation algorithm for Permanent-Magnet Synchronous Motor (PMSM) based on Sliding-Mode Observer (SMO).

.



Availability

This IQ module is available in one interface format:

1) The C interface version

Module Properties

Type: Target Independent, Application Dependent

Target Devices: x281x or x280x

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

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

Item	C version	Comments
Code Size□ (x281x/x280x)	221/221 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" SMOPOS structure consumes 36 words in the data memory

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

C Interface

Object Definition

The structure of SMOPOS object is defined by following structure definition

```
typedef struct {  _iq  Valpha;      // Input: Stationary alpha-axis stator voltage
                  _iq  Ealpha;      // Variable: Stationary alpha-axis back EMF
                  _iq  Zalpha;      // Output: Stationary alpha-axis sliding control
                  _iq  Gsmopos;     // Parameter: Motor dependent control gain
                  _iq  EstIalpha;   // Variable: Estimated stationary alpha-axis stator current
                  _iq  Fsmopos;     // Parameter: Motor dependent plant matrix
                  _iq  Vbeta;       // Input: Stationary beta-axis stator voltage
                  _iq  Ebeta;       // Variable: Stationary beta-axis back EMF
                  _iq  Zbeta;       // Output: Stationary beta-axis sliding control
                  _iq  EstIbeta;    // Variable: Estimated stationary beta-axis stator current
                  _iq  Ialpha;      // Input: Stationary alpha-axis stator current
                  _iq  IalphaError; // Variable: Stationary alpha-axis current error
                  _iq  Kslide;      // Parameter: Sliding control gain
                  _iq  Ibeta;       // Input: Stationary beta-axis stator current
                  _iq  IbetaError;  // Variable: Stationary beta-axis current error
                  _iq  Kslf;        // Parameter: Sliding control filter gain
                  _iq  Theta;       // Output: Compensated rotor angle
                  void  (*calc)();  // Pointer to calculation function
               } SMOPOS;

typedef SMOPOS * SMOPOS_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format*	Range(Hex)
Inputs	Valpha	stationary d-axis stator voltage	GLOBAL_Q	80000000-7FFFFFFF
	Vbeta	stationary q-axis stator voltage	GLOBAL_Q	80000000-7FFFFFFF
	Ialpha	stationary d-axis stator current	GLOBAL_Q	80000000-7FFFFFFF
	Ibeta	stationary q-axis stator current	GLOBAL_Q	80000000-7FFFFFFF
Outputs	Theta	rotor position angle	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
	Zalfa	stationary d-axis sliding control	GLOBAL_Q	80000000-7FFFFFFF
	Zbeta	stationary q-axis sliding control	GLOBAL_Q	80000000-7FFFFFFF
SMOPOS parameter	Fsmopos	Fsmopos = exp(-Rs*T/Ls)	GLOBAL_Q	80000000-7FFFFFFF
	Gsmopos	Gsmopos = (Vb/Ib)*(1- exp(-Rs*T/Ls))/Rs	GLOBAL_Q	80000000-7FFFFFFF
	Kslide	sliding mode control gain	GLOBAL_Q	80000000-7FFFFFFF
	Kslf	sliding control filter gain	GLOBAL_Q	80000000-7FFFFFFF
Internal	Ealpha	stationary d-axis back EMF	GLOBAL_Q	80000000-7FFFFFFF
	Ebeta	stationary q-axis back EMF	GLOBAL_Q	80000000-7FFFFFFF
	EstIalpha	stationary d-axis estimated current	GLOBAL_Q	80000000-7FFFFFFF
	EstIbeta	stationary q-axis estimated current	GLOBAL_Q	80000000-7FFFFFFF
	IalphaError	stationary d-axis current error	GLOBAL_Q	80000000-7FFFFFFF
	IbetaError	stationary q-axis current error	GLOBAL_Q	80000000-7FFFFFFF

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

Special Constants and Data types

SMOPOS
The module definition is created as a data type. This makes it convenient to instance an interface to the sliding-mode rotor position observer of Permanent-Magnet Synchronous Motor module. To create multiple instances of the module simply declare variables of type SMOPOS.

SMOPOS_handle
User defined Data type of pointer to SMOPOS module

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

Methods

void smopos_calc(SMOPOS_handle);

This definition implements one method viz., the sliding-mode rotor position observer of Permanent-Magnet Synchronous Motor computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two SMOPOS objects

```
SMOPOS  smo1, smo2;
```

Initialization
To Instance pre-initialized objects

```
SMOPOS fe1 = SMOPOS_DEFAULTS;
SMOPOS fe2 = SMOPOS_DEFAULTS;
```

Invoking the computation function

```
smo1.calc(&smo1);
smo2.calc(&smo2);
```

Example

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

```
main()
{
        smo1.Fsmopos = parem1_1;            // Pass parameters to smo1
        smo1.Gsmopos = parem1_2;            // Pass parameters to smo1
        smo1.Kslide = parem1_3;             // Pass parameters to smo1
        smo1.Kslf = parem1_4;               // Pass parameters to smo1
        smo2.Fsmopos = parem2_1;              // Pass parameters to smo2
        smo2.Gsmopos = parem2_2;              // Pass parameters to smo2
        smo2.Kslide = parem2_3;               // Pass parameters to smo2
        smo2.Kslf = parem2_4;                 // Pass parameters to smo2

}

void interrupt periodic_interrupt_isr()
{
        smo1.Valpha = voltage_dq1.d;          // Pass inputs to smo1
        smo1.Vbeta = voltage_dq1.q;           // Pass inputs to smo1
        smo1.Ialpha =current_dq1.d;           // Pass inputs to smo1
        smo1.Ibeta =current_dq1.q;            // Pass inputs to smo1

        smo2.Valpha = voltage_dq2.d;          // Pass inputs to smo2
        smo2.Vbeta = voltage_dq2.q;           // Pass inputs to smo2
        smo2.Ialpha =current_dq2.d;           // Pass inputs to smo2
        smo2.Ibeta =current_dq2.q;            // Pass inputs to smo2

        smopos1.calc(&smopos1)                // Call compute function for smopos1
        smopos2.calc(&smopos2);               // Call compute function for smopos2

        angle1 = smopos1.Theta;               // Access the outputs of smopos1
        angle2 = smopos2.Theta;               // Access the outputs of smopos2

}
```

Constant Computation Function

Since the sliding-mode rotor position observer of Permanent-Magnet Synchronous Motor module requires two constants (Fsmopos and Gsmopos) to be input basing on the machine parameters, base quantities, mechanical parameters, and sampling period. These two constants can be internally computed by the C function (smopos_const.c, smopos_const.h). The followings show how to use the C constant computation function.

Object Definition

The structure of SMOPOS_CONST object is defined by following structure definition

```
typedef struct    { float32  Rs;               // Input: Stator resistance (ohm)
                    float32  Ls;               // Input: Stator inductance (H)
                    float32  Ib;               // Input: Base phase current (amp)
                    float32  Vb;               // Input: Base phase voltage (volt)
                    float32  Ts;               // Input: Sampling period in sec
                    float32  Fsmopos;          // Output: constant using in observed current cal.
                    float32  Gsmopos;          // Output: constant using in observed current cal.
                    void   (*calc)();          // Pointer to calculation function
                  } SMOPOS_CONST;

typedef SMOPOS_CONST *SMOPOS_CONST_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format	Range(Hex)
Inputs	Rs	Stator resistance (ohm)	Floating	N/A
	Ls	Stator inductance (H)	Floating	N/A
	Ib	Base phase current (amp)	Floating	N/A
	Vb	Base phase voltage (volt)	Floating	N/A
	Ts	Sampling period (sec)	Floating	N/A
Outputs	Fsmopos	constant using in observed current calculation	Floating	N/A
	Gsmopos	constant using in observed current calculation	Floating	N/A

Special Constants and Data types

SMOPOS_CONST
The module definition is created as a data type. This makes it convenient to instance an interface to the sliding-mode rotor position observer of Permanent-Magnet Synchronous Motor constant computation module. To create multiple instances of the module simply declare variables of type SMOPOS_CONST.

SMOPOS_CONST_handle
User defined Data type of pointer to SMOPOS_CONST module

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

Methods

void smopos_const_calc(SMOPOS_CONST_handle);

This definition implements one method viz., the sliding-mode rotor position observer of Permanent-Magnet Synchronous Motor constant computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two SMOPOS_CONST objects
SMOPOS_CONST smopos1_const, smopos2_const;

Initialization
To Instance pre-initialized objects
SMOPOS_CONST smopos1_const = SMOPOS_CONST_DEFAULTS;
SMOPOS_CONST smopos2_const = SMOPOS_CONST_DEFAULTS;

Invoking the computation function
smopos1_const.calc(&smopos1_const);
smopos2_const.calc(&smopos2_const);

Example

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

```
main()
{

        smopos1_const.Rs = Rs1;         // Pass floating-point inputs to smopos1_const
        smopos1_const.Ls = Ls1;         // Pass floating-point inputs to smopos1_const
        smopos1_const.Ib = Ib1;         // Pass floating-point inputs to smopos1_const
        smopos1_const.Vb = Vb1;         // Pass floating-point inputs to smopos1_const
        smopos1_const.Ts = Ts1;         // Pass floating-point inputs to smopos1_const

        smopos2_const.Rs = Rs2;         // Pass floating-point inputs to smopos2_const
        smopos2_const.Ls = Ls2;         // Pass floating-point inputs to smopos2_const
        smopos2_const.Ib = Ib2;         // Pass floating-point inputs to smopos2_const
        smopos2_const.Vb = Vb2;         // Pass floating-point inputs to smopos2_const
        smopos2_const.Ts = Ts2;         // Pass floating-point inputs to smopos2_const

        smopos1_const.calc(&smopos1_const); // Call compute function for smopos1_const
        smopos2_const.calc(&smopos2_const); // Call compute function for smopos2_const

        // Access the outputs of smopos1_const
        smopos1.Fsmopos = _IQ(smopos1_const.Fsmopos);
        smopos1.Gsmopos = _IQ(smopos1_const.Gsmopos);
        // Access the outputs of smopos2_const
        smopos2.Fsmopos = _IQ(smopos2_const.Fsmopos);
        smopos2.Gsmopos = _IQ(smopos2_const.Gsmopos);
}
```

Technical Background

Figure 1 is an illustration of a permanent-magnet synchronous motor control system based on field orientation principle. The basic concept of field orientation is based on knowing the position of rotor flux and positioning the stator current vector at orthogonal angle to the rotor flux for optimal torque output. The implementation shown in Figure 1 derives the position of rotor flux from encoder feedback. However, the encoder increases system cost and complexity.

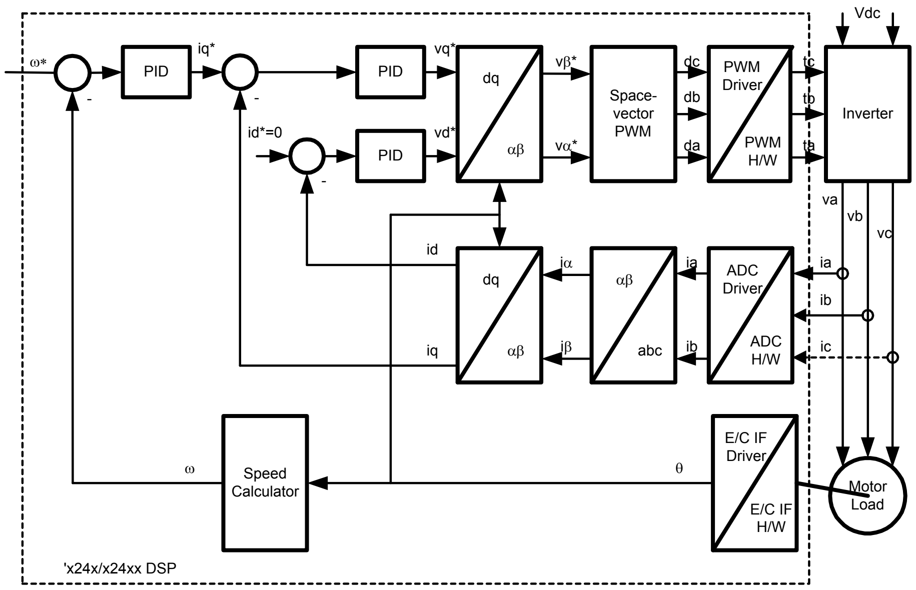


Figure 1 Field Oriented Control of PMSM

Therefore for cost sensitive applications, it is ideal if the rotor flux position information can be derived from measurement of voltages and currents. Figure 2 shows the block diagram of a sensorless PMSM control system where rotor flux position is derived from measurement of motor currents and knowledge of motor voltage commands.

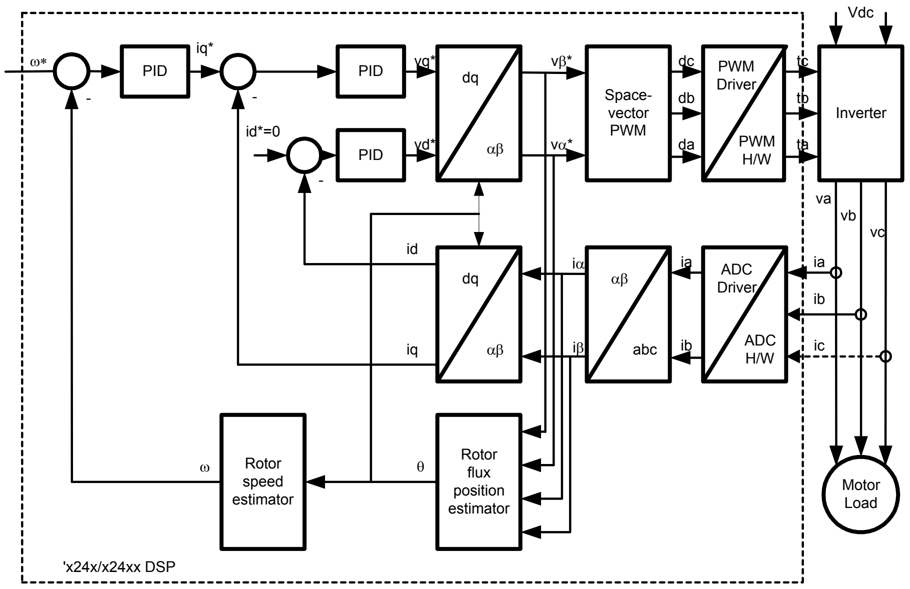


Figure 2 Sensorless Field Oriented Control of PMSM

This software module implements a rotor flux position estimator based on a sliding mode current observer. As shown in Figure 3, the inputs to the estimator are motor phase currents and voltages expressed in α-β coordinate frame.

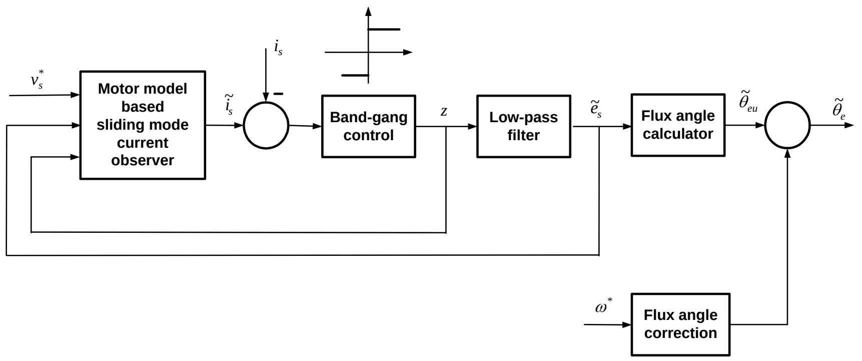


Figure 3 Sliding Mode Observer Based Rotor Flux Position Estimator

Figure 4 is an illustration of the coordinate frames and voltage and current vectors of PMSM, with *a*, *b* and *c* being the phase axes, α and β being a fixed Cartesian coordinate frame aligned with phase *a*, and *d* and *q* being a rotating Cartesian coordinate frame aligned with rotor flux. v_s, i_s and e_s are the motor phase voltage, current and back emf vectors (each with two coordinate entries). All vectors are expressed in α-β coordinate frame for the purpose of this discussion. The α-β frame expressions are obtained by applying Clarke transformation to their corresponding three phase representations.

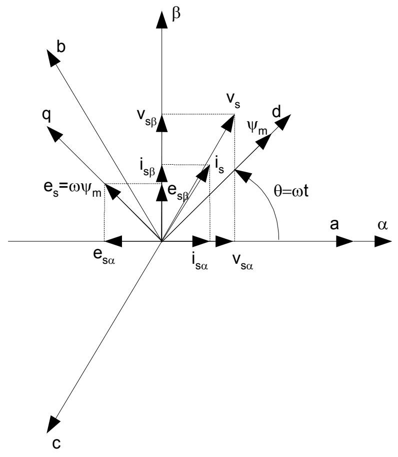


Figure 4 PMSM Coordinate Frames and Vectors

Equation 1 is the mathematical model of PMSM in α-β coordinate frame.

$$\frac{d}{dt}i_s = Ai_s + B(v_s - e_s) \qquad (1)$$

The matrices *A* and *B* are defined as $A = -\frac{R}{L}I_2$ and $B = \frac{1}{L}I_2$ with $L = \frac{3}{2}L_m$, where L_m and *R* are the magnetizing inductance and resistance of stator phase winding and I_2 is a 2 by 2 identity matrix. Next the mathematical equations for the blocks in Figure 3 are discussed.

1. Sliding Mode Current Observer

The sliding mode current observer consists of a model based current observer and a bang-bang control generator driven by error between estimated motor currents and actual motor currents. The mathematical equations for the observer and control generator are given by Equations 2 and 3.

$$\frac{d}{dt}\tilde{i}_s = A\tilde{i}_s + B(v_s{}^* - \tilde{e}_s + z) \qquad (2)$$

$$z = k\, sign(\tilde{i}_s - i_s) \qquad (3)$$

The goal of the bang-bang control *z* is to drive current estimation error to zero. It is achieved by proper selection of *k* and correct formation of estimated back emf, e_s. Note that the symbol ~ indicates that a variable is estimated. The symbol * indicates that a variable is a command.

The discrete form of Equations 2 and 3 are given by Equations 4 and 5.

$$\tilde{i}_s(n+1) = F\,\tilde{i}_s(n) + G\,(v_s^{*}(n) - \tilde{e}_s(n) + z(n)) \tag{4}$$

$$z(n) = k\,sign(\tilde{i}_s(n) - i_s(n)) \tag{5}$$

The matrices *F* and *G* are given by $F = e^{-\frac{R}{L}T_s} I_2$ and $G = \frac{1}{R}(1 - e^{-\frac{R}{L}T_s})I_2$ where T_s is the sampling period.

2. Estimated Back EMF

Estimated back emf is obtained by filtering the bang-bang control, *z*, with a first order low-pass filter described by Equation 6.

$$\frac{d}{dt}\tilde{e}_s = -\omega_0\tilde{e}_s + \omega_0 z \tag{6}$$

The parameter ω_0 is defined as $\omega_0 = 2\pi f_0$, where f_0 represents the cutoff frequency of the filter. The discrete form of Equation 6 is given by Equation 7.

$$\tilde{e}_s(n+1) = \tilde{e}_s(n) + 2\pi f_0\,(z(n) - \tilde{e}_s(n)) \tag{7}$$

3. Rotor Flux Position Calculation

Estimated rotor flux angle is obtained based on Equation 8 for back emf.

$$e_s = \frac{3}{2}k_e\omega\begin{pmatrix}-\sin\theta \\ \cos\theta\end{pmatrix} \tag{8}$$

Therefore given the estimated back emf, estimated rotor position can be calculated based on Equation 9.

$$\tilde{\theta}_{eu} = \arctan(-\tilde{e}_{s\alpha}, \tilde{e}_{s\beta}) \tag{9}$$

Next, Table 1 shows the correspondence of notations between variables used here and variables used in the program (i.e., smopos.c, smopos.h). The software module requires that both input and output variables are in per unit values.

	Equation Variables	**Program Variables**
Inputs	$v_{s\alpha}^*$	Valpha
	$v_{s\beta}^*$	Vbeta
	$i_{s\alpha}$	Ialpha
	$i_{s\beta}$	Ibeta
Outputs	$\tilde{\theta}_e$	Theta
	z_α	Zalpha
	z_β	Zbeta
Others	$\tilde{i}_{s\alpha}$	EstIalpha
	$\tilde{i}_{s\beta}$	EstIbeta
	$\tilde{e}_{s\alpha}$	Ealpha
	$\tilde{e}_{s\beta}$	Ebeta
	$e^{-\frac{R}{L}T_s}$	Fsmopos
	$\frac{1}{R}(1-e^{-\frac{R}{L}T_s})$	Gsmopos
	k	Kslide
	$2\pi f_0$	Kslf

Table 1: Correspondence of notations