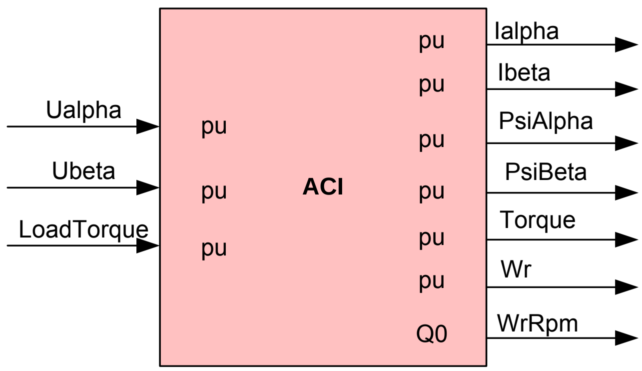
ACI	*Emulated (per-unit) 3-phase Induction Motor*

Description

This module implements a discrete equivalent of a 3-phase induction motor using trapezoidal approximation with predictor-corrector. The induction model is normalized by the adjustable base quantities of voltage, current, Torque, frequency, and flux linkage. The induction motor is modeled in the stationary reference frame. The outputs of this module are the stator currents, rotor flux linkages, electromagnetic Torque, electrically angular velocity, and actual rotor speed in rpm. All of these outputs are in per-unit except the actual rotor speed.



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: aci.c, aci.h

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

Item	C version	Comments
Code Size□ (x281x/x280x)	740/740 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" ACI structure consumes 46 words in the data memory

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

C Interface

Object Definition

The structure of ACI object is defined by following structure definition

```
typedef struct { _iq  Ualpha;          // Input: alpha-axis phase voltage
                 _iq  Ubeta;           // Input: beta-axis phase voltage
                 _iq  LoadTorque;      // Input: load Torque
                 _iq  Ialpha;          // Output: alpha-axis phase current
                 _iq  Ibeta;           // Output: beta-axis phase current
                 _iq  PsiAlpha;        // Output: alpha-axis rotor flux
                 _iq  PsiBeta;         // Output: beta-axis rotor flux
                 _iq  Torque;          // Output: electromagnetic Torque
                 _iq  Wr;              // Output: electrically angular velocity
                 int32  WrRpm;         // Output: motor speed in rpm (Q0)
                 _iq  K1;              // Parameter: constant using in rotor flux calculation
                 _iq  K2;              // Parameter: constant using in rotor flux calculation
                 _iq  K3;              // Parameter: constant using in rotor flux calculation
                 _iq  K4;              // Parameter: constant using in stator current cal
                 _iq  K5;              // Parameter: constant using in stator current cal
                 _iq  K6;              // Parameter: constant using in stator current cal
                 _iq  K7;              // Parameter: constant using in stator current cal
                 _iq  K8;              // Parameter: constant using in Torque calculation
                 _iq  K9;              // Parameter: constant using in rotor speed cal
                 _iq  K10;             // Parameter: constant using in rotor speed cal
                 Uint32  BaseRpm;      // Parameter: base motor speed in rpm
                 _iq  Alpha;           // Parameter: trapezoidal integration parameter (0-1)
                 void   (*calc)();     // Pointer to calculation function
               } ACI;

typedef ACI *ACI_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format*	Range(Hex)
Inputs	Ualpha	alpha-axis phase voltage	GLOBAL_Q	80000000-7FFFFFFF
	Ubeta	beta-axis phase voltage	GLOBAL_Q	80000000-7FFFFFFF
	LoadTorque	load Torque	GLOBAL_Q	80000000-7FFFFFFF
Outputs	Ialpha	alpha-axis phase current	GLOBAL_Q	80000000-7FFFFFFF
	Ibeta	beta-axis phase current	GLOBAL_Q	80000000-7FFFFFFF
	PsiAlpha	alpha-axis rotor flux	GLOBAL_Q	80000000-7FFFFFFF
	PsiBeta	beta-axis rotor flux	GLOBAL_Q	80000000-7FFFFFFF
	Torque	electromagnetic Torque	GLOBAL_Q	80000000-7FFFFFFF
	Wr	electrically angular velocity	GLOBAL_Q	80000000-7FFFFFFF
	WrRpm	motor speed in rpm	Q0	80000000-7FFFFFFF
ACI parameter	K1	constant using in rotor flux cal	GLOBAL_Q	80000000-7FFFFFFF
	K2	constant using in rotor flux cal	GLOBAL_Q	80000000-7FFFFFFF
	K3	constant using in rotor flux cal	GLOBAL_Q	80000000-7FFFFFFF
	K4	constant using in stator current cal	GLOBAL_Q	80000000-7FFFFFFF
	K5	constant using in stator current cal	GLOBAL_Q	80000000-7FFFFFFF
	K6	constant using in stator current cal	GLOBAL_Q	80000000-7FFFFFFF
	K7	constant using in stator current cal	GLOBAL_Q	80000000-7FFFFFFF
	K8	constant using in Torque cal	GLOBAL_Q	80000000-7FFFFFFF
	K9	constant using in rotor speed cal	GLOBAL_Q	80000000-7FFFFFFF
	K10	constant using in rotor speed cal	GLOBAL_Q	80000000-7FFFFFFF
	BaseRpm	base speed in rpm	Q0	80000000-7FFFFFFF

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

Special Constants and Data types

ACI
The module definition is created as a data type. This makes it convenient to instance an interface to the 3-phase Induction Motor module. To create multiple instances of the module simply declare variables of type ACI.

ACI_handle
User defined Data type of pointer to ACI module

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

Methods

void aci_calc(ACI_handle);

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

Module Usage

Instantiation
The following example instances two ACI objects
ACI aci1, aci2;

Initialization
To Instance pre-initialized objects
ACI aci1 = ACI_DEFAULTS;
ACI aci2 = ACI_DEFAULTS;

Invoking the computation function
aci1.calc(&aci1);
aci2.calc(&aci2);

Example

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

```
main()
{
        aci1.K1 = parem1_1;                 // Pass parameters to aci1
               .                                          .
               .                                          .
               .                                          .
        aci1.K10 = parem1_10;               // Pass parameters to aci1

        aci2.K1 = parem2_1;                 // Pass parameters to aci2
               .                                          .
               .                                          .
               .                                          .
        aci2.K10 = parem2_10;               // Pass parameters to aci2

}

void interrupt periodic_interrupt_isr()
{
        aci1.Ualpha = valpha1;              // Pass inputs to aci1
        aci1.Ubeta = vbeta1;                // Pass inputs to aci1
        aci1.LoadTorque = T_load1;          // Pass inputs to aci1

        aci2.Ualpha = valpha2;              // Pass inputs to aci2
        aci2.Ubeta = vbeta2;                // Pass inputs to aci2
        aci2.LoadTorque = T_load2;          // Pass inputs to aci2

        aci1.calc(&aci1);                   // Call compute function for aci1
        aci2.calc(&aci2);                   // Call compute function for aci2

        Ialpha1 = aci1.Ialpha;              // Access the outputs of aci1
        Ibeta1 = aci1.Ibeta;                // Access the outputs of aci1
        psi_alfa1 = aci1.PsiAlpha;          // Access the outputs of aci1
        psi_beta1 = aci1.PsiBeta;           // Access the outputs of aci1
        Wr1 = aci1.Wr;                      // Access the outputs of aci1
        WrRpm1 = aci1.WrRpm;                // Access the outputs of aci1

        Ialpha2 = aci2.Ialpha;              // Access the outputs of aci2
        Ibeta2 = aci2.Ibeta;                // Access the outputs of aci2
        psi_alfa2 = aci2.PsiAlpha;          // Access the outputs of aci2
        psi_beta2 = aci2.PsiBeta;           // Access the outputs of aci2
```

```
    Wr2 = aci2.Wr;                    // Access the outputs of aci2
    WrRpm2 = aci2.WrRpm;              // Access the outputs of aci2

}
```

Constant Computation Function

Since the Induction motor module requires ten constants (K1,…, K10) to be input basing on the machine parameters, base quantities, mechanical parameters, and sampling period. These ten constants can be internally computed by the C function (aci_const.c, aci_const.h). The followings show how to use the C constant computation function.

Object Definition

The structure of ACI_CONST object is defined by following structure definition

```
typedef struct { float32  Rs;           // Input: Stator resistance (ohm)
                 float32  Rr;           // Input: Rotor resistance (ohm)
                 float32  Ls;           // Input: Stator inductance (H)
                 float32  Lr;           // Input: Rotor inductance (H)
                 float32  Lm;           // Input: Magnetizing inductance (H)
                 float32  p;            // Input: Number of poles
                 float32 B;             // Input: Damping coefficient (N.m.sec/rad)
                 float32  J;            // Input: Moment of inertia of rotor mass (kg.m^2)
                 float32  Ib;           // Input: Base phase current (amp)
                 float32  Vb;           // Input: Base phase voltage (volt)
                 float32  Wb;           // Input: Base electrically angular velocity (rad/sec)
                 float32  Tb;           // Input: Base Torque (N.m)
                 float32  Lb;           // Input: Base flux linkage (volt.sec/rad)
                 float32  Ts;           // Input: Sampling period (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
                 float32  K5;           // Output: constant using in stator current calculation
                 float32  K6;           // Output: constant using in stator current calculation
                 float32  K7;           // Output: constant using in stator current calculation
                 float32  K8;           // Output: constant using in Torque calculation
                 float32  K9;           // Output: constant using in rotor speed calculation
                 float32  K10;          // Output: constant using in rotor speed calculation
                 void   (*calc)();      // Pointer to calculation function
               } ACI_CONST;

typedef ACI_CONST *ACI_CONST_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format	Range(Hex)
Inputs	Rs	Stator resistance (ohm)	Floating	N/A
	Rr	Rotor resistance (ohm)	Floating	N/A
	Ls	Stator inductance (H)	Floating	N/A
	Lr	Rotor inductance (H)	Floating	N/A
	Lm	Magnetizing inductance (H)	Floating	N/A
	p	Number of poles	Floating	N/A
	B	Damping coefficient (N.m.sec/rad)	Floating	N/A
	J	Moment of inertia (kg.m^2)	Floating	N/A
	Ib	Base phase current (amp)	Floating	N/A
	Vb	Base phase voltage (volt)	Floating	N/A
	Wb	Base electrical angular speed (rad/sec)	Floating	N/A
	Tb	Base Torque (N.m)	Floating	N/A
	Lb	Base flux linkage (volt.sec/rad)	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
	K5	constant using in stator current cal.	Floating	N/A
	K6	constant using in stator current cal.	Floating	N/A
	K7	constant using in stator current cal.	Floating	N/A
	K8	constant using in Torque calculation	Floating	N/A
	K9	constant using in rotor speed cal.	Floating	N/A
	K10	constant using in rotor speed cal.	Floating	N/A

Special Constants and Data types

ACI_CONST
The module definition is created as a data type. This makes it convenient to instance an interface to the 3-phase Induction Motor constant computation module. To create multiple instances of the module simply declare variables of type ACI_CONST.

ACI_CONST_handle
User defined Data type of pointer to ACI_CONST module

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

Methods

void aci_const_calc(ACI_CONST_handle);

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

Module Usage

Instantiation

The following example instances two ACI_CONST objects
ACI_CONST aci1_const, aci2_const;

Initialization

To Instance pre-initialized objects
ACI_CONST aci1_const = ACI_CONST_DEFAULTS;
ACI_CONST aci2_const = ACI_CONST_DEFAULTS;

Invoking the computation function

aci1_const.calc(&aci1_const);
aci2_const.calc(&aci2_const);

Example

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

```
main()
{

    aci1_const.Rs = Rs1;        // Pass floating-point inputs to aci1_const
    aci1_const.Rr = Rr1;        // Pass floating-point inputs to aci1_const
    aci1_const.Ls = Ls1;        // Pass floating-point inputs to aci1_const
    aci1_const.Lr = Lr1;        // Pass floating-point inputs to aci1_const
    aci1_const.Lm = Lm1;        // Pass floating-point inputs to aci1_const
    aci1_const.p = p1;          // Pass floating-point inputs to aci1_const
    aci1_const.B = B1;          // Pass floating-point inputs to aci1_const
    aci1_const.J = J1;          // Pass floating-point inputs to aci1_const
    aci1_const.Ib = Ib1;        // Pass floating-point inputs to aci1_const
    aci1_const.Vb = Vb1;        // Pass floating-point inputs to aci1_const
    aci1_const.Wb = Wb1;        // Pass floating-point inputs to aci1_const
    aci1_const.Tb = Tb1;        // Pass floating-point inputs to aci1_const
    aci1_const.Lb = Lb1;        // Pass floating-point inputs to aci1_const
    aci1_const.Ts = Ts1;        // Pass floating-point inputs to aci1_const

    aci2_const.Rs = Rs2;        // Pass floating-point inputs to aci2_const
    aci2_const.Rr = Rr2;        // Pass floating-point inputs to aci2_const
    aci2_const.Ls = Ls2;        // Pass floating-point inputs to aci2_const
    aci2_const.Lr = Lr2;        // Pass floating-point inputs to aci2_const
    aci2_const.Lm = Lm2;        // Pass floating-point inputs to aci2_const
    aci2_const.p = p2;          // Pass floating-point inputs to aci2_const
    aci2_const.B = B2;          // Pass floating-point inputs to aci2_const
    aci2_const.J = J2;          // Pass floating-point inputs to aci2_const
    aci2_const.Ib = Ib2;        // Pass floating-point inputs to aci2_const
    aci2_const.Vb = Vb2;        // Pass floating-point inputs to aci2_const
    aci2_const.Wb = Wb2;        // Pass floating-point inputs to aci2_const
    aci2_const.Tb = Tb2;        // Pass floating-point inputs to aci2_const
    aci2_const.Lb = Lb2;        // Pass floating-point inputs to aci2_const
    aci2_const.Ts = Ts2;        // Pass floating-point inputs to aci2_const
```

```
    aci1_const.calc(&aci1_const);    // Call compute function for aci1_const
    aci2_const.calc(&aci2_const);    // Call compute function for aci2_const

    aci1.K1 = _IQ(aci1_const.K1);    // Access the floating-point outputs of aci1_const
              .                                        .
              .                                        .
              .                                        .
    aci1.K10 = _IQ(aci1_const.K10); // Access the floating-point outputs of aci1_const


    aci2.K1 = _IQ(aci2_const.K1);    // Access the floating-point outputs of aci2_const
              .                                        .
              .                                        .
              .                                        .
    aci2.K10 = _IQ(aci2_const.K10); // Access the floating-point outputs of aci2_const

}
```

Technical Background

The mathematic model of a 3-phase induction motor in the stationary reference frame is described by the fifth-order differential equations as follows:

$$\frac{d\psi_{\beta r}}{dt} = f_1(\psi_{\beta r}, \psi_{\alpha r}, i_{\beta s}, \omega_r) = -\alpha\psi_{\beta r} + \omega_r\psi_{\alpha r} + \alpha L_m i_{\beta s} \quad (1)$$

$$\frac{d\psi_{\alpha r}}{dt} = f_2(\psi_{\beta r}, \psi_{\alpha r}, i_{\alpha s}, \omega_r) = -\omega_r\psi_{\beta r} - \alpha\psi_{\alpha r} + \alpha L_m i_{\alpha s} \quad (2)$$

$$\frac{di_{\beta s}}{dt} = f_3(\psi_{\beta r}, \psi_{\alpha r}, i_{\beta s}, v_{\beta s}, \omega_r) = \alpha\beta\psi_{\beta r} - \beta\omega_r\psi_{\alpha r} - \gamma i_{\beta s} + \frac{1}{\sigma L_s} v_{\beta s} \quad (3)$$

$$\frac{di_{\alpha s}}{dt} = f_4(\psi_{\beta r}, \psi_{\alpha r}, i_{\alpha s}, v_{\alpha s}, \omega_r) = \beta\omega_r\psi_{\beta r} + \alpha\beta\psi_{\alpha r} - \gamma i_{\alpha s} + \frac{1}{\sigma L_s} v_{\alpha s} \quad (4)$$

$$\frac{d\omega_r}{dt} = f_5(\psi_{\beta r}, \psi_{\alpha r}, i_{\beta s}, i_{\alpha s}, \omega_r) = -\frac{B}{J}\omega_r + \frac{n_p}{J}(T_e - T_l) \quad (5)$$

$$T_e = \frac{3}{2}\frac{L_m n_p}{L_r}(\psi_{\alpha r} i_{\beta s} - \psi_{\beta r} i_{\alpha s}) \quad (6)$$

where $\sigma = 1 - \frac{L_m^2}{L_s L_r}$, $\gamma = \frac{(L_m^2 R_r + L_r^2 R_s)}{\sigma L_s L_r^2}$, $\alpha = \frac{1}{\tau_r} = \frac{R_r}{L_r}$, and $\beta = \frac{L_m}{\sigma L_s L_r}$

Equations (1)-(4) can be compactly reWritten in the state-space model as follows:

$$\dot{\mathbf{x}} = \mathbf{A}(\omega_r)\mathbf{x} + \mathbf{B}\mathbf{u} \quad (7)$$

where $\mathbf{x} = \begin{bmatrix} \psi_{\beta r} \\ \psi_{\alpha r} \\ i_{\beta s} \\ i_{\alpha s} \end{bmatrix}$, $\mathbf{A}(\omega_r) = \begin{bmatrix} -\alpha & \omega_r & \alpha L_m & 0 \\ -\omega_r & -\alpha & 0 & \alpha L_m \\ \alpha\beta & -\beta\omega_r & -\gamma & 0 \\ \beta\omega_r & \alpha\beta & 0 & -\gamma \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{\sigma L_s} & 0 \\ 0 & \frac{1}{\sigma L_s} \end{bmatrix}$, and

$\mathbf{u} = \begin{bmatrix} v_{\beta s} \\ v_{\alpha s} \end{bmatrix}$. Then, the differential equations seen in equations (5) and (7) are discretized by the trapezoidal integration method with the predictor-corrector.

Predictors:

$$\mathbf{x_p}(k) = \mathbf{x}(k-1) + T\left(\mathbf{A}(\omega_r(k-1))\mathbf{x}(k-1) + \mathbf{B}\mathbf{u}(k)\right) \quad (8)$$

$$\omega_{r,p}(k) = \omega_r(k-1) + T\left(-\frac{B}{J}\omega_r(k-1) + \frac{n_p}{J}(T_e(k) - T_l(k))\right) \quad (9)$$

Correctors:

$$\mathbf{x}(k) = \mathbf{x}(k-1) + \frac{T}{2}\begin{pmatrix} (1+a)\left(\mathbf{A}(\omega_r(k-1))\mathbf{x_p}(k) + \mathbf{Bu}(k)\right) + \\ (1-a)\left(\mathbf{A}(\omega_r(k-1))\mathbf{x}(k-1) + \mathbf{Bu}(k)\right) \end{pmatrix} \tag{10}$$

$$\omega_r(k) = \omega_r(k-1) + \frac{T}{2}\begin{pmatrix} (1+a)\left(-\frac{B}{J}\omega_{r,p}(k) + \frac{n_p}{J}\left(T_e(k) - T_l(k)\right)\right) + \\ (1-a)\left(-\frac{B}{J}\omega_r(k-1) + \frac{n_p}{J}\left(T_e(k) - T_l(k)\right)\right) \end{pmatrix} \tag{11}$$

$$T_e(k) = \frac{3}{2}\frac{L_m n_p}{L_r}\left(\psi_{\alpha r}(k) i_{\beta s}(k) - \psi_{\beta r}(k) i_{\alpha s}(k)\right) \tag{12}$$

where T is sampling period (sec) and a is the weight of predictor in between 0 and 1.

Next, the discretized equations in (10) and (11) are normalized by the base quantities of voltage (V_b), current (I_b), flux linkage (ψ_b), Torque (T_b), and electrically angular velocity (ω_b). As a result, the per-unit, discrete-time equations of induction motor can be summarized as follows:

Predictors:

$$\psi_{\beta r,p}(k) = \psi_{\beta r}(k-1) - K_1\psi_{\beta r}(k-1) + K_2\omega_r(k-1)\psi_{\alpha r}(k-1) + K_3 i_{\beta s}(k-1) \tag{13}$$

$$\psi_{\alpha r,p}(k) = \psi_{\alpha r}(k-1) - K_1\psi_{\alpha r}(k-1) - K_2\omega_r(k-1)\psi_{\beta r}(k-1) + K_3 i_{\alpha s}(k-1) \tag{14}$$

$$i_{\beta s,p}(k) = i_{\beta s}(k-1) + K_4\psi_{\beta r}(k-1) - K_5\omega_r(k-1)\psi_{\alpha r}(k-1) - K_6 i_{\beta s}(k-1) + K_7 v_{\beta s}(k) \tag{15}$$

$$i_{\alpha s,p}(k) = i_{\alpha s}(k-1) + K_4\psi_{\alpha r}(k-1) + K_5\omega_r(k-1)\psi_{\beta r}(k-1) - K_6 i_{\alpha s}(k-1) + K_7 v_{\alpha s}(k) \tag{16}$$

$$\omega_{r,p}(k) = \omega_r(k-1) - K_9\omega_r(k-1) + K_{10}\left(T_e(k) - T_l(k)\right) \tag{17}$$

Correctors:

$$\Delta\psi_{\beta r,p}(k) = -K_1\psi_{\beta r,p}(k) + K_2\omega_r(k-1)\psi_{\alpha r,p}(k) + K_3 i_{\beta s,p}(k) \tag{18}$$

$$\Delta\psi_{\alpha r,p}(k) = -K_1\psi_{\alpha r,p}(k) - K_2\omega_r(k-1)\psi_{\beta r,p}(k) + K_3 i_{\alpha s,p}(k) \tag{19}$$

$$\Delta i_{\beta s,p}(k) = K_4\psi_{\beta r,p}(k) - K_5\omega_r(k-1)\psi_{\alpha r,p}(k) - K_6 i_{\beta s,p}(k) + K_7 v_{\beta s}(k) \tag{20}$$

$$\Delta i_{\alpha s,p}(k) = K_4\psi_{\alpha r,p}(k) + K_5\omega_r(k-1)\psi_{\beta r,p}(k) - K_6 i_{\alpha s,p}(k) + K_7 v_{\alpha s}(k) \tag{21}$$

$$\Delta\omega_{r,p}(k) = -K_9\omega_{r,p}(k) + K_{10}\left(T_e(k) - T_l(k)\right) \tag{22}$$

$$\Delta\psi_{\beta r}(k) = -K_1\psi_{\beta r}(k-1) + K_2\omega_r(k-1)\psi_{\alpha r}(k-1) + K_3 i_{\beta s}(k-1) \tag{23}$$

$$\Delta\psi_{\alpha r}(k) = -K_1\psi_{\alpha r}(k-1) - K_2\omega_r(k-1)\psi_{\beta r}(k-1) + K_3 i_{\alpha s}(k-1) \tag{24}$$

$$\Delta i_{\beta s}(k) = K_4\psi_{\beta r}(k-1) - K_5\omega_r(k-1)\psi_{\alpha r}(k-1) - K_6 i_{\beta s}(k-1) + K_7 v_{\beta s}(k) \tag{25}$$

$$\Delta i_{\alpha s}(k) = K_4\psi_{\alpha r}(k-1) + K_5\omega_r(k-1)\psi_{\beta r}(k-1) - K_6 i_{\alpha s}(k-1) + K_7 v_{\alpha s}(k) \tag{26}$$

$$\Delta\omega_r(k) = -K_9\omega_r(k-1) + K_{10}\left(T_e(k) - T_l(k)\right) \quad (27)$$

$$\psi_{\beta r}(k) = \psi_{\beta r}(k-1) + 0.5\left((1+a)\Delta\psi_{\beta r,p}(k) + (1-a)\Delta\psi_{\beta r}(k)\right) \quad (28)$$

$$\psi_{\alpha r}(k) = \psi_{\alpha r}(k-1) + 0.5\left((1+a)\Delta\psi_{\alpha r,p}(k) + (1-a)\Delta\psi_{\alpha r}(k)\right) \quad (29)$$

$$i_{\beta s}(k) = i_{\beta s}(k-1) + 0.5\left((1+a)\Delta i_{\beta s,p}(k) + (1-a)\Delta i_{\beta s}(k)\right) \quad (30)$$

$$i_{\alpha s}(k) = i_{\alpha s}(k-1) + 0.5\left((1+a)\Delta i_{\alpha s,p}(k) + (1-a)\Delta i_{\alpha s}(k)\right) \quad (31)$$

$$\omega_r(k) = \omega_r(k-1) + 0.5\left((1+a)\Delta\omega_{r,p}(k) + (1-a)\Delta\omega_r(k)\right) \quad (32)$$

$$T_e = K_8\left(\psi_{\alpha r}(k) i_{\beta s}(k) - \psi_{\beta r}(k) i_{\alpha s}(k)\right) \quad (33)$$

where $K_1 = T\alpha$, $K_2 = T\omega_b$, $K_3 = T\alpha L_m \frac{I_b}{\psi_b}$, $K_4 = T\alpha\beta\frac{\psi_b}{I_b}$, $K_5 = T\beta\frac{\psi_b\omega_b}{I_b}$,

$K_6 = T\gamma$, $K_7 = T\frac{1}{\sigma L_s}\frac{V_b}{I_b}$, $K_8 = 1.5n_p\frac{L_m}{L_r}\frac{\psi_b I_b}{T_b}$, $K_9 = T\frac{B}{J}$, $K_{10} = T\frac{n_p}{J}\frac{T_b}{\omega_b}$

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

	Equation Variables	**Program Variables**
Inputs	$v_{\alpha s}$	Ualpha
	$v_{\beta s}$	Ubeta
	T_l	LoadTorque
Outputs	$i_{\alpha s}$	Ialpha
	$i_{\beta s}$	Ibeta
	$\psi_{\alpha r}$	PsiAlpha
	$\psi_{\beta r}$	PsiBeta
	T_e	Torque
	ω_r	Wr
Others	K1	K1
	K2	K2
	K3	K3
	K4	K4
	K5	K5
	K6	K6
	K7	K7
	K8	K8
	K9	K9
	K10	K10
	a	Alpha

Table 1: Correspondence of notations