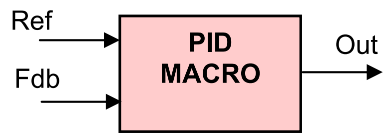
PID_REG3	*Digital PID Controller with Anti-windup*

Description

This module implements a 32-bit digital PID controller with anti-windup correction. It can be used for PI or PD controller as well. In this digital PID controller, the differential equation is transformed to the difference equation by means of the backward approximation.



Availability

This IQ module is available in one interface format:

1) The C interface version

Module Properties

Type: Target Independent, Application Dependent

Target Devices: 28x Fixed and Floating Point devices

C Version File Names: pid_reg3.h

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

C Interface

Object Definition

The structure of PID_REG3 object is defined by following structure definition

```
typedef struct {  _iq  Ref;          // Input: Reference input
                  _iq  Fdb;          // Input: Feedback input
                  _iq  Err;          // Variable: Error
                  _iq  Kp;           // Parameter: Proportional gain
                  _iq  Up;           // Variable: Proportional output
                  _iq  Ui;           // Variable: Integral output
                  _iq  Ud;           // Variable: Derivative output
                  _iq  OutPreSat;    // Variable: Pre-saturated output
                  _iq  OutMax;       // Parameter: Maximum output
                  _iq  OutMin;       // Parameter: Minimum output
                  _iq  Out;          // Output: PID output
                  _iq  SatErr;       // Variable: Saturated difference
                  _iq  Ki;           // Parameter: Integral gain
                  _iq  Kc;           // Parameter: Integral correction gain
                  _iq  Kd;           // Parameter: Derivative gain
                  _iq  Up1;          // History: Previous proportional output
               } PIDREG3;

typedef PIDREG3 *PIDREG3_handle;
```

Module Terminal Variables/Macros

Item	Name	Description	Format*	Range(Hex)
Input	Ref	Reference input	GLOBAL_Q	80000000-7FFFFFFF
	Fdb	Feedback input	GLOBAL_Q	80000000-7FFFFFFF
	OutMax	Maximum PID32 module output	GLOBAL_Q	80000000-7FFFFFFF
	OutMin	Minimum PID32 module output	GLOBAL_Q	80000000-7FFFFFFF
Output	Out	PID Output (Saturated)	GLOBAL_Q	80000000-7FFFFFFF
PID parameter	Kp	Proportional gain	GLOBAL_Q	80000000-7FFFFFFF
	Ki	Integral gain	GLOBAL_Q	80000000-7FFFFFFF
	Kd	Derivative gain	GLOBAL_Q	80000000-7FFFFFFF
	Kc	Integral correction gain	GLOBAL_Q	80000000-7FFFFFFF
Internal	Err	Error=Reference-feedback	GLOBAL_Q	80000000-7FFFFFFF
	SatErr	SatErr=output-preSatOut	GLOBAL_Q	80000000-7FFFFFFF
	Up	Proportional output	GLOBAL_Q	80000000-7FFFFFFF
	Up1	Previous proportional output	GLOBAL_Q	80000000-7FFFFFFF
	Ui	Integral output	GLOBAL_Q	80000000-7FFFFFFF
	Ud	Differential output	GLOBAL_Q	80000000-7FFFFFFF
	OutPreSat	PID output before saturation	GLOBAL_Q	80000000-7FFFFFFF

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

Special Constants and Data types

PIDREG3
The module definition is created as a data type. This makes it convenient to instance an interface to the PID module. To create multiple instances of the module simply declare variables of type PIDREG3.

PIDREG3_handle
User defined Data type of pointer to PID_REG3 module

PIDREG3_DEFAULTS
Structure symbolic constant to initialize PID_REG3 module. This provides the initial values to the terminal variables as well as method pointers.

Methods

PID_MACRO(PIDREG3_handle);

This macro implements the digital PID controller (IQ implementation) using backward approximation technique. The input argument to this macro is the module handle.

Module Usage

Instantiation
The following example instances two PID objects
```
PIDREG3  pid1, pid2;
```

Initialization
To Instance pre-initialized objects
```
PIDREG3 pid1 = PIDREG3_DEFAULTS;
PIDREG3 pid2 = PIDREG3_DEFAULTS;
```

Invoking the computation macro
```
PID_MACRO(pid1);
PID_MACRO(pid2);
```

Example

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

```
/* Instance the PID_REG3 module */
PIDREG3   pid1=PIDREG3_DEFAULTS;
PIDREG3   pid2=PIDREG3_DEFAULTS;

main()
{

        pid1.Kp = _IQ(0.5);              // Pass _iq parameters to pid1
        pid1.Ki  = _IQ(0.001);           // Pass _iq parameters to pid1
        pid1.Kd = _IQ(0.01);             // Pass _iq parameters to pid1
        pid1.Kc = _IQ(0.9);              // Pass _iq parameters to pid1
```

```
        pid2.Kp = _IQ(0.8);                 // Pass _iq parameters to pid2
        pid2.Ki  = _IQ(0.0001);             // Pass _iq parameters to pid2
        pid2.Kd = _IQ(0.02);                // Pass _iq parameters to pid2
        pid2.Kc = _IQ(0.8);                 // Pass _iq parameters to pid2

}

void interrupt periodic_interrupt_isr()
{
        pid1.Ref = input1_1;                // Pass _iq inputs to pid1
        pid1.Fdb = input1_2;                // Pass _iq inputs to pid1
        pid2.Ref = input2_1;                // Pass _iq inputs to pid2
        pid2.Fdb = input2_2;                // Pass _iq inputs to pid2

        PID_MACRO(pid1);                    // Call compute macro for pid1
        PID_MACRO(pid2);                    // Call compute macro for pid2

        output1 = pid1.Out;                 // Access the output of pid1
        output2 = pid2.Out;                 // Access the output of pid2
}
```

Technical Background

The block diagram of a conventional PID controller with anti-windup correction can be shown in Figure 1.

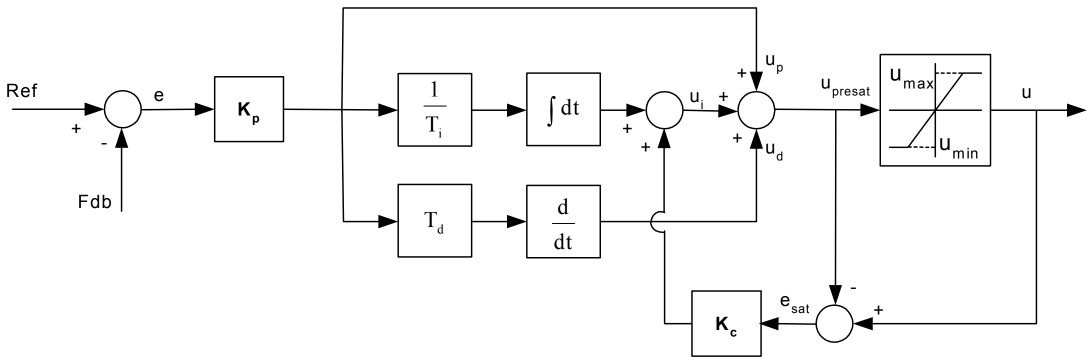


Figure 1: Block diagram of PID controller with anti-windup

The differential equation for PID controller with anti-windup before saturation is described in the following equation [1].

$$u_{presat}(t) = u_p(t) + u_i(t) + u_d(t) \quad (1)$$

Each term can be expressed as follows:

Proportional term: $$u_p(t) = K_P e(t) \quad (2)$$

Integral term with saturation correction:

$$u_i(t) = \frac{K_P}{T_i}\int_0^t e(\varsigma)d\varsigma + K_c\big(u(t) - u_{presat}(t)\big) \quad (3)$$

Derivative term: $$u_d(t) = K_P T_d \frac{de(t)}{dt} \quad (4)$$

where

- $u(t)$ is the output of PID controller
- $u_{presat}(t)$ is the output before saturation
- $e(t)$ is the error between the reference and feedback variables
- K_p is the proportional gain of PID controller
- T_i is the integral time (or reset time) of PID controller
- T_d is the derivative time of PID controller
- K_c is the integral correction gain of PID controller

Equations (1)-(4) can be discretized using backward approximation as follows:

Pre-saturated output:

$$u_{presat}(k) = u_p(k) + u_i(k) + u_d(k) \quad (5)$$

Proportional term:

$$u_p(k) = K_P e(k) \tag{6}$$

Integral term with saturation correction:

$$u_i(k) = u_i(k-1) + K_P \frac{T}{T_i} e(k) + K_c\left(u(k) - u_{presat}(k)\right) \tag{7}$$

Derivative term:

$$u_d(k) = K_P \frac{T_d}{T}\left(e(k) - e(k-1)\right) \tag{8}$$

Defining $K_i = \frac{T}{T_i}$, and $K_d = \frac{T_d}{T}$, then integral with saturation correction and derivative terms finally become

$$u_i(k) = u_i(k-1) + K_i u_p(k) + K_c\left(u(k) - u_{presat}(k)\right) \tag{9}$$

$$u_d(k) = K_d\left(u_p(k) - u_p(k-1)\right) \tag{10}$$

where T is sampling period (sec).

Table 1 shows the correspondence of notation between variables used here and variables used in the program (i.e., pid_reg3.c and pid_reg3.h). The software module requires that both input and output variables are in per unit values.

	Equation Variables	**Program Variables**
Inputs	Ref	Ref
	Fdb	Fdb
Output	$u(k)$	Out
Others	$e(k)$	Err
	$u_p(k)$	Up
	$u_p(k-1)$	Up1
	$u_i(k)$	Ui
	$u_d(k)$	Ud
	$u_{presat}(k)$	OutPreSat
	$e_{sat}(k)$	SatErr
	K_p	Kp
	K_i	Ki
	K_d	Kd
	K_c	Kc

Table 1: Correspondence of notations

References:

[1] G.F. Franklin, D.J. Powell, and M.L. Workman, Digital Control of Dynamic Systems, Addison-Wesley, 1990.