ADSP-21990: Implementation of PI Controllers

AN21990-13

Table of Contents

SUMMARY 3

1 CHARACTERISATION OF THE PI CONTROLLER 3

1.1 The continuous time domain 3

1.2 The discrete time domain 4
1.2.1 Zero order hold (ZOH) 4
1.2.2 First order hold (FOH) 5

1.3 Implementation on a digital signal processor 5

1.4 Accuracy of the routine 6

1.5 Anti Wind-up 6

2 USING THE PI ROUTINES 7

2.1 Determination of the coefficients 7

2.2 Usage of the controller routine 8

2.3 Usage of the DSP registers 8

2.4 Access to the library through the header file: pi.h 8

2.5 The program code 9

3 SOFTWARE EXAMPLE: PI CONTROL 9

3.1 Using the PI routines: Calculation of coefficients 9

3.2 The main program: main.dsp 9

3.3 Main.h 10

4 EXPERIMENTAL RESULTS 10

Summary

PI controllers are universally known because of their flexibility combined with the relatively easy tuning. This application note describes the conversion from the continuous to the discrete time domain, which is essential for every implementation on a digital processor. A routine is then presented that implements the discrete time representation of the PI controller.

1 Characterisation of the PI controller

1.1 The continuous time domain

PI controllers are in most cases analysed and tuned in the continuous time domain. The corresponding transfer function is given as

$$U(s) = \left(K_P + \frac{K_I}{s}\right) \cdot I(s) \tag{1.1}$$

where I and U denote the input error signal and the controller's output, respectively, s is the Laplace variable and K_P and K_I are the two parameters of the PI controller associated with the proportional (P) and integral (I) part. However, tuning a PI controller is not very intuitive when specifying these two parameters. The equation above may be rewritten as follows:

$$U(s) = K_P \cdot \left(1 + \frac{K_I}{K_P} \cdot \frac{1}{s}\right) \cdot I(s) = K_P \cdot \omega_{PI} \cdot \left(\frac{\frac{s}{\omega_{PI}} + 1}{s}\right) \cdot I(s) \tag{1.2}$$

where $\omega_{PI} = \frac{K_I}{K_P}$ (expressed in [rad/s]). Evidently, this transfer function presents a pole in the origin and a zero located at ω_{PI}. The gain at high frequencies is given by K_P itself. The following figure illustrates a typical bode diagram of a PI controller, with K_P and ω_{PI} set to 0.25 (-12dB) and 50Hz (314rad/s), respectively.

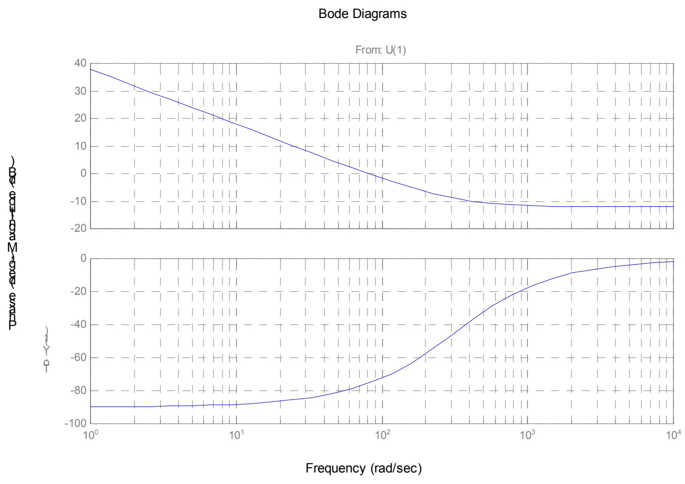

Figure 1 Typical bode diagram of a PI controller

In the following, the controller is supposed to be tuned in the continuous time domain by K_P and ω_{PI}.

1.2 The discrete time domain

The transition from the continuous to the discrete time domain entails that the integral operation has to be approximated by a discrete summation. There are several methods for replacing the integral. Two of them will be discussed hereafter.

1.2.1 Zero order hold (ZOH)

With this approach, the signal is sampled at the instant k and held constant until the next sampling instant k+1. The following figure illustrates this.

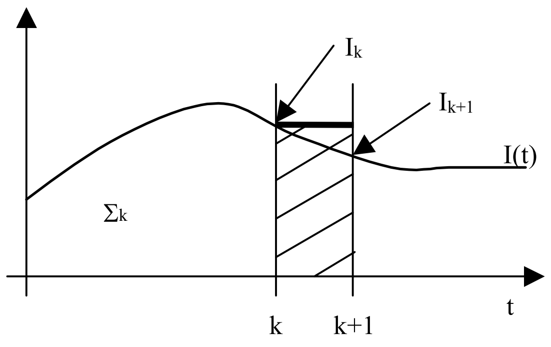

Figure 2 Integral with ZOH approximation

The integral operation is approximated by accumulating the rectangular areas. Denoting the sum at instant k with Σ_k, the signal at instant k with I_k and the sample time with T_{sample}, the "integration" is achieved by:

$$\Sigma_{k+1} = \Sigma_k + I_k \cdot T_{sample} \tag{1.3}$$

As known, the same equation may be expressed in the z-domain by

$$z \cdot \Sigma(z) = \Sigma(z) + I(z) \cdot T_{sample} \tag{1.4}$$

which leads to

$$\Sigma(z) = \frac{T_{sample}}{z-1} I(z) \tag{1.5}$$

Therefore the continuous integration $\frac{1}{s}$ in equation (1.1) is replaced by the 1st factor in the equation above. The transfer function in the discrete domain is obtained from (1.1) and (1.5) as:

$$U(z) = K_P \cdot \left(1 + \omega_{PI} \cdot \frac{T_{sample}}{z-1}\right) I(z) = \frac{K_P z + K_P \cdot (\omega_{PI} \cdot T_{sample} - 1)}{z-1} I(z) \tag{1.6}$$

This corresponds to the following difference equation:

$$U_{k+1} = K_P \cdot I_{k+1} + K_P (\omega_{PI} \cdot T_{sample} - 1) \cdot I_k + U_k \tag{1.7}$$

1.2.2 First order hold (FOH)

A slightly improved approach is shown in the following figure.

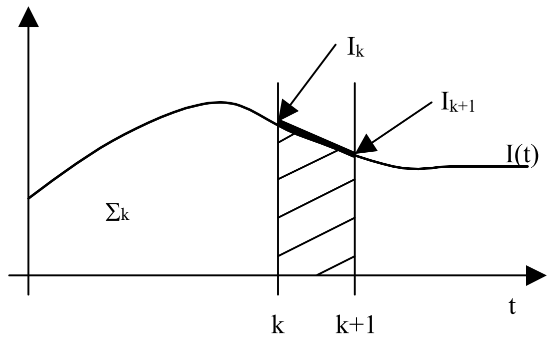

Figure 3 Integral with FOH approximation

The integral operation is approximated by accumulating the trapezoidal areas. Denoting the sum at instant k with Σ_k, the signal at instant k with I_k and the sample time with T_{sample}, the "integration" is achieved by:

$$\Sigma_{k+1} = \Sigma_k + \frac{I_k + I_{k+1}}{2} \cdot T_{sample} \tag{1.8}$$

where the last term is derived from the known formula for the area of a trapezoid. Following a similar procedure as in the previous section, this may be expressed in the z-domain by

$$z\Sigma(z) = \Sigma(z) + I(z) \cdot \frac{1+z}{2} \cdot T_{sample} \tag{1.9}$$

which results in

$$\Sigma(z) = \frac{T_{sample}}{2} \frac{z+1}{z-1} \cdot I(z) \tag{1.10}$$

This time, the continuous integration $\frac{1}{s}$ in equation (1.1) is replaced by the whole factor preceding I(z). The transfer function in the discrete domain is obtained from (1.1) and (1.10) as:

$$U(z) = K_P \cdot \left(1 + \omega_{PI} \cdot \frac{T_{sample}}{2} \frac{z+1}{z-1}\right) I(z) = \frac{K_P \cdot \left(\frac{\omega_{PI} \cdot T_{sample}}{2} + 1\right) z + K_P \cdot \left(\frac{\omega_{PI} \cdot T_{sample}}{2} - 1\right)}{z-1} I(z) \tag{1.11}$$

This corresponds to the following difference equation:

$$U_{k+1} = K_P \left(\frac{\omega_{PI} \cdot T_{sample}}{2} + 1\right) \cdot I_{k+1} + K_P \left(\frac{\omega_{PI} \cdot T_{sample}}{2} - 1\right) \cdot I_k + U_k \tag{1.12}$$

1.3 Implementation on a digital signal processor

It is easily seen that both the difference equations of (1.7) and (1.12) are of the same form as reported below:

$$U_{k+1} = A_1 \cdot I_{k+1} + A_0 \cdot I_k + U_k \tag{1.13}$$

This is a sequence of multiply and accumulate operations that are ideally suited for implementation on a DSP such as the ADSP-21990. However, care must be taken due to the 16 bit fixed-point representation of the values. For correct operation, all coefficients and signals are to be scaled to the common 1.15 format. Assuming the input error I is already in this format, the coefficients may generally have to be scaled to lie within this range. This is achieved by introducing a scale factor B_0 and reformulating equation (1.13) as follows:

$$U_{k+1} \cdot B_0 = A_1 \cdot B_0 \cdot I_{k+1} + A_0 \cdot B_0 \cdot I_k + U_k \cdot B_0 = A_1^{sc} \cdot I_{k+1} + A_0^{sc} \cdot I_k + U_k \cdot B_0 \tag{1.14}$$

where the apex 'sc' denotes the scaled values of the coefficients. B_0 is chosen such that both coefficients are in the range between –1 and 1 with the maximal precision. In addition, if it is chosen as a power of 2, the final de-scaling is a simple shift operation. This will be explained in more detail in Section 2.1.

1.4 Accuracy of the routine

It is clear from equation (1.14) that the increments of U_k at each iteration becomes smaller as the input error decreases and as I_{k+1} does not change significantly compared to its previois value I_k. In addition, the scaled coefficients may be small. It might therefore in some cases occur that U_k is no longer incremented even with non-zero input error, which leads to a steady state error. One way to improve the controller is to increase the word length of U_k. Hence, the proposed library includes a 32-bit version of the algorithm. An example output will be included in Section 4.

1.5 Anti Wind-up

The phenomenon of wind-up of the integral part is easily avoided by saturating the current sum U_{k+1} whenever it exceeds +1 or is less than –1. This feature is incorporated into the routines presented here. Refer to Section 2.5 for more details on this issue.

2 Using the PI routines

2.1 Determination of the coefficients

Once the PI controller is tuned in the continuous time domain (therefore given K_P and ω_{PI}), the user has to choose a suitable sample time T_{sample}. It may be shown that for values such that $\omega_{PI} \cdot T_{sample} \le \frac{1}{20}$ or $\omega_{PI} \cdot T_{sample} \le \frac{1}{10}$ in the cases of the zero or first order hold, respectively, the approximation error is less than 3%. For values that are higher than these limits, the discrete controller will no longer behave like its continuous pendant. Next, A_1 and A_0 are to be determined by comparison of the coefficients of equation (1.13) with those in (1.7) and (1.12) for ZOH and FOH, respectively. At last, an appropriate scale factor has to be found. It is easily seen that all coefficients of equation (1.13) are converted into the 1.15 format if they are divided by the following value:

$$B_{real} = \max\left(|A_1|,|A_0|,1\right) \tag{2.1}$$

However, in order to simplify the de-scaling and the anti wind-up procedure, the scale factor is chosen to be a power of two, as mentioned in section 1.3. If n denotes a non-negative integer values such that $B_0 = 2^{-n}$, it is clear that n may be found by

$$n = \overline{\log_2\left(B_{real}\right)} \tag{2.2}$$

where the bar above the expression stands for the operation of rounding towards the next <u>higher</u> integer. The following table reassumes all operations that are required to implement the PI controller.

Table 1 Determination of the coefficients

Parameter	***ZOH***	***FOH***
T_{sample}	$T_{sample} \le \frac{1}{20} \cdot \omega_{PI}^{-1}$	$T_{sample} \le \frac{1}{10} \cdot \omega_{PI}^{-1}$
A_1	$A_1 = K_P$	$A_1 = K_P\left(\frac{\omega_{PI} \cdot T_{sample}}{2} + 1\right)$
A_0	$A_0 = K_P\left(\omega_{PI} \cdot T_{sample} - 1\right)$	$A_0 = K_P\left(\frac{\omega_{PI} \cdot T_{sample}}{2} - 1\right)$
n	$n = \overline{\log_2\left(\max\left(\|A_1\|,\|A_0\|,1\right)\right)}$	
B_0	$B_0 = 2^{-n}$	
A_1^{sc}	$A_1^{sc} = A_1 \cdot B_0$	
A_0^{sc}	$A_0^{sc} = A_0 \cdot B_0$	

2.2 Usage of the controller routine

The routines are developed as an easy-to-use library, which has to be linked to the user's application. The library consists of two files. The file "pi.dsp" contains the assembly code for the subroutines. This package has to be compiled and can then be linked to an application. The user has to include the header file "pi.h", which provides a function-like call to the routines. The example file in the following section will demonstrate the usage of the controller. The following table reassumes the set of macros defined in this library.

Table 2 Implemented routines

Precision	*Operation*	*Usage*	*Input*	*Output*
32 bit	Initialisation	**PI32_Init**(Delay_line, Initial_value);	none	none
	PI	**PI32**(Delay_line, Coefficients, Scale_Shift);	ar	sr1

As will be seen in more detail in Section 2.5, the PI control routine requires the coefficients A_1, A_0 stored (in this order) in program memory. This is done with a circular buffer called *Coefficients* in Table 2. Similarly, I_{k+1} and U_{k+1}(MSW and LSW) need to be stored in a circular buffer called the *Delay_line* (in data memory and in this order) at the end of the computation since they are required for the next call to the routine. *Scale_Shift* denotes the integer value n that was defined by equation (2.2). *Initial_value* is the initial output of the PI (corresponding to the initial value of the integrator in the continuous case). A call to PI32_Init initialises the buffer Delay_line by clearing I_k and loading U_k with this parameter (resets the PI if *Initial_value* is equal to zero). At last, the input error for the PI routine (defined as difference between reference value and feedback signal) is required to be stored in the ar register (1.15 format). The output value (1.15 format) is contained in the sr1 register after executing the routine.

2.3 Usage of the DSP registers

In this library, two macros are defined, as shown in Table 2. Table 3 gives an overview of the DSP core registers that are modified by them. Besides, the delay line is also modified.

Table 3 Usage of DSP core registers for the subroutines

Precision	*Usage*	*Modified registers*
32 bit	**PI32_Init**(Delay_line, Initial_value);	I3, L3, ar, B3, M3
	PI32(Delay_line, Coefficients, Scale_Shift);	I3, L3, ax0, B3, M3, I7, L7, ay0, mx1, mx0, mr1, mr0, ar, se, sr, my0

2.4 Access to the library through the header file: pi.h

The library may be accessed by including the header file "pi.h" in the application code. The header file is intended to provide function-like calls to the PI routines. It defines the calls shown in Table 2. The usage of them requires that the buffers **Delay_line** and **Coefficients** be set-up correctly. The parameter

Initial_value serves as reset value for the initialisation routine. Note how I3 is used to point the delay line buffer in data memory and I7 points the coefficient buffer in program memory. For more information, please refer to the comments in the "pi.h" file.

2.5 The program code: pi.dsp

In this file, the ***PI32_Control_*** routine is defined. Assuming that the DAG registers I3 and I7 are correctly set (in the macro "**PI32**"), then the delay line is loaded into the core registers. The multiplication by B_0 of the last term of equation (1.14) is achieved by a shift over the entire 32-bit value U_k. Next, equation (1.14) is executed. Then the routine checks if the final shift by n (contained in ay0) will produce a resulting U_{k+1} that is bigger than 1 in absolute value. If not, the shift is executed and the output value is stored in the delay line for the next cycle. If the shifted result would be bigger than one, U_{k+1} is saturated to +1 or –1 (depending on the sign of the result) and this value is stored. For more detail, please refer to "pi.dsp".

3 Software Example: PI Control

3.1 Using the PI routines: Calculation of coefficients

The example demonstrates how to use the routines and may be used for reference in own applications. Here, the example file implements a PI controller of the following parameters:

$K_P = 0.25$

$\omega_{PI} = 314$rad/s (corresponding to a zero at 50Hz)

$T_{sample} = 100\mu s$ (corresponding to a controller frequency of 10kHz)

Note that $\omega_{PI} \cdot T_{sample} = 3.14e^{-2} < \frac{1}{20}$ so that both the ZOH and the FOH approximation introduce errors that are less than 3%. Following the procedure of Section 2.1yields the following coefficients:

	ZOH	**FOH**
$A_1^{sc} =$	0.25	0.253927
$A_0^{sc} =$	-0.242146	-0.246073
n =	0	0
$B_0 =$	1	1

The last step is the conversion of those values into hexadecimal values in 1.15 format (showing only the case of the ZOH):

$$A_1^{sc} = 0x2000$$

$$A_0^{sc} = 0xE101$$

The exponent n=0 is the third parameter of the PI macro. The corresponding label in the example code is PI_SF32.

3.2 The main program: main.dsp

It contains the initialisation and PWMSYNC and PWMTRIP interrupt service routines. Please note that this file is written for demonstrating how to use the routine, the user has to modify the program for their application. The general systems constants and PWM configuration constants[1] (main.h-see the next section) are included. Also, the PWM library (pwm21990.h) and the **PI library** (pi.h) are included.

Then, some variables such as the PI parameters are defined and initialised. Here, the PI delay line (I_k (16-bit), U_k (32-bits, split into 2 16-bit data memory locations)) is declared.

At the start of the program, initialisations such as setting the data memory page registers to accessing the internal memory is achieved. Then the PWM block and the PI controller are initialised.

The PWMSYNC interrupt service routine is executed at the PWM switching rate (10kHz here). First, the PWMSYNC interrupt status in the PWM status register is cleared. The error signal is computed by subtracting the feedback signal from the command (reference) signal. Please note that, assuming both the reference value and the feedback signal being in 1.15 format, the difference may assume values comprised between –2 and 2. Clearly, those values can no longer be represented in 1.15 format. However, by enabling the ALU saturation mode, the difference is saturated to values in the range of –1 to 1. This should normally have only minimal effects on the overall behaviour, since errors that are larger than full scale should appear only in small transition times. If nonetheless the saturation of the error is unacceptable, it is necessary to divide it by two, call the PI routine and multiply the output by two. Note also that if the reference and feedback signals are both uni-polar, the error is represented in 1.15 format and the enabling of the saturation mode is not required. Then, this error signal is stored in the AR register. As stated before, this file has to be modified by the user, here; the reference and the feedback signals have to be defined by the user according to the application. The error then serves as an input into the PI controller and the PI control routine is implemented. Thus, the sampling time of the PI controller is dictated by the PWM switching frequency, which is 10kHz here. The output from the controller is stored in the SR1 register.

In the PWMTRIP interrupt service routine, the users can define what to implement in case of a PWM fault. In this example, the status bit of the PWMTRIP interrupt in the PWM status register is cleared.

For more details on the "main.dsp", please refer to the comments in the file.

3.3 Main.h

The use of this file is to define general system parameters and constants, such as the crystal clock [kHz], core clock [kHz] and the peripheral clock [kHz] (also denoted as H_{clk}). Besides, any constants required by the library files will be defined here. In this case, the constants needed by the PWM library, such as the PWM switching frequency [Hz], the dead time [nsec] and the PWM sync pulse time [nsec] will be included in this file. Two files, the "ADSP-21990.h" and the "macro.h" are also included in the "main.h" file. The "ADSP-21990.h" defines the peripheral registers of the ADSP-21990 while the "macro.h" defines the most commonly used macro.

4 Experimental Results

The following figures show the step response (from 0 to 30%) of the PI controller, using the configuration as stated in the previous section. The output from the PI controller directly serves as the feedback signal input, thus this corresponds to a control system of unity gain. Figure 4 shows the simulation result while Figure 5 shows the results obtained from the ADSP-21990. Please note that the results from Figure 5 is obtained from the Digital to Analog Converter (DAC) such that the voltage output ranges from 0V to 2V representing –1 to +1 in digital value. Hence, 1V means a value of zero, and it is denoted as the zero line on Figure 5. As shown in Figure 4, it takes 90ms for the feedback signal to reach the reference input. This result matches the one implemented in the ADSP-21990, as shown in Figure 5. Besides, one can also see that the error dies down to zero at this point.

[1] Please refer to the comments in the "pwm21990.h" and "pwm21990.dsp" for more information.

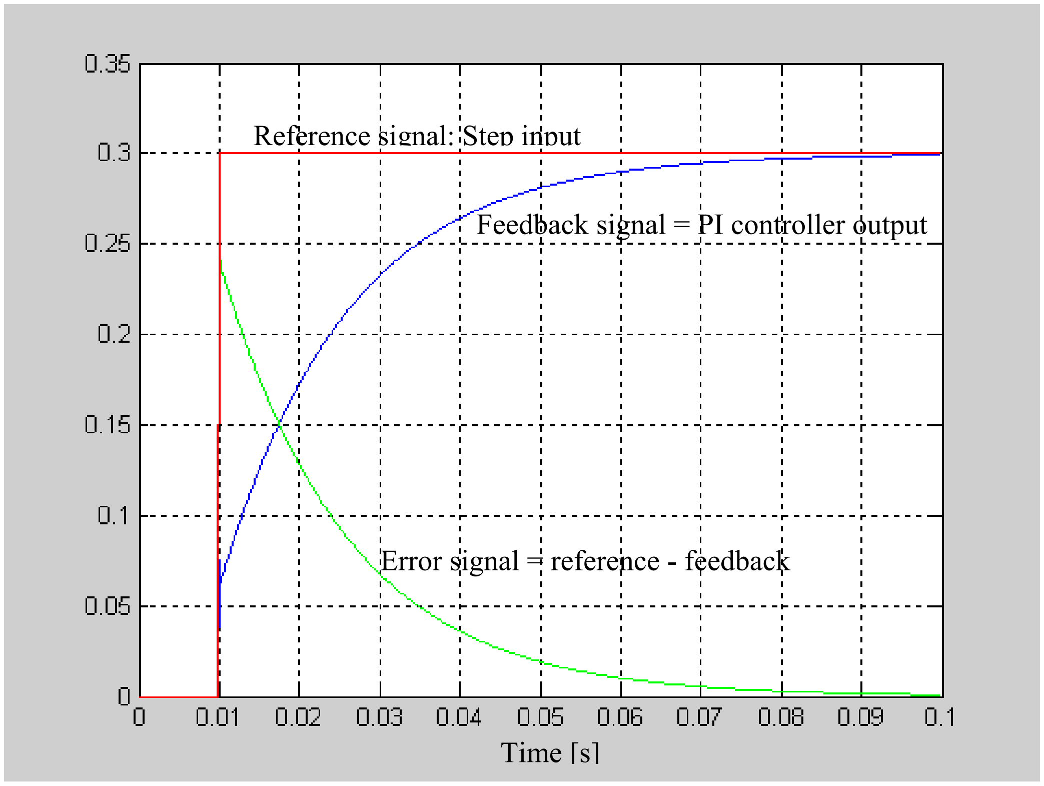

Figure 4 Simulation result of PI controller

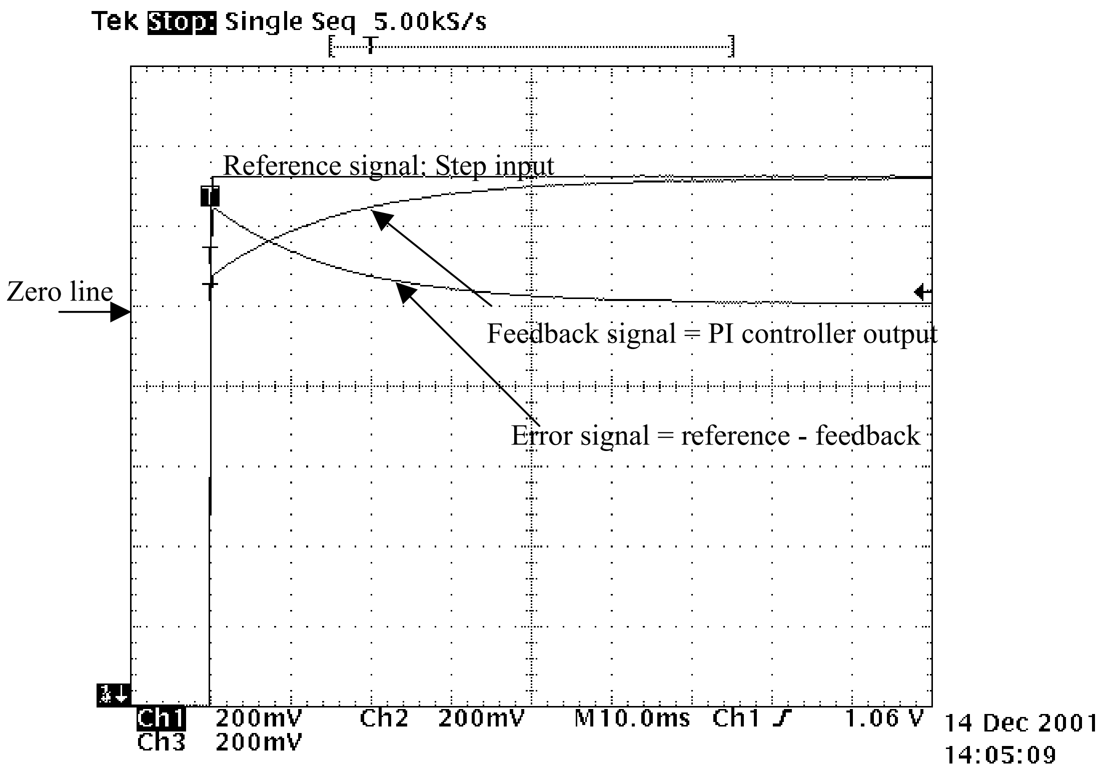

Figure 5 DSP implementation result of PI controller