

SMPS Control Library Help

Table of Contents

1 SMPS Control Library	1-1
-------------------------------	------------

1 SMPS Control Library

Files

Name	Description
smps_control.h	This header file lists the interfaces used by the Switch Mode Power Supply compensator library.

Description

1.1 Introduction

SMPS Control Library for Microchip Microcontrollers

This [library](#) is a collection of optimized controller functions commonly used in Switch Mode Power Supply (SMPS) applications.

Description

The SMPS Control [library](#) contains function blocks that are optimized for the dsPIC33F and dsPIC33E family of Digital Signal Controllers (DSC). The [library](#) functions are designed to be used within an application framework for realizing an efficient and flexible way of implementing the control of an SMPS application.

The block diagram in Figure-1 shows a typical usage scenario. The user-developed SMPS application interfaces to the DSC peripherals while using function calls into this [library](#) to perform majority of the time-critical operations.

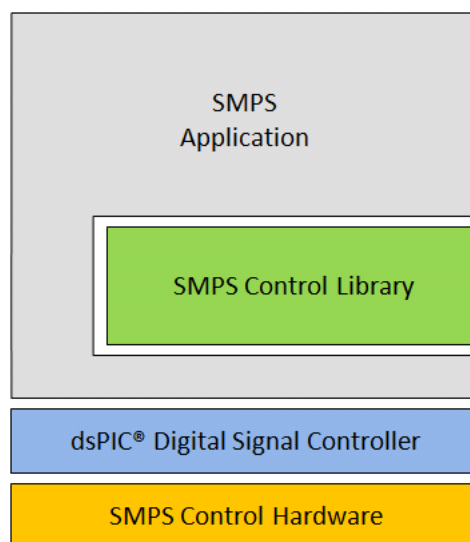


Figure-1: Block diagram of a typical [library](#) usage scenario.

1.2 Release Notes

SMPS Control Library Version : 0.10 Release Date: December 11th, 2013

New:

This is the first release of the [library](#). The interface can change in the beta and/or 1.0 release.

Changes:

None.

Fixes:

None.

Known Issues:

None.

Development Tools:

This version of the [library](#) is tested to be compatible with the following:

- XC16 v1.11 compiler (only)
- MPLAB X IDE v1.90 and later

Performance and functional correctness of the [library](#) cannot be guaranteed if this version of the [library](#) is used with versions of the development tools other than those listed above.

1.3 SW License Agreement

(c) 2013 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

1.4 Library Overview

This topic describes the basic architecture of the SMPS Control Library and provides information and examples on how to use it.

The SMPS Control Library hosts functions as defined in the Interface Header File.

Interface Header File: [smps_control.h](#)

The interfaces to the SMPS Control [library](#) are defined in the "[smps_control.h](#)" header file. Any C language source (.c) file that uses the SMPS Control [library](#) should include the "[smps_control.h](#)".

Library Files: [libsmps_control_dspic33f-elf.a](#)

The SMPS Control [library](#) archive (.a) files installed with the [library](#) release. The prototypes for [library](#) functions hosted by the archive files are described in the [smps_control.h](#) file. Both of these archive files released with the [library](#) are built using the ELF-type of Object Module Format (OMF).

1.4.1 Library Sections

The [library](#) interface routines for each of the controllers is divided into two sub-sections. Each sub-section addresses one of the classes of operation in the SMPS Control [library](#).

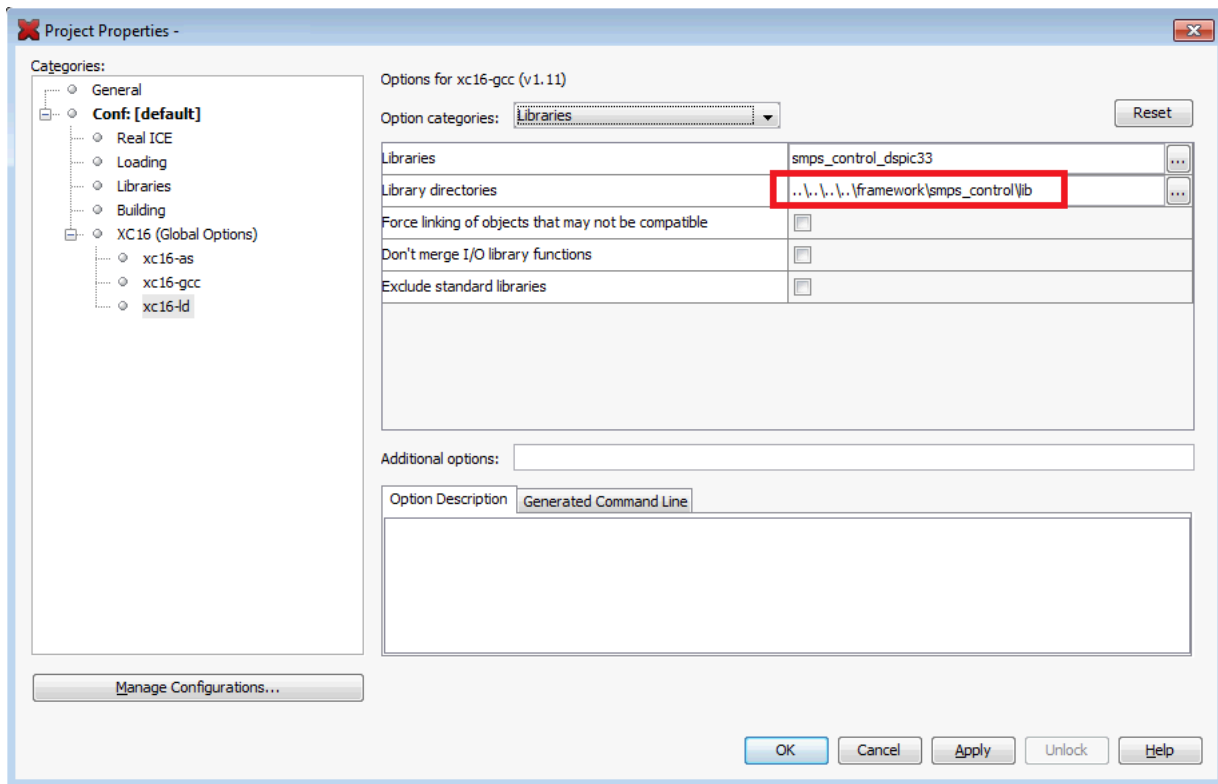
Library Interface Section	Description
Controller Initialization	This function clears the controller data structure arrays
Controller Implementation	This function calls the Digital controller

1.4.2 Library Usage Model

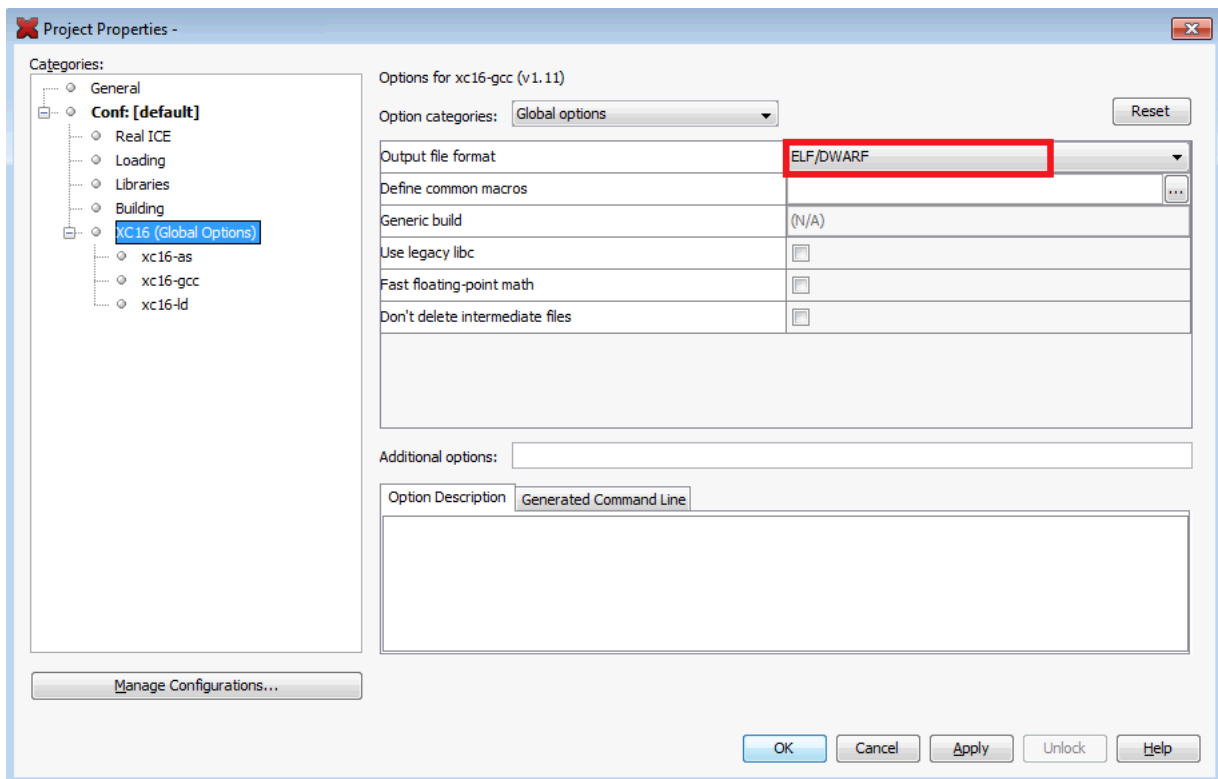
This topic describes the typical usage model for this [library](#).

In order to use the [library](#) in the user application:

1. Include the [library](#) archive file into the application project. Add the [library](#) archive directory into the Project Properties -> xc16-ld -> (Option categories) Libraries field as shown below.



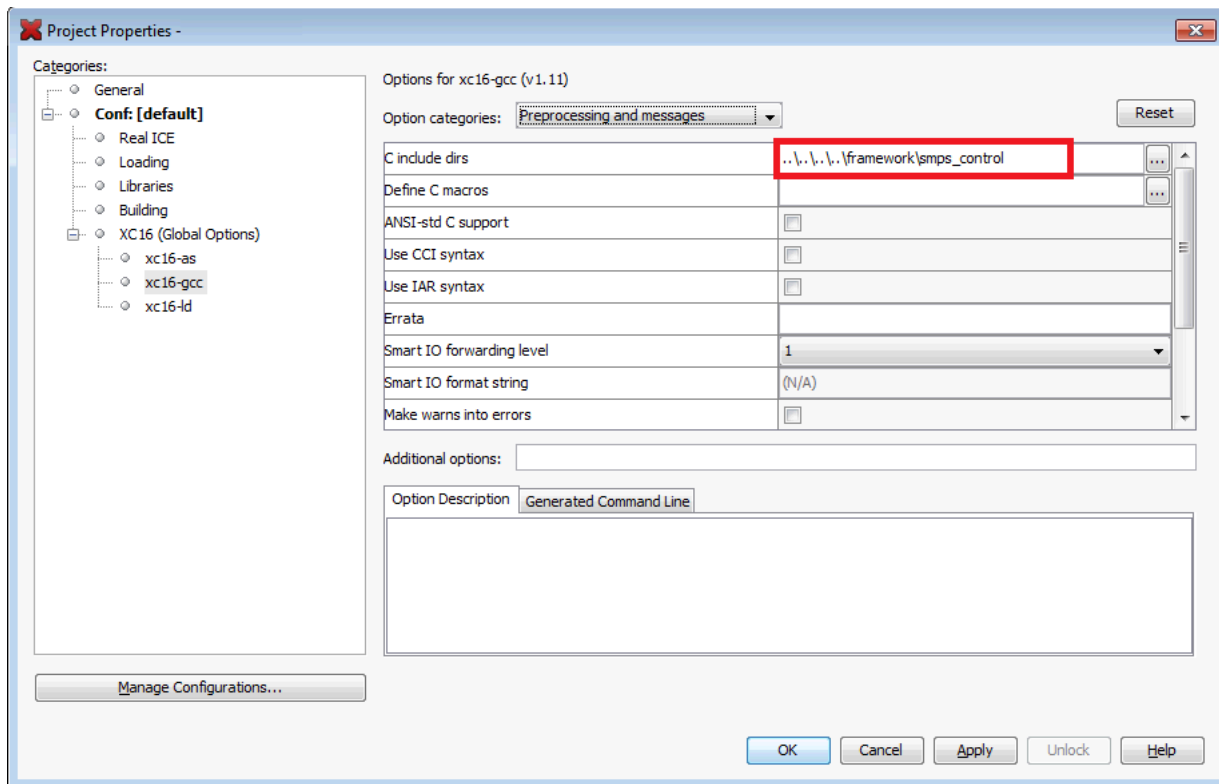
2. Ensure that the application project is configured to use ELF/DWARF type of output file format.



3. Include the `smcps_control.h` file in all C language source (.c) file that use the SMPS Control [library](#).

```
#include "smcps_control.h"
```

4. Add the [library](#) path to the C include directory field in Project Properties -> xc16-gcc -> (Option categories) Preprocessing and messages -> C include dirs.



1.4.3 Using the 3P3Z Controller

The 3P3Z controller is the digital implementation of the analog type III controller. This is a filter which generates a compensator characteristic considering three poles and three zeros. This controller requires four feedback error multiplied by their associated coefficients plus the three latest controller output values multiplied by their associated coefficients along the delay line to provide proper compensation of the power converter. The coefficients are determined externally using simulation tools.

Three Pole Three Zero (3P3Z) - Controller:

This controller was designed with a good trade-off between speed and accuracy

Filename Description:

SMPS_Controller3P3Z stands for Switch Mode Power Supply 3-pole 3-zero controller. The coefficients can cover a wide range of numbers depending on the performance requirements for the system or plant. This controller was programmed to work with Q15 numbers only, therefore the coefficients have to be normalized to the range between -1 and +1 before implementation.

File Usage:

1) Variable Declarations

To use this controller in an application, one or more controller(s) can be defined as following:

- Include stdint.h for declaration in the 16-bit integer format

```
#include <stdint.h>
```

- Declare a 3P3Z Data Structure (e.g. controller3P3Z)

```
SMPS_3P3Z_T controller3P3Z;
```

The controller3P3Z data structure contains a pointer to derived coefficients in X-space and pointer to controller and error

history in Y-space. So declare variables for the derived coefficients and the controller history samples, this can be done in main.h

```
int16_t controller3P3ZACoefficient[3] __attribute__((section(".xbss")));
int16_t controller3P3ZControlHistory[3] __attribute__((section(".ybss")));
int16_t controller3P3ZBCoefficient[4] __attribute__((section(".xbss")));
int16_t controller3P3ZErrorHistory[4] __attribute__((section(".ybss")));
```

2) Controller Initialization

Before the controller can be used, it has to be initialized. First, the data structure has to be filled by copying the pointers to the coefficients, error and controller history arrays into the structure, in addition the physical clamping limits of the output value need to be defined, for example:

```
controller3P3Z.aCoefficients = &controller3P3ZACoefficient[0]; // Set up pointer to derived coefficients
controller3P3Z.bCoefficients = &controller3P3ZBCoefficient[0]; // Set up pointer to derived coefficients
controller3P3Z.controlHistory = &controller3P3ZControlHistory[0]; // Set up pointer to controller history
controller3P3Z.errorHistory = &controller3P3ZErrorHistory[0]; // Set up pointer to error history
controller3P3Z.preShift = (e.g. 5); // Normalization shift for error amplifier into Q15 format
controller3P3Z.postShift = (e.g. 1); // Normalization shift for control loop results to peripheral
controller3P3Z.postScaler = (e.g. 2); // Normalization shift for control loop results to peripheral
controller3P3Z.minOutput = (e.g. min duty cycle); // Clamp value for minimum duty ratio
controller3P3Z.maxOutput = (e.g. max duty cycle); // Clamp value for maximum duty ratio
```

It's recommended to clean up the error-history and controller-history arrays before start-up using the following instruction:

```
SMPS\_Controller3P3ZInitialize(&controller3P3Z); // Clear history
```

3) Calling the Controller

As soon as the coefficients have been loaded into their arrays, the controller can be called using the following instruction:

```
SMPS\_Controller3P3ZUpdate(&controller3P3Z,&ADCBUF0,controlReference,&PDC1)
```

This function call includes the pointer to the controller data structure, pointer of the input source register, control reference value, and to the pointer to the output register.

CONTROL LOOP DEFINITIONS

Transfer Function **for** a Digital 3P3Z Controller

$$H(z) = \frac{u(z)}{e(z)} = \frac{B0 + B1 z^{-1} + B2 z^{-2} + B3 z^{-3}}{-A3 z^{-3} - A2 z^{-2} - A1 z^{-1} + 1}$$

The Linear Difference Equation becomes:

$$u(n) = B0 e(n) + B1 e(n-1) + B2 e(n-2) + B3 e(n-3) + A1 u(n-1) + A2 u(n-2) + A3 u(n-3)$$

The [SMPS_3P3Z_T](#) data structure contains a pointer to derived coefficients in X-space and pointer to error/control history samples in Y-space. So declare variables for the derived coefficients and the error history samples.

The abCoefficients referenced by the [SMPS_3P3Z_T](#) data structure are derived from the coefficients B0-B3 plus A1-A3. These will be declared in external arrays. The [SMPS_3P3Z_T](#) data structure just holds pointers to these arrays.

The coefficients will be determined by simulation tools, which output is given as floating point numbers. These numbers will be copied into the declared arrays after they have been converted into 16-bit integer numbers.

1.4.4 Using the 2P2Z Controller

The 2P2Z controller is the digital implementation of the analog type II controller. This is a filter which generates a compensator characteristic considering two poles and two zeros. This controller requires three feedback error multiplied by their associated coefficients plus the two latest controller output values multiplied by their associated coefficients along the delay line to provide proper compensation of the power converter. The coefficients are determined externally using simulation tools.

Two-Pole Two-Zero (2P2Z) - Controller:

This controller was designed with a good trade-off between speed and accuracy

Filename Description:

SMPS_Controller2P2Z stand for Switch Mode Power Supply 2-pole 2-zero controller. The coefficients can cover a wide range of numbers depending on the performance requirements for the system or plant. This controller was programmed to work with Q15 numbers only, therefore the coefficients have to be normalized to the range between -1 and +1 before implementation.

File Usage:

1) Variable Declarations

To use this controller in an application, one or more controller(s) can be defined as following:

- Include stdint.h for declaration in the 16-bit integer format

```
#include <stdint.h>
```

- Declare a 2P2Z Data Structure (e.g. controller2P2Z)

```
SMPS_2P2Z_T controller2P2Z;
```

The controller2P2Z data structure contains a pointer to derived coefficients in X-space and pointer to controller and error history in Y-space. So declare variables for the derived coefficients and the controller history samples, this can be done in main.h

```
int16_t controller2P2ZACoefficient[2] __attribute__((section(".xbss")));
```

```
int16_t controller2P2ZControlHistory[2] __attribute__((section(".ybss")));
```

```
int16_t controller2P2ZBCoefficient[3] __attribute__((section(".xbss")));
```

```
int16_t controller2P2ZErrorHistory[3] __attribute__((section(".ybss")));
```

2) Controller Initialization

Before the controller can be used, it has to be initialized. First, the data structure has to be filled by copying the pointers to the coefficients, error and controller history arrays into the structure, in addition the physical clamping limits of the output value need to be defined, for example:

```
controller2P2Z.aCoefficients = &controller2P2ZACoefficient[0]; // Set up pointer to derived coefficients
```

```
controller2P2Z.bCoefficients = &controller2P2ZBCoefficient[0]; // Set up pointer to derived coefficients
```

```
controller2P2Z.controlHistory = &controller2P2ZControlHistory[0]; // Set up pointer to controller history
```

```
controller2P2Z.errorHistory = &controller2P2ZErrorHistory[0]; // Set up pointer to error history
```

```
controller2P2Z.preShift = (e.g. 5); // Normalization shift for error amplifier into Q15 format
```

```
controller2P2Z.postShift = (e.g. 1); // Normalization shift for control loop results to peripheral
```

```
controller2P2Z.postScaler = (e.g. 2); // Normalization shift for control loop results to peripheral
```

```
controller2P2Z.minOutput = (e.g. min duty cycle); // Clamp value for minimum duty ratio
```

controller2P2Z.maxOutput = (e.g. max duty cycle); // Clamp value for maximum duty ratio

It's recommended to clean up the error-history and controller-history arrays before start-up using the following instruction:

`SMPS_Controller2P2ZInitialize(&controller2P2Z);` // Clear history

3) Calling the Controller

As soon as the coefficients have been loaded into their arrays, the controller can be called using the following instruction:

`SMPS_Controller2P2ZUpdate(&controller2P2Z,&ADCBUF0,controlReference,&PDC1)`

This function call includes the pointer to the controller data structure, pointer of the input source register, control reference value, and to the pointer to the output register.

CONTROL LOOP DEFINITIONS

Transfer Function **for** a Digital 2P2Z Controller

$$H(z) = \frac{u(z)}{e(z)} = \frac{B0 + B1 z^{-1} + B2 z^{-2}}{-A2 z^{-2} - A1 z^{-1} + 1}$$

The Linear Difference Equation becomes:

$$u(n) = B0 e(n) + B1 e(n-1) + B2 e(n-2) + A1 u(n-1) + A2 u(n-2)$$

The `SMPS_2P2Z_T` data structure contains a pointer to derived coefficients in X-space and pointer to error/control history samples in Y-space. So declare variables for the derived coefficients and the error history samples.

The abCoefficients referenced by the `SMPS_2P2Z_T` data structure are derived from the coefficients B0-B2 plus A1-A2. These will be declared in external arrays. The `SMPS_2P2Z_T` data structure just holds pointers to these arrays.

The coefficients will be determined by simulation tools, which output is given as floating point numbers. These numbers will be copied into the declared arrays after they have been converted into 16-bit integer numbers.

1.4.5 Using the PID Controller

The digital implementation of a PID controller is a filter which generates a compensator characteristic considering the values of the coefficients KA, KB, KC these coefficients will determine the converter's frequency response. These coefficients are determined externally using simulation tools.

Filename Description:

SMPS_ControllerPID stands for Proportional Integral Derivative controller for switch mode power supply. This controller was programmed to operate using Q15 numbers only. Therefore the coefficients have to be normalized to the range between -1 and +1 before implementation.

File Usage:

1) Variable Declarations

To use this controller in an application, one or more controller(s) can be defined as following:

- Include stdint.h for declaration in the 16-bit integer format (int16_t)

`#include <stdint.h>`

- Declare a PID Data Structure (e.g. controllerPID)

`SMPS_PID_T controllerPID;`

The `SMPS_PID_T` data structure contains a pointer to derived coefficients in X-space and pointer to controller and error history in Y-space. So declare variables for the derived coefficients and the controller history samples, this can be done in main.h

```
int16_t controllerPIDCoefficientABC[3] __attribute__((section (".xbss")));  
int16_t controllerPIDControlHistory[1] __attribute__((section (".ybss")));  
int16_t controllerPIDErrorHistory[3] __attribute__((section (".ybss")));
```

2) Controller Initialization

Before the controller can be used, it has to be initialized. First, the data structure has to be filled by copying the pointers to the coefficients, error and controller history arrays into the structure, in addition the physical clamping limits of the output value need to be defined, for example:

```
controllerPID.abcCoefficients = &controllerPIDCoefficientABC[0]; // Set up pointer to derived coefficients  
controllerPID.controlHistory = &controllerPIDControlHistory[0]; // Set up pointer to controller history  
controllerPID.errorHistory = &controllerPIDErrorHistory[0]; // Set up pointer to error history  
controllerPID.preShift = (e.g. 5); // Normalization shift for error amplifier results in Q15 format  
controllerPID.postShift = (e.g. 1); // Normalization shift for control loop results to peripheral  
controllerPID.postScaler = (e.g. 2); // Normalization shift for control loop results to peripheral  
controllerPID.minOutput = (e.g. min duty cycle); // Clamp value for minimum duty ratio  
controllerPID.maxOutput = (e.g. max duty cycle); // Clamp value for maximum duty ratio
```

It's recommended to clean up the error-history and controller-history arrays before start-up using the following instruction:

```
SMPS\_ControllerPIDInitialize(&controllerPID);
```

3) Calling the Controller

As soon as the coefficients have been loaded into their arrays, the controller can be called using the following instruction:

```
SMPS\_ControllerPIDUpdate(&controllerPID,&ADCBUF0,controlReference,&PDC1)
```

This function call includes the pointer to the controller data structure, pointer of the input source register, control reference value, and to the pointer to the output register.

1.5 Modifying the Library

At release, the [library](#) includes one archive files:

1. [libsmps_control_dspic33f-elf.a](#) - To be used with dsPIC33F family of devices.

This archive file released with the [library](#) are built using the ELF type Object Module Format (OMF). The source (.s) files that are used to build these archive files are also provided with the [library](#), in the [/src](#) folder. These source files are provided for reference and need not be used directly in a typical [library](#) usage scenario.

Users may also utilize the flexibility provided by the [library](#) to modify the source files and re-build their own archive files. In order to help users to get started, a [library](#) project is included in the [/mplabx](#) folder of the [library](#):

1. [libsmps_control_dspic33f-elf.X](#) - To be used with dsPIC33F family of devices.

These [library](#) projects assemble the source files from the [/src](#) folder using an assembly API file, and then archive the assembled output object files into a binary archive. The binary archive is by default, saved in the [/mplabx/libsmps_control_dspic33f-elf.X/dist/default/production](#) folder.

1.6 Performance

The following table lists the approximate number of instruction cycles required to:

1. Save the arguments
2. Call into the [library](#) function
3. Return from the [library](#) function

Function Name	Instruction Cycle Usage (Minimum)
SMPS_Controller3P3ZInitialize	18
SMPS_Controller2P2ZInitialize	16
SMPS_ControllerPIDInitialize	15
SMPS_Controller3P3ZUpdate	63
SMPS_Controller2P2ZUpdate	58
SMPS_ControllerPIDUpdate	52

Note:

The above performance numbers were measured on the SMPS Control Library Version : 0.10 Release




1.7 Register Usage

The register usage and handling behavior of the [library](#) functions are as described below.




1. Assembly implementation: Register W0 - W7 are caller saved. The calling function must preserve these values before the [library](#) function call if their value is required subsequently from the [library](#) function call. The stack is a good place to preserve these values.
2. Assembly implementation: Register W8 - W14 are saved by the [library](#) function if they are used within the [library](#) function.
3. Register W0 - W7 may be used for argument transmission.
4. Accumulator (A and B) registers are not saved by any of the [library](#) functions. If the calling function requires the accumulator registers to be unchanged after the [library](#) function call, the calling function will have to save the accumulator registers before the [library](#) function call.
5. Core Control Register (CORCON): certain [library](#) functions require CORCON register to be setup in a certain state in order to operate correctly. Due to this requirement, these [library](#) functions save the CORCON register on the stack in the beginning of the function and restore it before the function return. After saving the CORCON register, [library](#) functions write to all bits of the CORCON register. Thus, for the brief duration when these [library](#) functions are executing, the state of CORCON register may be different from its state as set by the function caller. This may temporarily change the CPU core behavior with respect to exception processing latency, DO loop termination, CPU interrupt priority level and DSP-engine behavior.

1.8 Library Interface

Controller Initialization Functions

	Name	Description
	SMPS_Controller3P3ZInitialize	This function clears the SMPS_3P3Z_T data history structure arrays
	SMPS_Controller2P2ZInitialize	This function clears the SMPS_2P2Z_T data structure arrays
	SMPS_ControllerPIDInitialize	This function clears the SMPS_PID_T data structure arrays

Types

	Name	Description
	SMPS_3P3Z_T	Data type for the 3-pole 3-zero (3P3Z) controller
	SMPS_2P2Z_T	Data type for the 2-pole 2-zero (2P2Z) controller
	SMPS_PID_T	Data type for the PID controller




Description

This section describes the functions and type defines provided by the SMPS Control [library](#).

1.8.1 Types

The [library](#) functions require input and output data to be organized in structures of specific types. The type defines, described in this section, are provided in the [smps_control.h](#) file.

Structures

	Name	Description
	SMPS_3P3Z_T	Data type for the 3-pole 3-zero (3P3Z) controller
	SMPS_2P2Z_T	Data type for the 2-pole 2-zero (2P2Z) controller
	SMPS_PID_T	Data type for the PID controller

1.8.1.1 SMPS_3P3Z_T Structure

C

```
typedef struct {
    int16_t* aCoefficients;
    int16_t* bCoefficients;
    int16_t* controlHistory;
    int16_t* errorHistory;
    uint16_t preShift;
    int16_t postShift;
    int16_t postScaler;
    uint16_t minOutput;
    uint16_t maxOutput;
} SMPS_3P3Z_T;
```

Description

Data type for the 3-pole 3-zero (3P3Z) controller

The 3P3Z controller is the digital implementation of the analog type III controller. This is a filter which generates a compensator characteristic considering three poles and three zeros. This controller requires four feedback error multiplied by their associated coefficients plus the three latest controller output values multiplied by their associated coefficients along the delay line to provide proper compensation of the power converter. The coefficients are determined externally using

simulation tools.

The SMPS_3P3Z_T data structure contains a pointer to derived coefficients in X-space and pointer to error/control history samples in Y-space. User must declare variables for the derived coefficients and the error history samples.

The abCoefficients referenced by the SMPS_3P3Z_T data structure are derived from the coefficients B0-B3 plus A1-A3. These will be declared in external arrays. The SMPS_3P3Z_T data structure just holds pointers to these arrays.

The coefficients will be determined by simulation tools, which output is given as floating point numbers. These numbers will be copied into the declared arrays after they have been converted into 16-bit integer numbers.

Members

Members	Description
int16_t* aCoefficients;	Pointer to A coefficients located in X-space
int16_t* bCoefficients;	Pointer to B coefficients located in X-space
int16_t* controlHistory;	Pointer to 3 delay-line samples located in Y-space with the first sample being the most recent
int16_t* errorHistory;	Pointer to 4 delay-line samples located in Y-space with the first sample being the most recent
uint16_t preShift;	Normalization from ADC-resolution to Q15 (R/W)
int16_t postShift;	Normalization bit-shift from Q15 to PWM register resolution (R/W)
int16_t postScaler;	Controller output post-scaler (R/W)
uint16_t minOutput;	Minimum output value used for clamping (R/W)
uint16_t maxOutput;	Maximum output value used for clamping (R/W)

1.8.1.2 SMPS_2P2Z_T Structure

C

```
typedef struct {
    int16_t* aCoefficients;
    int16_t* bCoefficients;
    int16_t* controlHistory;
    int16_t* errorHistory;
    uint16_t preShift;
    int16_t postShift;
    int16_t postScaler;
    uint16_t minOutput;
    uint16_t maxOutput;
} SMPS_2P2Z_T;
```

Description

Data type for the 2-pole 2-zero (2P2Z) controller

The 2P2Z controller is the digital implementation of the analog type II controller. This is a filter which generates a compensator characteristic considering two poles and two zeros. This controller requires three feedback error multiplied by their associated coefficients plus the two latest controller output values multiplied by their associated coefficients along the delay line to provide proper compensation of the power converter. The coefficients are determined externally using simulation tools.

The SMPS_2P2Z_T data structure contains a pointer to derived coefficients in X-space and pointer to error/control history samples in Y-space. User must declare variables for the derived coefficients and the error history samples.

The abCoefficients referenced by the SMPS_2P2Z_T data structure are derived from the coefficients B0-B2 plus A1-A2. These will be declared in external arrays. The SMPS_2P2Z_T data structure and just holds pointers to these arrays.

The coefficients will be determined by simulation tools, which output is given as floating point numbers. These numbers will be copied into the declared arrays after they have been converted into 16-bit integer numbers.

Members

Members	Description
int16_t* aCoefficients;	Pointer to A coefficients located in X-space
int16_t* bCoefficients;	Pointer to B coefficients located in X-space
int16_t* controlHistory;	Pointer to 2 delay-line samples located in Y-space with the first sample being the most recent
int16_t* errorHistory;	Pointer to 3 delay-line samples located in Y-space with the first sample being the most recent
uint16_t preShift;	Normalization from ADC-resolution to Q15 (R/W)
int16_t postShift;	Normalization bit-shift from Q15 to PWM register resolution (R/W)
int16_t postScaler;	Controller output post-scaler (R/W)
uint16_t minOutput;	Minimum output value used for clamping (R/W)
uint16_t maxOutput;	Maximum output value used for clamping (R/W)

1.8.1.3 SMPS_PID_T Structure**C**

```
typedef struct {
    int16_t* abcCoefficients;
    int16_t* errorHistory;
    int16_t controlHistory;
    int16_t postScaler;
    int16_t preShift;
    int16_t postShift;
    uint16_t minOutput;
    uint16_t maxOutput;
} SMPS_PID_T;
```

Description

Data type for the PID controller

Data type for the Proportional Integral Derivative (PID) controller

This digital implementation of a PID controller is a filter which generates a compensator characteristic considering the values of the coefficients KA, KB, KC these coefficients will determine the converter's frequency response. These coefficients are determined externally using simulation tools.

This function call includes the pointer to the controller data structure, pointer of the input source register, control reference value, and to the pointer to the output register.

Members

Members	Description
int16_t* abcCoefficients;	Pointer to A, B & C coefficients located in X-space These coefficients are derived from the PID gain values - Kp, Ki and Kd
int16_t* errorHistory;	Pointer to 3 delay-line samples located in Y-space with the first sample being the most recent
int16_t controlHistory;	Stores the most recent controller output (n-1)
int16_t postScaler;	PID basic Coefficient scaling Factor
int16_t preShift;	Normalization from ADC-resolution to Q15 (R/W)
int16_t postShift;	Normalization from DSP to PWM register
uint16_t minOutput;	Minimum output value used for clamping
uint16_t maxOutput;	Maximum output value used for clamping

1.8.2 Controller Initialization Functions

This sections lists and describes the initialization functions used in the SMPS Control Library.

Functions

	Name	Description
⇒	SMPS_Controller3P3ZInitialize	This function clears the SMPS_3P3Z_T data history structure arrays
⇒	SMPS_Controller2P2ZInitialize	This function clears the SMPS_2P2Z_T data structure arrays
⇒	SMPS_ControllerPIDInitialize	This function clears the SMPS_PID_T data structure arrays

1.8.2.1 SMPS_Controller3P3ZInitialize Function

C

```
void SMPS_Controller3P3ZInitialize(  
    SMPS_3P3Z_T * controllerData  
) ;
```

Description

This function clears the [SMPS_3P3Z_T](#) data history structure arrays. It's recommended to clear the error-history and controller-history arrays before 3P3Z controller implementation.

Preconditions

None.

Parameters

Parameters	Description
SMPS_3P3Z_T *	This parameter is a pointer to a SMPS_3P3Z_T type structure

Returns

Void.

Example

```
SMPS_3P3Z_T controller3P3Z;  
SMPS_3P3ZInitialize(&controller3P3Z);
```

1.8.2.2 SMPS_Controller2P2ZInitialize Function

C

```
void SMPS_Controller2P2ZInitialize(  
    SMPS_2P2Z_T * controllerData  
) ;
```

Description

This function clears the [SMPS_2P2Z_T](#) data history structure arrays. It's recommended to clear the error-history and controller-history arrays before 2P2Z controller implementation.

Preconditions

None.

Parameters

Parameters	Description
SMPS_2P2Z_T*	This parameter is a pointer to a SMPS_2P2Z_T type structure

Returns

void

Example

```
SMPS_2P2Z_T controller2P2Z;
SMPS_Controller2P2ZUpdateInitialize(&controller2P2Z);
```

1.8.2.3 SMPS_ControllerPIDInitialize Function

C

```
void SMPS_ControllerPIDInitialize(
    SMPS_PID_T * controllerData
);
```

Description

This function clears the [SMPS_PID_T](#) data history structure arrays. It's recommended to clear the error-history and controller-history arrays before PID controller implementation.

Preconditions

None.

Parameters

Parameters	Description
SMPS_PID_T*	This parameter is a pointer to a SMPS_PID_T type structure

Returns

void

Example

```
SMPS_PID_T controllerPID;
SMPS_ControllerPIDUpdateInitialize(&controllerPID);
```

1.8.3 Controller Functions

Functions

	Name	Description
⇒	SMPS_Controller3P3ZUpdate	This function calls the SMPS_Controller3P3ZUpdate controller
⇒	SMPS_Controller2P2ZUpdate	This function calls the SMPS_Controller2P2ZUpdate controller
⇒	SMPS_ControllerPIDUpdate	This function calls the SMPS_ControllerPIDUpdate controller

Description

This sections lists and describes the function calls for each of the controllers included in the SMPS Control Library.

1.8.3.1 SMPS_Controller3P3ZUpdate Function

C

```
void SMPS_Controller3P3ZUpdate(
    SMPS_3P3Z_T* controllerData,
    volatile uint16_t* controllerInputRegister,
    int16_t reference,
    volatile uint16_t* controllerOutputRegister
);
```

Description

This function updates the 3P3Z controller and can be called as soon as the coefficients have been loaded into their arrays.

Preconditions

Before the controller can be used, it has to be initialized. The data structure has to be filled by copying the pointers to the coefficient, error and controller history arrays to the structure and the physical clamping limits of the output value. In the function call pointers to the Input source register, reference value, and pointer to the output register need to be called.

Parameters

Parameters	Description
SMPS_3P3Z_T * controllerData	This parameter is a pointer to a SMPS_3P3Z_T type structure
uint16_t* controllerInputRegister	This parameter is a pointer to the input source register or variable being tracked by the 3P3Z (e.g. ADCBUF0).
int16_t reference	This parameter is a signed integer value that will be used by the controller as the feedback reference or set-point.
uint16_t* controllerOutputRegister	This parameter is a pointer to the Control loop target register of the calculated result(e.g. PDC1).

Returns

void

Example

```
int16_t controlReference;
SMPS_3P3Z_T controller3P3Z;
SMPS_Controller3P3ZUpdate(&controller3P3Z,&ADCBUF0,controlReference,&PDC1)
```

1.8.3.2 SMPS_Controller2P2ZUpdate Function

C

```
void SMPS_Controller2P2ZUpdate(
    SMPS_2P2Z_T* controllerData,
    volatile uint16_t* controllerInputRegister,
    int16_t reference,
    volatile uint16_t* controllerOutputRegister
);
```

Description

This function updates the 2P2Z controller and can be called as soon as the coefficients have been loaded into their arrays.

Preconditions

Before the controller can be used, it has to be initialized. The data structure has to be filled by copying the pointers to the coefficient, error and controller history arrays to the structure and the physical clamping limits of the output value. In the function call pointers to the Input source register, reference value, and pointer to the output register need to be called.

Parameters

Parameters	Description
SMPS_2P2Z_T * controllerData	This parameter is a pointer to a SMPS_2P2Z_T type structure
uint16_t* controllerInputRegister	This parameter is a pointer to the input source register or variable being tracked by the 2P2Z (e.g. ADCBUF0).
int16_t reference	This parameter is a signed integer value that will be used by the controller as the feedback reference or set-point.
uint16_t* controllerOutputRegister	This parameter is a pointer to the Control loop target register of the calculated result(e.g. PDC1).

Returns

void

Example

```
int16_t controlReference;
SMPS_2P2Z_T controller2P2Z;
SMPS_Controller2P2ZUpdate(&controller2P2Z,&ADCBUF0,controlReference,&PDC1)
```

1.8.3.3 SMPS_ControllerPIDUpdate Function**C**

```
void SMPS_ControllerPIDUpdate(
    SMPS_PID_T* controllerData,
    volatile uint16_t* controllerInputRegister,
    int16_t reference,
    volatile uint16_t* controllerOutputRegister
);
```

Description

This function updates the PID controller and can be called as soon as the coefficients have been loaded into their arrays.

Preconditions

Before the controller can be used, it has to be initialized. The data structure has to be filled by copying the pointers to the coefficient, error and controller history arrays to the structure and the physical clamping limits of the output value. In the function call pointers to the Input source register, reference value, and pointer to the output register need to be called.

Parameters

Parameters	Description
SMPS_PID_T * controllerData	This parameter is a pointer to a SMPS_PID_T type structure
uint16_t* controllerInputRegister	This parameter is a pointer to the input source register or variable being tracked by the PID (e.g. ADCBUF0).
int16_t reference	This parameter is a signed integer value that will be used by the controller as the feedback reference or set-point.
uint16_t* controllerOutputRegister	This parameter is a pointer to the Control loop target register of the calculated result(e.g. PDC1).

Returns

void

Example

```
int16_t controlReference;
SMPS_PID_T controllerPID;
SMPS_ControllerPIDUpdate(&controllerPID,&ADCBUF0,controlReference,&PDC1)
```

1.9 Files

Files







Name	Description
smps_control.h	This header file lists the interfaces used by the Switch Mode Power Supply compensator library .

1.9.1 smps_control.h




SMPS Control (Compensator) [library](#) interface header file

This header file lists the type defines for structures used by the SMPS [library](#). Library function definitions are also listed along with information regarding the arguments of each [library](#) function.

Functions

	Name	Description
	SMPS_Controller2P2ZInitialize	This function clears the SMPS_2P2Z_T data structure arrays
	SMPS_Controller2P2ZUpdate	This function calls the SMPS_Controller2P2ZUpdate controller
	SMPS_Controller3P3ZInitialize	This function clears the SMPS_3P3Z_T data history structure arrays
	SMPS_Controller3P3ZUpdate	This function calls the SMPS_Controller3P3ZUpdate controller
	SMPS_ControllerPIDInitialize	This function clears the SMPS_PID_T data structure arrays
	SMPS_ControllerPIDUpdate	This function calls the SMPS_ControllerPIDUpdate controller

Structures

	Name	Description
	SMPS_2P2Z_T	Data type for the 2-pole 2-zero (2P2Z) controller
	SMPS_3P3Z_T	Data type for the 3-pole 3-zero (3P3Z) controller
	SMPS_PID_T	Data type for the PID controller

File Name

smps_interfaces.h

Company

Microchip Technology Inc.

Index

C

Controller Functions 1-19

Controller Initialization Functions 1-18

F

Files 1-22

I

Introduction 1-2

L

Library Interface 1-15

Library Overview 1-5

Library Sections 1-5

Library Usage Model 1-5

M

Modifying the Library 1-12

P

Performance 1-13

R

Register Usage 1-14

Release Notes 1-3

S

SMPS Control Library 1-1

SMPS_2P2Z_T structure 1-16

SMPS_3P3Z_T structure 1-15

smpls_control.h 1-22

SMPS_Controller2P2ZInitialize function 1-18

SMPS_Controller2P2ZUpdate function 1-20

SMPS_Controller3P3ZInitialize function 1-18

SMPS_Controller3P3ZUpdate function 1-19

SMPS_ControllerPIDInitialize function 1-19

SMPS_ControllerPIDUpdate function 1-21

SMPS_PID_T structure 1-17

SW License Agreement 1-4

T

Types 1-15

U

Using the 2P2Z Controller 1-9

Using the 3P3Z Controller 1-7

Using the PID Controller 1-10