C2000™ Digital Controller Library

User's Guide

Literature Number: SPRUI31
July 2015

Contents

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

List of Figures

Copyright © 2015, Texas Instruments Incorporated

List of Tables

Copyright © 2015, Texas Instruments Incorporated

Read This First

About This Manual

This user's guide contains information relating to the C2000 Digital Controller Library (DCL). Here you will find technical descriptions of the library functions and how to use them. The User's Guide does not contain information on specific control applications or on the underlying theory.

The software described in this document relates only to the C2000 microcontroller (MCU) from Texas Instruments. While the structure and usage of all the controllers described here are widespread in industry, the functions in the DCL are written exclusively for the C2000, and will not run on any other MCU platform.

How to Use This Manual

The information presented in this document is divided into the following chapters:

- **Chapter 1:** introduces the library and provides information on its use.
- **Chapter 2:** describes the structure of the library and introduces naming conventions.
- **Chapter 3 and Chapter 4**: describe the PID and PI controller implementations in the DCL.
- **Chapter 5, Chapter 6 and Chapter 7**: describe the three ARMA configurations in the library: a third order controller in "Direct Form 1", a second order "Direct Form 2", and a third order "Direct Form 2".

  The reader is advised to begin by deciding on the type of controller he or she wishes to use. Performance and other important information in the corresponding chapter should then be read carefully to ensure the library algorithm is suitable. Once suitability has been ascertained, Chapter 1 and Chapter 2 provide a useful overview of how to add the library code to an existing user C program, and should be read carefully by all users. The chapter describing the selected controller type should then be read in its entirety. Finally, if data array management is part of the system requirements, Appendix A provides information on a data logger utility that is provided with the library and may be of interest.

Related Documentation From Texas Instruments

For a complete list of related documentation and development tools for the C2000 device, visit the C2000 page on the Texas Instruments website at www.ti.com/c2000.

If You Need Assistance

Technical support is available online at the TI "E2E Community: e2e.ti.com/support/microcontrollers/c2000.

C2000, Code Composer Studio are trademarks of Texas Instruments.
Matlab, Simulink are registered trademarks of The MathWorks, Inc.
All other trademarks are the property of their respective owners.
C2000, Code Composer Studio are trademarks of Texas Instruments.
Matlab, Simulink are registered trademarks of The MathWorks, Inc.
All other trademarks are the property of their respective owners.

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

Introduction

This chapter contains a brief introduction to the Texas Instruments C2000 Digital Controller Library.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

1.1 Supported Devices

The Digital Controller Library (DCL) only supports C2000 devices that contain a 32-bit floating-point unit (FPU). Among these devices are:

- TMS320F2837x
- TMS320F2807x
- TMS320F2833x
- TMS320C2834x
- TMS320F2806x
- TMS320F28M35x
- TMS320F28M36x

1.2 Overview of the Library

The C2000 Digital Controller Library provides a suite of robust and easy-to-use software functions for developers of control applications using the C2000 MCU platform from Texas Instruments. The functions are intended for use in any digital control system in which a C2000 device is used.

The DCL functions are supplied as C callable assembly language functions. The library includes functions that support the Control Law Accelerator (CLA). The CLA is an independent 32-bit floating-point core designed to compute control functions with minimum latency and execution time.

The following functions are included in version 1.0 the library:

- Full-featured proportional-integral-derivative (PID) controller
- Proportional-integral (PI) controller
- Third order ARMA controller in Direct Form 1
- Second and third order ARMA controllers in Direct Form 2
- Precomputed controllers for all Direct Form types
- CLA compatible functions for all above controllers
- Utility to create and manage data arrays

1.3 Sample Rate Selection

All the controllers described here run independently of sample rate; the sample rate is fixed by whatever hardware event the user chooses to call the controller function. However, the performance of all discrete time control systems is critically dependent on the rate at which they run; consequently the selection of a suitable sampling rate is among the most important decisions the designer takes. All digital design calculations depend on this parameter; a poor choice can have a profound effect on the performance of the control system.

General guidelines on sample rate selection can be found in many publications [3], [4]. In general, sampling at too slow a rate reduces CPU loading at the cost of degraded control performance. Sampling at too fast a rate typically improves performance yet places increased computational burden on the processor, which in extreme cases can result in missed sample updates. In both cases, the consequences can be potentially damaging.

In control systems, an indication of the minimum sample rate required for good performance can be obtained from the “rise time” of the plant. In general, a sample period of at most one quarter of the rise time yields acceptable control performance. Rise time is typically defined as the time taken for the output of the plant to change from 10% to 90% of its steady state value. In systems that exhibit initial undershoot the time is measured from the application of the test input. For more information, see [4].

Copyright © 2015, Texas Instruments Incorporated

1.4 Numerical Representation

All input, output and internal variables are in 32-bit (single precision) floating-point format compliant with IEEE.754. In the C28x implementation, the 32-bit floating-point numeric format comprises one sign bit (s), a 23-bit mantissa (f), and an 8-bit exponent (e).

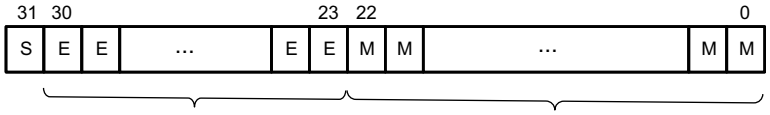

Figure 1-1. IEEE.754 Single Precision Floating-Point Format

The numeric value is determined from the 32-bit structure according to the following five cases:

• Case 1: if e = 255 and f != 0,	then v = [(-1)^s]*infinity
• Case 2: if e = 255 and f = 0,	
• Case 3: if 0 < e < 255,	then v=[(-1)^s]*[2^(e-127)]*(1.f)
• Case 4: if e = 0 and f != 0,	then v=[(-1)^s]*[2^(-126)]*(0.f)
• Case 5: if e = 0 and f = 0,	then v=[(-1)^s]*0

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

Using the Digital Controller Library

This chapter contains general information about the structure and usage of the C2000 Digital Controller Library.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

2.1 What the Library Contains

The Digital Controller Library provides a set of software functions optimized for real-time digital control applications. The library is supplied as part of the “controlSUITE” download package from Texas Instruments.

Library functions are supplied in C callable assembly form. This form allows the user to freely inspect and modify the code, yet maintains high cycle efficiency and small memory footprint. The format also means that performance is independent of compiler and optimizer settings.

All functions have filenames beginning with “DCL_”. Functions intended for use with the CLA are identified by names suffixed with the letter “c”. This distinction is necessary because the CPU and CLA instruction sets are different and it is conceivable that users will run similar functions on the two cores in the same CCS project, resulting in two object files with similar names.

Certain controllers are supplied in “precomputed” form to reduce control loop latency. In this form, the controller consists of two functions: an “immediate” function that supplies the controller result in the current sample interval and a “partial” function that pre-computes a partial result for use in the next sample interval. These two functions are identified by an “i” and “p” suffix to the function name, respectively.

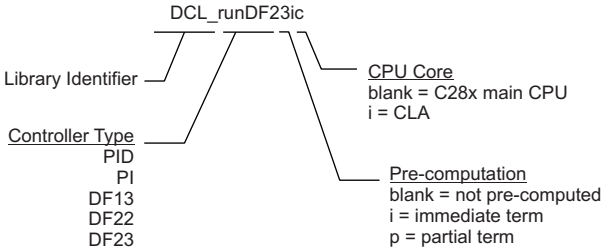

Figure 2-1. DCL Function Naming Convention

The following files are included in the library.

2.1.1 Header Files

2.1.1.1 DCL.h

This header file contains type definitions and function prototypes. It must be included in the project and visible to each C file that references library variables or calls library functions.

2.1.1.2 DCL_fdlog.h

This header file contains a floating-point data-logger utility. These functions are useful when working with buffers of data as well as when instrumenting C code for testing or debugging purposes. No source code is associated with this utility. It can be added to any C2000 C project regardless of whether the DCL is used.

Copyright © 2015, Texas Instruments Incorporated

2.1.2 Source Files

The following C28x source files are included with the library package:

- DCL_PID.asm
- DCL_PI.asm
- DCL_DF13.asm
- DCL_DF22.asm
- DCL_DF23.asm

The following CLA source files are included with the library package:

- DCL_PID.asm
- DCL_PI.asm
- DCL_DF13.asm
- DCL_DF22.asm
- DCL_DF23.asm

Table 2-1 lists of the controller functions contained in the DCL.

Table 2-1. List of Controller Functions

Function Name	Description
DCL_runPID	PID controller for C28x CPU
DCL_runPIDc	PID controller for CLA
DCL_runPI	PI controller for C28x CPU
DCL_runPIc	PI controller for CLA
DCL_runDF13	Third order DF1 for C28x CPU
DCL_runDF13i	Third order DF1 for C28x CPU with pre-computation (immediate)
DCL_runDF13p	Third order DF1 for C28x CPU with pre-computation (partial)
DCL_runDF13c	Third order DF1 for CLA
DCL_runDF13ic	Third order DF1 for CLA with pre-computation (immediate)
DCL_runDF13pc	Third order DF1 for CLA with pre-computation (partial)
DCL_runDF22	Second order DF2 for C28x CPU
DCL_runDF22i	Second order DF2 for C28x CPU with pre-computation (immediate)
DCL_runDF22p	Second order DF2 for C28x CPU with pre-computation (partial)
DCL_runDF22c	Second order DF2 for CLA
DCL_runDF22ic	Second order DF2 for CLA with pre-computation (immediate)
DCL_runDF22pc	Second order DF2 for CLA with pre-computation (partial)
DCL_runDF23	Third order DF2 for C28x CPU
DCL_runDF23i	Third order DF2 for C28x CPU with pre-computation (immediate)
DCL_runDF23p	Third order DF2 for C28x CPU with pre-computation (partial)
DCL_runDF23c	Third order DF2 for CLA
DCL_runDF23ic	Third order DF2 for CLA with pre-computation (immediate)
DCL_runDF23pc	Third order DF2 for CLA with pre-computation (partial)

Copyright © 2015, Texas Instruments Incorporated

The location of each controller function is shown in Table 2-2.

Table 2-2. Function Allocation by File

Controller	Filename	Type	Function
PID	DCL_PID	asm	DCL_runPID
	DCL_PID_CLA		DCL_runPIDc
PI	DCL_PI	asm	DCL_runPI
	DCL_PI_CLA		DCL_runPIc
DF1-3	DCL_DF13	asm	DCL_runDF13
			DCL_runDF13i
			DCL_runDF13p
	DCL_DF13_CLA	asm	DCL_runDF13c
			DCL_runDF13ic
			DCL_runDF13pc
DF2-2	DCL_DF22	asm	DCL_runDF22
			DCL_runDF22i
			DCL_runDF22p
	DCL_DF22_CLA	asm	DCL_runDF22c
			DCL_runDF22ic
			DCL_runDF22pc
DF2-3	DCL_DF23	asm	DCL_runDF23
			DCL_runDF23i
			DCL_runDF23p
	DCL_DF23_CLA	asm	DCL_runDF23c
			DCL_runDF23ic
			DCL_runDF23pc

Note that full and precomputed controllers are located in the same source file for each controller type.

2.2 How to Add the DCL to Your Code

The Digital Controller Library is intended to be used with a project written in the C programming language. All controller functions are written in C2000 assembly language in a form that should be called from a C program.

Typically, the controller functions for the C28x would be inserted into an Interrupt Service Routine (ISR) triggered by a hardware event to ensure they are executed at a fixed rate. Functions for use on the CLA would be called from a CLA task, which would typically be triggered by a hardware event.

1. Specify the include file.

   Before you can begin using the library, you must add the library header file to your project.

   ```
   #include    "DCL.h"
   ```

   This must be done in such a way that the `DCL` header file is visible to all program source files that reference controller variables or functions. You must also ensure that the library files are copied to a directory location where the compiler can find them. The include file options in Code Composer Studio™ (CCS) allow users to specify header file paths.

2. Add the source files to the project.

   The source files for the controllers you wish to use must be added to the CCS project. You can copy the files into your project directory, or specify the library pathname in the CCS compiler options.

Copyright © 2015, Texas Instruments Incorporated

3. Allocate the controller functions in the linker command file.

   DCL functions that execute on the C28x core can be allocated to a specific memory block in the linker command file. This allows the user to maximize execution speed by placing controller functions in zero wait state RAM. This step is unnecessary with CLA controllers since all CLA functions run from internal zero wait state RAM.

   C28x library functions are placed in the user-defined code section “dclfuncs”. An example showing how this section might be mapped into the internal L4 RAM memory block is shown below:

   ```
   dclfuncs        : > RAML4,        PAGE = 0
   ```

   More details of section allocation can be found in the *TMS320C28x Assembly Language Tools User's Guide* (SPRU513).

4. Create an instance of the controller.

   You must declare an instance of the controller you wish to use. Examples can be found for each controller type in the corresponding section that describes it. For example, to create an instance of a PID controller named “pid1”, you would add the following line to the variable list in your C source:

   ```
   PID pid1 = DCL_PID_DEFAULTS;
   ```

   This creates a variable of type “PID”, the elements of which are initialized to those default values specified in the `DCL.h` header file. Like any C variable, the structure must be visible to any source files that reference it.

   Note that CLA variables must be initialized at run-time by user code. They cannot be initialized at the variable declaration. Typically this would be done using a CLA task.

5. Declare variables

   In addition to a pointer to the controller structure, each controller function requires certain input variables to be passed as arguments to the function. You should declare instances of these variables in your code and ensure they are compiler visible to all files that call the controller functions. For example:

   ```
   float uk;         // control
   float rk = 0.0f;  // reference
   float yk = 0.0f;  // feedback
   float lk = 1.0f;  // saturation
   ```

   Again, CLA variables cannot be initialized at the variable declaration.

6. Initialize the controller.

   The elements of the (CPU) controller structure were initialized to default settings in step 3. The user program must configure any controller elements with specific values before the function is called. For example:

   ```
   pid.Kp = 9.4f       // set proportional gain to 9.4
   pid.Umax = 10.0f;   // upper output clamp limit = 10
   ```

   If a CLA-based controller is being used, its parameters must always be initialized using a separate task. For more information on the CLA C compiler, see the *CLA Compiler* chapter of the *TMS320C28x Optimizing C/C++ Compiler User's Guide* (SPRU514).

   ARMA control structures incorporate one or two delay lines, which hold previous controller output data. These must be initialized to zero before calling the controller functions. It is possible that uninitialized delay line data, especially in the recursive path, might cause the controller to saturate or deliver unreliable results. The initialization of the delay line elements is the responsibility of the user.

7. Call the controller function.

   Typically the controller functions would be inserted into an Interrupt Service Routine (ISR) that is triggered by a hardware timer. This ensures that the control law is executed at a deterministic and fixed time interval. Each control function returns a single floating-point variable, which represents the controller output. An example of a controller function call is shown below:

   ```
   uk = DCL_runPID (&pid1, rk, yk  lk);
   ```


Copyright © 2015, Texas Instruments Incorporated

2.3 Benchmarks

Table 2-3 lists the execution cycle count for each function in the Digital Controller Library. The execution cycles shown include the C function calling overhead. The code size is given in (16-bit) words for each file.

Table 2-3. Execution and Code Size Benchmarks

Controller	Filename	Type	Function	CPU	Cycles	Size
PID	DCL_PID	asm	DCL_runPID	C28	70	103
	DCL_PID_CLA		DCL_runPIDc	CLA	52	68
PI	DCL_PI	asm	DCL_runPI	C28	46	58
	DCL_PI_CLA		DCL_runPIc	CLA	33	40
DF1-3	DCL_DF13	asm	DCL_runDF13	C28	59	175
			DCL_runDF13i		22	
			DCL_runDF13p		65	
	DCL_DF13_CLA	asm	DCL_runDF13c	CLA	59	192
			DCL_runDF13ic		20	
			DCL_runDF13pc		57	
DF2-2	DCL_DF22	asm	DCL_runDF22	C28	42	121
			DCL_runDF22i		21	
			DCL_runDF22p		36	
	DCL_DF22_CLA	asm	DCL_runDF22c	CLA	32	90
			DCL_runDF22ic		17	
			DCL_runDF22pc		33	
DF2-3	DCL_DF23	asm	DCL_runDF23	C28	56	157
			DCL_runDF23i		20	
			DCL_runDF23p		49	
	DCL_DF23_CLA	asm	DCL_runDF23c	CLA	44	140
			DCL_runDF23ic		20	
			DCL_runDF23ip		44	

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

PID Controller

This chapter describes the Proportional Integral Derivative (PID) controller included in the Digital Controller Library.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

3.1 Background

The controller described here is a feature-rich PID controller that includes several features not commonly found in basic PID designs. This complexity is reflected the benchmark figures. Applications that do not require derivative action, or are more sensitive to cycle efficiency, may be better served by the simpler PI controller structure described in Chapter 4.

The first use of PID control can be traced back to ship and aircraft applications in the early part of the 20th century and is today probably the most widespread type of controller in industry. The PID controller has the advantage that the physical effect of each of its three gain terms is clearly visualized in the features of the transient response: a fact that greatly simplifies manual tuning of the control loop. Good technical descriptions of the PID can be found in many books and technical papers, a few of which are listed in Chapter 8.

3.2 Technical Description

PID control is widely used in systems that employ output feedback control. In such systems, the controlled output is measured and fed back to a summing point where it is subtracted from the reference input. The difference between the reference and feedback corresponds to the control loop error (or servo error) and forms the input to the PID controller.

The PID controller output is the parallel sum of three paths that act, respectively, on the error, error integral, and error derivative. The relative weight of each path is adjusted by the user to optimize transient response.

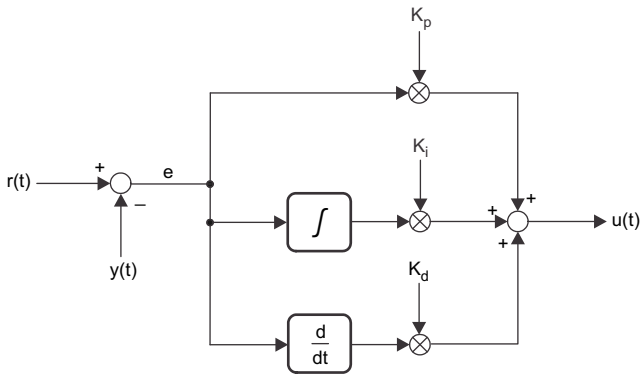

Figure 3-1. Basic PID Controller Structure

Figure 3-1 shows the structure of a basic continuous time PID controller. The output of this so-called "ideal" PID controller is captured in Equation 1.

$$u(t) = K_p\, e(t) + K_i \int_{-\infty}^{t} e(\tau)\, dt + K_d \frac{de(t)}{dt} \tag{1}$$

Conceptually, the controller comprises three separate paths connected in parallel. The upper path contains an adjustable gain term (Kp). Its effect is to fix the open loop gain of the control system. Since loop gain is proportional to this term, Kp is known as proportional gain.

A second path contains an integrator that accumulates error history. A separate gain term acts on this path. The output of the integral path changes continuously as long as a non-zero error (e) is present at the controller input. A small but persistent servo error has the effect of driving the output of the integrator such that the loop error will eventually disappear. The principal effect of the integral path is therefore to eliminate steady state error. The effect of the integral gain term is to change the rate at which this happens. Integral action is especially important in applications that are required to maintain accurate regulation over long periods of time.

The third path contains a differentiator. The output of this path is large whenever the rate of change of the error is large. The principal effect of the derivative action is to damp oscillation and reduce transients.

The operation of the PID controller is best visualized in terms of the transient error following a change of load or set-point.

Copyright © 2015, Texas Instruments Incorporated

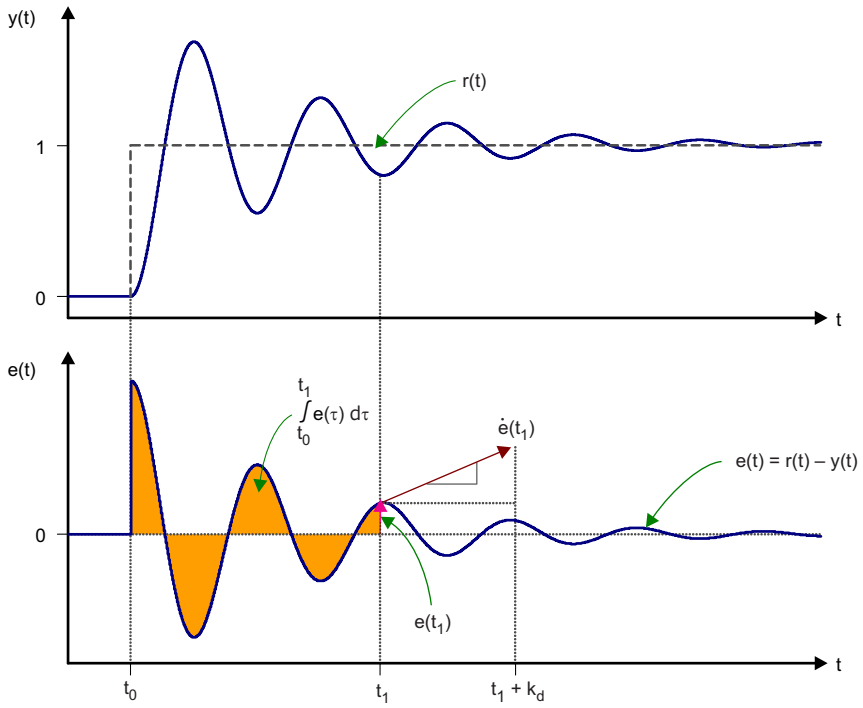

Figure 3-2. PID Control Action

Figure 3-2 shows the action of the PID controller in terms of the control loop error at time t1. The proportional term contributes a control effort that is proportional to the instantaneous loop error. The output of the integral path is the accumulated error history: the shaded area in the lower plot. The contribution of the derivative path is proportional to the rate of change of the loop error. Derivative gain fixes the time interval over which a tangential line to the error curve is projected into the future.

Tuning the PID controller is a matter of finding the optimum combination of these three effects. This in turn means finding the best balance of the three gain terms. For more information on PID control, see [4].

The PID module contained in the DCL implements PID control action with the following enhancements:

- Programmable output saturation
- Independent reference weighting on proportional path
- Independent reference weighting on derivative path
- Anti-windup integrator reset
- Programmable low-pass derivative filter
- Internal servo error calculation
- Adjustable output saturation

Copyright © 2015, Texas Instruments Incorporated

The controller described here is a discrete time PID controller intended for use in real-time digital control applications. Typical applications include switched power control systems such as motor drives and power supplies. A block diagram of the internal controller structure is shown in Figure 3-3.

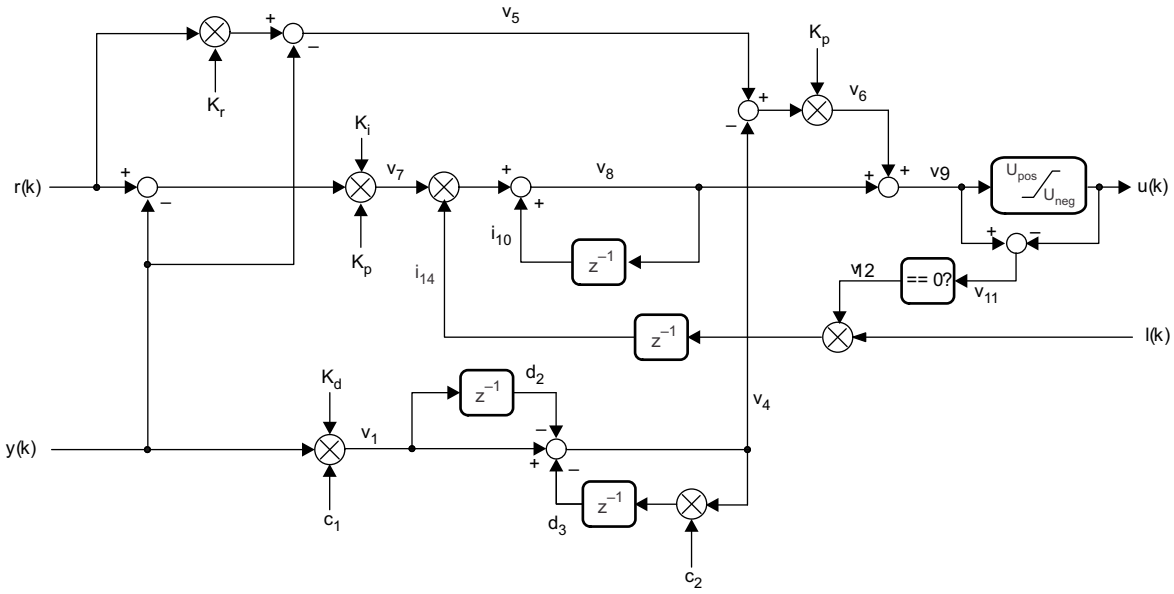

Figure 3-3. Digital PID Controller

Variables denoted ‘v’ are local to the function and are not preserved between function calls. Variables denoted ‘i’ or ‘d’ appear as elements in the PID controller structure and are therefore preserved between function calls.

The user software interacts with the PID function via four floating-point variables and one data structure. The variables pass data in and out of the device, as follows:

- Reference input: r(k)
- Feedback input: y(k)
- Saturation input: l(k)
- Control output: u(k)

The PID data structure holds variables that are internal to the controller, including gain settings and stored integration variables. The elements of the controller data structure are:

- Proportional gain (K_p)
- Integral gain (K_i)
- Derivative gain (K_d)
- Set point weight (K_r)
- Integrator storage (i_{10})
- Derivative filter storage (d_2 and d_3)
- Derivartive filter coefficients (c_1 and c_2)
- Output saturation limits (u_{max} and u_{min})
- Saturation storage (i_{14})

Copyright © 2015, Texas Instruments Incorporated

3.2.1 Proportional Path

The servo error is the difference between the reference input and the feedback input. Proportional gain is usually applied directly to servo error, however, a feature of this controller is that sensitivity of the proportional path to the reference input can be weighted differently from that of the feedback input. This is achieved through the Kr variable, and provides an additional degree of freedom when tuning the controller. The proportional control law is shown in Equation 2.

$$v_5(k) = K_r r(k) - y(k) \quad (2)$$

In most situations the weighting gain Kr will be unity. For information on when and how to adjust this parameter, see step 6 of the tuning guide in Section 3.3.

3.2.2 Integral Path

The integral path consists of a discrete integrator that is pre-multiplied by a scalar gain (Ki) and a term derived from the output saturation module. The saturation input term (i14) is either zero or one, and provides a means to stall the integrator path when loop saturation occurs. This feature prevents the integrator from “winding up” and improves recovery time following saturation. The integrator law used here is based on backwards approximation. Note that the proportional gain is applied before the integrator. This enables glitch-less change of proportional gain when the integrator output is non-zero.

$$v_8(k) = v_8(k-1) + i_{14}(k) K_p K_i [r(k) - y(k)] \quad (3)$$

The controller includes a feature to limit the integrator output if an external component in the control loop reaches saturation. Provision for this is made using the l(k) input. This floating-point input must be provided by the user and typically originates in a saturation limit block elsewhere in the application code.

Under normal operating conditions the controller expects l(k) to have a value of one. If l(k) is set to zero, integrating action ceases and the output of the integrator (v8) is maintained at its current value. This feature is popularly known as “anti-windup reset”. In this design, anti-windup reset is implemented by setting the integrator input to zero.

$$i_{14}(k) = \begin{cases} 1: v_{11}(k-1) == 0 \quad AND \quad l(k-1) == 1 \\ 0: v_{11}(k-1) \neq 0 \quad OR \quad l(k-1) == 0 \end{cases} \quad (4)$$

The variable i14 must be preserved between function calls and is, therefore, an element in the PID structure.

3.2.3 Derivative Path

In this design, the input to the derivative path is the feedback y(k) rather than the loop error. This avoids sharp discontinuous inputs when the reference set-point r(k) is changed suddenly from one level to another. Discontinuities in the input to the derivative path produce large transients in the derivative path that can disturb the loop; a phenomenon known as “derivative kick”. The use of y(k) alone for the derivative path means the rate of change of the input is limited by the speed of the plant while the damping properties of derivative action are unaffected.

$$v_1(k) = K_d c_1 y(k) \quad (5)$$

The gain of a pure differentiator increases with frequency, so the use of differential gain in a controller raises the possibility of high frequency noise amplification. In some situations this can lead to poor regulation and for this reason most practical PID controller designs include a low-pass filter in series with the derivative path.

The filter implemented here is a simple first order lag filter converted into discrete form using the Tustin transform. The difference equation of the filtered differentiator is shown in Equation 6.

$$v_4(k) = v_1(k) - d_2(k) - d_3(k) \quad (6)$$

The temporary storage elements d2 and d3 interval must be preserved from the $(k - 1)^{th}$ interval, so the following must be computed after the differentiator update.

$$d_2(k) = v_1(k - 1) \quad (7)$$

$$d_3(k) = c_2 v_4(k - 1) \quad (8)$$

Copyright © 2015, Texas Instruments Incorporated

The derivative filter coefficients are:

$$c_1 = \frac{2}{T + 2\tau} \tag{9}$$

$$c_2 = \frac{T - 2\tau}{T + 2\tau} \tag{10}$$

Both the sample period (T) and filter time constant are required for definition of this filter. The time constant (τ) is the reciprocal of the desired filter bandwidth in radians per second.

3.2.4 Output Path

The output is the parallel sum of the three controller paths. The Kp gain is applied to the sum of the proportional and derivative terms, and the resultant summed with the integral term. It has been found that this arrangement reduces coupling between the three principal gain terms and facilitates tuning.

$$v_9(k) = v_8(k) + K_p[v_4(k) = v_5(k)] \tag{11}$$

A programmable saturation module allows the user to clamp the controller output to pre-defined maximum and minimum levels. If the output reaches either threshold the integrator is disabled.

$$v_{12}(k) = \begin{cases} 1 : v_9(k) == u(k) \\ 0 : v_9(k) \neq 0 \end{cases} \tag{12}$$

3.3 Tuning

This subsection suggests a manual tuning procedure for the PID controller. Several procedures for fixing the controller gains based on plant response measurements can be found in the literature but these are rarely used outside process control. In power electronics applications, tuning is most often performed by iteratively adjusting one parameter at a time to optimize one or two performance objectives before moving to a different parameter. This continues until the user judges that a satisfactory response has been obtained.

A degree of optimality is possible in by assigning a performance index or cost function. Such functions are typically based on integrating the transient error over a fixed time interval. Information on performance indices and their application to PID tuning can be found in [4]. A function to compute the ITAE index is included in the data logger utility described in Appendix A.

The following list of steps constitutes a suggested procedure for manually tuning the PID controller described here. It is assumed that the user understands the general features of the transient response and that realistic performance objectives have been set.

CAUTION

Adjustments to the controller parameters may cause sudden and unpredictable changes to control, which may damage the system. Depending on the nature of the system, sudden control changes may cause injury or death. Always proceed with extreme caution when tuning any control loop.

1. Initialize controller gains.

   Set the proportional gain (Kp) to a known safe initial value. When selecting an initial value for Kp, be sure to set the gain significantly lower than the expected optimum value so that the gain can be safely increased without risk of the control loop becoming unstable. Ensure integral and derivative gains are set to zero, and that the reference weighting coefficient (Kr) is set to one.

2. Apply transient disturbance.

   Apply a disturbance to the control loop in such a way as to induce an observable transient at the system output. In many cases, the disturbance will be a sudden change in reference set-point, or a change in output load. Observe the transient part of the output response. The data logger utility may be useful for this task.

Copyright © 2015, Texas Instruments Incorporated

3. Adjust proportional gain.

   Adjust proportional gain (Kp) as required and repeat the response test to achieve optimum response. A summary of transient tuning characteristics can be found in [4].

   Changes to Kp should be made in small increments and the transient response test repeated each time. Keep in mind that many systems will become unstable if the proportional loop gain exceeds a certain value; a consequence of one or more root loci entering the RHP [4]. If a value of Kp can be found that meets all the performance objectives the tuning procedure can be terminated at this point.

4. Adjust integral gain.

   Depending on the nature of the control loop, steady state error can sometimes be eliminated with the introduction of integral control action. The effect of integral control depends on the open loop transfer function and the type of test stimulus applied. In industrial applications the test stimulus is often a step change of reference input, and zero steady state error is achieved when at least one integrator is present inside the loop. For more information, see [1], [2], [3].

   If necessary, gradually increase integral gain (Ki) to reduce the steady state output error. Increasing the integral gain increases the rate at which the response converges on steady state, but may introduce or amplify overshoot and oscillation. Depending on the plant dynamics, the response may be very sensitive to the integral term and may become unstable, so be sure to start with a very small gain. In general, the faster the response of the plant, the greater will be the sensitivity of the control loop to integral gain.

   The primary reason for adding integral gain is to force zero steady state error; however, integral gain typically results in increased overshoot and oscillation, so it may be necessary to simultaneously decrease the Kp term to find the best balance.

5. Configure derivative filter.

   Oscillatory effects in the transient response can sometimes be reduced by applying derivative gain. To do this, first select a derivative filter cut-off frequency and determine the reciprocal time constant. If filtering is not required, use infinite cut-off frequency (zero time constant). Apply Equation 9 and Equation 10 to compute the derivative filter coefficients c1 and c2. Load these values into the PID structure.

6. Adjust derivative gain.

   Apply a small amount of derivative gain (Kd) and repeat the transient test. Depending on the nature of the plant, the control may be strongly or weakly dependent on this parameter. If small amount of derivative action makes no significant difference, it may be necessary to use progressively larger increments until a difference is seen. In general, the faster the plant, the less sensitive is the closed loop response to derivative action.

   It may be necessary to re-adjust Kp and Ki gains at this point. Typically, tuning involves repeated adjustments to the gain terms to find the best achievable response.

7. Adjust set-point weighting.

   In some cases it is advantageous to “weight” the control error input to the proportional path differently from that of the integral path. This is achieved by changing the Kr term in the PID structure. Examples include systems with time delay or right half plane zeros. Typically the optimum value of Kr gain is a little less than one.

   The application of set-point weighting is usually a matter of trial-and-error and no specific guidelines are given here. Note that the majority of control systems will not benefit materially from adjustment of Kr.

Copyright © 2015, Texas Instruments Incorporated

3.4 Adding the PID Controller to C Code

This subsection describes how to add the PID controller module to a user C code project.

3.4.1 Controller Structure Definition

The C type definition of the PID controller data structure is shown below:

```
typedef volatile struct {
    float Kp;       // proportional gain
    float Ki;       // integral gain
    float Kd;       // derivative gain
    float Kr;       // set point weight
    float c1;       // D filter coefficient 1
    float c2;       // D filter coefficient 2
    float d2;       // D filter storage 1
    float d3;       // D filter storage 2
    float i10;      // I storage
    float i14;      // sat storage
    float Umax;     // upper saturation limit
    float Umin;     // lower saturation limit
} PID;
```

A list of default floating-point settings is included for initialization purposes. These configure the proportional path to have unity gain, and disable the other paths.

```
#define  PID_DEFAULTS {   1.0f, \
                          0.0f, \
                          0.0f, \
                          1.0f, \
                          0.0f, \
                          0.0f, \
                          0.0f, \
                          0.0f, \
                          0.0f, \
                          0.0f, \
                          1.0f, \
                          -1.0f \
                      }
```

An example of the C declaration of an initialized PID structure is:

```
PID pid1 = PID_DEFAULTS;
```

3.4.2 Implementing a PID Controller

The steps required to add the PID controller module to an existing user project are as follows.

1. Include the `DCL.h` header file to the user project. This file contains all the definitions for the Digital Control Library and needs to be visible to all project source files that reference them.

   ```
   #include  “DCL.h”
   ```

2. If working with the main CPU, add the `DCL_PID.asm` source file to the project. If working with the CLA, add the `DCL_PID_CLA.asm` source file to the project.

   Create an instance of a PID controller and initialize it (CPU only). This can be accomplished in the following declaration.

   ```
   PID pid1 = PID_DEFAULTS;
   ```

3. Call the PID controller function. Typically, this function would be placed in an interrupt service routine to ensure it is executed at a fixed and deterministic rate. In the following CPU example, uk, rk, yk, and lk are floating-point variables. The argument &pid1 is a pointer to the PID structure described in Section 3.4.1 and in step 2 of this subsection.

   ```
   pid1 is a pointer to the PID structure described above and in the previous step.
   ```

   If the CLA controller is being used, the controller call must be made from a CLA task. Data variables would typically be passed between the main CPU and the CLA using the message RAMs. An equivalent CLA function call would be:

   ```
   uk = DCL_runPIDc(&pid1, rk, yk, lk);
   ```


Copyright © 2015, Texas Instruments Incorporated

3.4.3 Configuring the PID Controller in PI Mode

To configure the controller in PI mode, ensure that derivative gain Kd and the internal variables d2 and d3 are set to zero in the PID structure. The derivative filter coefficients c1 and c2 are then immaterial. The code in the derivative path will still be executed but will make no contribution to the output and the controller will act as a PI controller. In many applications, the execution overhead of the unused derivative path is insignificant.

3.5 Performance

This subsection provides information on the performance of this PID controller, together with some test results. The response plots that follow are intended to illustrate some of the common artefacts found in physical control systems.

3.5.1 Benchmarks

The code size and execution cycles of the PID controller are shown in Table 3-1.

Table 3-1. PID Controller Benchmarks

	Code Size (words)	Execution (cycles)
CPU	103	70
CLA	68	52

3.5.2 Typical Response Results

This subsection presents a series of test results for a simple digital control loop. The plant model used here is linear third order, with one LHP zero.

$$G(s) = \frac{s+1}{s^3 + 3s^2 + 5s + 1} \tag{13}$$

The test method involved simulating the digital control system using Matlab®/Simulink® to generate a set of test data. Each test data set comprised four arrays of input and output data for a specific test scenario. The input data arrays were then loaded into internal device memory using CCS, and the software controller executed to generate a buffer of control data. The control data was then imported into Matlab where it was compared with the original simulation data. In this way the behavior of the simulated and coded controllers was validated.

Figure 3-4 shows the stimulus and closed loop response after tuning. The test stimulus is a 10% return step.

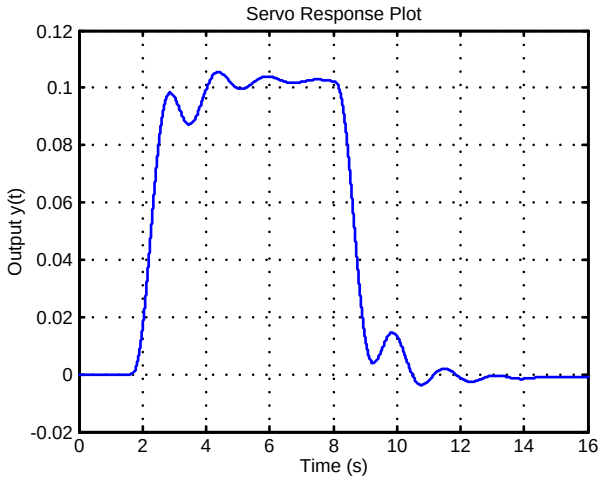

Figure 3-4. Example Return Step Output Response

Copyright © 2015, Texas Instruments Incorporated

The return step plot is fairly typical of a tuned control loop. Tuning objectives such as over-shoot and rise time would be balanced using the PID controller gains as described in Section 3.3.

- Proportional gain = 22.5
- Integral gain = 0.001
- Derivative gain = 0.6
- Reference weighting = 1
- Derivative filter bandwidth = 500 Hz

3.5.3 Saturation Limit Activation

The purpose and design of the saturation limit feature have been described in Section 3.2.2. This test validates correct operation of the anti-windup reset feature of the integral path.

Figure 3-5 shows the output response in the presence of limit activation. Controller settings were unchanged from the previous test.

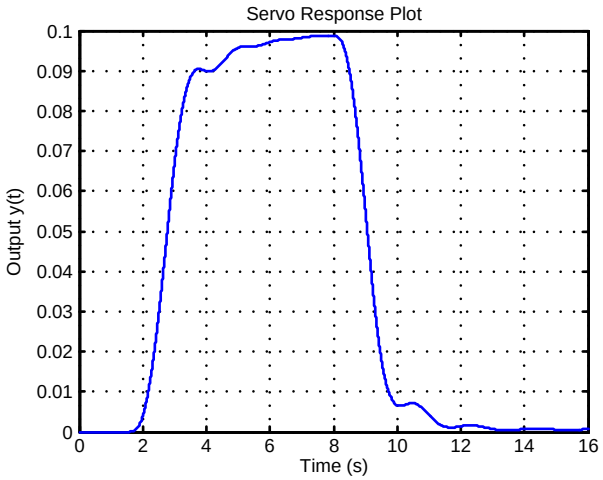

Figure 3-5. Integrator Anti-Windup

Figure 3-6 and Figure 3-7 show the PID control effort u(k) and the limit input to the integrator (i14), respectively. The PID output clamp limits (*Umax*, *Umin*) were set to ±0.5 for this test.

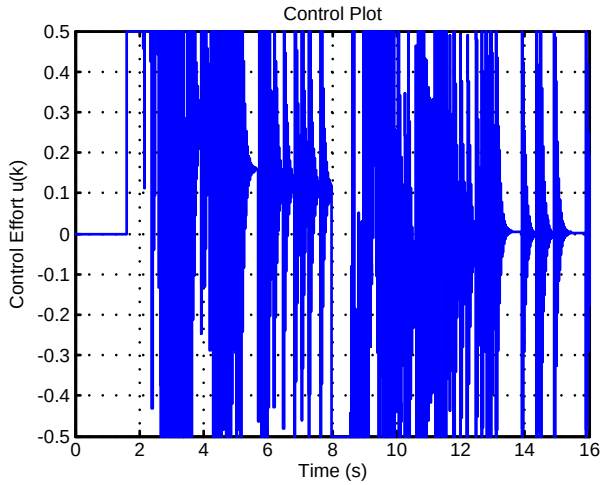

Figure 3-6. Control Effort With Output Saturation

Copyright © 2015, Texas Instruments Incorporated

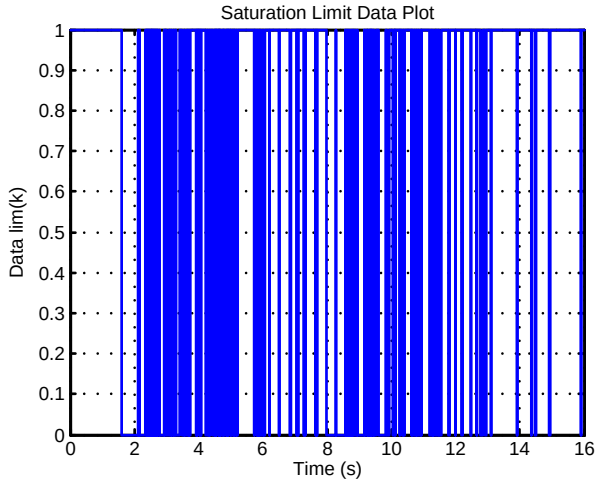

Figure 3-7. Output Saturation Limit Active

Figure 3-8 and Figure 3-9 show the effect of limit activation on the integral path. The first shows the integrator output over the complete test sequence, while the second shows a magnified portion of the integrator curve. Notice that the integrator is output is held constant while the saturation limit lk is zero.

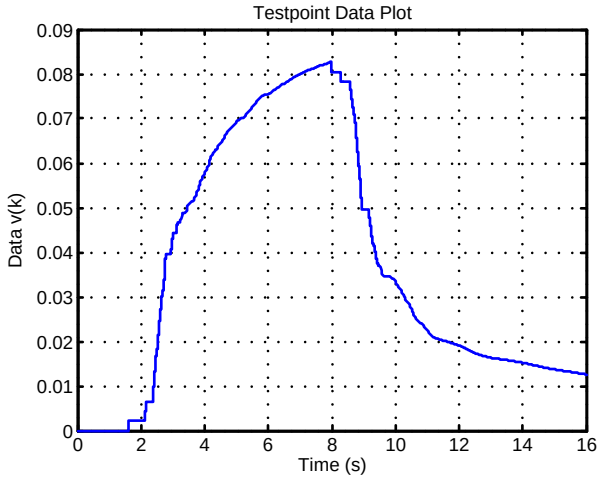

Figure 3-8. Integrator Action

Copyright © 2015, Texas Instruments Incorporated

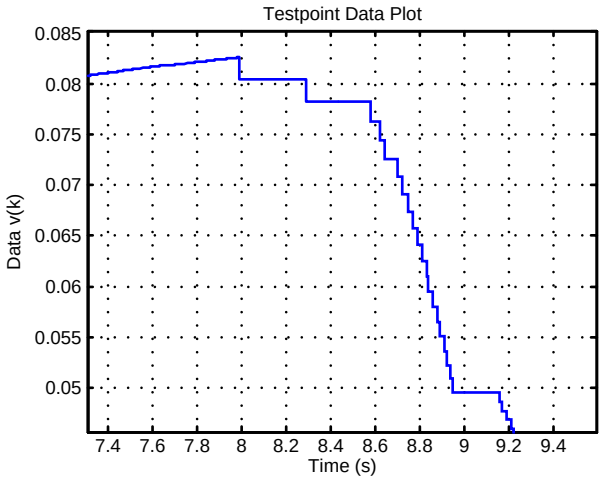

Figure 3-9. Integrator Action (zoomed)

3.5.4 Influence of Resolution Loss

The effects of low resolution in the feedback path are in general detrimental to performance. Resolution loss in the forward path (after the controller) also degrades control and, in some cases, can induce sustained limit cycles. This phenomenon appears when control resolution falls below that of the feedback input and is well documented in relation to digital power supplies. Derivative action can help to reduce limit cycle magnitude but the effect is rarely satisfactory. Limit cycles of this kind should be avoided by addressing the resolution problem directly, usually by increasing the number of effective bits of D/A resolution.

Copyright © 2015, Texas Instruments Incorporated

3.5.5 Influence of Feedback Noise

Random noise may be introduced into the control loop from various sources, through the feedback sensor. Figure 3-10 shows the same system with the same controller gains. In the first case, the derivative filter bandwidth has been reduced to reduce noise amplification. The output response plots on the left are similar; however, the magnitude of control effort is clearly much larger in the second case as the controller has to work harder to regulate the output.

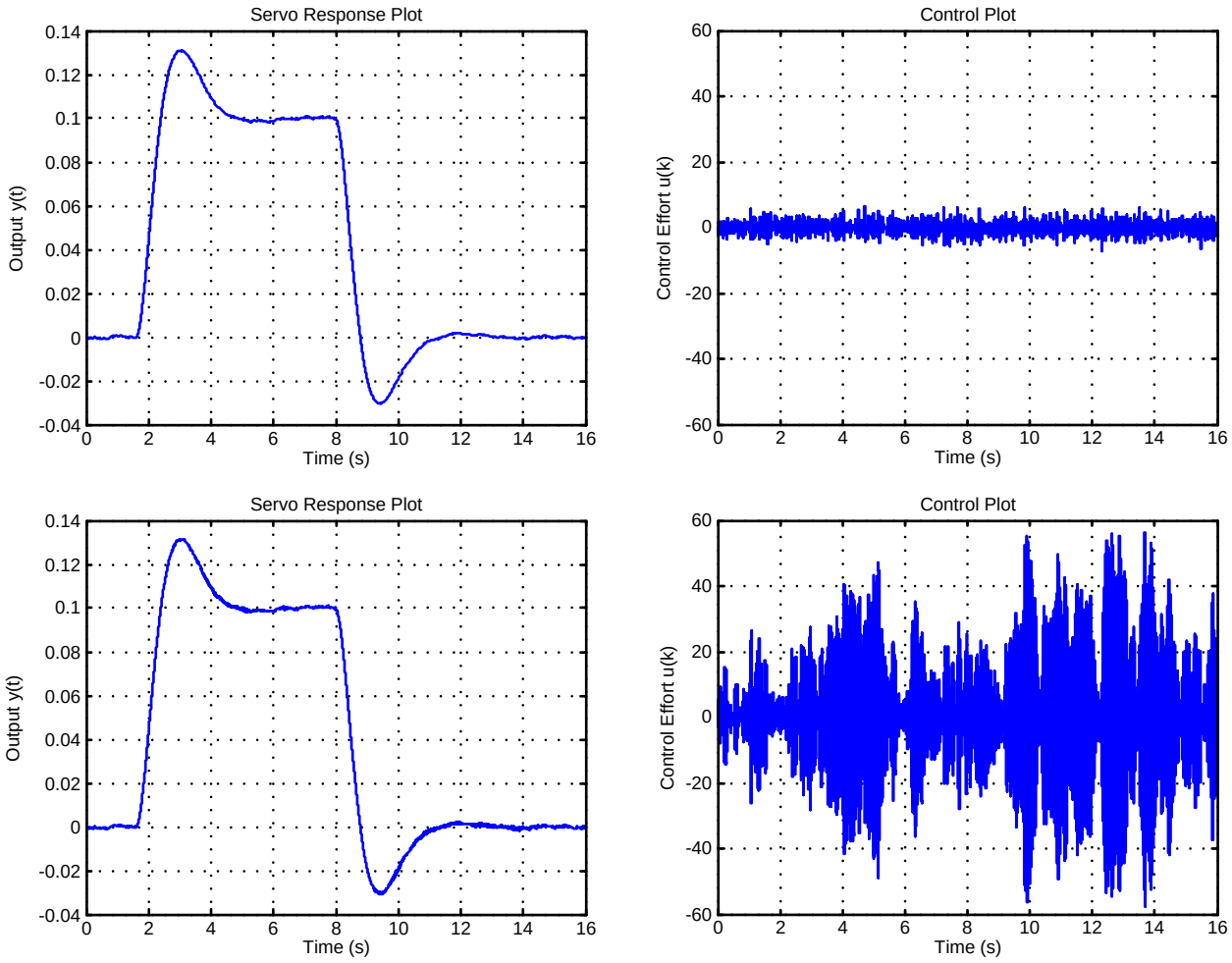

Figure 3-10. Comparison of Feedback Noise Performance

Copyright © 2015, Texas Instruments Incorporated

3.5.6 Effect of Time Delay

Delay in the feedback loop is typically detrimental to control. Delay in the time domain is equivalent to phase lag in the frequency domain and this typically erodes phase margin and reduces control loop robustness to plant errors (component tolerance, temperature drift, and so on). In severe cases time delay alone can result in loss of stability.

Time delay is difficult to compensate effectively, although both derivative action and set-point weighting can help. The plot below compares the nominal system output response with that of the same system with time delay. In both cases, controller parameters were manually adjusted for best response measured using an ITAE index. Oscillation visible in the response of the time delayed system is attributable to loss of phase margin.

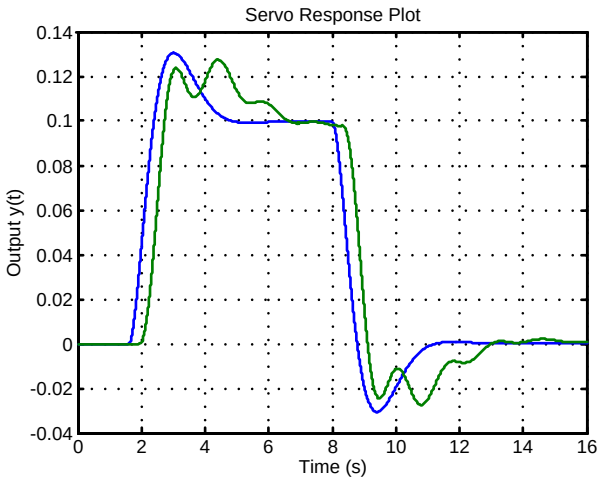

Figure 3-11. Effect of Plant Model Time Delay

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

PI Controller

This chapter describes the Proportional Integral (PI) controller included in the Digital Controller Library.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

4.1 Background

In addition to the PID controller described in Chapter 3, the Digital Controller Library includes a simplified PI controller. This controller is commonly found in motor control and other applications where derivative action is unnecessary. The controller includes a saturation block at its output and implements integrator anti-windup reset.

The theoretical background for the PI controller is outlined in Chapter 3. The following features are not present in the PI controller described here.

- Derivative action
- Set-point weighting
- Set-point weighting

With the exception of these features, the tuning procedure described in Chapter 3 can be applied directly to the PI controller. Conceptually, the PI controller is constructed as shown in Figure 4-1.

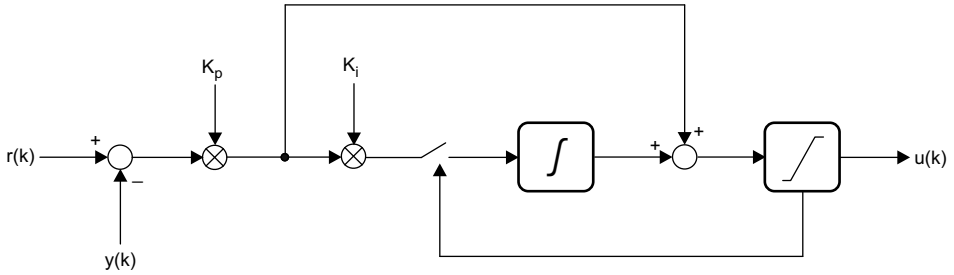

Figure 4-1. PI Controller High-Level Diagram

The input to the controller is the loop error in the current (k^{th}) sample interval. The error is multiplied by the proportional gain and this term is then added to its own integral to form the control output applied to the plant. If desired, the output can be clamped to avoid saturating other components in the loop. If the controller output exceeds the clamp limits the integrator path is disabled to prevent windup, as described in Chapter 3.

4.2 Technical Description

A more detailed diagrammatic representation of the PI controller is shown inFigure 4-2.

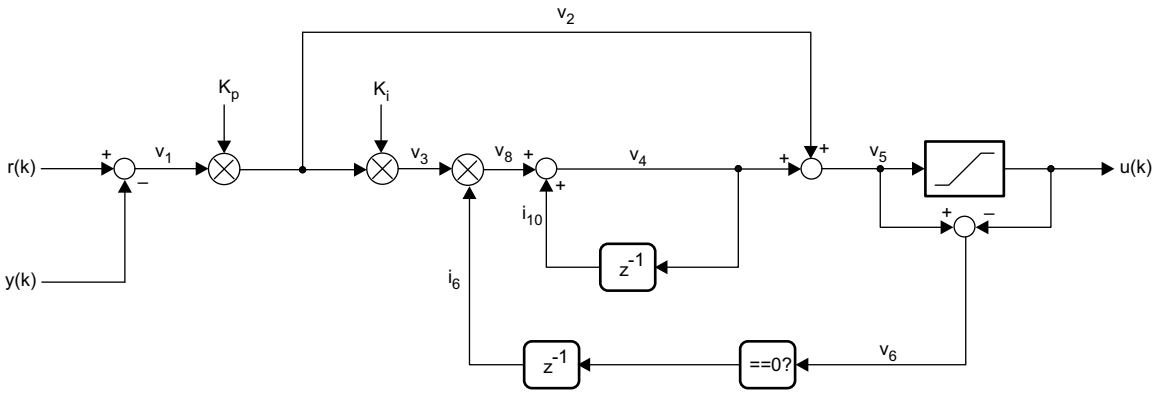

Figure 4-2. DCL PI Controller

Equation 14 represents the PI operation in the k^{th} sample interval.

$$v_5(k) = [r(k) - y(k)]\, K_p \{1 + K_i\, i_6(k)\} + i_{10}(k) \quad (14)$$

The variable i_{10} is the integrator output in the previous sample period.

$$i_{10}(k) = v_4(k - 1) \quad (15)$$

Copyright © 2015, Texas Instruments Incorporated

The variable i_6 captures the status of the output saturation block in the previous sample interval.

$$i_6(k) = \begin{cases} 1: v_5(k-1) = u(k-1) \\ 0: v_5(k-1) \neq u(k-1) \end{cases} \quad (16)$$

Maximum and minimum limits on v_5 are set by the variables *Umax* and *Umin* in the PI structure.

4.3 Adding the PI Controller to C Code

This subsection describes how to add the PI controller module to a user C code project. The method is similar to that of the other library modules.

4.3.1 Controller Structure Definition

The C type definition of the PI controller data structure is shown below:

```
typedef volatile struct {
    float Kp;       // proportional gain
    float Ki;       // integral gain
    float i10;      // I storage
    float Umax;     // upper saturation limit
    float Umin;     // lower saturation limit
    float i6;       // saturation storage
 } PI;
```

The `DCL.h` library header file includes a set of default floating-point values that can be used for initialization purposes. These configure the proportional path to have unity gain, zero integral gain, output saturation limits of ±1.

```
#define PI_DEFAULTS {   1.0f,  \
                        0.0f,  \
                        0.0f,  \
                        1.0f,  \
                        -1.0f, \
                        1.0f   \
                    }
```

An example of the C declaration of an initialized PI structure is:

```
PI pi1 = PI_DEFAULTS;
```

4.3.2 Implementing a PI Controller

The steps required to add the PID controller module to an existing user project are as follows.

1. Include the `DCL.h` header file to the user project. This file contains all the definitions for the Digital Control Library and needs to be visible to all project source files that reference them.

   ```
   #include  "DCL.h"
   ```

2. If working with the C28x CPU, add the `DCL_PI.asm` source file to the project. If working with the CLA, add the `DCL_PI_CLA.asm` source file to the project.
3. Create an instance of a PI controller and initialize it, if required (CPU only). This can be accomplished in the following declaration:

   ```
   PID pi1 = PI_DEFAULTS;
   ```

4. Call the PI controller function as described in Chapter 3. Typically, this function would be placed in an interrupt service routine to ensure it is executed at a fixed and deterministic rate. In the following example, uk, rk, and yk are floating-point variables. &pi1 is a pointer to the PI structure described in Section 4.3.1 and in step 3 of this subsection.

   ```
   uk = DCL_runPI(&pi1, rk, yk);
   ```

   Use of the PI controller with the CLA is similar to that of the PID controller described in Chapter 3, except that the function name is suffixed "c" to indicate that it runs on the CLA.

   ```
   uk = DCL_runPIc (&pi1, rk, yk);
   ```


Copyright © 2015, Texas Instruments Incorporated

4.4 Performance

This subsection contains benchmarks and test results pertaining to the PI controller. Guidelines on tuning of the PI controller are broadly similar to those given for the PID controller in Section 3.3.

4.4.1 Benchmarks

The code size and execution cycles of the PI controller are shown in Table 4-1.

Table 4-1. PI Controller Benchmarks

	Code Size (words)	Execution (cycles)
DCL_PI	58	46
DCL_PIc	40	33

4.4.2 Typical Response Results

For typical transient response plots, see Section 3.5.

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

DF13 Controller

This chapter describes a Direct Form 1 implementation of a third order control law.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

5.1 Background

Many controllers are designed to meet specifications on the open loop frequency response. Typical specifications are: gain cross-over frequency, gain margin, and phase margin. Applications that often specify control performance in this way include industrial power supplies, solar converters, and digital lighting systems. The digital control law results can be conveniently implemented in one of several Auto-Regressive, Moving Average (ARMA) structures.

The selection of compensator poles and zeros lies outside the scope of this user's guide. Descriptions of the under-lying theory can be found in many texts [1], [3].

The transfer function of an arbitrary order discrete time compensator having m zeros and n poles is shown in Equation 17.

$$F(z) = K\frac{(z+z_1)(z+z_2)...(z+z_m)}{(z+p_1)(z+p_2)...(z+p_n)} \tag{17}$$

Expanding the polynomials and absorbing the gain K into the numerator coefficients, see Equation 18.

$$F(z) = \frac{\beta_0 z^m + ... + \beta_{m-1} z + \beta_m}{\alpha_0 z^n + ... + \alpha_{n-1} z + \alpha_n} \tag{18}$$

The term "ARMA" is applied to a class of controllers described by the transfer function in Equation 18. In Equation 18, β_i and α_i represent real numerator and denominator coefficients, respectively. The transfer function is often normalized to the highest power term in the denominator so that all powers of z are negative.

In power electronic applications, ARMA controllers are most commonly either second order or third order. These are sometimes known as "2-pole, 2-zero" (2P2Z) and "3-Pole, 3-Zero" (3P3Z), respectively. The DCL code described here implements a discrete time third order ARMA controller with the transfer function shown in Equation 19.

$$F(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} \tag{19}$$

Notice that the coefficients have been adjusted to normalize the highest power of z in Equation 19.

NOTE: Coefficient Notation

There is no notational standard for the numbering of the controller coefficients. The notation used here has the advantage that the coefficient suffixes are the same as the delay line elements, and this helps with clarity of the assembly code. Other notation may be found in the literature.

The corresponding difference equation is shown in Equation 20.

$$u(k) = b_0 e(k) + b_1 e(k-1) + b_2 e(k-2) + b_3 e(k-3) - a_1 (k-1) - a_2 (k-2) - a_3 (k-3) \tag{20}$$

Copyright © 2015, Texas Instruments Incorporated

5.2 Implementation

The DF1 controller uses two, three-element delay lines to store previous input and output data required to compute u(k). A diagrammatic representation is shown in Figure 5-1.

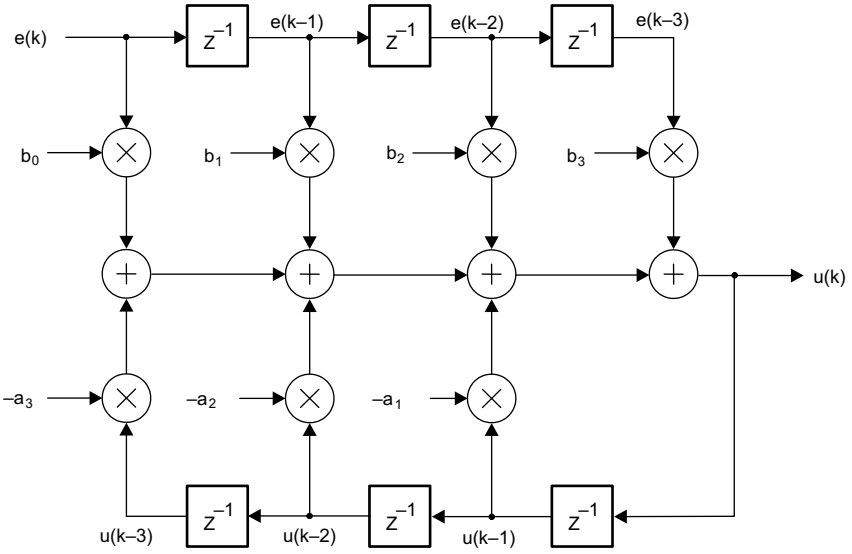

Figure 5-1. Third Order Direct Form 1 (DF1) Controller Structure

The second order (2P2Z) transfer function in Direct Form 1 is shown in Equation 21.

$$F(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{21}$$

Emulation of a second order ARMA controller is achieved by setting the b3 and a3 coefficients to zero. In such a case, the controller algorithm will compute all seven partial products in the difference equation, however, two will have a zero result.

The DF13 control law can be re-structured to reduce control latency by pre-computing six of the seven partial products in the previous sample interval. This allows the control computation in the next sample interval to be reduced to one multiplication and one addition. The control law is then broken into two parts: the “immediate” part and the “partial” part.

In the k^{th} interval, the immediate part is computed as shown in Equation 22.

$$u(k) = b_0 e(k) + v(k - 1) \tag{22}$$

Next, the v(k) partial product is precomputed for use in the $(k+1)^{th}$ interval as shown in Equation 23.

$$v(k) = b_1 e(k) + b_2 e(k - 1) + b_3 e(k - 2) - a_1(k) - a_2(k - 1) - a_3(k - 2) \tag{23}$$

Copyright © 2015, Texas Instruments Incorporated

Structurally, the control law looks similar to Figure 5-2.

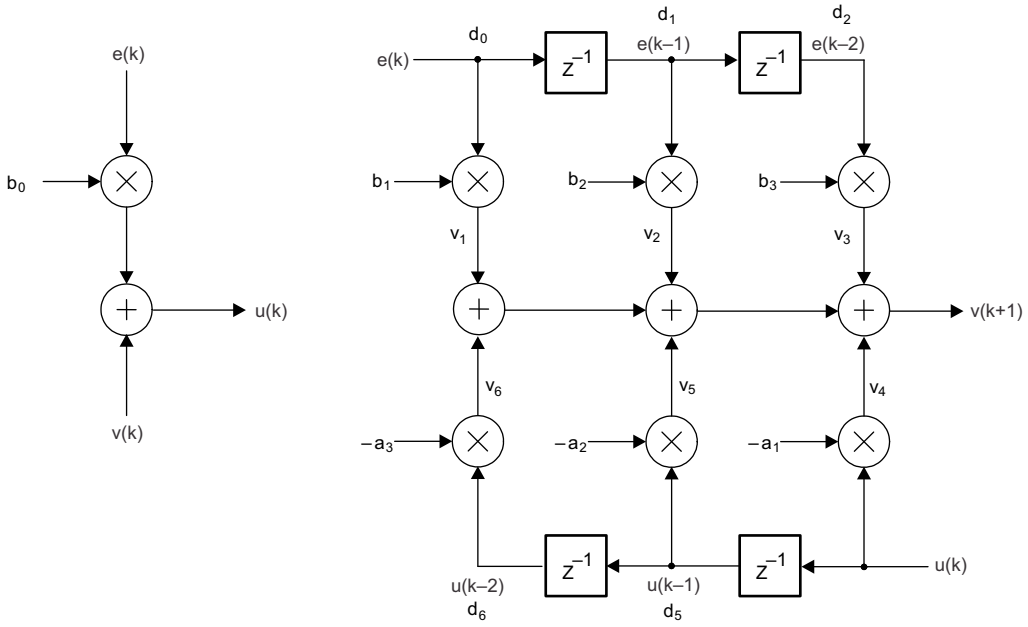

Figure 5-2. Precomputed Third Order Direct Form 1 (DF13) Controller Structure

5.3 Adding the DF13 Controller to C Code

5.3.1 Adding the DF13 Controller to C Code

The DF13 controller structure is defined in the `DCL.h` header file.

```
typedef volatile struct {
    float c[8];        // coefficients
    float d[8];        // data
    } DF13;
```

The structure consists of two arrays, each of eight elements. The first holds the transfer function coefficients; the second is used for delay line data storage. The structure elements are arranged in memory as shown in Figure 5-3.

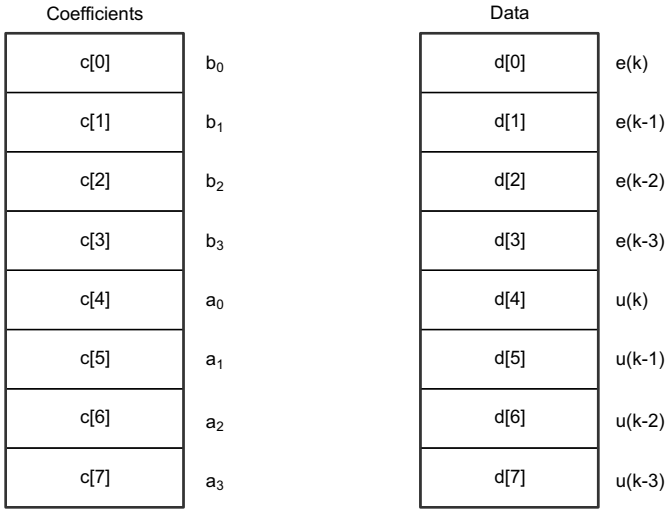

Figure 5-3. DF13 Structure Layout

Copyright © 2015, Texas Instruments Incorporated

It is the responsibility of the user to initialize both arrays prior to use. A set of default values is defined in the library header file and can be used with the variable declaration. An example of variable declaration is shown below:

```
DF13 arma1 = ARMA_DEFAULTS;
```

The controller is implemented as a C callable assembly function that is called with two arguments: a pointer to the structure and the current input e(k). The function returns a 32-bit floating-point variable corresponding to the controller output, u(k). The function prototype is shown below:

```
float DCL_runDF13(DF1_3 *p, float ek);
```

Use of the CLA version of the controller requires that the function call be made from a CLA task. The equivalent CLA function call is shown below:

```
float DCL_runDF13c(DF1_3 *p, float ek);
```

5.3.2 Using Pre-Computation With the DF13 Controller

In control applications, the time between sampling the feedback and generating a corrective output is known as “sample-to-output delay”. This delay imparts a phase lag into the loop that, in general, degrades performance. The computation time of the controller (and therefore the phase lag) can be reduced by pre-computing those terms in the difference equation, which is already known when the new input sample arrives. The controller need then only compute one partial product, regardless of the order of the controller.

The full control law in the k^{th} sample interval has already been given (see Equation 20).

If pre-computation is applied, the control law in the same sample interval reduces to:

$$u(k) = b_0e(k) + v(k - 1) \quad (24)$$

The result can be used immediately, and the remaining terms precomputed for the $(k+1)^{th}$ sample interval.

$$v(k) = b_1e(k - 1) + b_2e(k - 2) + b_3e(k - 3) - a_1(k - 1) - a_2(k - 2) - a_3(k - 3) \quad (25)$$

The DCL contains two functions that can be used to reduce the sample-to-output delay with the DF13 controller.

The first function computes the “immediate” part of the result. This takes the newest feedback sample (ek) and combines it with a precomputed partial product (vk) from the previous sample interval. The C prototype for the immediate controller function is shown below:

```
float DCL_runDF13i(DF13 *p, float ek, float vk);
```

The second function pre-computes that part of the control law needed in the next sample interval. This function requires both the input (ek) and output (uk) in the current sample interval. The C prototype for the precomputed controller function is shown below:

```
float DCL_runDF13p(DF13 *p, float ek, float uk);
```

To apply the precomputed controller, place the above functions on either side of the code that requires the result. The C code will look something like the following:

```
uk = DCL_runDF13i(&df1, ek, vk);
// ..use uk result here...
vk = DCL_runDF13p(&df1, ek, uk);
```

The precomputed term (vk) must be declared as a global or static variable in the user code in order that its’ value be preserved between function calls.

```
float DCL_runDF13ic(DF13 *p, float ek, float vk);
float DCL_runDF13pc(DF13 *p, float ek, float uk);
```


Copyright © 2015, Texas Instruments Incorporated

5.4 Performance

5.4.1 Benchmarks

Table 5-1. DF13 Controller Benchmarks

Function	Code Size (words)	Execution (cycles)
DCL_DF13	175	59
DCL_DF13i		22
DCL_DF13p		65
DCL_DF13c	192	59
DCL_DF13ic		20
DCL_DF13pc		57

5.5 PID Emulation

The properties of the basic PID controller can be emulated using a second order ARMA structure. The relationship between ARMA coefficients and the P, I, & D gains of the basic PID can be found using any of the standard discrete transformation methods. An example using the Tustin transformation is provided in [3]. The three numerator coefficients are obtained from Equation 26 through Equation 28.

$$b_2 = K_p + K_i \frac{T}{2} + K_d \frac{2}{T} \tag{26}$$

$$b_1 = K_i T - K_d \frac{4}{T} \tag{27}$$

$$b_0 = -K_p + K_i \frac{T}{2} + K_d \frac{2}{T} \tag{28}$$

Note that these coefficients are dependent on the sample period, T. The remaining ARMA coefficients are: a3 = a1 = b3 = 0, a0 = 1, a2 = -1. The controller transfer function is shown in Equation 29.

$$F(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-2}} \tag{29}$$

This approach allows the tuning of transient response properties to be accomplished within a standard ARMA structure; however, the refinements of set-point weighting and anti-windup reset present in the full PID controller are not available.

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

DF22 Controller

This chapter describes the DCL implementation of a second order Direct Form 2 controller, denoted “DF22”.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

6.1 Background

The Digital Controller Library contains one second order and one third order implementation of the Direct Form 2 controller structure. These controllers are denoted “DF22” and “DF23”, respectively. This subsection describes the DF22 controller. The DF23 controller is described in Chapter 7.

The second order structure is sometimes referred to as a “bi-quad” filter and is commonly used in a cascaded chain to build up digital filters of high order. The theoretical background for the Direct Form 2 controller is similar to that of the DF13. For more details, see Section 5.1.

The transfer function of an arbitrary order discrete time compensator having m zeros and n poles is shown in Equation 18.

In Equation 21, β_i and α_i represent real numerator and denominator coefficients, respectively. After normalizing to the α_0 term, we have for the second order case.

The corresponding difference equation is shown in Equation 30.

$$u(k) = b_0 e(k) + b_1 e(k-1) + b_2 e(k-2) - a_1 (k-1) - a_2 (k-2) \quad (30)$$

Figure 6-1 shows a representation of the second order Direct Form 2 controller.

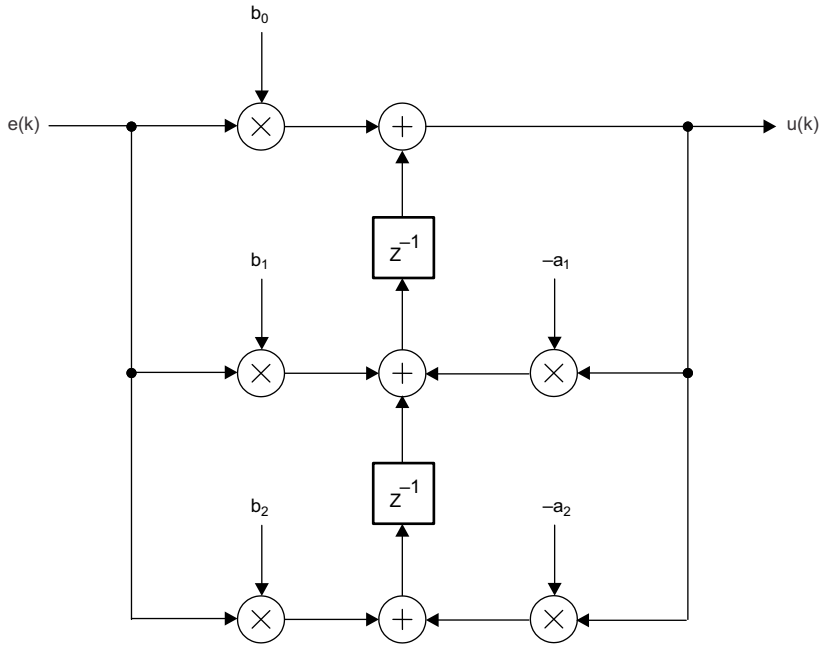

Figure 6-1. Second Order Direct Form 2 (DF22) Controller Structure

An advantage of the Direct Form 2 structure is that only one delay line need be maintained. This reduces the number of variables that must be preserved across function calls and reduces the execution time of the controller algorithm. A disadvantage of the Direct Form 2 structure is that, in general, it is more sensitive to coefficient variation than the Direct Form 1.

The direct part, from e(k) to u(k) through coefficient b0, is easier to separate in the DF22 case. This allows the time critical part of the controller to be computed without re-structuring the algorithm, as would have to be done in the DF1 case.

Copyright © 2015, Texas Instruments Incorporated

As with the DF13 controller, it is possible to apply pre-computation to the DF22 structure to reduce sample-to-output delay. The DF22 controller structure with pre-computation is shown in Figure 6-2.

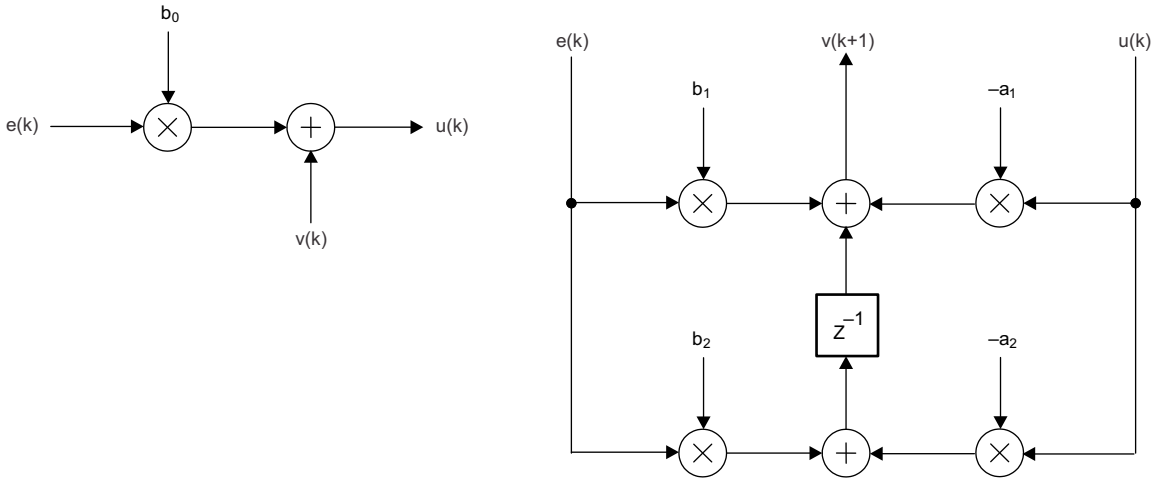

Figure 6-2. Precomputed Second Order Direct Form 2 (DF22) Controller Structure

6.2 Adding the DF22 Controller to C Code

This subsection describes how to add the DF22 controller module to a C code project. The method is similar to that of the other library modules.

6.2.1 Controller Structure Definition

The C type definition of the DF22 controller data structure is shown below:

```
typedef volatile struct {
    float b0;
    float b1;
    float b2;
    float a1;
    float a2;
    float x1;
    float x2;
} DF22;
```

The DCL.h library header file includes a set of default floating-point values that can be used to initialize the controller structure. These configure the structure to have a direct unity gain input-output connection (b0 coefficient set to 1 and all other coefficients and internal data set to 0).

```
#define DF22_DEFAULTS { 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f }
```

An example of the C declaration of an initialized DF22 structure is shown below:

```
DF22 arma2 = DF22_DEFAULTS;
```

6.2.2 Implementing a DF22 Controller

The steps required to add the DF22 controller module to an existing user project are as follows:

1. Include the DCL.h header file to the user project. This file contains all the definitions for the Digital Control Library and needs to be visible to all project source files that reference them.
   ```
   #include  “DCL.h”
   ```
2. Add the DCL_DF22.asm or DCL_DF22_CLA.asm source files to the project.
3. Create an instance of the controller and initialize it (C28x only), if required. This can be accomplished in the following declaration.
   ```
   DF22 arma2 = DF22_DEFAULTS;
   ```


Copyright © 2015, Texas Instruments Incorporated

4. Call the controller function as required. Typically, this function would be placed in an interrupt service routine to ensure it is executed at a fixed and deterministic rate. In the following example, “ek” and “uk” are floating-point variables corresponding to the controller input and output, respectively. The argument *&arma2* is a pointer to the controller structure described in Section 6.2.1 and in step 2 of this subsection.

```
uk = DCL_runDF22(&arma2, ek);
```

The equivalent CLA function is shown below:

```
uk = DCL_runDF22c(&arma2, ek);
```

6.2.3 Implementing a Precomputed DF22 Controller

When using the precomputed form of the DF22 controller, the description changes for step 4 in Section 6.2.2. There will be two controller functions: the “immediate” and “partial” functions, as described for the DF13 controller in Chapter 5.

```
float DCL_runDF22i(DF22 *p, float ek);
void DCL_runDF22p(DF22 *p, float ek, float uk);
```

Note that the result of the partial controller appears as one of the states in the direct form 2 structure. It is, therefore, unnecessary to return this from the partial controller function or to pass it as an argument to the immediate function in the following sample interval, as was done in the DF13 case.

To apply the precomputed DF22 controller, place the above functions on either side of the code that requires the result. The C code will look something like the following:

```
uk = DCL_runDF22i (&arma2, ek);
// ...use uk result here
CL_runDF22p (&arma2, ek, uk);
```

The use of CLA functions is identical to that shown above, except that the function names carry a “c” suffix. The CLA function prototypes are shown below:

```
float DCL_runDF22ic(DF22 *p, float ek);
void DCL_runDF22pc(DF22 *p, float ek, float uk);
```

6.3 Performance

6.3.1 Benchmarks

Table 6-1. DF22 Controller Benchmarks

Function	Code Size (words)	Execution (cycles)
DCL_DF22	121	42
DCL_DF22i		21
DCL_DF22p		36
DCL_DF22c	90	32
DCL_DF22ic		17
DCL_DF22pc		33

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

DF23 Controller

This chapter describes the DCL implementation of a third order Direct Form 2 controller, denoted “DF23”.

Topic **Page**

Copyright © 2015, Texas Instruments Incorporated

7.1 Background

The theoretical background for the DF23 controller is identical to that of the DF13. For details, see Section 5.1.

The normalized third order transfer function is shown in Equation 19.

The corresponding difference equation is shown in Equation 20.

Figure 7-1 shows a representation of the third order DF23 controller.

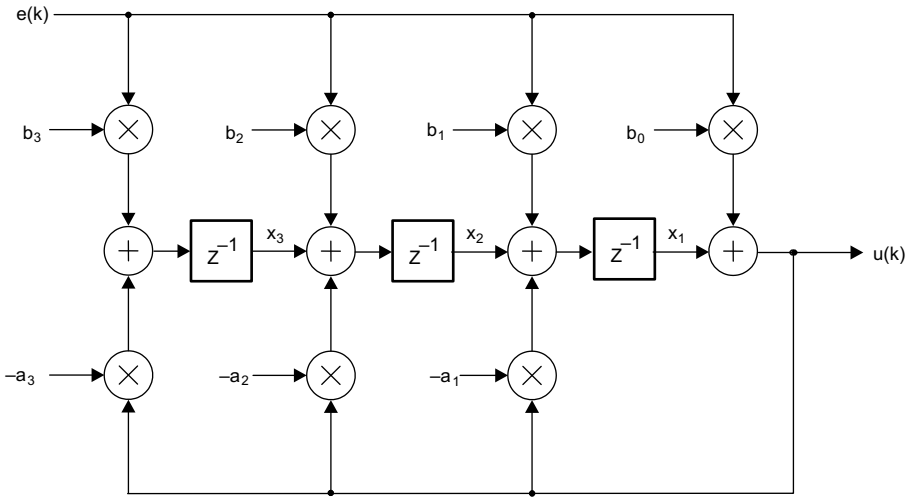

Figure 7-1. Third Order Direct Form 2 (DF23) Controller Structure

The precomputed form of the DF23 controller is shown in Figure 7-2.

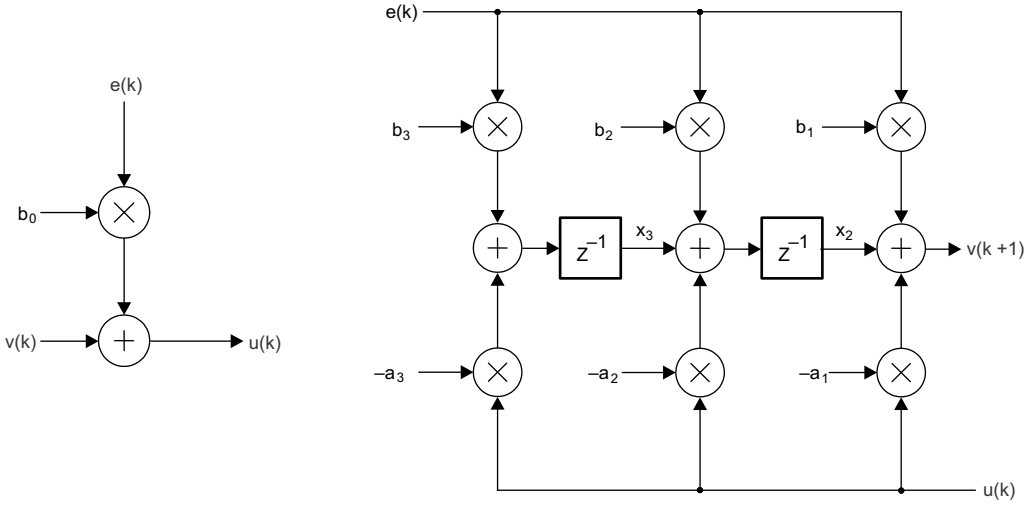

Figure 7-2. Precomputed Third Order Direct Form 2 (DF23) Controller Structure

Copyright © 2015, Texas Instruments Incorporated

7.2 Adding the DF23 Controller to C Code

This subsection describes how to add the DF23 controller module to a C code project. The method is similar to that of the other library modules.

7.2.1 Controller Structure Definition

The C type definition of the DF23 controller data structure is shown below:

```
typedef volatile struct {
    float b0;
    float b1;
    float b2;
    float b3;
    float a1;
    float a2;
    float a3;
    float x1;
    float x2;
    float x3;
} DF23;
```

The `DCL.h` library header file includes a set of default floating-point values that can be used to initialize the controller structure. These configure the structure to have a direct unity gain input-output connection (b0 coefficient set to 1 and all other coefficients and internal data set to 0).

```
#define DF23_DEFAULTS { 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f }
```

An example of the C declaration of an initialized DF23 structure is shown below:

```
DF23 arma3 = DF23_DEFAULTS;
```

7.2.2 Implementing a DF23 Controller

The steps required to add the DF23 controller module to an existing user project are as follows:

1. Include the `DCL.h` header file to the user project. This file contains all the definitions for the Digital Control Library and needs to be visible to all project source files that reference them.

   ```
   #include  “DCL.h”
   ```

2. Add the `DCL_DF23.asm` or `DCL_DF23_CLA.asm` source files to the project.
3. Create an instance of the controller and initialize it, if required. For the C28x, this can be accomplished in the following declaration.

   ```
   DF23 arma3 = DF23_DEFAULTS;
   ```

4. Call the controller function as required. Typically, this function would be placed in an interrupt service routine to ensure it is executed at a fixed and deterministic rate. In the following example “ek” and “uk” are floating-point variables corresponding to the controller input and output, respectively. The argument *&arma3* is a pointer to the controller structure described in Section 7.2.1 and in step 3 of this subsection.

   ```
   uk = DCL_runDF23 (&arma3, ek);
   ```

   The equivalent CLA function is shown below:

   ```
   uk = DCL_runDF23c (&arma3, ek);
   ```

   As with the other controllers, pre-computation can be applied to reduce control loop latency. To apply the precomputed DF23 controller, place the above functions on either side of the code that requires the result. The C code will look something like this:

   ```
   uk = DCL_runDF23i (&arma3, ek);
   // ...use uk result here
   DCL_runDF23p (&arma3, ek, uk);
   ```

   The use of CLA functions are identical to that shown above, except that the function names carry a “c” suffix. The prototypes are shown below:

   ```
   float DCL_runDF23ic(DF23 *p, float ek);
   void DCL_runDF23pc(DF23 *p, float ek, float uk);
   ```


Copyright © 2015, Texas Instruments Incorporated

7.3 Performance

This subsection contains benchmarks and test results pertaining to the DF23 controller.

7.3.1 Benchmarks

Table 7-1. DF23 Controller Benchmarks

Function	Code Size (words)	Execution (cycles)
DCL_DF23	157	56
DCL_DF23i		20
DCL_DF23p		49
DCL_DF23c	140	44
DCL_DF23ic		20
DCL_DF23pc		44

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

References

1. Information on a series of technical seminars in control theory can be found at www.controltheoryseminars.com.
2. J.J.DiStefano, A.R.Stubberud & I.J.Williams, *Feedback & Control Systems*, Schaum, 2011.
3. G.F.Franklin, J.D.Powell & M.L.Workman, *Digital Control of Dynamic Systems*, Addison-Wesley, 1998.
4. R.Poley, *Control Theory Fundamentals*, CreateSpace, 2015.
5. *TMS320C28x Assembly Language Tools User's Guide* (SPRU513)
6. *TMS320C28x Optimizing C/C++ Compiler User's Guide* (SPRU514)

Copyright © 2015, Texas Instruments Incorporated

Copyright © 2015, Texas Instruments Incorporated

Data Logger Utility

Included in the Digital Controller Library is a general-purpose data logger utility that is useful when testing and debugging control applications.

A.1 Technical Description

The data logger utility is supplied in the form of a C header file. No source code is associated with this utility and it may be used on any C2000 device irrespective of whether the DCL is used. The utility may not be used on the CLA.

The data logger stores arrays of 32-bit floating-point data. The location, size and indexing of each array is defined by a set of three pointers capturing the start address, end address, and data index address. All three pointers are held in a C structure.

```
typedef volatile struct {
    float *fptr;
    float *lptr;
    float *dptr;
} FDLOG;
```

Conceptually, the relationship between the array pointers and the elements of a data array of length “N” is shown in Figure A-1.

The intended use of the data logger utility is to capture a stream of data values in a block of memory for subsequent analysis. The data index pointer (dptr) advances through the memory block as each new value is written into the log. On reaching the end of the log, the pointer is reset to the first address in the log and old data is over-written. The data logger header file contains a set of C macros and in-lined C functions to access and manipulate data logs.

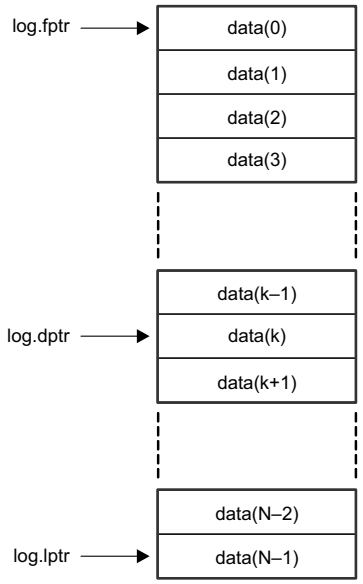

Figure A-1. Data Logger Pointer Assignment

Copyright © 2015, Texas Instruments Incorporated

Initially, the indexing pointer (dptr) is equal to the first address in the array (dptr = fptr). As data is added to the log, the index pointer advances through the memory block until it reaches the last address pointer (dptr = lptr), at which point the buffer is full. The next data element will be added to the first element in the array, over-writing any existing data at that address. In this way, the logger implements a circular buffer, the bounds of which are set by the fptr and lptr pointers.

Structure pointers may be read at any time by user code. Note that the index pointer (dptr) always points to the address in the buffer to which the next incoming data point will be written.

A.2 Using the Data Logger

To use the data logger, the `DCL_fdlog.h` header file must be added to the user project and made visible to each source file that references it. A variable may be declared with this data type as follows.

```
FDLOG log = FDLOG_DEFAULTS;
```

This declaration initializes all pointers to address zero. Subsequently, the user code may assign pointers to the desired memory addresses.

A.3 Data Logger Functions

The following functions are included in the data logger utility:

- Initializes the contents of the log to zero

```
DCL_clearLog(FDLOG *fdlog);
```

- Assigns addresses to the data log start and end pointers and assigns dptr to fptr

```
DCL_createLog(FDLOG *fdlog, float *StartAddr, unsigned int buflen);
```

- Assigns all buffer pointer addresses to zero

```
DCL_deleteLog(FDLOG *fdlog);
```

- Copies a specified data value to all elements in the log

```
DCL_fillLog(FDLOG *fdlog, float data);
```

- Returns the value current value and advances dptr by 1 element, wrapping, if necessary

```
DCL_readLog(FDLOG *fdlog);
```

- Assigns dtpr to fptr

```
DCL_resetLog(FDLOG *fdlog);
```

- Returns value at current dptr address and then replaces the element with new data. The data pointer (dptr) is then incremented, wrapping on the last element. The return element allows the function to be used to implement a fixed length delay.

```
DCL_writeLog(FDLOG *fdlog, float data);
```

A.4 Data Logger Macros

The following C macros are included in the data logger utility:

- Number of elements between dptr and lptr

```
FDLOG_SPACE(buf)
```

- Total number of elements in the log

```
FDLOG_SIZE(buf)
```

- Number of elements between dptr and lptr

```
FDLOG_SPACE(buf)
```

- Index number of the current element in the log

```
FDLOG_ELEMENT(buf)
```

- True, if dptr points to the first element in the log

```
FDLOG_START(buf)
```

- True, if dptr points to the last element in the log

```
FDLOG_END(buf)
```


Copyright © 2015, Texas Instruments Incorporated

- True, if dptr points outside the log range

```
FDLOG_ FDLOG_OUT_OF_RANGE(buf)
```

- True, if log size is non-zero

```
FDLOG_EXISTS(buf)
```

A.5 Instrumentation Functions

The DCL data logger utility currently contains one instrumentation function that computes a performance index based on the absolute servo error.

```
DCL_runITAE(FDLOG *log1, FDLOG *log2, float period);
```

The arguments log1 and log2 are buffers that hold the input control loop reference and measured output, respectively. Both logs must have the same length. The third argument is the sample period in seconds. The return value is the time weighted integral of absolute servo error (ITAE).

In Equation 31, r and y represent the input and feedback, and x is the ITAE index. N represents the log length and T is the sample period.

$$x = T \sum_{i=0}^{N-1} i \left| r_i - y_i \right| \tag{31}$$

The ITAE index is a useful indication of the effectiveness of control action following a transient input. The index is concave with respect to most control variables and is therefore useful in manual control loop tuning. Typically, parametric adjustments are made iteratively to minimize the index. Obviously, the integration interval (buffer lengths) must be fixed in order for comparative measurements to be meaningful.

Copyright © 2015, Texas Instruments Incorporated

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2015, Texas Instruments Incorporated