Module Document: CNTL_PI

Texas Instruments – C2000 DSP System Applications Group

Contents

Figures

Tables

1 CNTL_PI module documentation

This module implements a proportional-integral controller, by implementing the second order difference equation shown in equation (1) below.

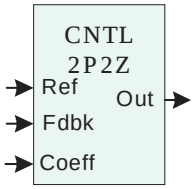


Figure 1. PI controller module

$$U(n) = K_P \cdot e(n) + K_I \cdot I(n) + K_F \cdot R(n)$$
$$I(n) = I(n-1) + e(n) \quad \text{..... (1)}$$

1.1 Module Properties

This section describes module properties, such as compatible devices, components, invocation etc. The CNTL_PI module has the following dependencies:

Module	Dependency
CPU dependency	C28x
Device dependency	C28x
Target application	Closed loop control.
Math format (precision)	32 bit fixed Q

Table 1. CNTL_PI module dependencies

The CNTL_PI module has the following components:

Component	Present
C-based initialization	No
ASM interrupt initialization	Yes
ASM runtime macro	Yes

Table 2. CNTL_PI module components

The CNTL_PI module has the following miscellaneous properties:

Property name	Property value
Multiple instance support	Yes
Reentrant	No
Accessible from 'C' environment	Yes
Full configuration from 'C' environment	Yes
Input / Output connection	Pointer to signal net.

Table 3. CNTL_PI module miscellaneous properties

Component	Files
Macro source:	C:\tidcs\DPS_C280x\vXYZ[1]\dplib280x\dplib280x.inc
C Interface:	C:\tidcs\DPS_C280x\vXYZ\dplib280x\dplib280x.h
C source	C:\tidcs\DPS_C280x\vXYZ\dplib280x\dplib280x.src
Object file archive	C:\tidcs\DPS_C280x\vXYZ\dplib280x\dplib280x.lib

Table 4. CNTL_PI module component files

1.2 CNTL_PI Module inputs

Input name	Description	Data Format	Range
Ref	Reference Set point	Pointer to 16-bit fixed point input data	Q15: [-1, 1] or [-32768, 32767]
Fdbk	Feedback, used to calculate the error term	Pointer to 16-bit fixed point input data	Q15: [-1, 1] or [-32768, 32767]
Coeff	Pointer to coefficient structure	Pointer to structure	NA

Table 5. CNTL_PI module inputs

1.3 CNTL_PI Module output definitions

Output name	Description	Data Format	Range
Out	Controller output	Pointer to 16-bit fixed point output data	Q15: [-1, 1] or [-32768, 32767]

Table 6. CNTL_PI module outputs

[1] The xyz represents the version number directory level. For instance, a 1.00 release would have v100 in its directory path, and v210 would indicate a release 2.10.

1.4 Module coefficient configuration

The controller coefficients equation (1) above are specified in a record in memory. All coefficients are 32 bits wide, and are in a Q format as described below. The co-efficient formats are shown in Table 7 below.

Coefficient	Description	Format	Range
Kp	Controller coefficient	32-bit fixed point	Q24: [-32, 31.99999]
Ki	Controller coefficient	32-bit fixed point	Q24: [-32, 31.99999]
Kf	Controller coefficient	32-bit fixed point	Q24: [-32, 31.99999]
A2	Controller coefficient	32-bit fixed point	Q24: [-32, 31.99999]
IsatUpperLimit	Integral upper saturation bound	32-bit fixed point	Q31: [-1, 1)
IsatLowerLimit	Integral lower saturation bound	32-bit fixed point	Q31: [-1, 1)
OutputUpperLimit	Control output upper saturation bound	32-bit fixed point	Q31: [-1, 1)
OutputLowerLimit	Control output lower saturation bound	32-bit fixed point	Q31: [-1, 1)

Table 7. CNTL_PI module coefficient formats

The coefficients can be declared in a C-client file and a co-efficient pointer passed to the controller via the coefficient input which expects a pointer to a coefficient structure.

```
struct CNTL_PI_COEFF_STRUCT
{       long int IsatUpperLimit;
        long int IsatLowerLimit;
        long Ki;
        long Kp;
        long Kf;
        long int OutputUpperLimit;
        long int OutputLowerLimit;
};
```

A structure of type CNTL_PI_COEFF_STRUCT must be instanced and populated with the desired values.

1.5 Module API description: CNTL_PI_INIT

Function Name: CNTL_PI_INIT

Prototype: CNTL_PI_INIT nInstance

Return value: None

Preconditions: None

This function is the assembler initialization macro, and must be called in addition to the C language initialization routine, for proper operation of the runtime macro routine. This initialization routine must be executed as part of an assembler initialization routine. This macro routine declares variables, initializes variables to known values, and sets up constants for the runtime macro routines.

- **nInstance:** Specifies the instance number of the current instance.
  - **Valid Range:** Limited only by available memory in the application.

Example: Call the CNTL_PI_INIT to initialize controller module.

```
;---------------------------------------------------------------
; ISR Initialization
;---------------------------------------------------------------
_ISR_Init:  . . . . . .  ; Other init routines go in here

            CNTL_PIZ_INIT     1

            . . . . . .  ; Other init routines go in here

            LRETR
```

1.6 Module API description: CNTL_PI

Function Name: CNTL_PI

Prototype: CNTL_PI nInstance

Return value: None.

Preconditions: The following preconditions must be satisfied:

- The ISR initialization macro CNTL_PI_INIT must be instanced in an assembler initialization routine, and must run prior to this instance of the controller routine.
- This function is the assembler run time macro, and this creates code that runs the run time computation for the PI controller. This computes the control output depending on the current reference and feedback values, and coefficients.
- nInstance**:** Specifies which controller instance is computed.
  - **Valid Range:** Limited only by available memory in the application.

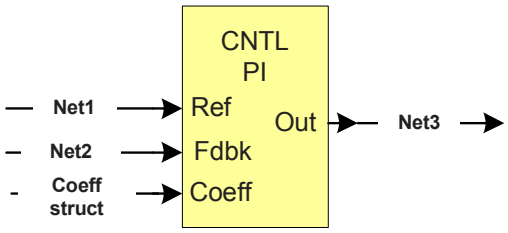
Example: Call the CNTL_PI in an assembler ISR

```
;---------------------------------------------------------
; Runtime interrupt service routine
;---------------------------------------------------------
_ISR_Run:   CONTEXT_SAVE            ;call macro


            CNTL_PI     1
;---------------------------------------------------------
EXIT_ISR: ;Interrupt management before exit
;---------------------------------------------------------
            MOVW  DP,#ETCLR1>>6
            MOV   @ETCLR1,#0x01     ; Clear EPWM1 Int flag

;---------------------------------------------------------
; Restore context & return
;---------------------------------------------------------
            CONTEXT_REST
            IRET
```

1.7 Usage Example:



Step1: Instantiate the INIT macro in assembly (this is one-time pass through code)

```
       CNTL_PI_INIT  1
```

Step2. Instantiate the run time macro in assembly (this is usually looped or ISR code)

```
; “call” the main macro
       CNTL_PI                  1
```

Step3. (optional) Declare “Signal Nets” to “connect” the module to in “C”

```
int    Net1, Net2, Net3;
```

Step4. Declare the module “Terminal pointers” in “C”

```
// CNTL_PI terminal external references for 1st instantiation
extern int *CNTL_PI_Ref1, *CNTL_PI_Fdbk1, *CNTL_PI_Out1;
struct CNTL_PI_COEFF_STRUCT   *CNTL_PI_Coeff1;
```

Step5. “Connect” the module terminals to the Signal Nets in “C”.

```
// CNTL_PI(1) connections
Struct CNTL_PI_COEFF_STRUCT LoopCoeff;
       LoopCoeff.IsatUpperLimit    = 0x38000000;
       LoopCoeff.IsatLowerLimit    = 0xC0000000;
       LoopCoeff.Ki                = 0x02000000;
       LoopCoeff.Kp                = 0x00000000;
       LoopCoeff.Kf                = 0x00000000;
       LoopCoeff.OutputLowerLimit  = 0x00000000;
       LoopCoeff.OutputUpperLimit  = 0x7eeffff;

       CNTL_PI_Ref1  = &Net1;
       CNTL_PI_Out1  = &Net2;
       CNTL_PI_Fdbk1 = &Net3;
       CNTL_PI_Coeff = &LoopCoeff;
```