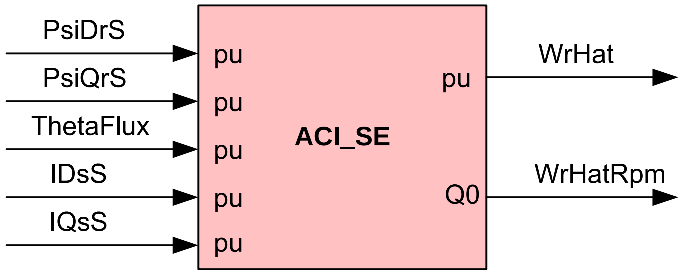
ACI_SE	*Speed estimator of the 3-ph induction motor*

Description

This software module implements a speed estimator of the 3-ph induction motor based upon its mathematics model. The estimator's accuracy relies heavily on knowledge of critical motor parameters.



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: aci_se.c, aci_se.h

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

Item	C version	Comments
Code Size□ (x281x/x280x)	180/180 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" ACISE structure consumes 32 words in the data memory

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

C Interface

Object Definition

The structure of ACISE object is defined by following structure definition

```
typedef struct {  _iq  IQsS;            // Input: Stationary q-axis stator current
                  _iq  PsiDrS;          // Input: Stationary d-axis rotor flux
                  _iq  IDsS;            // Input: Stationary d-axis stator current
                  _iq  PsiQrS;          // Input: Stationary q-axis rotor flux
                  _iq  K1;              // Parameter: Constant using in speed computation
                 _iq  SquaredPsi;       // Variable: Squared rotor flux
                 _iq  ThetaFlux;        // Input: Rotor flux angle
                 _iq21  K2;             // Parameter: Constant using in differentiator (Q21)
                 _iq  OldThetaFlux;     // Variable: Previous rotor flux angle
                 _iq  K3;               // Parameter: Constant using in low-pass filter
                 _iq21  WPsi;           // Variable: Synchronous rotor flux speed in pu  (Q21)
                 _iq  K4;               // Parameter: Constant using in low-pass filter
                 _iq  WrHat;            // Output: Estimated speed in per unit
                 Uint32  BaseRpm;       // Parameter: Base rpm speed (Q0)
                 int32  WrHatRpm;       // Output: Estimated speed in rpm (Q0)
                 void  (*calc)();       // Pointer to calculation function
             } ACISE;

typedef ACISE *ACISE_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format*	Range(Hex)
Inputs	PsiDrS	stationary d-axis rotor flux	GLOBAL_Q	80000000-7FFFFFFF
	PsiDrS	stationary q-axis rotor flux	GLOBAL_Q	80000000-7FFFFFFF
	ThetaFlux	rotor flux linkage angle	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
	IDsS	stationary d-axis stator current	GLOBAL_Q	80000000-7FFFFFFF
	IQsS	stationary q-axis stator current	GLOBAL_Q	80000000-7FFFFFFF
Outputs	WrHat	estimated rotor speed	GLOBAL_Q	80000000-7FFFFFFF
	WrHatRpm	estimated rotor speed in rpm	Q0	80000000-7FFFFFFF
ACI_SE parameter	K1	K1 = 1/(Wb*Tr)	GLOBAL_Q	80000000-7FFFFFFF
	K2	K2 = 1/(fb*T)	Q21	80000000-7FFFFFFF
	K3	K3 = Tau/(Tau+T)	GLOBAL_Q	80000000-7FFFFFFF
	K4	K4 = T/(Tau+T)	GLOBAL_Q	80000000-7FFFFFFF
	BaseRpm	base speed in rpm	Q0	80000000-7FFFFFFF
Internal	OldThetaFlux	previous rotor flux linkage angle	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
	WPsi	synchronous speed	GLOBAL_Q	80000000-7FFFFFFF
	SquaredPsi	squared magnitude of rotor flux	GLOBAL_Q	80000000-7FFFFFFF

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

Special Constants and Data types

ACISE
The module definition is created as a data type. This makes it convenient to instance an interface to the speed estimator of Induction Motor module. To create multiple instances of the module simply declare variables of type ACISE.

ACISE_handle
User defined Data type of pointer to ACISE module

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

Methods

void aci_se_calc(ACISE_handle);

This definition implements one method viz., the speed estimator of Induction Motor computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two ACISE objects
ACISE se1, se2;

Initialization
To Instance pre-initialized objects
ACISE se1 = ACISE_DEFAULTS;
ACISE se2 = ACISE_DEFAULTS;

Invoking the computation function
se1.calc(&se1);
se2.calc(&se2);

Example

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

```
main()
{
        se1.K1 = parem1_1;                    // Pass parameters to se1
        se1.K2 = parem1_2;                    // Pass parameters to se1
        se1.K3 = parem1_3;                    // Pass parameters to se1
        se1.K4 = parem1_4;                    // Pass parameters to se1
        se1.BaseRpm = base_speed_1;           // Pass parameters to se1

        se2.K1 = parem2_1;                    // Pass parameters to se2
        se2.K2 = parem2_2;                    // Pass parameters to se2
        se2.K3 = parem2_3;                    // Pass parameters to se2
        se2.K4 = parem2_4;                    // Pass parameters to se2
        se2.BaseRpm = base_speed_2;           // Pass parameters to se2

}
```

```
void interrupt periodic_interrupt_isr()
{
        se1.PsiDrS= flux_dq1.d;                // Pass inputs to se1
        se1.PsiQrS= flux_dq1.q;                // Pass inputs to se1
        se1.IDsS=current_dq1.d;                // Pass inputs to se1
        se1.IQsS=current_dq1.q;                // Pass inputs to se1
        se1.ThetaFlux=angle1;                  // Pass inputs to se1

        se2.PsiDrS= flux_dq2.d;                // Pass inputs to se2
        se2.PsiQrS= flux_dq2.q;                // Pass inputs to se2
        se2.IDsS=current_dq2.d;                // Pass inputs to se2
        se2.IQsS=current_dq2.q;                // Pass inputs to se2
        se2.ThetaFlux=angle2;                  // Pass inputs to se2

        se1.calc(&se1);                        // Call compute function for se1
        se2.calc(&se2);                        // Call compute function for se2

        speed_pu1 = se1.WrHat;                 // Access the outputs of se1
        speed_rpm1 = se1.WrHatRpm;             // Access the outputs of se1

        speed_pu2 = se2.WrHat;                 // Access the outputs of se2
        speed_rpm2 = se2.WrHatRpm;             // Access the outputs of se2
}
```

Constant Computation Function

Since the speed estimator of Induction motor module requires four constants (K1,…, K4) 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 (aci_se_const.c, aci_se_const.h). The followings show how to use the C constant computation function.

Object Definition

The structure of ACISE_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  fc;      // Input: Cut-off frequency of low-pass filter (Hz)
                    float32  Ts;      // Input: Sampling period in sec
                    float32  K1;      // Output: constant using in rotor flux calculation
                    float32  K2;      // Output: constant using in rotor flux calculation
                    float32  K3;      // Output: constant using in rotor flux calculation
                    float32  K4;      // Output: constant using in stator current calculation
                    void   (*calc)();   // Pointer to calculation function
                  } ACISE_CONST;

typedef ACISE_CONST *ACISE_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
	fc	Cut-off frequency of low-pass filter (Hz)	Floating	N/A
	Ts	Sampling period (sec)	Floating	N/A
Outputs	K1	constant using in rotor flux calculation	Floating	N/A
	K2	constant using in rotor flux calculation	Floating	N/A
	K3	constant using in rotor flux calculation	Floating	N/A
	K4	constant using in stator current cal.	Floating	N/A

Special Constants and Data types

ACISE_CONST
The module definition is created as a data type. This makes it convenient to instance an interface to the speed estimation of Induction Motor constant computation module. To create multiple instances of the module simply declare variables of type ACISE_CONST.

ACISE_CONST_handle
User defined Data type of pointer to ACISE_CONST module

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

Methods

void aci_se_const_calc(ACISE_CONST_handle);

This definition implements one method viz., the speed estimator of Induction Motor constant computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two ACISE_CONST objects
ACISE_CONST se1_const, se2_const;

Initialization
To Instance pre-initialized objects
ACISE_CONST se1_const = ACISE_CONST_DEFAULTS;
ACISE_CONST se2_const = ACISE_CONST_DEFAULTS;

Invoking the computation function
se1_const.calc(&se1_const);
se2_const.calc(&se2_const);

Example

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

```
main()
{

    se1_const.Rr = Rr1;             // Pass floating-point inputs to se1_const
    se1_const.Lr = Lr1;             // Pass floating-point inputs to se1_const
    se1_const.fb = Fb1;             // Pass floating-point inputs to se1_const
    se1_const.fc = Fc1;             // Pass floating-point inputs to se1_const
    se1_const.Ts = Ts1;             // Pass floating-point inputs to se1_const

    se2_const.Rr = Rr2;             // Pass floating-point inputs to se2_const
    se2_const.Lr = Lr2;             // Pass floating-point inputs to se2_const
    se2_const.fb = Fb2;             // Pass floating-point inputs to se2_const
    se2_const.fc = Fc2;             // Pass floating-point inputs to se2_const
    se2_const.Ts = Ts2;             // Pass floating-point inputs to se2_const

    se1_const.calc(&se1_const);     // Call compute function for se1_const
    se2_const.calc(&se2_const);     // Call compute function for se2_const

    se1.K1 = _IQ(se1_const.K1);   // Access the floating-point outputs of se1_const
    se1.K2 = _IQ(se1_const.K2);   // Access the floating-point outputs of se1_const
    se1.K3 = _IQ(se1_const.K3);   // Access the floating-point outputs of se1_const
    se1.K4 = _IQ(se1_const.K4);   // Access the floating-point outputs of se1_const

    se2.K1 = _IQ(se2_const.K1);   // Access the floating-point outputs of se2_const
    se2.K2 = _IQ(se2_const.K2);   // Access the floating-point outputs of se2_const
    se2.K3 = _IQ(se2_const.K3);   // Access the floating-point outputs of se2_const
    se2.K4 = _IQ(se2_const.K4);   // Access the floating-point outputs of se2_const

}
```

Technical Background

The open-loop speed estimator [1] is derived basing on the mathematics equations of induction motor in the stationary reference frame. The precise values of machine parameters are unavoidably required, otherwise the steady-state speed error may happen. However, the structure of the estimator is much simple comparing with other advanced techniques. All equations represented here are in the stationary reference frame (with superscript "s"). Firstly, the rotor flux linkage equations can be shown as below:

$$\lambda^s_{dr} = L_r i^s_{dr} + L_m i^s_{ds} \tag{1}$$

$$\lambda^s_{qr} = L_r i^s_{qr} + L_m i^s_{qs} \tag{2}$$

where L_r, and L_m are rotor, and magnetizing inductance (H), respectively.
According to equations (1)-(2), the rotor currents can be expressed as

$$i^s_{dr} = \frac{1}{L_r}\left(\lambda^s_{dr} - L_m i^s_{ds}\right) \tag{3}$$

$$i^s_{qr} = \frac{1}{L_r}\left(\lambda^s_{qr} - L_m i^s_{qs}\right) \tag{4}$$

Secondly, the rotor voltage equations are used to find the rotor flux linkage dynamics.

$$0 = R_r i^s_{dr} + \omega_r \lambda^s_{qr} + \frac{d\lambda^s_{dr}}{dt} \tag{5}$$

$$0 = R_r i^s_{qr} - \omega_r \lambda^s_{dr} + \frac{d\lambda^s_{qr}}{dt} \tag{6}$$

where ω_r is electrically angular velocity of rotor (rad/sec), and R_r is rotor resistance (Ω).
Substituting the rotor currents from (3)-(4) into (5)-(6), then the rotor flux linkage dynamics can be found as

$$\frac{d\lambda^s_{dr}}{dt} = -\frac{1}{\tau_r}\lambda^s_{dr} + \frac{L_m}{\tau_r} i^s_{ds} - \omega_r \lambda^s_{qr} \tag{7}$$

$$\frac{d\lambda^s_{qr}}{dt} = -\frac{1}{\tau_r}\lambda^s_{qr} + \frac{L_m}{\tau_r} i^s_{qs} + \omega_r \lambda^s_{dr} \tag{8}$$

where $\tau_r = \dfrac{L_r}{R_r}$ is rotor time constant (sec).

Suppose that the rotor flux linkages in (7)-(8) are known, therefore, its magnitude and angle can be computed as

$$\lambda^s_r = \sqrt{\left(\lambda^s_{dr}\right)^2 + \left(\lambda^s_{qr}\right)^2} \tag{9}$$

$$\theta_{\lambda_r} = \tan^{-1}\left(\frac{\lambda^s_{qr}}{\lambda^s_{dr}}\right) \tag{10}$$

Next, the rotor flux (i.e., synchronous) speed, ω_e, can be easily calculated by derivative of the rotor flux angle in (10).

$$\omega_e = \frac{d\theta_{\lambda_r}}{dt} = \frac{d\left(\tan^{-1}\left(\frac{\lambda_{qr}^s}{\lambda_{dr}^s}\right)\right)}{dt} \tag{11}$$

Referring to the derivative table, equation (11) can be solved as

$$\frac{d\left(\tan^{-1} u\right)}{dt} = \frac{1}{1+u^2}\frac{du}{dt} \tag{12}$$

where $u = \frac{\lambda_{qr}^s}{\lambda_{dr}^s}$, yields

$$\omega_e = \frac{d\theta_{\lambda_r}}{dt} = \frac{\left(\lambda_{dr}^s\right)^2}{\left(\lambda_r^s\right)^2}\left(\frac{\lambda_{dr}^s \frac{d\lambda_{qr}^s}{dt} - \lambda_{qr}^s \frac{d\lambda_{dr}^s}{dt}}{\left(\lambda_{dr}^s\right)^2}\right) \tag{13}$$

Substituting (7)-(8) into (13), and rearranging, then finally it gives

$$\omega_e = \frac{d\theta_{\lambda_r}}{dt} = \omega_r + \frac{1}{\left(\lambda_r^s\right)^2}\frac{L_m}{\tau_r}\left(\lambda_{dr}^s i_{qs}^s - \lambda_{qr}^s i_{ds}^s\right) \tag{14}$$

The second term of the left hand in (14) is known as slip that is proportional to the electromagnetic torque when the rotor flux magnitude is maintaining constant. The electromagnetic torque can be shown here for convenience.

$$T_e = \frac{3}{2}\frac{p}{2}\frac{L_m}{L_r}\left(\lambda_{dr}^s i_{qs}^s - \lambda_{qr}^s i_{ds}^s\right) \tag{15}$$

where p is the number of poles. Thus, the rotor speed can be found as

$$\omega_r = \omega_e - \frac{1}{\left(\lambda_r^s\right)^2}\frac{L_m}{\tau_r}\left(\lambda_{dr}^s i_{qs}^s - \lambda_{qr}^s i_{ds}^s\right) \tag{16}$$

Now, the per-unit concept is applied to (16), then, the equation (16) becomes

$$\omega_{r,pu} = \omega_{e,pu} - \frac{1}{\omega_b \tau_r}\left(\frac{\lambda_{dr,pu}^s i_{qs,pu}^s - \lambda_{qr,pu}^s i_{ds,pu}^s}{\left(\lambda_{r,pu}^s\right)^2}\right) \quad \text{pu} \tag{17}$$

where $\omega_b = 2\pi f_b$ is the base electrically angular velocity (rad/sec), $\lambda_b = L_m I_b$ is the base flux linkage (volt.sec), and I_b is the base current (amp). Equivalently, another form is

$$\omega_{r,pu} = \omega_{e,pu} - K_1\left(\frac{\lambda_{dr,pu}^s i_{qs,pu}^s - \lambda_{qr,pu}^s i_{ds,pu}^s}{\left(\lambda_{r,pu}^s\right)^2}\right) \quad \text{pu} \tag{18}$$

where $K_1 = \frac{1}{\omega_b \tau_r}$.

The per-unit synchronous speed can be calculated as

$$\omega_{e,pu} = \frac{1}{2\pi f_b}\frac{d\theta_{\lambda_r}}{dt} = \frac{1}{f_b}\frac{d\theta_{\lambda_r,pu}}{dt} \qquad \text{pu} \qquad (19)$$

where f_b is the base electrical (supplied) frequency (Hz) and 2π is the base angle (rad).

Discretizing equation (19) by using the backward approximation, yields

$$\omega_{e,pu}(k) = \frac{1}{f_b}\left(\frac{\theta_{\lambda_r,pu}(k) - \theta_{\lambda_r,pu}(k-1)}{T}\right) \qquad \text{pu} \qquad (20)$$

where T is the sampling period (sec). Equivalently, another form is

$$\omega_{e,pu}(k) = K_2\left(\theta_{\lambda_r,pu}(k) - \theta_{\lambda_r,pu}(k-1)\right) \qquad \text{pu} \qquad (21)$$

where $K_2 = \frac{1}{f_b T}$ is usually a large number.

In practice, the typical waveforms of the rotor flux angle, $\theta_{\lambda_r,pu}$, in both directions can be seen in Figure 1. To take care the discontinuity of angle from 360^o to 0^o (CCW) or from 0^o to 360^o (CW), the differentiator is simply operated only within the differentiable range as seen in this Figure. This differentiable range does not significantly lose the information to compute the estimated speed.

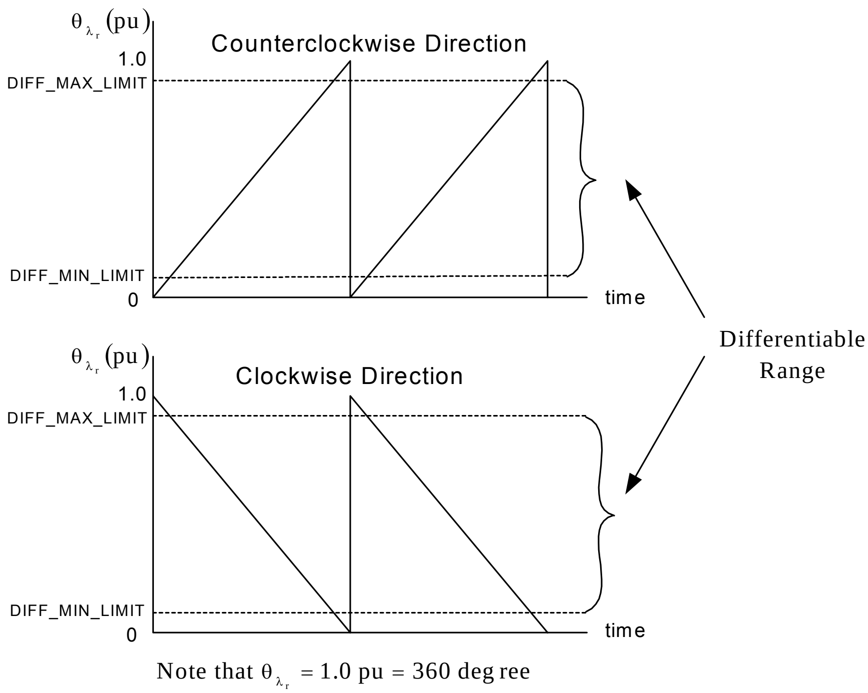


Figure 1: The waveforms of rotor flux angle in both directions

In addition, the synchronous speed in (21) is necessary to be filtered out by the low-pass filter in order to reduce the amplifying noise generated by the pure differentiator in (21). The simple 1st-order low-pass filter is used, then the actual synchronous speed to be used is the output of the low-pass filter, $\hat{\omega}_{e,pu}$, seen in following equation. The continuous-time equation of 1st-order low-pass filter is as

$$\frac{d\hat{\omega}_{e,pu}}{dt} = \frac{1}{\tau_c}\left(\omega_{e,pu} - \hat{\omega}_{e,pu}\right) \quad \text{pu} \qquad (22)$$

where $\tau_c = \dfrac{1}{2\pi f_c}$ is the low-pass filter time constant (sec), and f_c is the cut-off frequency (Hz). Using backward approximation, then (22) finally becomes

$$\hat{\omega}_{e,pu}(k) = K_3\hat{\omega}_{e,pu}(k-1) + K_4\omega_{e,pu}(k) \quad \text{pu} \qquad (23)$$

where $K_3 = \dfrac{\tau_c}{\tau_c + T}$, and $K_4 = \dfrac{T}{\tau_c + T}$.

In fact, only three equations (18), (21), and (23) are mainly employed to compute the estimated speed in per-unit. The required parameters for this module are summarized as follows:

<u>The machine parameters:</u>
- number of poles (p)
- rotor resistance (R_r)
- rotor leakage inductance (L_{rl})
- magnetizing inductance (L_m)

<u>The based quantities:</u>
- base current (I_b)
- base electrically angular velocity (ω_b)

<u>The sampling period:</u>
- sampling period (T)

<u>Low-pass filter:</u>
- cut-off frequency (f_c)

Notice that the rotor self inductance is $L_r = L_{rl} + L_m$ (H).

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

	Equation Variables	Program Variables
Inputs	λ_{dr}^{s}	PsiDrS
	λ_{qr}^{s}	PsiQrS
	θ_{λ_r}	ThetaFlux
	i_{ds}^{s}	IDsS
	i_{qs}^{s}	IQsS
Output	ω_r	WrHat
Others	$\left(\lambda_r^s\right)^2$	SquaredPsi
	ω_e	WPsi

Table 1: Correspondence of notations

References:

[1] A.M. Trzynadlowski, The Field Orientation Principle in Control of Induction Motors, Kluwer Academic Publishers, 1994, pp. 176-180.