SMPS Control Library Help

Table of Contents

1 Structs, Records, Enums

The following table lists %CATEGORYL% in this documentation.

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
SMPS_Controller_Options_T	Optional data type for the controllers using the hardware accelerated functions.

1.1 SMPS_2P2Z_T Structure

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

File

smps_control.h

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)

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.

1.2 SMPS_3P3Z_T Structure

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

File

smps_control.h

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)

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 errors 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, their outputs are given as floating point numbers. These numbers will be copied into the declared arrays after they have been converted into 16-bit integer numbers.

1.3 SMPS_PID_T Structure

Data type for the PID controller

File

smps_control.h

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

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.

1.4 SMPS_Controller_Options_T Structure

Optional data type for the controllers using the hardware accelerated functions.

File

smps_control.h

Members

Members	Description
uint16_t triggerSelectFlag;	00 = No Trigger Enabled 01 = Trigger On-Time Enabled 10 = Trigger Off-Time Enabled
volatile unsigned int* trigger;	Pointer to trigger source
volatile unsigned int* period;	Pointer to time base (i.e., PTPER) register

Description

Optional data type that may be used with any of the controllers for changing the location of the trigger in real time. Additional structure members may be added by the user for additional options and/or necessary features.

The hardware accelerated functions come with the option of configuring the trigger value (location) in real time. The options that are included are:

1. No trigger option is selected.
2. 50 % On/Off time trigger plus some user defined delay to account for gate drive delay.

The following two equations show how these values are computed:

On-Time Trigger: TRIGx = PDCx/2 + Delay

Off-Time Trigger: TRIGx = PDCx + (PTPER - PDCx)/2 + Delay

In order to make use of this trigger option, the trigger options structure must be initialized.

Please refer to the smps_control.h file for further details.

2 Release Notes

SMPS Control Library Version : 1.0 Release Date: May 13th, 2015

New:

This new release includes the previously released software based archive files and the addition of four hardware accelerated based functions which make use of the persistent alternate working register sets. These functions also include trigger options whereby the trigger location can be updated in real time.

Software based libraries for dsPIC33 devices:

1. smps_2p2z_dspic.s
2. smps_3p3z_dspic.s
3. smps_pid_dspic.s

Hardware accelerated based libraries using persistent alternate working register sets:

1. smps_2p2z_dspic_v2.s
2. smps_3p3z_dspic_v2.s
3. smps_4p4z_dspic_v2.s
4. smps_pid_dspic_v2.s

The new Application Programming Interface (API) includes function prototypes for both library versions.

Changes:

New text has been added to reference the controller functions making use of the alternate working register set features.

Fixes:

Minor miscellaneous textual corrections where appropriate.

Known Issues:

None.

Development Tools:

This version of the library is tested to be compatible with the following compiler and IDE:

- XC16 v1.25 compiler and later revisions.
- MPLAB X IDE v2.35 and later revisions.

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.

3 SW License Agreement

4 Introduction

SMPS Control Library for

Microchip Microcontrollers

The libraries are 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 specifically 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 the majority of the time-critical operations.

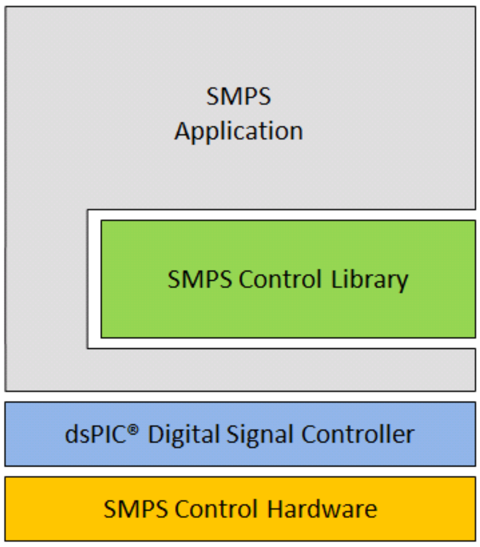


Figure-1: Block diagram of a typical library usage scenario.

5 Library Interface

This section describes both the initialization functions and and the controller update functions provided by the SMPS Control library.

5.1 Controller Update Functions

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

Functions

	Name	Description
◆	SMPS_Controller2P2ZUpdate_HW_Accel	This function calls the 2 pole 2 zero compensator using the alternate working register sets
◆	SMPS_Controller3P3ZUpdate_HW_Accel	This function calls the the 3 pole 3 zero compensator using the alternate working register sets
◆	SMPS_Controller4P4ZUpdate_HW_Accel	This function calls the 4 pole 4 zero compensator using the alternate working register sets
◆	SMPS_ControllerPIDUpdate_HW_Accel	This function calls the PID compensator using the alternate working register sets
◆	SMPS_Controller2P2ZUpdate	This function calls the 2 pole 2 zero compensator
◆	SMPS_Controller3P3ZUpdate	This function calls the 3 pole 3 zero compensator
◆	SMPS_ControllerPIDUpdate	This function calls the PID compensator

5.1.1 SMPS_Controller2P2ZUpdate_HW_Accel Function

This function calls the 2 pole 2 zero compensator using the alternate working register sets

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

```
void SMPS_Controller2P2ZUpdate_HW_Accel (void);
```

This function call executes a 2 pole 2 zero compensator and updates the control output target register

Conditions

Before the controller can be used, all of the alternate working registers have to be pre-initialized with the appropriate shifts, scalars, min/max clamps, input/output registers, and array pointers. Please refer to 'Register **Usage**' section for alternate working register assignments.

Example

```
mov _controllerControlReference, w0
call _SMPS_Controller2P2ZUpdate_HW_Accel
```

5.1.2 SMPS_Controller3P3ZUpdate_HW_Accel Function

This function calls the the 3 pole 3 zero compensator using the alternate working register sets

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

```
void SMPS_Controller3P3ZUpdate_HW_Accel (void);
```

This function call executes a 3 pole 3 zero compensator and updates the control output target register

Conditions

Before the controller can be used, all of the alternate working registers have to be pre-initialized with the appropriate shifts, scalars, min/max clamps, input/output registers, and array pointers. Please refer to 'Register **Usage**' section for alternate working register assignments.

Example

```
mov _controllerControlReference, w0
call _SMPS_Controller3P3ZUpdate_HW_Accel
```

5.1.3 SMPS_Controller4P4ZUpdate_HW_Accel Function

This function calls the 4 pole 4 zero compensator using the alternate working register sets

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

```
void SMPS_Controller4P4ZUpdate_HW_Accel (void);
```

This function call executes a 4 pole 4 zero compensator and updates the control output target register

Conditions

Before the controller can be used, all of the alternate working registers have to be pre-initialized with the appropriate shifts, scalars, min/max clamps, input/output registers, and array pointers. Please refer to 'Register **Usage**' section for alternate working register assignments.

Example

```
mov _controllerControlReference, w0
call _SMPS_Controller4P4ZUpdate_HW_Accel
```

5.1.4 SMPS_ControllerPIDUpdate_HW_Accel Function

This function calls the PID compensator using the alternate working register sets

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

```
void SMPS_ControllerPIDUpdate_HW_Accel (void);
```

This function call executes a PID compensator and updates the control output target register

Conditions

Before the controller can be used, all of the alternate working registers have to be pre-initialized with the appropriate shifts, scalars, min/max clamps, input/output registers, and array pointers. Please refer to 'Register **Usage**' section for alternate working register assignments.

Example

```
mov _controllerControlReference, w0
call _SMPS_ControllerPIDUpdate_HW_Accel
```

5.1.5 SMPS_Controller2P2ZUpdate Function

This function calls the 2 pole 2 zero compensator

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

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

This function call executes a 2 pole 2 zero compensator and updates the control output target register.

Conditions

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.

Example

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

5.1.6 SMPS_Controller3P3ZUpdate Function

This function calls the 3 pole 3 zero compensator

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

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

This function call executes a 3 pole 3 zero compensator and updates the control output target register

Conditions

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.

Example

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

5.1.7 SMPS_ControllerPIDUpdate Function

This function calls the PID compensator

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function:

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

This function call executes a PID compensator and updates the control output target register

Conditions

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.

Example

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

5.2 Controller Initialization Functions

The following are the functions that are used by the software based libraries to initialize the controllers by both clearing the error and control histories and by initializing the controller coefficients.

Functions

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

5.2.1 SMPS_Controller2P2ZInitialize Function

This function clears the SMPS_2P2Z_T data structure arrays

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function: void SMPS_2P2ZInitialize(void)

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.

Conditions

None.

Example

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

5.2.2 SMPS_Controller3P3ZInitialize Function

This function clears the SMPS_3P3Z_T data history structure arrays

File

smps_control.h

Parameters

Parameters	Description
Return	Void.

Description

Function: void SMPS_3P3ZInitialize(SMPS_3P3Z_T *controller_data)

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.

Conditions

None.

Example

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

5.2.3 SMPS_ControllerPIDInitialize Function

This function clears the SMPS_PID_T data structure arrays

File

smps_control.h

Parameters

Parameters	Description
Return	void

Description

Function: void SMPS_PIDInitialize(SMPS_PID_T *controller_data)

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.

Conditions

None.

Example

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

6 Performance

The following table tabulates the expected cycle count for both the initialization functions and the controller update functions. The controller library file include the software based functions as well as the new hardware accelerated functions. The critical paths have also been tabulated for comparison purposes.

The first column in Table 1 below lists the library functions.

The second column 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.

The third column:

4. Critical path time from the function call to the time that the control output target register is updated.

Function Name (Software Based Libraries)	**Total Instruction Cycle Count Usage (Minimum)**	**Critical Path Cycle Usage (Minimum)**
SMPS_Controller3P3ZInitialize	18	N/A
SMPS_Controller2P2ZInitialize	16	N/A
SMPS_ControllerPIDInitialize	15	N/A
SMPS_Controller3P3ZUpdate	63	52
SMPS_Controller2P2ZUpdate	58	46
SMPS_ControllerPIDUpdate	53	38
Function Name (Hardware Accelerated Based Libraries)[1]		
SMPS_Controller2P2ZUpdate_HW_Accel	53	25
SMPS_Controller3P3ZUpdate_HW_Accel	59	25
SMPS_Controller4P4ZUpdate_HW_Accel	66	25
SMPS_ControllerPIDUpdate_HW_Accel	51	31

Table 1 - Library cycle usage tabulation.

Note:

The above performance numbers were measured on the SMPS Control Library.

Note 1:

The hardware accelerated based library functions can only be used by devices that have the alternate working register set features. This is for the case where no trigger option has been selected.

Version

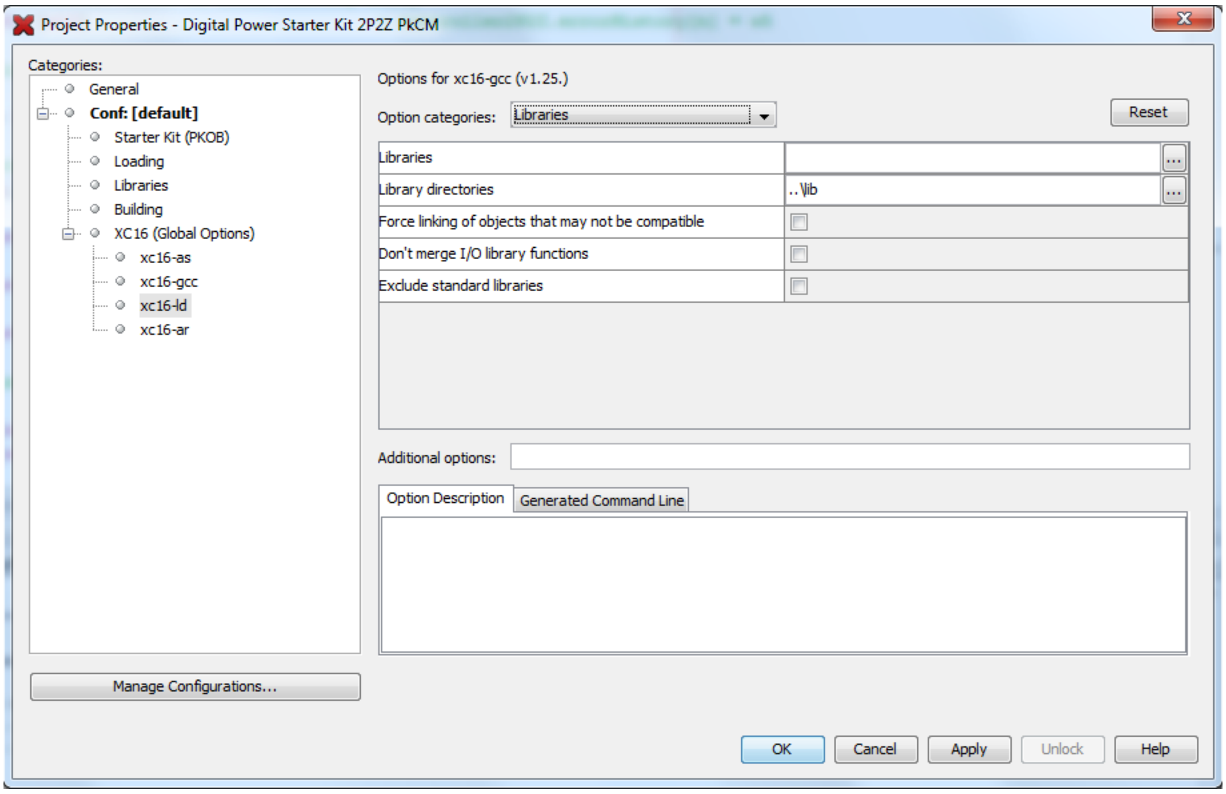
1.0 Release

7 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.



Project Properties - Digital Power Starter Kit 2P2Z PkCM

Categories:
- General
- Conf: [default]
  - Starter Kit (PKOB)
  - Loading
  - Libraries
  - Building
  - XC16 (Global Options)
    - xc16-as
    - xc16-gcc
    - xc16-ld
    - xc16-ar

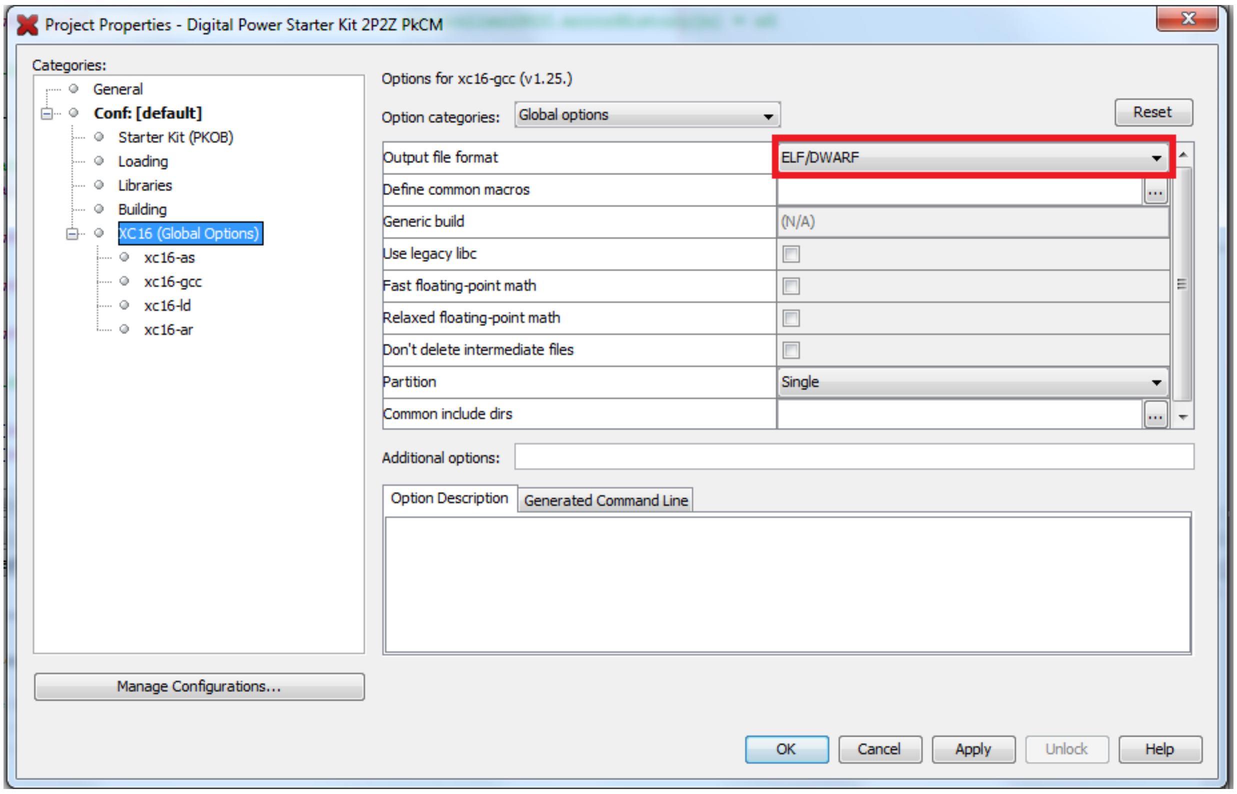
Options for xc16-gcc (v1.25.)

Option categories: Libraries

Libraries	
Library directories	..\lib
Force linking of objects that may not be compatible	☐
Don't merge I/O library functions	☐
Exclude standard libraries	☐

Additional options:

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


Project Properties - Digital Power Starter Kit 2P2Z PkCM

Categories:
- General
- Conf: [default]
  - Starter Kit (PKOB)
  - Loading
  - Libraries
  - Building
  - XC16 (Global Options)
    - xc16-as
    - xc16-gcc
    - xc16-ld
    - xc16-ar

Options for xc16-gcc (v1.25.)

Option categories: Global options

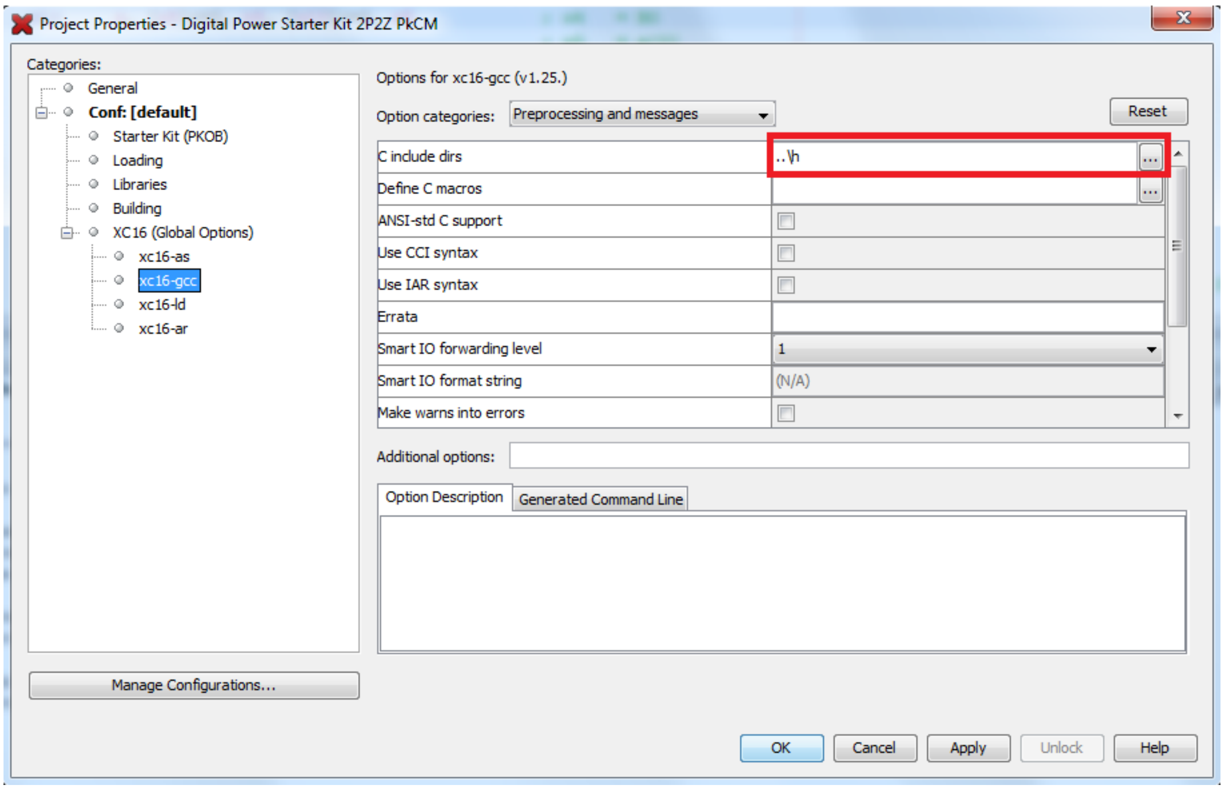
Output file format	ELF/DWARF
Define common macros	
Generic build	(N/A)
Use legacy libc	☐
Fast floating-point math	☐
Relaxed floating-point math	☐
Don't delete intermediate files	☐
Partition	Single
Common include dirs	

Additional options:

3. Include the smps_control.h file in all C language source (.c) file that use the SMPS Control library.

```
#include "smps_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.


Project Properties - Digital Power Starter Kit 2P2Z PkCM

Categories:
- General
- Conf: [default]
  - Starter Kit (PKOB)
  - Loading
  - Libraries
  - Building
  - XC16 (Global Options)
    - xc16-as
    - xc16-gcc
    - xc16-ld
    - xc16-ar

Options for xc16-gcc (v1.25.)

Option categories: Preprocessing and messages

C include dirs	..\h
Define C macros	
ANSI-std C support	☐
Use CCI syntax	☐
Use IAR syntax	☐
Errata	
Smart IO forwarding level	1
Smart IO format string	(N/A)
Make warns into errors	☐

Additional options:

8 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, also referred to as the Application Programming Interface (API).

Interface Header File: smps_control.h

The interfaces to the SMPS control libraries are defined in the "smps_control.h" header file. Any C language source (.c) files that use the SMPS Control library should include the "smps_control.h" file.

Library Files: libsmps_control_dspic33e-elf.a

This is the SMPS Control library archive (.a) file installed with the library release. The prototypes for the library functions hosted by the archive file are described in the smps_control.h file. The archive file released with the library was built using the ELF-type of Object Module Format (OMF).

Please refer to the respective assembly file for compensator type, implementation details and specific register usage.

8.1 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

9 Modifying the Library

At release, the library includes one archive file:

1. libsmps_control_dspic33e-elf.a - To be used with dsPIC33E family of devices.

The archive file released with the library is 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:

2. libsmps_control_dspic33e-elf.X - To be used with dsPIC33E and 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_dspic33e-elf.X/dist/default/production folder. If the user wishes to use this library with the dsPIC33F family of devices, the user must select the appropriate device under 'Device' in the "Project Properties' pull down menu and recompile to generate the new library archive file.

10 SMPS Control Library

Files

Name	Description
smps_control.h	This header file lists the interfaces used by the switch mode power supply compensator library.

Library Function Requirements

The controller was designed to work with Q15 numbers only. Therefore, the coefficients have to be normalized to be between -1 and +1 before implementation. The library function makes use of MAC instructions and therefore pointers must be declared to point to both X-space and Y-space. The MAC instruction and other associated instructions can concurrently fetch two data operands from memory while multiplying two working registers. Please refer to the **16-Bit MCU and DSC Programmer's Reference Manual** (**DS70157**) for further details. Because the library SW makes use of MAC instructions, two arrays need to be created to process the required operands.

11 Hardware Accelerated Function Register Assignment

When using the hardware accelerated based controller functions, the following register assignments must be followed.

When using the **2P2Z**, **3P3Z**, and **4P4Z** hardware accelerated functions:

Alternate Working Registers	Assignment
w0	Control Reference value
w1	Address of the Source Register (Input) - ADCBUFx
w2	Address of the Target Register (Output) - PDCx/CMPxDAC
w3, w4, w5	ACCAx ... and misc operands
w6	postScalar
w7	postShift
w8	Library options structure pointer
w9	ACoefficients/BCoefficients array base address
w10	ErrorHistory/ControlHistory array base address
w11	minClamp
w12	maxClamp
w13, w14	user defined, misc use

When using the **PID** hardware accelerated function:

Alternate Working Registers	Assignment
w0	Control Reference value
w1	Address of the Source Register (Input) - ADCBUFx
w2	Address of the Target Register (Output) - PDCx/CMPxDAC
w3, w4, w5	ACCAx ... and misc operands
w6	postScalar
w7	postShift
w8	Library options structure pointer
w9	ACoefficients/BCoefficients array base address
w10	ErrorHistory/ControlHistory array base address
w11	minClamp
w12	maxClamp
w13	preShift
w14	user defined, misc use

11.1 Using the Alternate Working Register Features

Alternate Working Register (Context) Set Overview

Specific dsPIC33 devices have alternate working register sets which are a subset of the default working registers (w0-w15). Please see device datasheet in question to verify if these features are implemented. These registers however only include registers w0 through w14. They are persistent registers meaning that they keep their contents when switching to another context. This is useful when wanting to keep the contents of the registers for later processing by not having to reload the previous register values. Unlike the default context 0 working registers, which are memory mapped, these registers are mapped to special function register (SFR) addresses. These special features are invaluable in time critical applications. In summary:

- Mapped to SFR memories in contrast to being memory mapped as is done with the default register arrays

- Persistent: Register contents do not change whenever the CPU switches to another register set thereby saving time by

reducing the amount of clock cycles needed for restoring register contents; useful for time critical applications

- Can be associated with a user defined Interrupt Priority Level (IPL)

Context Switching

There are two types of context switching. They are as follows:

1. Manual Context Switching

Manual context switching requires the use of the CTXTSWP instruction. The CTXTSWP instruction is referenced in the Instruction Set Overview of the device datasheet. Manual context switching consists of manually inserting code including the CTXTSWP instruction to initiate a context switch. These instructions should be included just prior to the manual swap and post manual swap. In the following example, just prior to making the compensator function call, a manual swap is performed to switch over to the alternate working register set #2. After the function call, a manual swap is performed to switch back to the default context #0.

```
void __attribute__ ((__interrupt__, no_auto_psv, context)) _ADCAN3Interrupt()
{
asm("CTXTSWP #0x2"); // Switch to context #2
asm("mov controllerControlReference, w0");
SMPS_Controller2P2ZUpdate_HW_Accel(); // Call 2P2Z controller function
asm("CTXTSWP #0x0"); // Switch back to #0 (default context)
```

2. Automatic (Hardware) Context Switching

Hardware context switching is the method by which both an interrupt and an alternate working register set are tied to the same IPL. When an interrupt occurs, the CPU checks both the IPL of the ISR and the alternate working register set. The context whose IPL matches that of the ISR is the context that is ‘switched on’ for the ISR call. Note that the IPL of all alternate working register sets must be unique. The IPL for each context is specified by the CTXTn<2:0> bits in the FALTREG configuration register, where ‘n’ is the alternate working register set's number. Each can be assigned an Interrupt

Priority Level between 1 and 7.

If using automatic context switching, both the ISR and the context have to be assigned to the same IPL. This informs the compiler that the ISR has been assigned a particular IPL and that it has been assigned to a particular alternate working register set. For example, to assign an interrupt priority level of 7 to both the alternate working register context #1 and to the ADC AN1 analog input, the following two lines of code need to be included:

```
_FALTREG(CTXT1_IPL7) // Assign a context of #1 an interrupt priority level (IPL) of 7 - usually in main.c or one of its .h file

IPC27bits.ADCAN1IP = 7; // Set ADC interrupt priority level (IPL) to 7 - normally included in initialization code

void __attribute__((__interrupt__, no_auto_psv, context)) _ADCAN3Interrupt()
```

Any time that there is an interrupt service request for the ADC AN1 input, an automatic swap is made from the current context to the context that matches the IPL of the AN1 channel input.

Note that for extremely time critical applications, it is recommended to perform context swaps by way of the automatic context switching method where both an interrupt and an alternative working register set are tied to the same IPL. This method removes the overhead of having to manually perform the context swaps in software. The preferred sequence is to have the compensator function using the alternate working register set called within an ISR written in assembly. If using C, then using inline asm instructions would be the secondary preferred method. This way the compiler does not destroy any data that should be persistent in the alternate working registers.

11.2 Register Usage

The register usage described below apply when using the hardware accelerated based library functions.

Alternate Working Register (Context) Set Overview

Specific dsPIC33 devices have alternate working register sets which are a subset of the default working registers (w0-w15). Please see device datasheet in question to verify if these features are implemented. These registers however only include registers w0 through w14. They are persistent registers meaning that they keep their contents when switching to another context. This is useful when wanting to keep the contents of the registers for later processing by not having to reload the previous register values. Unlike the default context 0 working registers, which are memory mapped, these registers are mapped to special function register (SFR) addresses. These special features are invaluable in time critical applications. In summary:

- Mapped to SFR memories in contrast to being memory mapped as is done with the default register arrays

- Persistent: Register contents do not change whenever the CPU switches to another register set thereby saving time by

reducing the amount of clock cycles needed for restoring register contents; useful for time critical applications

- Can be associated with a user defined Interrupt Priority Level (IPL)

Context Switching

There are two types of context switching. They are as follows:

1. Manual Context Switching

Manual context switching requires the use of the CTXTSWP instruction. The CTXTSWP instruction is referenced in the Instruction Set Overview of the device datasheet. Manual context switching consists of manually inserting code including the CTXTSWP instruction to initiate a context switch. These instructions should be included just prior to the manual swap and post manual swap. In the following example, just prior to making the compensator function call, a manual swap is performed to switch over to the alternate working register set #2. After the function call, a manual swap is performed to switch back to the default context #0.

```
void __attribute__ ((__interrupt__, no_auto_psv)) _ADCAN3Interrupt()

{

asm("CTXTSWP #0x2"); // Switch to context #2

asm("mov controllerControlReference, w0");

SMPS_Controller2P2ZUpdate_HW_Accel(); // Call 2P2Z controller function

asm("CTXTSWP #0x0"); // Switch back to #0 (default context)

}
```

2. <u>Automatic (Hardware) Context Switching</u>

Hardware context switching is the method by which both an interrupt and an alternate working register set are tied to the same IPL. When an interrupt occurs, the CPU checks both the IPL of the ISR and the alternate working register set. The context whose IPL matches that of the ISR is the context that is ‘switched on’ for the ISR call. Note that the IPL of all alternate working register sets must be unique. The IPL for each context is specified by the CTXTn<2:0> bits in the FALTREG configuration register, where ‘n’ is the alternate working register set's number. Each can be assigned an Interrupt Priority Level between 1 and 7.

If using automatic context switching, both the ISR and the context have to be assigned to the same IPL. This informs the compiler that the ISR has been assigned a particular IPL and that it has been assigned to a particular alternate working register set. For example, to assign an interrupt priority level of 7 to both the alternate working register context #1 and to the ADC AN1 analog input, the following two lines of code need to be included:

```
_FALTREG(CTXT1_IPL7) // Assign a context of #1 an interrupt priority level (IPL) of 7 - usually in main.c or one of its .h file

IPC27bits.ADCAN1IP = 7; // Set ADC interrupt priority level (IPL) to 7 - normally included in initialization code
```

Note that when using the automatic context switching, the 'context' label has to be added to the ISR attribute. The context attribute may be used to associate the current routine with an alternate register set. Typically this is used with interrupt service routines to reduce the amount of context that must be preserved, which will improve interrupt latency.

```
void __attribute__((__interrupt__, no_auto_psv, context)) _ADCAN3Interrupt()
```

Any time that there is an interrupt service request for the ADC AN1 input, an automatic swap is made from the current context to the context that matches the IPL of the AN1 channel input.

Note that for extremely time critical applications, it is recommended to perform context swaps by way of the automatic context switching method where both an interrupt and an alternative working register set are tied to the same IPL. This method removes the overhead of having to manually perform the context swaps in software. The preferred sequence is to have the compensator function using the alternate working register set called within an ISR written in assembly. If using C, then using inline asm instructions would be the secondary preferred method. This way the compiler does not destroy any data that should be persistent in the alternate working registers.

The register usage descriptions described below apply when using the software based libraries.

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. CPU Core Control Register (CORCON): certain library functions require the 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 at the beginning of the function and restore it before the function return (exit). After saving the CORCON register, the library function writes to all bits of the CORCON register. Thus, for the brief duration when these library functions are executing, the state of the 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.

12 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.

CONTROL LOOP DEFINITIONS

```
 Transfer Function for a Digital 2P2Z Controller
 --------------------------------------------------
|        u(z)     B0 + B1 z^(-1) + B2 z^(-2)       |
| H(z) = ---- = ----------------------------       |
|        e(z)    - 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)      |
 ---------------------------------------------------------------------
```

There are two types of controllers. A software based controller and an hardware accelerated based controller.

1. Software based library 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:

a) 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")));

b) 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

c) 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.

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.

2. Hardware accelerated based library controller

Filename Description

smps_2p2z_dspic_v2.s stands for Switch Mode Power Supply 2-pole 2-zero controller. The 'v2' suffix is added to differentiate it from the software based controller function mentioned above. The hardware accelerated controller based function can only be used with devices having the alternate working register set special features. The alternate working registers are persistent, meaning they keep their values between function calls. Because the registers are persistent, the registers do not have to be reloaded during each library function call. Currently, the devices with these features are the dsPIC33E devices. Please refer to the device datasheet for further details and to verify whether or not these features are supported with the device(s) in question. The new 2P2Z library is a derivative of the original software based library controller. It has been re-designed to minimize control output target register update latency by taking advantage of the alternate working register set features. By comparing the two functions, it is shown that the control output register update latency for the new hardware accelerated based function has been reduced by more than 50 % relative to the software based library. Please see table in the performance section.

a) Variable Declarations

Two arrays are created. One contains the values of the coefficients in X-space and the other the error and control history values in Y-space.

```
volatile int16_t controllerErrorControlHistory[4]__attribute__ ((space (ymemory), far));
volatile int16_t controllerABCoefficients[5]__attribute__ ((section (".xbss")));
```

b) Controller Initialization

Before the controller can be used, it has to be initialized. First, the coefficients array has to be initialized, second the error and control history array needs to be cleared, and third the alternate working registers have to be initialized, in no particular order.

The following instructions initialize the coefficients array (note that the coefficients are obtained via simulation tools) and are performed in a c application file:

```
controllerABCoefficients[0] = CONVERTER_COMP_2P2Z_COEFF_B0;
controllerABCoefficients[1] = CONVERTER_COMP_2P2Z_COEFF_B1;
controllerABCoefficients[2] = CONVERTER_COMP_2P2Z_COEFF_B2;

controllerABCoefficients[3] = CONVERTER_COMP_2P2Z_COEFF_A1;
controllerABCoefficients[4] = CONVERTER_COMP_2P2Z_COEFF_A2;
```

Second, the error and control history array needs to be cleared. This can be done by creating a macro and making a function call to it in the application SW.

```
#define MACRO_CLR_CONVERTERHISTORY() \
controllerErrorControlHistory[0] = 0; \
controllerErrorControlHistory[1] = 0; \
controllerErrorControlHistory[2] = 0; \
controllerErrorControlHistory[3] = 0;
```

Third, the alternate working register set has to be initialized. Please see the **Hardware Accelerated Function Register Assignment** section for further details. The CTXTSWP instruction is used to swap between the different register banks. To load a particular alternative working register context, this instruction must be called.

Included in this new library is the option by the user to modify the trigger location. A structure as been created for this purpose. The three options

that are included are:

1. No change in trigger location.

2. 50 % On-Time trigger plus a delay if applicable to account for gate drive switching.

3. 50 % Off-Time trigger plus a delay if applicable to account for gate drive switching.

The user makes this selection in the application SW. Please refer to the library for the trigger option descriptions.

The user makes this selection in the application SW. Please refer to the library for the trigger option descriptions.

c) Calling the Controller

After the requirements from steps one and two have been met, the controller can be called using the following instructions:

```
mov _controllerControlReference, w0 ; The control reference has to be pre-loaded prior to each function call
call _SMPS_Controller2P2ZUpdate_HW_Accel ; Make library function call
```

13 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 errors 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.

CONTROL LOOP DEFINITIONS

```
 Transfer function for the digital 3P3Z controller:
-----------------------------------------------------------------
|         u(z)     B0 + B1 z^(-1) + B2 z^(-2) + B3 z^(-3)         |
| H(z) = ---- = ----------------------------------------         |
|         e(z)    -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)                     |
----------------------------------------------------------------
```

There are two types of controllers. A software based controller and an hardware accelerated based controller.

1. Software based library controller.

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

Filename Description:

smps_3p3z_dspic.s stands for Switch Mode Power Supply 3-pole 3-zero compensator. The coefficients can cover a wide range of numbers depending on the performance requirements for the system or plant. This controller was created to work with Q15 numbers only. Therefore the coefficients have to be normalized between -1 and +1 before implementation.

File Usage:

a) 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")));

b) 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

c) 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.

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.

2. Hardware accelerated based library controller

Filename Description

smps_3p3z_dspic_v2.s stands for Switch Mode Power Supply 3-pole 3-zero controller. The 'v2' suffix is added to differentiate it from the software based controller function mentioned above. The hardware accelerated based controller function can only be used with devices having the alternate working register set special features. The alternate working registers are persistent, meaning they keep their values between function calls. Because the registers are persistent, the registers do not have to be reloaded during each library function call. Currently, the devices with these features are the dsPIC33E devices. Please refer to the device datasheet for further details and to verify whether or not these features are supported with the device(s) in question. The new 3P3Z library is a derivative of the original software based function controller. It has been re-designed to minimize control output target register update latency by taking advantage of the alternate working register set features. By comparing the two functions, it is shown that the control output the register update latency for the new hardware accelerated based function has been reduced by more than 50 % relative to the software based function. Please see table in the performance section.

a) Variable Declarations

Two arrays are created. One contains the values of the coefficients in X-space and the other the error and control history values in Y-space.

```
volatile int16_t controllerErrorControlHistory[6]__attribute__ ((space (ymemory), far));

volatile int16_t controllerABCoefficients[7]__attribute__ ((section (".xbss")));
```

b) Controller Initialization

Before the controller can be used, it has to be initialized. First, the coefficients array has to be initialized, second the error and control history array needs to be cleared, and third the alternate working registers have to be initialized, in no particular order.

The following instructions initialize the coefficients array (note that the coefficients are obtained via simulation tools) and are performed in a c application file:

```
controllerABCoefficients[0] = CONVERTER_COMP_3P3Z_COEFF_B0;
controllerABCoefficients[1] = CONVERTER_COMP_3P3Z_COEFF_B1;
controllerABCoefficients[2] = CONVERTER_COMP_3P3Z_COEFF_B2;
controllerABCoefficients[3] = CONVERTER_COMP_3P3Z_COEFF_B3;

controllerABCoefficients[4] = CONVERTER_COMP_3P3Z_COEFF_A1;
controllerABCoefficients[5] = CONVERTER_COMP_3P3Z_COEFF_A2;
controllerABCoefficients[6] = CONVERTER_COMP_3P3Z_COEFF_A3;
```

Second, the error and control history array needs to be cleared. This can be done by creating a macro and making a function call to it in the application SW.

```
#define MACRO_CLR_CONVERTERHISTORY() \
controllerErrorControlHistory[0] = 0; \
controllerErrorControlHistory[1] = 0; \
controllerErrorControlHistory[2] = 0; \
controllerErrorControlHistory[3] = 0; \
```

controllerErrorControlHistory[4] = 0; \

controllerErrorControlHistory[5] = 0;

Third, the alternate working register set has to be initialized. Please see the **Hardware Accelerated Function Register Assignment** section for further details. The CTXTSWP instruction is used to swap between the different register banks. To load a particular alternative working register context, this instruction must be called.

Included in this new library is the option by the user to modify the trigger location. A structure as been created for this purpose. The three options

that are included are:

1. No change in trigger location.

2. 50 % On-Time trigger plus a delay if applicable to account for gate drive switching.

3. 50 % Off-Time trigger plus a delay if applicable to account for gate drive switching.

The user makes this selection in the application SW. Please refer to the library for the trigger option descriptions.

c) Calling the Controller

After the requirements from steps one and two have been met, the controller can be called using the following instructions:

mov _controllerControlReference, w0 ; The control reference has to be pre-loaded prior to each function call

call _SMPS_Controller3P3ZUpdate_HW_Accel ; Make library function call

14 Using the 4P4Z Controller

The 4P4Z controller is a filter which generates a compensator characteristic considering four poles and four zeros. This controller requires five feedback errors multiplied by their associated coefficients plus the four 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.

CONTROL LOOP DEFINITIONS

```
 Transfer function for the digital 4P4Z controller:
----------------------------------------------------------------------
|        u(z)     B0 + B1 z^(-1) + B2 z^(-2) + B3 z^(-3) + B4 z^(-4)  |
| H(z) = ---- = -----------------------------------------------------|
|        e(z)    -A4 z^(-4) -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) + B4 e(n-4) |
|        + A1 u(n-1) + A2 u(n-2) + A3 u(n-3) + A4 u(n-4)         |
-----------------------------------------------------------------
```

There are two types of controllers. A software based controller and an hardware accelerated based controller.

1. Software based library controller.

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

Filename Description:

smps_4p4z_dspic.s stands for Switch Mode Power Supply 4-pole 4-zero compensator. The coefficients can cover a wide range of numbers depending on the performance requirements for the system or plant. This controller was created to work with Q15 numbers only. Therefore the coefficients have to be normalized between -1 and +1 before implementation.

2. Hardware accelerated based library controller

Filename Description

smps_4p4z_dspic_v2.s stands for Switch Mode Power Supply 4-pole 4-zero controller. The 'v2' suffix is added to differentiate it from the software based controller function mentioned above. The hardware accelerated controller based function can only be used with devices having the alternate working register set special features. The alternate working registers are persistent, meaning they keep their values between function calls. Because the registers are persistent, the registers do not have to be reloaded during each function call. Currently, the devices with these features are the dsPIC33E devices. Please refer to the device datasheet for further details and to verify whether or not these features are supported with the device(s) in question. The new 4P4Z library is a derivative of the 3P3Z controller function. It has been re-designed to minimize the control output target register update latency by taking advantage of the alternate working register set features.

a) Variable Declarations

Two arrays are created. One contains the values of the coefficients in X-space and the other the error and control history values in Y-space.

```
volatile int16_t controllerErrorControlHistory[8]__attribute__ ((space (ymemory), far));
volatile int16_t controllerABCoefficients[9]__attribute__ ((section (".xbss")));
```

b) Controller Initialization

Before the controller can be used, it has to be initialized. First, the coefficients array has to be initialized, second the error and control history array needs to be cleared, and third the alternate working registers have to be initialized, in no particular order.

The following instructions initialize the coefficients array (note that the coefficients are obtained via simulation tools) and are performed in a c application file:

```
controllerABCoefficients[0] = CONVERTER_COMP_3P3Z_COEFF_B0;
controllerABCoefficients[1] = CONVERTER_COMP_3P3Z_COEFF_B1;
controllerABCoefficients[2] = CONVERTER_COMP_3P3Z_COEFF_B2;
controllerABCoefficients[3] = CONVERTER_COMP_3P3Z_COEFF_B3;
controllerABCoefficients[4] = CONVERTER_COMP_3P3Z_COEFF_B4;

controllerABCoefficients[5] = CONVERTER_COMP_3P3Z_COEFF_A1;
controllerABCoefficients[6] = CONVERTER_COMP_3P3Z_COEFF_A2;
controllerABCoefficients[7] = CONVERTER_COMP_3P3Z_COEFF_A3;
controllerABCoefficients[8] = CONVERTER_COMP_3P3Z_COEFF_A4;
```

Second, the error and control history array needs to be cleared. This can be done by creating a macro and making a function call to it in the application SW.

```
#define MACRO_CLR_CONVERTERHISTORY() \
controllerErrorControlHistory[0] = 0; \
controllerErrorControlHistory[1] = 0; \
controllerErrorControlHistory[2] = 0; \
controllerErrorControlHistory[3] = 0; \
controllerErrorControlHistory[4] = 0; \
controllerErrorControlHistory[5] = 0; \
controllerErrorControlHistory[6] = 0; \
controllerErrorControlHistory[7] = 0;
```

Third, the alternate working register set has to be initialized. Please see the **Hardware Accelerated Function Register Assignment** section for further details. The CTXTSWP instruction is used to swap between the different register banks. To load a particular alternative working register context, this instruction must be called.

Included in this new library is the option by the user to modify the trigger location. A structure as been created for this purpose. The three options

that are included are:

1. No change in trigger location.

2. 50 % On-Time trigger plus a delay if applicable to account for gate drive switching.

3. 50 % Off-Time trigger plus a delay if applicable to account for gate drive switching.

The user makes this selection in the application SW. Please refer to the library for the trigger option descriptions.

c) Calling the Controller

After the requirements from steps one and two have been met, the controller can be called using the following instructions:

```
mov _controllerControlReference, w0 ; The control reference has to be pre-loaded prior to each function call
call _SMPS_Controller4P4ZUpdate_HW_Accel ; Make library function call
```

15 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.

CONTROL LOOP DEFINITIONS

```
 Transfer Function for a Digital PID Controller
 --------------------------------------------------
|        u(z)     a + b z^(-1) + c z^(-2)          |
| H(z) = ---- = ----------------------------       |
|        e(z)          1 - z^(-1)                  |
 --------------------------------------------------

 The Linear Difference Equation becomes:
 --------------------------------------------------
| u[n] = u[n-1] + e[n]*a + e[n-1]*b + e[n-2]*c     |
 --------------------------------------------------
```

There are two types of controllers. A software based controller and an hardware accelerated based controller.

1. Software based library controller.

File Usage:

a) 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")));

b) 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);

c) 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.

2. Hardware accelerated based library controller

Filename Description

smps_pid_dspic_v2.s stands for Switch Mode Power Supply PID controller. The 'v2' suffix is added to differentiate it from the software based function above. The software based function can only be used with devices having the alternate working register set special features. The alternate working registers are persistent, meaning they keep their values between function calls. Because the registers are persistent, the registers do not have to be reloaded during each library function call. Currently, the devices with these features are the dsPIC33E devices. Please refer to the device datasheet for further details and to verify whether or not the device in question has these features implemented. The new PID library is a derivative of the original PID software based function. It has been re-designed to minimize the control output target register update latency. By comparing the two functions, it is shown that the the control output register update latency for the new function has been reduced by approximately 24 % relative to the software based library.

a) Variable Declarations

Two arrays are created. One contains the values of the coefficients in X-space and the other the error and control history values in Y-space.

volatile int16_t controllerPIDCoefficients[3]__attribute__ ((section (".xbss")));

volatile int16_t controllerPIDErrorHistory[3]__attribute__ ((space (ymemory), far));

b) Controller Initialization

Before the controller can be used, it has to be initialized. First, the coefficients array has to be initialized, second the error and control history array needs to be cleared, and third the alternate working registers have to be initialized, in no particular order.

The following instructions initialize the coefficients array (note that the coefficients are obtained via simulation tools) and are performed in a c application file:

```
controllerPIDCoefficients[0] = CONVERTER_COMP_PID_COEFF_Ka;
controllerPIDCoefficients[1] = CONVERTER_COMP_PID_COEFF_Kb;
controllerPIDCoefficients[2] = CONVERTER_COMP_PID_COEFF_Kc;
```

Second, the error and control history array needs to be cleared. This can be done by creating a macro and making a function call to it in the application SW.

```
#define MACRO_CLR_BUCKHISTORY() \
controllerPIDErrorHistory[0] = 0; \
controllerPIDErrorHistory[1] = 0; \
controllerPIDErrorHistory[2] = 0;
```

Third, the alternate working register set has to be initialized. Please see the **Hardware Accelerated Function Register Assignment** section for further details. The CTXTSWP instruction is used to swap between the different register banks. To load a particular alternative working register context, this instruction must be called.

Included in this new library is the option by the user to modify the trigger location. A structure as been created for this purpose. The three options

that are included are:

1. No change in trigger location.
2. 50 % On-Time trigger plus a delay if applicable to account for gate drive switching.
3. 50 % Off-Time trigger plus a delay if applicable to account for gate drive switching.

The user makes this selection in the application SW. Please refer to the library for the trigger option descriptions.

c) Calling the Controller

After the requirements from steps one and two have been met, the controller can be called using the following instructions:

```
mov _controllerControlReference, w0 ; The control reference has to be pre-loaded prior to each function call
call _SMPS_ControllerPIDUpdate_HW_Accel ; Make library function call
```

16 Symbol Reference

16.1 Macros

The following table lists macros in this documentation.

Macros

Name	Description
_SMPS_CONTROL_H	Guards against multiple inclusion

16.1.1 _SMPS_CONTROL_H Macro

File

smps_control.h

Description

Guards against multiple inclusion

16.2 Files

The following table lists files in this documentation.

Files

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

16.2.1 smps_control.h

This header file lists the interfaces used by the Switch Mode Power Supply compensator library.

Description

Company: Microchip Technology Inc. File Name: 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 2 pole 2 zero compensator
	SMPS_Controller2P2ZUpdate_HW_Accel	This function calls the 2 pole 2 zero compensator using the alternate working register sets
	SMPS_Controller3P3ZInitialize	This function clears the SMPS_3P3Z_T data history structure arrays
	SMPS_Controller3P3ZUpdate	This function calls the 3 pole 3 zero compensator
	SMPS_Controller3P3ZUpdate_HW_Accel	This function calls the the 3 pole 3 zero compensator using the alternate working register sets
	SMPS_Controller4P4ZUpdate_HW_Accel	This function calls the 4 pole 4 zero compensator using the alternate working register sets
	SMPS_ControllerPIDInitialize	This function clears the SMPS_PID_T data structure arrays
	SMPS_ControllerPIDUpdate	This function calls the PID compensator
	SMPS_ControllerPIDUpdate_HW_Accel	This function calls the PID compensator using the alternate working register sets

Macros

Name	Description
_SMPS_CONTROL_H	Guards against multiple inclusion

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_Controller_Options_T	Optional data type for the controllers using the hardware accelerated functions.
SMPS_PID_T	Data type for the PID controller

Index