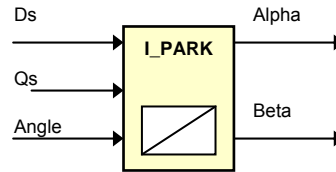


Description

This transformation projects vectors in orthogonal rotating reference frame into two phase orthogonal stationary frame.

**Availability**

This IQ module is available in one interface format:

- 1) The C interface version

Module Properties

Type: Target Independent, Application Independent

Target Devices: x281x or x280x

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

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

Item	C version	Comments
Code Size [□] (x281x/x280x)	45/45 words	
Data RAM	0 words [*]	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

^{*} Each pre-initialized “_iq” IPARK structure consumes 12 words in the data memory

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

C Interface

Object Definition

The structure of IPARK object is defined by following structure definition

```
typedef struct { _iq Alpha;      // Output: stationary d-axis stator variable
                _iq Beta;      // Output: stationary q-axis stator variable
                _iq Angle;     // Input: rotating angle (pu)
                _iq Ds;        // Input: rotating d-axis stator variable
                _iq Qs;        // Input: rotating q-axis stator variable
                void (*calc)(); // Pointer to calculation function
            } IPARK;
```

```
typedef IPARK *IPARK_handle;
```

Item	Name	Description	Format	Range(Hex)
Inputs	Ds	Direct axis(D) component of transformed signal in rotating reference frame	GLOBAL_Q	80000000-7FFFFFFF
	Qs	Quadrature axis(Q) component of transformed signal in rotating reference frame	GLOBAL_Q	80000000-7FFFFFFF
	Angle	Phase angle between stationary and rotating frame	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
Outputs	Alpha	Direct axis(d) component of the transformed signal	GLOBAL_Q	80000000-7FFFFFFF
	Beta	Quadrature axis(q) component of the transformed signal	GLOBAL_Q	80000000-7FFFFFFF

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

Special Constants and Data types

IPARK

The module definition is created as a data type. This makes it convenient to instance an interface to the Inverse Park variable transformation. To create multiple instances of the module simply declare variables of type IPARK.

IPARK_handle

User defined Data type of pointer to IPARK module

IPARK_DEFAULTS

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

Methods

```
void ipark_calc(IPARK_handle);
```

This definition implements one method viz., the inverse Park variable transformation computation function. The input argument to this function is the module handle.

Module Usage

Instantiation

The following example instances two IPARK objects
 IPARK ipark1, ipark2;

Initialization

To Instance pre-initialized objects

IPARK ipark1 = IPARK_DEFAULTS;

IPARK ipark2 = IPARK_DEFAULTS;

Invoking the computation function

ipark1.calc(&ipark1);

ipark2.calc(&ipark2);

Example

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

```
main()
{
}

void interrupt periodic_interrupt_isr()
{
    ipark1.Ds = de1;           // Pass inputs to ipark1
    ipark1.Qs = qe1;           // Pass inputs to ipark1
    ipark1.Angle = ang1;       // Pass inputs to ipark1

    ipark2.Ds = de2;           // Pass inputs to ipark2
    ipark2.Qs = qe2;           // Pass inputs to ipark2
    ipark2.Angle = ang2;       // Pass inputs to ipark2

    ipark1.calc(&ipark1);      // Call compute function for ipark1
    ipark2.calc(&ipark2);      // Call compute function for ipark2

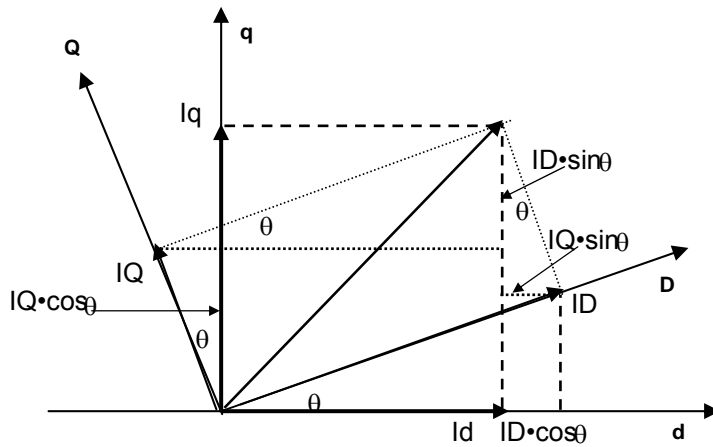
    ds1 = ipark1.Alpha;        // Access the outputs of ipark1
    qs1 = ipark1.Beta;         // Access the outputs of ipark1

    ds2 = ipark2.Alpha;        // Access the outputs of ipark2
    qs2 = ipark2.Beta;         // Access the outputs of ipark2
}
```

Technical Background

Implements the following equations:

$$\begin{cases} Id = ID \times \cos \theta - IQ \times \sin \theta \\ Iq = ID \times \sin \theta + IQ \times \cos \theta \end{cases}$$



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

	Equation Variables	Program Variables
Inputs	ID	Ds
	IQ	Qs
	θ	Angle
Outputs	id	Alpha
	iq	Beta

Table 1: Correspondence of notations