PID Motor Control

Controller Functions

Functions

__STATIC_FORCEINLINE **float32_t**	**arm_pid_f32** (**arm_pid_instance_f32** ***S**, **float32_t** in) Process function for the floating-point PID Control. More...
__STATIC_FORCEINLINE **q31_t**	**arm_pid_q31** (**arm_pid_instance_q31** ***S**, **q31_t** in) Process function for the Q31 PID Control. More...
__STATIC_FORCEINLINE **q15_t**	**arm_pid_q15** (**arm_pid_instance_q15** ***S**, **q15_t** in) Process function for the Q15 PID Control. More...
void	**arm_pid_init_f32** (**arm_pid_instance_f32** ***S**, int32_t resetStateFlag) Initialization function for the floating-point PID Control. More...
void	**arm_pid_init_q15** (**arm_pid_instance_q15** ***S**, int32_t resetStateFlag) Initialization function for the Q15 PID Control. More...
void	**arm_pid_init_q31** (**arm_pid_instance_q31** ***S**, int32_t resetStateFlag) Initialization function for the Q31 PID Control. More...
void	**arm_pid_reset_f32** (**arm_pid_instance_f32** ***S**) Reset function for the floating-point PID Control. More...
void	**arm_pid_reset_q15** (**arm_pid_instance_q15** ***S**) Reset function for the Q15 PID Control. More...
void	**arm_pid_reset_q31** (**arm_pid_instance_q31** ***S**) Reset function for the Q31 PID Control. More...

Description

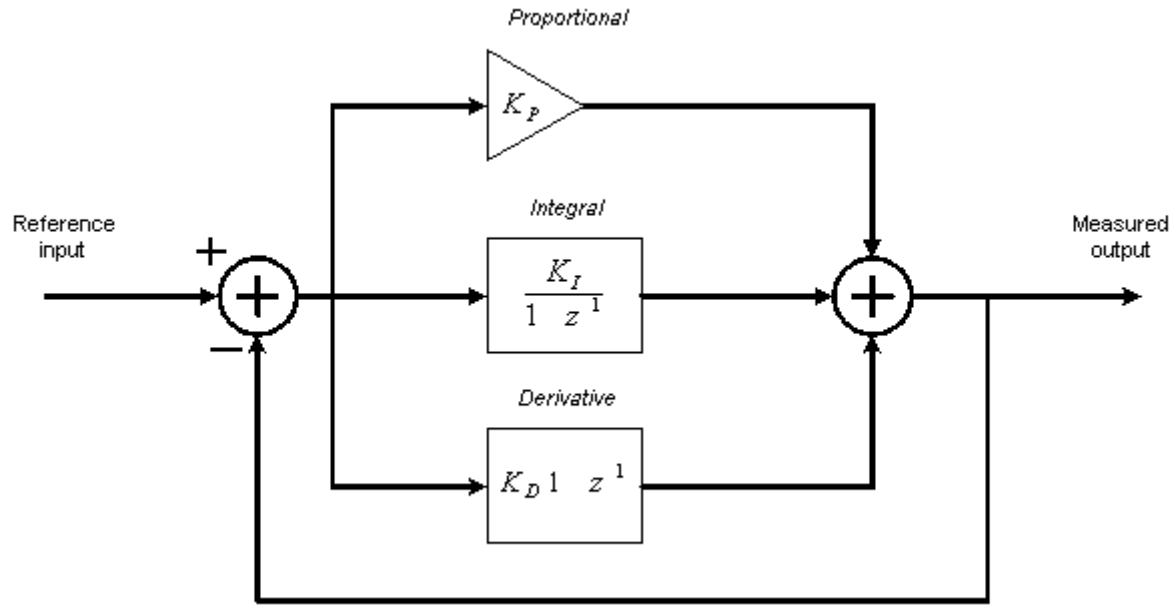
A Proportional Integral Derivative (PID) controller is a generic feedback control loop mechanism widely used in industrial control systems. A PID controller is the most commonly used type of feedback controller.

This set of functions implements (PID) controllers for Q15, Q31, and floating-point data types. The functions operate on a single sample of data and each call to the function returns a single processed value. `S` points to an instance of the PID control data structure. `in` is the input sample value. The functions return the output value.

Algorithm:

```
y[n] = y[n-1] + A0 * x[n] + A1 * x[n-1] + A2 * x[n-2]
A0 = Kp + Ki + Kd
A1 = (-Kp ) - (2 * Kd )
A2 = Kd
```

where `Kp` is proportional constant, `Ki` is Integral constant and `Kd` is Derivative constant



Proportional Integral Derivative Controller

The PID controller calculates an "error" value as the difference between the measured output and the reference input. The controller attempts to minimize the error by adjusting the process control inputs. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing.

Instance Structure

The Gains A0, A1, A2 and state variables for a PID controller are stored together in an instance data structure. A separate instance structure must be defined for each PID Controller. There are separate instance structure declarations for each of the 3 supported data types.

Reset Functions

There is also an associated reset function for each data type which clears the state array.

Initialization Functions

There is also an associated initialization function for each data type. The initialization function performs the following operations:

- Initializes the Gains A0, A1, A2 from Kp,Ki, Kd gains.
- Zeros out the values in the state buffer.

Instance structure cannot be placed into a const data section and it is recommended to use the initialization function.

Fixed-Point Behavior

Care must be taken when using the fixed-point versions of the PID Controller functions. In particular, the overflow and saturation behavior of the accumulator used in each function must be considered. Refer to the function specific documentation below for usage guidelines.

Function Documentation

```
__STATIC_FORCEINLINE float32_t arm_pid_f32 ( arm_pid_instance_f32 * S,
                                             float32_t              in
                                           )
```

Parameters

[in,out]	**S**	is an instance of the floating-point PID Control structure
[in]	**in**	input sample to process

Returns

processed output sample.

void arm_pid_init_f32 (arm_pid_instance_f32 * S,
int32_t resetStateFlag
)

Parameters

[in,out]	**S**	points to an instance of the PID structure
[in]	**resetStateFlag**	• value = 0: no change in state • value = 1: reset state

Returns
none

Details
The `resetStateFlag` specifies whether to set state to zero or not.
The function computes the structure fields: `A0`, `A1` `A2` using the proportional gain(`Kp`), integral gain(`Ki`) and derivative gain(`Kd`) also sets the state variables to all zeros.

void arm_pid_init_q15 (arm_pid_instance_q15 * S,
int32_t resetStateFlag
)

Parameters

[in,out]	**S**	points to an instance of the Q15 PID structure
[in]	**resetStateFlag**	• value = 0: no change in state • value = 1: reset state

Returns
none

Details
The `resetStateFlag` specifies whether to set state to zero or not.
The function computes the structure fields: `A0`, `A1` `A2` using the proportional gain(`Kp`), integral gain(`Ki`) and derivative gain(`Kd`) also sets the state variables to all zeros.

void arm_pid_init_q31 (arm_pid_instance_q31 * S,
int32_t resetStateFlag
)

Parameters

[in,out]	**S**	points to an instance of the Q31 PID structure
[in]	**resetStateFlag**	• value = 0: no change in state • value = 1: reset state

Returns
none

Details
The `resetStateFlag` specifies whether to set state to zero or not.
The function computes the structure fields: `A0`, `A1` `A2` using the proportional gain(`Kp`), integral gain(`Ki`) and derivative gain(`Kd`) also sets the state variables to all zeros.

__STATIC_FORCEINLINE q15_t arm_pid_q15 (arm_pid_instance_q15 * S, q15_t in)

Parameters

[in,out] **S** points to an instance of the Q15 PID Control structure

[in] **in** input sample to process

Returns

processed output sample.

Scaling and Overflow Behavior

The function is implemented using a 64-bit internal accumulator. Both Gains and state variables are represented in 1.15 format and multiplications yield a 2.30 result. The 2.30 intermediate results are accumulated in a 64-bit accumulator in 34.30 format. There is no risk of internal overflow with this approach and the full precision of intermediate multiplications is preserved. After all additions have been performed, the accumulator is truncated to 34.15 format by discarding low 15 bits. Lastly, the accumulator is saturated to yield a result in 1.15 format.

__STATIC_FORCEINLINE q31_t arm_pid_q31 (arm_pid_instance_q31 * S, q31_t in)

Parameters

[in,out] **S** points to an instance of the Q31 PID Control structure

[in] **in** input sample to process

Returns

processed output sample.

Scaling and Overflow Behavior

The function is implemented using an internal 64-bit accumulator. The accumulator has a 2.62 format and maintains full precision of the intermediate multiplication results but provides only a single guard bit. Thus, if the accumulator result overflows it wraps around rather than clip. In order to avoid overflows completely the input signal must be scaled down by 2 bits as there are four additions. After all multiply-accumulates are performed, the 2.62 accumulator is truncated to 1.32 format and then saturated to 1.31 format.

void arm_pid_reset_f32 (arm_pid_instance_f32 * S)

Parameters

[in,out] **S** points to an instance of the floating-point PID structure

Returns

none

Details

The function resets the state buffer to zeros.

void arm_pid_reset_q15 (arm_pid_instance_q15 * S)

Parameters

`[in,out]` **S** points to an instance of the Q15 PID structure

Returns

none

Details

The function resets the state buffer to zeros.

void arm_pid_reset_q31 (arm_pid_instance_q31 * S)

Parameters

`[in,out]` **S** points to an instance of the Q31 PID structure

Returns

none

Details

The function resets the state buffer to zeros.