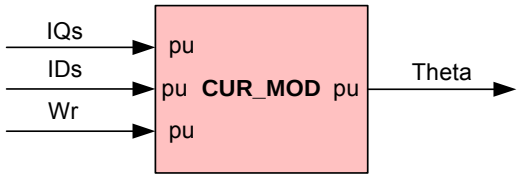
CUR_MOD	*Current Model*

Description

This module takes as input both IQs and IDs, 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: x281x or x280x

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

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

Item	C version	Comments
Code Size□ (x281x/x280x)	72/72 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" CURMOD structure consumes 18 words in the data memory

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

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
                 void  (*calc)(); // Pointer to calculation function
                } CURMOD;

typedef CURMOD *CURMOD_handle;
```

Module Terminal Variables/Functions

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*wb)	GLOBAL_Q	80000000-7FFFFFFF
	K	K = T*fb	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

void cur_mod_calc(CURMOD_handle);

This definition implements one method viz., the current model computation function. The input argument to this function 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 function
cm1.calc(&cm1);
cm2.calc(&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

        cm1.calc(&cm1);              // Call compute function for cm1
        cm2.calc(&cm2);              // Call compute function for cm2

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

}
```

Constant Computation Function

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 C function (cur_const.c, cur_const.h). The followings show how to use the C constant computation function.

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
                   void   (*calc)();  // Pointer to calculation function
                  } CURMOD_CONST;

typedef CURMOD_CONST *CURMOD_CONST_handle;
```

Module Terminal Variables/Functions

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

void cur_mod_const_calc(CURMOD_CONST_handle);

This definition implements one method viz., the current model constant computation function. The input argument to this function 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 function
cm1_const.calc(&cm1_const);
cm2_const.calc(&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

        cm1_const.calc(&cm1_const);   // Call compute function for cm1_const
        cm2_const.calc(&cm2_const);   // Call compute function 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:

$$i_{dS} = T_R \frac{di_{mR}}{dt} + i_{mR}$$

$$fs = \frac{1}{\omega_b}\frac{d\theta}{dt} = n + \frac{i_{qS}}{T_R i_{mR} \omega_b}$$

Where We have:

- θ 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.
- fs is the rotor flux speed
- ω_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:

$$i_{mR_{k+1}} = i_{mR_k} + \frac{T}{T_R}(i_{dS_k} - i_{mR_k})$$

$$f_{S_{k+1}} = n_{k+1} + \frac{1}{T_R \omega \omega_b}\frac{i_{qS_k}}{i_{mR_{k+1}}}$$

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 (fs) has been calculated, the necessary rotor flux position in per-unit (θ) 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**
Inputs	i_{qS}	IQs
	i_{dS}	IDs
	n	Wr
Output	θ	Theta
Others	i_{mR}	IMDs

Table 1: Correspondence of notations