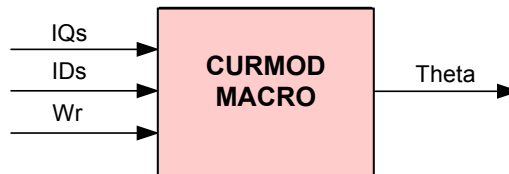


**Description**

This module takes as input both IQs and ID<sub>s</sub>, currents coming from the PARK transform, as well as the rotor mechanical speed and gives the rotor flux position.

**Availability**

This IQ module is available in one interface:

- 1) The C interface version

**Module Properties**

**Type:** Target Independent, Application Dependent

**Target Devices:** 28x Fixed and Floating Point devices

**C Version File Names:** cur\_mod.h

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

## C Interface

### Object Definition

The structure of CURMOD object is defined by following structure definition

```
typedef struct { _iq IDs;      // Input: Syn. rotating d-axis current
                _iq IQs;      // Input: Syn. rotating q-axis current
                _iq Wr;        // Input: Rotor electrically angular velocity
                _iq IMDs;      // Variable: Syn. rotating d-axis magnetizing current
                _iq Theta;     // Output: Rotor flux angle
                _iq Kr;        // Parameter: constant using in magnetizing current calc
                _iq Kt;        // Parameter: constant using in slip calculation
                _iq K;         // Parameter: constant using in rotor flux angle calculation
            } CURMOD;

typedef CURMOD *CURMOD_handle;
```

### Module Terminal Variables

Item	Name	Description	Format	Range(Hex)
Inputs	IDs	Syn. rotating d-axis current	GLOBAL_Q	80000000-7FFFFFFF
	IQs	Syn. rotating d-axis current	GLOBAL_Q	80000000-7FFFFFFF
	Wr	Rotor electrically angular velocity	GLOBAL_Q	80000000-7FFFFFFF
Outputs	Theta	Rotor flux angle	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
CUR_MOD parameter	Kr	$Kr = T/Tr$	GLOBAL_Q	80000000-7FFFFFFF
	Kt	$Kt = 1/(Tr \cdot \omega_b)$	GLOBAL_Q	80000000-7FFFFFFF
	K	$K = T \cdot f_b$	GLOBAL_Q	80000000-7FFFFFFF
Internal	Wslip	Slip frequency	GLOBAL_Q	80000000-7FFFFFFF
	We	Synchronous frequency	GLOBAL_Q	80000000-7FFFFFFF

GLOBAL\_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

### Special Constants and Data types

#### CURMOD

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

#### CURMOD\_handle

User defined Data type of pointer to CURMOD module

#### CURMOD\_DEFAULTS

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

## Methods

**CURMOD\_MACRO(CURMOD\_handle);**

This definition implements one method viz., the current model computation macro. The input argument to this macro is the module handle.

## Module Usage

### Instantiation

The following example instances two CURMOD objects  
CURMOD cm1, cm2;

### Initialization

To Instance pre-initialized objects  
CURMOD cm1 = CURMOD\_DEFAULTS;  
CURMOD cm2 = CURMOD\_DEFAULTS;

### Invoking the computation macro

CURMOD\_MACRO(cm1);  
CURMOD\_MACRO(cm2);

## Example

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

```
main()
{
    cm1.Kr = parem1_1;           // Pass parameters to cm1
    cm1.Kt = parem1_2;           // Pass parameters to cm1
    cm1.K = parem1_3;            // Pass parameters to cm1

    cm2.Kr = parem2_1;           // Pass parameters to cm2
    cm2.Kt = parem2_2;           // Pass parameters to cm2
    cm2.K = parem2_3;            // Pass parameters to cm2
}

void interrupt periodic_interrupt_isr()
{
    cm1.IDs = de1;               // Pass inputs to cm1
    cm1.IQs = qe1;               // Pass inputs to cm1
    cm1.Wr = Wr1;                // Pass inputs to cm1

    cm2.IDs = de2;               // Pass inputs to cm2
    cm2.IQs = qe2;               // Pass inputs to cm2
    cm2.Wr = Wr2;                // Pass inputs to cm2

    CURMOD_MACRO(cm1);           // Call compute macro for cm1
    CURMOD_MACRO(cm2);           // Call compute macro for cm2

    ang1 = cm1.Theta;            // Access the outputs of cm1
    ang2 = cm2.Theta;            // Access the outputs of cm2
}
```

## Constant Computation Macro

Since the current model module requires three constants (Kr, Kt, and K) to be input basing on the machine parameters, base quantities, mechanical parameters, and sampling period. These four constants can be internally computed by the macro (cur\_const.h). The followings show how to use the C constant computation macro.

### Object Definition

The structure of CURMOD\_CONST object is defined by following structure definition

```
typedef struct { float32 Rr;    // Input: Rotor resistance (ohm)
                float32 Lr;    // Input: Rotor inductance (H)
                float32 fb;    // Input: Base electrical frequency (Hz)
                float32 Ts;    // Input: Sampling period (sec)
                float32 Kr;    // Output: constant using in magnetizing current calculation
                float32 Kt;    // Output: constant using in slip calculation
                float32 K;     // Output: constant using in rotor flux angle calculation
            } CURMOD_CONST;

typedef CURMOD_CONST *CURMOD_CONST_handle;
```

### Module Terminal Variables

Item	Name	Description	Format	Range(Hex)
Inputs	Rr	Rotor resistance (ohm)	Floating	N/A
	Lr	Rotor inductance (H)	Floating	N/A
	fb	Base electrical frequency (Hz)	Floating	N/A
	Ts	Sampling period (sec)	Floating	N/A
Outputs	Kr	constant using in current model calculation	Floating	N/A
	Kt	constant using in current model calculation	Floating	N/A
	K	constant using in current model calculation	Floating	N/A

### Special Constants and Data types

#### CURMOD\_CONST

The module definition is created as a data type. This makes it convenient to instance an interface to the current model constant computation module. To create multiple instances of the module simply declare variables of type CURMOD\_CONST.

#### CURMOD\_CONST\_handle

User defined Data type of pointer to CURMOD\_CONST module

#### CURMOD\_CONST\_DEFAULTS

Structure symbolic constant to initialize CURMOD\_CONST module. This provides the initial values to the terminal variables as Well as method pointers.

## Methods

**CURMOD\_CONST\_MACRO(CURMOD\_CONST\_handle);**

This definition implements one method viz., the current model constant computation macro. The input argument to this macro is the module handle.

## Module Usage

### Instantiation

The following example instances two CURMOD\_CONST objects  
CURMOD\_CONST cm1\_const, cm2\_const;

### Initialization

To Instance pre-initialized objects

CURMOD\_CONST cm1\_const = CURMOD\_CONST\_DEFAULTS;  
CURMOD\_CONST cm2\_const = CURMOD\_CONST\_DEFAULTS;

### Invoking the computation macro

CURMOD\_CONST\_MACRO (cm1\_const);  
CURMOD\_CONST\_MACRO (cm2\_const);

## Example

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

```
main()
{

    cm1_const.Rr = Rr1;           // Pass floating-point inputs to cm1_const
    cm1_const.Lr = Lr1;           // Pass floating-point inputs to cm1_const
    cm1_const.fb = Fb1;           // Pass floating-point inputs to cm1_const
    cm1_const.Ts = Ts1;           // Pass floating-point inputs to cm1_const

    cm2_const.Rr = Rr2;           // Pass floating-point inputs to cm2_const
    cm2_const.Lr = Lr2;           // Pass floating-point inputs to cm2_const
    cm2_const.fb = Fb2;           // Pass floating-point inputs to cm2_const
    cm2_const.Ts = Ts2;           // Pass floating-point inputs to cm2_const

    CURMOD_CONST_MACRO (cm1_const); // Call compute macro for cm1_const
    CURMOD_CONST_MACRO (cm2_const); // Call compute macro for cm2_const

    cm1.Kr = _IQ(cm1_const.Kr);   // Access the floating-point outputs of cm1_const
    cm1.Kt = _IQ(cm1_const.Kt);   // Access the floating-point outputs of cm1_const
    cm1.K = _IQ(cm1_const.K);     // Access the floating-point outputs of cm1_const

    cm2.Kr = _IQ(cm2_const.Kr);   // Access the floating-point outputs of cm2_const
    cm2.Kt = _IQ(cm2_const.Kt);   // Access the floating-point outputs of cm2_const
    cm2.K = _IQ(cm2_const.K);     // Access the floating-point outputs of cm2_const

}
```

## Technical Background

With the asynchronous drive, the mechanical rotor angular speed is not by definition, equal to the rotor flux angular speed. This implies that the necessary rotor flux position cannot be detected directly by the mechanical position sensor used with the asynchronous motor (QEP or tachometer). The current model module be added to the generic structure in the regulation block diagram to perform a current and speed closed loop for a three phases ACI motor in FOC control.

The current model consists of implementing the following two equations of the motor in d,q reference frame:

$$\begin{aligned} i_{dS} &= T_R \frac{di_{mR}}{dt} + i_{mR} \\ f_S &= \frac{1}{\omega_b} \frac{d\theta}{dt} = n + \frac{i_{qS}}{T_R i_{mR} \omega_b} \end{aligned}$$

Where We have:

- $\theta$  is the rotor flux position
- $i_{mR}$  is the magnetizing current
- $T_R = \frac{L_R}{R_R}$  is the rotor time constant with  $L_R$  the rotor inductance and  $R_R$  the rotor resistance.
- $f_S$  is the rotor flux speed
- $\omega_b$  is the electrical nominal flux speed.

Knowledge of the rotor time constant is critical to the correct functioning of the current model as it is this system that outputs the rotor flux speed that will be integrated to get the rotor flux position.

Assuming that  $i_{qS_{k+1}} \approx i_{qS_k}$  the above equations can be discretized as follows:

$$\begin{aligned} i_{mR_{k+1}} &= i_{mR_k} + \frac{T}{T_R} (i_{dS_k} - i_{mR_k}) \\ f_{S_{k+1}} &= n_{k+1} + \frac{1}{T_R \omega_b} \frac{i_{qS_k}}{i_{mR_{k+1}}} \end{aligned}$$

In this equation system, T represents the Main loop control period. In a FOC control this usually corresponds to the Timer 1 underflow interrupt period.

Let the two above equations constants  $\frac{T}{T_R}$  and  $\frac{1}{T_R \omega_b}$  be renamed respectively  $K_t$  and  $K_R$ . These two constants need to be calculated according to the motor parameters and initialize into the cur\_mod.c file.

Once the motor flux speed ( $f_s$ ) has been calculated, the necessary rotor flux position in per-unit ( $\theta$ ) is computed by the integration formula:

$$\theta = \theta_{k-1} + K f_{s_k}$$

where  $K = T f_b$

The user should be aware that the current model module constants depend on the motor parameters and need to be calculated for each type of motor. The information needed to do so are the rotor resistance, the rotor inductance (which is the sum of the magnetizing inductance and the rotor leakage inductance ( $L_R = L_H + L_{\sigma R}$ )).

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

	Equation Variables	Program Variables
<b>Inputs</b>	$i_{qS}$	IQs
	$i_{dS}$	IDs
	$n$	Wr
<b>Output</b>	$\theta$	Theta
<b>Others</b>	$i_{mR}$	IMDs

Table 1: Correspondence of notations