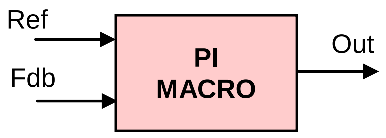
PI Controller

Description This module implements a simple 32-bit digital PI controller with anti-windup correction.



Availability C interface version

Module Properties **Type:** Target Independent

Target Devices: 28x Fixed or Floating Point

C Version File Names: pi.h

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

C Interface

Object Definition

The structure of PI object is defined by following structure definition

```
typedef struct {
                Ref         // Input: reference set-point
                Fbk         // Input: feedback
                Out         // Output: controller output
                Kp          // Parameter: proportional loop gain
                Ki          // Parameter: integral gain
                Umax        // Parameter: upper saturation limit
                Umin        // Parameter: lower saturation limit
                up          // Data: proportional term
                ui          // Data: integral term
                v1          // Data: pre-saturated controller output
                i1          // Data: integrator storage: ui(k-1)
                w1          // Data: saturation record: [u (k-1) - v(k-1)]
               } PI;
```

Module Terminal Variables/Macros

Item	Name	Description	Format*	Range(Hex)
Input	Ref	Reference input	GLOBAL_Q	80000000-7FFFFFFF
	Fbk	Feedback input	GLOBAL_Q	80000000-7FFFFFFF
	UMax	Maximum PI32 module output	GLOBAL_Q	80000000-7FFFFFFF
	UMin	Minimum PI32 module output	GLOBAL_Q	80000000-7FFFFFFF
Output	Out	PI Output (Saturated)	GLOBAL_Q	80000000-7FFFFFFF
PI parameter	Kp	Proportional gain	GLOBAL_Q	80000000-7FFFFFFF
	Ki	Integral gain	GLOBAL_Q	80000000-7FFFFFFF
Internal	up	Proportional term	GLOBAL_Q	80000000-7FFFFFFF
	ui	Integral term	GLOBAL_Q	80000000-7FFFFFFF
	v1	Pre-saturated controller output	GLOBAL_Q	80000000-7FFFFFFF
	i1	Integrator storage	GLOBAL_Q	80000000-7FFFFFFF
	w1	Saturation record	GLOBAL_Q	80000000-7FFFFFFF

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

Special Constants and Data types

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

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

Module Usage

Instantiation
The following example instances two PI objects
PI pi1, pi2;

Initialization
To Instance pre-initialized objects
PI pi1 = PI_DEFAULTS;
PI pi2 = PI_DEFAULTS;

Invoking the computation macro
PI_MACRO(pi1);
PI_MACRO(pi2);

Example

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

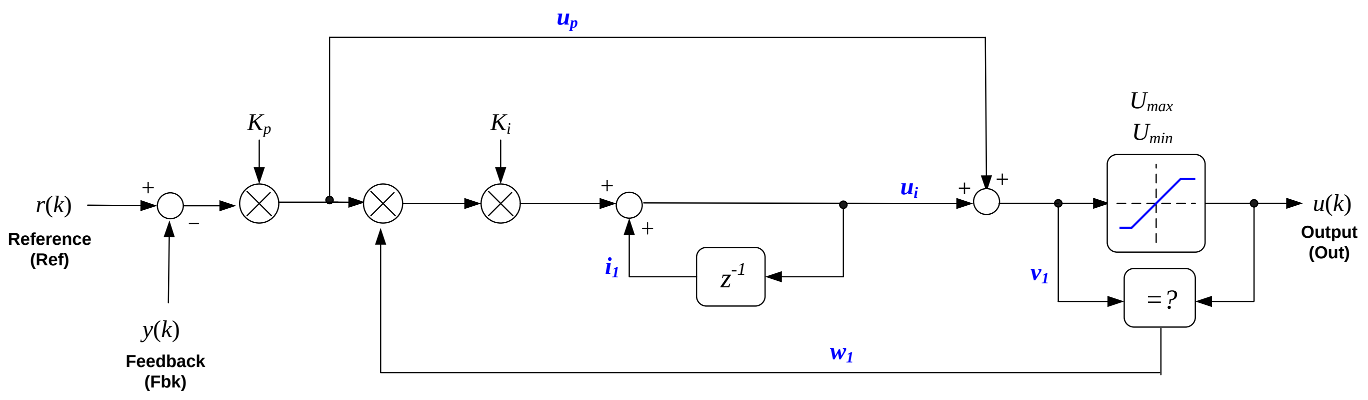
```
 /* Instance the PI module */
PI   pi1=PI_DEFAULTS;
PI   pi2=PI_DEFAULTS;

main()
{

        pi1.Kp = _IQ(0.5);              // Pass _iq parameters to pi1
        pi1.Ki  = _IQ(0.001);           // Pass _iq parameters to pi1
        pi1.Umax =_IQ(0.9);             // Pass _iq parameters to pi1
        pi1.Umin  =_IQ(-0.9);           // Pass _iq parameters to pi1

        pi2.Kp = _IQ(0.8);              // Pass _iq parameters to pi2
        pi2.Ki  = _IQ(0.0001);          // Pass _iq parameters to pi2
        pi1.Umax =_IQ(0.9);             // Pass _iq parameters to pi2
        pi1.Umin  =_IQ(-0.9);           // Pass _iq parameters to pi2


}

void interrupt periodic_interrupt_isr()
{
        pi1.Ref = input1_1;             // Pass _iq inputs to pi1
        pi1.Fdb = input1_2;             // Pass _iq inputs to pi1
        pi2.Ref = input2_1;             // Pass _iq inputs to pi2
        pi2.Fdb = input2_2;             // Pass _iq inputs to pi2

        PI_MACRO(pi1);                  // Call compute macro for pi1
        PI_MACRO(pi2);                  // Call compute macro for pi2

        output1 = pi1.Out;              // Access the output of pi1
        output2 = pi2.Out;              // Access the output of pi2
}
```

Technical Background

The PI_cntl module implements a basic summing junction and P+I control law with the following features:

- Programmable output saturation
- Independent reference weighting on proportional path
- Anti-windup integrator reset

The PI controller is a sub-set of the PID controller. All input, output and internal data is in I8Q24 fixed-point format. A block diagram of the internal controller structure is shown below.



a) Proportional path

The proportional path is a direct connection between the error term and a summing junction with the integral path. The error term is:

$$u_p(k) = e(k) = K_p[r(k) - y(k)] \quad \textbf{(1)}$$

b) Integral path

The integral path consists of a discrete integrator which is pre-multiplied by a term derived from the output module. The term w1 is either zero or one, and provides a means to disable the integrator path when output saturation occurs. This prevents the integral term from “winding up” and improves the response time on recovery from saturation. The integrator law used is based on a backwards approximation.

$$u_i(k) = u_i(k-1) + K_i e(k) \quad \textbf{(2)}$$

c) Output path

The output path contains a summing block to sum the proportional and integral controller terms. The result is then saturated according to user programmable upper and lower limits to give the controller output.

The pre-and post-saturated terms are compared to determine whether saturation has occurred, and if so, a zero or one result is produced which is used to disable the integral path (see above). The output path law is defined as follows.

$$v_1(k) = u_p(k) + u_i(k) \quad \textbf{(3)}$$

$$u(k) = \begin{cases} U_{\max} : v_1(k) > U_{\max} \\ U_{\min} : v_1(k) < U_{\min} \\ v_1(k) : U_{\min} < v_1(k) < U_{\max} \end{cases} \quad \textbf{(4)}$$

$$w_1(k) = \begin{cases} 0 : v_1(k) \neq u(k) \\ 1 : v_1(k) = u(k) \end{cases} \quad \textbf{(5)}$$

Tuning the P+I controller

Default values for the controller coefficients are defined in the macro header file which apply unity gain through the proportional path, and disable both integral and derivative paths. A suggested general technique for tuning the controller is now described.

Step 1. Ensure integral is set to zero and proportional gain set to one.

Step 2. Gradually adjust proportional gain variable (K_p) while observing the step response to achieve optimum rise time and overshoot compromise.

Step 3. If necessary, gradually increase integral gain (K_i) to optimize the return of the steady state output to nominal. The controller will be very sensitive to this term and may become unstable so be sure to start with a very small number. Integral gain will result in an increase in overshoot and oscillation, so it may be necessary to slightly decrease the K_p term again to find the best balance. Note that if the integral gain is used then set to zero, a small residual term may persist in u_i.