Biricha Digital Power Ltd

Chip Support
Library DPWR for C2000
Version: v2.0
Date: 07/11/2012

© Biricha Digital Power Limited
Parkway Drive
Sonning
Reading
RG4 6XG
UNITED KINGDOM

ISSUE HISTORY

Author	Changes	Version	Date
Dr Chris Hossack	First draft	V2.0	07/11/2012

REFERENCES

Id	Version	Title
1.		
2.		
3.		
4.		

Contents

1 Introduction

This document covers the Digital Power Library (DPWR). All of the C2000 devices are supported.

The following modules are supported at the moment within the CSL.

Module Name	Peripheral Description
CLA	Control Law Accelerator

dpwr

2 Installation

You need to have installed TI controlSUITE before installing this library.
Run the self-extracting zip file and the files will be extracted to C:\TI\…

There are two versions of the DPWR libraries.

- Fixed point - dpwr_ml.lib
- Floating point - dpwr_fpu.lib

2.1 File Structure

When you install the DPWR library it creates a DPWR directory as shown below

C:\TI\controlSUITE\libs\csl\ latest

- include
  - dpwr.h
  - dpwr_cla_t0_Pub.h
  - dpwr_cntrl_Pub.h
  - dpwr_filt_Pub.h
  - dpwr_pfc_Pub.h
- lib
  - dpwr_ml.lib
  - dpwr_fpu.lib
- doc
  - DPWR_C2000.pdf

3 Revision Changes

None.

4 dpwr

4.1.1 Description

This is the main header file for the dpwr library. It pulls in all of the dpwr modules' header files.

dpwr_ml.lib
dpwr_ml_cla.lib
dpwr_ml_cla.lib

5 dpwr_cla_t0_

5.1.1 Description

Contains functions to execute a digital 3 pole 3 zero (3p3z) and 2 pole 2 zero (2p2z) algorithm for use in the control of switch mode power supplies (SMPS) using the CLA.

The CLA code must be declared using CLA_3p3zVMode() or CLA_2p2zVMode() and then initialized using CLA_config().

The control functions have been optimized in Assembler for maximum speed.

```
      type      instructions  us(60Mhz)
CLA_3p3zVMode()      46        .780us
CLA_2p2zVMode()      39        .650us
```

The controllers have sufficient information in the current time step to pre-calculate some of the result for the following time step. This pre-calculation is performed after the duty has been updated. Therefore,
due to the pre-calculation, the controller can calculate the current output and then update the duty within 290ns.

This means that, when using early ADC interrupts, the ADC can be sampled
and duty updated within 940ns. However, if shadow registers are turned on
then the duty will not take effect until the next PWM period.

```
              (see above)
   880ns     -------------
-------------|           |--------------
             --->290ns PWM update (1170ns from edge to update)

             (see above)
   650ns   -------------   early ADC
-----------|           |--------------
           --->290ns PWM update (940ns from edge to update)
```

The parameters for the 3p3z algorithm must be determined through control theory analysis of the system. The poles and zeros in the analogue frequency domain can be converted to the digital domain using the tool provided on the Biricha
Digital Power website <http://www.biricha.com/resources/converter.php?type=4>

Arguments are passed to the CLA_3p3zVMode() and CLA_2p2zVMode() functions as float numbers. Macros, constants or variables cannot be used.

In the function CLA_setRef(), the argument REF is compared to the feedback value from the system under control. The CLA code reads the feedback value from the ADC and stores it within

the structure during each cycle of the control loop. The CLA code is used to update the output value based on REF
and this feedback value.

5.1.2 Examples

Initializes the 3p3z structure with the correct coefficients. When ADC_MOD_7 has completed a conversion the CLA code begins execution. This reads the
value from the ADC result register. The duty is calculated and then PWM_MOD_3 duty is updated all within the CLA code.

```
CLA_3p3zVMode( ClaTask, 7, 3,
          +1.46818, -0.314933, -0.153248,
           1.784224053, -1.629063952, -1.780916725, 1.632371281,
           0.48, 0.0, 240.0 );

void main ( void )
{
  SYS_init();
  ADC_init();

  PWM_config( PWM_MOD_3, PWM_freqToTicks(200000), PWM_COUNT_DOWN );
  PWM_pin( PWM_MOD_3, PWM_CH_A, GPIO_NON_INVERT );
  PWM_setAdcSoc( PWM_MOD_3, PWM_CH_A, PWM_INT_ZERO );

  ADC_setEarlyInterrupt( 1 );
  ADC_config( ADC_MOD_1, ADC_SH_WIDTH_7, ADC_CH_A0, ADC_TRIG_EPWM3_SOCA );
  ADC_setCallback( ADC_MOD_1, 0, ADC_INT_7 );

  CLA_setRef( CLA_getCtrlPtr(ClaTask), 2048 );
  CLA_config( CLA_MOD_7, &ClaTask, CLA_INT_ADC );
  CLA_setCallback( CLA_MOD_7, IsrFunc );

  INT_enableGlobal( 1 );

  while( 1 )
  {}
}
```

5.1.3 Notes

At power up all of the CLA to CPU message RAM is set to zero and CLA task 8 is pre-configured for use with CLA_memSet().

5.2 Api

CLA_softStartConfig()
CLA_softStartUpdate()
CLA_softStartDirection()
CLA_setRef()
CLA_3p3zVMode()
CLA_getCtrlPtr()
CLA_2p2zVMode()
CLA_slopeCode()
CLA_2p2zIMode()
CLA_3p3zIMode()

5.2.1 CLA_softStartConfig

void CLA_softStartConfig(CLA_Ctrl* Ptr,uint32_t RampMs,uint32_t UpdatePeriodNs)

where:
Ptr -
RampMs -
UpdatePeriodNs -

5.2.1.1 Description

Configures and enables a soft start for the CLA control code.

The 'RampMs' argument is the time in milli-seconds for the reference to reach its steady state value. The period of execution for the update function, CLA_SoftStartUpdate(), is specified by the argument 'UpdatePeriodNs'.

After configuring the soft start using this function the soft start is executed by calling the update function CLA_SoftStartUpdate() at the
frequency determined by the period argument 'UpdatePeriodNs'. The update function should preferably be called from inside an idle loop.

5.2.1.2 Examples

This sets the soft start for 2 seconds with an update rate of 200kHz (T=5000ns).

```
CLA_SoftStartConfig( CLA_getCtrlPtr(Cntrl), 2000, 5000 );
```

5.2.1.3 Notes

5.2.2 CLA_softStartUpdate

void CLA_softStartUpdate(CLA_Ctrl* Ptr)

where:
Ptr -

5.2.2.1 Description

Performs an update of the CLA reference value according to the soft start parameters set using CLA_SoftStartConfig().

The reference value is updated with a value initially calculated within the function CLA_SoftStartConfig().

This function must be called at the frequency determined by the UpdatePeriodNs argument of the CLA_SoftStartConfig() function call. The update function should be called from within an idle loop.

5.2.2.2 Examples

Updates the current CLA reference with the soft ramp delta value from within the main idle loop. A delay is generated which lasts for the period specified in the configuration parameters.

```
while ( 1 )
{
   CLA_softStartUpdate( CLA_getCtrlPtr(Cntrl) );
   SYS_usDelay( 5 ); // Delay for 5000ns
}
```

5.2.3 CLA_softStartDirection

void CLA_softStartDirection(CLA_Ctrl* Ptr,int PowerUp)

where:
Ptr -
PowerUp -

5.2.3.1 Description

Converts the soft start to a soft stop or vice versa.

After a soft start has been configured the user may require a soft stop. This function will reverse the ramp value allowing for the
CLA_SoftStartUpdate() function to generate a soft stop.

The ramp value may be reversed again to generate a soft start. The soft start or soft stop is determined by the 'PowerUp' argument. If this is true the update function will generate a soft start. If this is false the update function will generate a soft stop. This parameter could be read from an input pin allowing the end user to generate a soft start or soft stop.

5.2.3.2 Examples

This configures the controller to perform a soft stop.

```
CLA_SoftStartDirection( CLA_getCtrlPtr(Cntrl), false );
```

5.2.3.3 Notes

5.2.4 CLA_setRef

void CLA_setRef(CLA_Ctrl* Ptr,uint16_t Ref)

where:
Ptr -
Ref -

5.2.4.1 Description

Sets the reference for the controller.

5.2.5 CLA_3p3zVMode

void CLA_3p3zVMode(void Name,void Adc,void Pwm,void A1,void A2,void A3,void B0,void B1,void B2,void B3,void K,void MiN,void MaX)

where:
Name -
Adc - ADC module number.
Pwm - PWM module number.
A1 -
A2 -
A3 -
B0 -
B1 -
B2 -
B3 -
K -
MiN - Minimum number of ticks that the duty can be set to.
MaX - Maximum number of ticks that the duty can be set to.

5.2.5.1 Description

This macro must be called at the top of the C file, before the main function begins.

The values passed to the function call must be literals. Constants, variables or macros cannot be used.

The function creates the CLA code for a 3p3z controller.

5.2.5.2 Examples

Creates the CLA function called ClaTask. This reads the ADC value from ADC_MOD_7 and writes the duty to PWM_MOD_3.

```
CLA_3p3zVMode( ClaTask, 7, 3,
         +1.46818, -0.314933, -0.153248,
          1.784224053, -1.629063952, -1.780916725, 1.632371281,
          0.48, 0, 240 );
```

5.2.6 CLA_getCtrlPtr

void CLA_getCtrlPtr(void Mod)

where:
Mod - Selects the CLA module.

5.2.6.1 Description

Returns a pointer to the CLA module controller structure that holds the reference value.

5.2.7 CLA_2p2zVMode

void CLA_2p2zVMode(void Name,void Adc,void Pwm,void A1,void A2,void B0,void B1,void B2,void K,void MiN,void MaX)

where:
Name -
Adc -
Pwm -
A1 -
A2 -
B0 -
B1 -
B2 -
K -
MiN - Minimum number of ticks that the duty can be set to.
MaX - Maximum number of ticks that the duty can be set to.

5.2.7.1 Description

This macro must be called at the top of the C file, before the main function begins.

The values passed to the function call must be literals. Constants, variables or macros cannot be used.

The function creates the CLA code for a 2p2z controller.

5.2.7.2 Examples

Creates the CLA function called ClaTask. This reads the ADC value from ADC_MOD_7 and writes the duty to PWM_MOD_3.

```
CLA_2p2zVMode( ClaTask, 7, 3,
         +1.46818, -0.314933,
          1.784224053, -1.629063952, -1.780916725
          0.48, 0, 240 );
```

5.2.8 CLA_slopeCode

void CLA_slopeCode(void Name,int Comp,int Pwm,float Delta,void Steps)

where:
Name - Name of CLA code.
Comp - CMP_MOD number 1..3
Pwm - PWM_MOD number 1..6
Delta - The delta added to the CMP_MOD DAC value
Steps - The number of time Delta is added to the DAC value.

5.2.8.1 Description

This macro must be called at the top of the C file, before the main function begins.

The values passed to the function call must be literals. Constants, variables or macros cannot be used.

This function creates the CLA code to adjust the CMP_MOD DAC value.

The current DAC value is adjusted by Delta every 50ns. This means that the new DAC value must be set before the CLA slope code is executed.

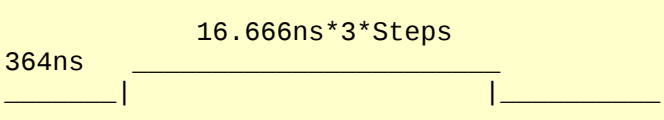
The DAC is adjusted after 364ns from the PWM interrupt.
The DAC value must be valid before 280ns after the interrupt value where it is read by the CLA code.
The CLA code also clears the PWM interrupt.

Each instruction takes 16.666ns to execute assuming a 60MHz system clock.
Three instructions are used to decrement the DAC register by the value Delta.
Therefore each decrement by Delta will occur at fixed intervals of 50ns (16.666ns*3).

The number of decrements that occur during each execution of the CLA task is determined by the argument Steps. The CLA task begins executing 280ns after the interrupt for the PWM module specified occurs.

Therefore the DAC value must be valid no greater than 280ns after the interrupt as it is read in to the CLA task code. Similarly, the CLA code must finish executing before the new DAC value is set by the control function.
Otherwise the DAC value will be overwritten by the CLA slope task value.

The designer must ensure that the CLA_slopeCode function finishes before the new DAC value is written and that the Delta value is not too large such that the DAC value wraps around from zero by the end of the number of steps.



```
-------
       ----
           ----  16.666ns*3
 DAC           ----
 value             ----
                       ---
                          ----------------
```

5.2.8.2 Examples

This creates the CLA function called SlopeTask. When PWM_MOD_3 generates an interrupt the CLA code is run where it decrements the CMP_MOD_1 DAC value by
1 every 50ns for 6 cycles.

```
CLA_slopeCode( SlopeTask, 1,3, -1.0, 6 );

//in the main code
//set up the comp and set the DAC value
CMP_config( CMP_MOD_1, CMP_SAMPLE_1, GPIO_NON_INVERT, CMP_DAC );
CMP_pin( CMP_MOD_1);
CMP_setDac( CMP_MOD_1, 100 );

//configure the CLA  to run after a PWM interrupt occurs
CLA_config( CLA_MOD_3, &SlopeTask, CLA_INT_PWM );

//configure the PWM
PWM_config( PWM_MOD_3, PWM_freqToTicks(200000), PWM_COUNT_DOWN );
PWM_pin( PWM_MOD_3, PWM_CH_A, GPIO_NON_INVERT );
PWM_setCallback(PWM_MOD_3, 0, PWM_INT_ZERO, PWM_INT_PRD_1 );
```

5.2.9 CLA_2p2zIMode

void CLA_2p2zIMode(void Name,void Adc,void Cmp,void A1,void A2,void B0,void B1,void B2,void K,void MiN,void MaX)

where:
Name -
Adc -
Cmp -
A1 -
A2 -
B0 -
B1 -
B2 -
K -
MiN - Minimum number of ticks that the duty can be set to.
MaX - Maximum number of ticks that the duty can be set to.

5.2.9.1 Description

This macro must be called at the top of the C file, before the main function begins.

The values passed to the function call must be literals. Constants, variables or macros cannot be used.

The function creates the CLA code for a 2p2z current mode controller.

5.2.9.2 Examples

Creates the CLA function called ClaTask. This reads the ADC value from ADC_MOD_7 and writes the value to CMP_MOD_3.

```
CLA_2p2zIMode( ClaTask, 7, 3,
         +1.46818, -0.314933,
          1.784224053, -1.629063952, -1.780916725
          0.48, 0, 240 );
```

5.2.10 CLA_3p3zIMode

void CLA_3p3zIMode(void Name,void Adc,void Cmp,void A1,void A2,void A3,void B0,void B1,void B2,void B3,void K,void MiN,void MaX)

where:
Name -
Adc - ADC module number.
Cmp - Comp module number.
A1 -
A2 -
A3 -
B0 -
B1 -
B2 -
B3 -
K -
MiN - Minimum number of ticks that the duty can be set to.
MaX - Maximum number of ticks that the duty can be set to.

5.2.10.1 Description

This macro must be called at the top of the C file, before the main function begins.

The values passed to the function call must be literals. Constants, variables or macros cannot be used.

The function creates the CLA code for a 3p3z current mode controller.

5.2.10.2 Examples

Creates the CLA function called ClaTask. This reads the ADC value from ADC_MOD_7 and writes the duty to CMP_MOD_3.

```
CLA_3p3zIMode( ClaTask, 7, 3,
         +1.46818, -0.314933, -0.153248,
          1.784224053, -1.629063952, -1.780916725, 1.632371281,
          0.48, 0, 240 );
```

5.3 Types

5.3.1 CLA_3p3zData

```
struct CLA_3p3zData
{
      float m_PreValue; /* +0 */
      float m_U[3];     /* +2 +4 +6 */
      float m_E[3];     /* +8 +10 +12 ram */
};
```

5.3.1.1 Description

This structure is used by the 3p3z controllers for internal values.
This structure is only readable by the CPU.

5.3.2 CLA_2p2zData

```
struct CLA_2p2zData
{
      float m_PreValue; /* +0 */
      float m_U[2];     /* +2 +4 */
      float m_E[2];     /* +6 +8 ram */
};
```

5.3.2.1 Description

This structure is used by the 2p2z controllers for internal values.
This structure is only readable by the CPU.

5.3.3 CLA_Ctrl

```
struct CLA_Ctrl
{
      long  m_Ref;      /* +0 */
      long  m_Delta;
      long  m_Max;
};
```

5.3.3.1 Description

This structure is used by both controllers to set the reference and for soft start.
This structure is readable and writeable by the CPU.

6 dpwr_cntrl_

6.1.1 Description

Contains functions to execute a digital 3 pole 3 zero (3p3z) and 2 pole 2 zero (2p2z) algorithm for use in the control of switch mode power supplies (SMPS).

The control structure must be declared and then initialized using CNTRL_3p3zInit()/CNTRL_2p2zInit() before the control function is run.

The control function has been optimized in Assembler for maximum speed. In standard C a 3p3z algorithm can take circa 170 cycles (1.7us based on a 10ns system clock).

```
CNTRL_3p3z()        71    .71us
CNTRL_3p3zInline()  53    .53us
CNTRL_2p2z()        64    .64us
CNTRL_2p2zInline()  44    .44us
```

The C wrapper contains a small time penalty when compared to pure assembly but it has the advantage that no knowledge of assembly is required.

The values for the 3p3z algorithm must be determined through control theory analysis of the system. The poles and zeros in the analogue frequency domain can be converted to the digital domain using the tool provided on the Biricha Digital Power website <http://www.biricha.com/resources/converter.php?type=4>

The arguments are passed as _iq26 numbers. The limits of these arguments are,

```
Value                Limit
A0-A2, B0-B3        -32  < value < 31.999999985
REF, MIN, MAX        0   < value < 1
```

The argument REF is the value that is compared to the feedback value from the system under control. The user code reads the feedback value from the system and stores it within the structure during each cycle of the control loop. The CNTRL_3p3z() function is used to update the output value based on REF and this feedback value.

6.1.2 Examples

Initializes the 3p3z structure with the correct coefficients. It then sets the m_IQ feedback value to the IQ value FDBK. The output value is then updated by running the control algorithm. Note that it is also possible to set the feedback value as an integer using the m_Int property of the structure.

```
CNTRL_3p3zInit(&CNTL_3P3Z_1,          // Structure
               REF                    // Ref
               A0,A1,A2               // a0,a1,a2
               B0,B1,B2,B3            // b0,b1,b2,b3
```

```
                K,MIN,MAX               // K, min, max
                );

  // Control
  CNTL_3P3Z_1.Fdbk.m_IQ = _IQ15(FDBK);   // Set feedback value
  CNTRL_3p3z(&CNTL_3P3Z_1 );             // Update
```

6.2 Api

CNTRL_2p2zFloatInline()
CNTRL_3p3zInit()
CNTRL_3p3z()
CNTRL_softStartConfig()
CNTRL_softStartUpdate()
CNTRL_softStartDirection()
CNTRL_2p2zInit()
CNTRL_2p2z()
CNTRL_2p2zSoftStartConfig()
CNTRL_2p2zSoftStartUpdate()
CNTRL_2p2zSoftStartDirection()
CNTRL_3p3zFloatInit()
CNTRL_3p3zFloat()
CNTRL_2p2zFloatInit()
CNTRL_2p2zFloat()
CNTRL_RampFloatConfig()
CNTRL_RampUpdate()
CNTRL_RampSetDirection()
CNTRL_RampSetMin()
CNTRL_RampSetMax()
CNTRL_2p2zFloatAsm()

6.2.1 CNTRL_2p2zFloatInline

void CNTRL_2p2zFloatInline(CNTRL_2p2zDataFloat* x)

where:
x -

6.2.1.1 Description

Performs the 2 pole 2 zero (2p2z) control algorithm.

This are the results of some simple tests runing at 60Mhz

```
opt=disabled  opt=0
2440ns        1190ns                inline "C" 2p2z function
1150ns        1130ns                inline asm 2p2z function
1070ns        1050ns                asm 2p2z function
```

6.2.1.2 Examples

Reads

```
// Control
CNTL_3P3Z_f.m_Ref = ADC_getValue(ADC_MOD_1,3);  // Read feedback
```

6.2.2 CNTRL_3p3zInit

void CNTRL_3p3zInit(CNTRL_3p3zData* Ptr,_iq15 Ref,_iq26 A1,_iq26 A2,_iq26 A3,_iq26 B0,_iq26 B1,_iq26 B2,_iq26 B3,_iq23 K,_iq15 Min,_iq15 Max)

where:
Ptr -
Ref -
A1 -
A2 -
A3 -
B0 -
B1 -
B2 -
B3 -
K -
Min -
Max -

6.2.2.1 Description

Initializes the 3 pole 3 zero (3p3z) structure with the required coefficients.

A structure, of type CNTRL_3p3zData, must be declared and passed as a reference to this function. This is the location where the function will
store the parameters. It will be used later on by the CNTRL_3p3z() function within the control loop.

This structure, the CNTRL_3p3zData* Ptr, must be aligned to 64 words,

```
// Structure is aligned to 64 words
#pragma DATA_ALIGN ( Cntrl3p3z , 64 );
CNTRL_3p3zData Cntrl3p3z;
```

The coefficients A1-A3 and B0-B3 are passed as _iq26 numbers. Therefore the coefficients must be within the range -32 < A < 31.999999985. Where A is the coefficient.

The argument REF is the value that is compared to the feedback value within the structure. This feedback value will most likely come from the ADC, which returns a value between 0 and 0.1249694824 (i.e. a 12 bit value 0xFFF stored as a _iq15).
REF is also stored as a _iq15 value. Therefore it is recommended that the REF argument is set with the desired return value of the ADC.

For example, if a 3.5V output value is required, then using an ADC that has a range of 0 (0V) to 4095 (3.3V) and a 1/2x prescaler (a potential divider)
on the input to the ADC pin,

```
_IQ15val = ( (REFval * Prescaler) * (ADCmax / ADCmaxV)
_IQ15val = ( (3.5    * 0.5      ) * (4095   / 3.3    ) ) = 2172
```

Therefore the argument REF can be passed as 2172 or _IQ15(0.06628417969).

The control algorithm will attempt to keep the output of the ADC feedback value around 2172 (out of the 4095 range in this case).

Min and Max are also stored as _IQ15 numbers in a 16 bit value. Therefore their range is also limited to >= 0.0 and < 1.0 and follow the same principle as above.

The parameter K is the scaling factor. It is determined using the following equation,

```
K = ( 1 / Prescaler ) * ( ADCmaxV / ADCmax ) * ( PWMperiod )
  = ( 1 /    0.5    ) * (   3.3   /  4095  ) * (    500    )
  = _IQ23(0.80586)
```

Where Prescaler is the potential divider scaling factor on the input to the ADC pin. PWMperiod is the period of the PWM signal as a number of PWM
ticks. This can be obtained from the function PWM_freqToTicks(). The
value of K is an _IQ23 number between 0 and 1.

6.2.2.2 Examples

Initializes the CNTL_3P3Z_1 structure with A1..A3, B0..B3, reference, min
and max values.

```
#pragma DATA_ALIGN ( CNTL_3P3Z_1 , 64 );
CNTRL_3p3zInit(&CNTL_3P3Z_1
     ,REF                       // Ref
     ,A1,A2,A3                  // a1,a2,a3
     ,B0,B1,B2,B3               // b0,b1,b2,b3
     ,_IQ23(1.0)                // K
     ,_IQ23(0.0),_IQ23(0.9999)  // min, max
     );
```

6.2.3 CNTRL_3p3z

void CNTRL_3p3z(CNTRL_3p3zData* Ptr)

where:
Ptr - Pointer to a 3p3z control structure.

6.2.3.1 Description

Performs the 3 pole 3 zero (3p3z) control algorithm using the information stored within the 3p3z control structure that is passed as a pointer to this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_3p3zInit().

The feedback value from the system being controlled must be read and stored within the 3p3z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the Out.m_Int property.

This function is a "C" wrapper around an assembly function. This gives faster execution time without requiring any assembly knowledge.

6.2.3.2 Examples

Reads the feedback value from the ADC, which will be >=0.0 and < 1.0 and
calls the 3p3z control algorithm. The ePWM module 1 duty for channel A is updated using the output of the control algorithm.

```
// Control
CNTL_3P3Z_1.Fdbk.m_Int = ADC_getValue(ADC_MOD_1,3);  // Read feedback
CNTRL_3p3z(&CNTL_3P3Z_1 );                           // Run algorithm
PWM_setDutyA(PWM_MOD_1, CNTL_3P3Z_1.Out.m_Int );     // Set new output
```

6.2.4 CNTRL_softStartConfig

void CNTRL_softStartConfig(CNTRL_3p3zData* Ptr,uint32_t RampMs,uint32_t UpdatePeriodNs)

where:
Ptr -
RampMs -
UpdatePeriodNs -

6.2.4.1 Description

Configures and enables a soft start for the 3p3z control code.

The 'RampMs' argument is the time in milli-seconds for the reference to reach its steady state value. The period of execution for the update function, CNTRL_softStartUpdate(), is specified by the argument 'UpdatePeriodNs'.

After configuring the soft start using this function the soft start is executed by calling the update function CNTRL_softStartUpdate() at the frequency determined by the period argument 'UpdatePeriodNs'. The update function should preferably be called from inside an idle loop.

6.2.4.2 Examples

This sets the soft start for 2 seconds with an update rate of 200kHz (T=5000ns).

```
CNTRL_softStartConfig( &Cntrl3p3z, 2000, 5000 );
```

6.2.5 CNTRL_softStartUpdate

void CNTRL_softStartUpdate(CNTRL_3p3zData* Ptr)

where:
Ptr -

6.2.5.1 Description

Performs an update of the 3p3z reference value according to the soft start parameters set using CNTRL_softStartConfig().

The reference value is updated with a value initially calculated within the function CNTRL_softStartConfig().

This function must be called at the frequency determined by the period argument of the CNTRL_softStartConfig() function call. The update function should be called from within an idle loop.

6.2.5.2 Examples

Updates the current 3p3z reference with the soft ramp delta value from
within the main idle loop. A delay is generated which lasts for the period specified in the configuration parameters.

```
while ( 1 )
{
   CNTRL_softStartUpdate( &Cntrl3p3z );
   SYS_usDelay( 5 ); // Delay for 5000ns
}
```

6.2.6 CNTRL_softStartDirection

void CNTRL_softStartDirection(CNTRL_3p3zData* Ptr,int PowerUp)

where:
Ptr -
PowerUp -

6.2.6.1 Description

Converts the soft start to a soft stop or vice versa.

After a soft start has been configured the user may require a soft stop. This function will reverse the ramp value allowing for the CNTRL_softStartUpdate() function to generate a soft stop.

The ramp value may be reversed again to generate a soft start. The soft start or soft stop is determined by the 'PowerUp' argument. If this is true the update function will generate a soft start. If this is false the update function will generate a soft stop. This parameter could be read from an input pin allowing the end user to generate a soft start or soft stop.

6.2.6.2 Examples

This configures the controller to perform a soft stop.

```
CNTRL_softStartDirection( &Cntrl3p3z, false );
```

6.2.6.3 Notes

6.2.7 CNTRL_2p2zInit

void CNTRL_2p2zInit(CNTRL_2p2zData* Ptr,_iq15 Ref,_iq26 A1,_iq26 A2,_iq26 B0,_iq26 B1,_iq26 B2,_iq23 K,_iq15 Min,_iq15 Max)

where:
Ptr -
Ref -
A1 -
A2 -
B0 -
B1 -
B2 -
K -
Min -
Max -

6.2.7.1 Description

Initializes the 2 pole 2 zero (2p2z) structure with the required coefficients.

A structure, of type CNTRL_2p2zData, must be declared and passed as a reference to this function. This is the location where the function will
store the parameters. It will be used later on by the CNTRL_2p2z() function within the control loop.

This structure, the CNTRL_2p2zData* Ptr, must be aligned to 64 words,

```
// Structure is aligned to 64 words
#pragma DATA_ALIGN ( Cntrl2p2z , 64 );
CNTRL_2p2zData Cntrl2p2z;
```

The coefficients A1-A2 and B0-B3 are passed as _iq26 numbers. Therefore the coefficients must be within the range -32 < A < 31.999999985. Where A is the coefficient.

The argument REF is the value that is compared to the feedback value within the structure. This feedback value will most likely come from the ADC, which returns a value between 0 and 0.1249694824 (i.e. a 12 bit value 0xFFF stored as a _iq15).
REF is also stored as a _iq15 value. Therefore it is recommended that the REF argument is set with the desired return value of the ADC.

For example, if a 3.5V output value is required, then using an ADC that has a range of 0 (0V) to 4095 (3.3V) and a 1/2x prescaler (a potential divider)
on the input to the ADC pin,

```
_IQ15val = ( (REFval * Prescaler) * (ADCmax / ADCmaxV)
_IQ15val = ( (3.5    * 0.5      ) * (4095   / 3.3    ) ) = 2172
```

Therefore the argument REF can be passed as 2172 or _IQ15(0.06628417969).

The control algorithm will attempt to keep the output of the ADC feedback value around 2172 (out of the 4095 range in this case).

Min and Max are also stored as _IQ15 numbers in a 16 bit value. Therefore their range is also limited to >= 0.0 and < 1.0 and follow the same principle as above.

The parameter K is the scaling factor. It is determined using the following equation,

```
K = ( 1 / Prescaler ) * ( ADCmaxV / ADCmax ) * ( PWMperiod )
  = ( 1 /    0.5    ) * (   3.3   /  4095  ) * (    500    )
  = _IQ23(0.80586)
```

Where Prescaler is the potential divider scaling factor on the input to the ADC pin. PWMperiod is the period of the PWM signal as a number of PWM ticks. This can be obtained from the function PWM_freqToTicks(). The value of K is an _IQ23 number between 0 and 1.

6.2.7.2 Examples

Initializes the CNTL_2P2Z_1 structure with A1..A3, B0..B3, reference, min and max values.

```
#pragma DATA_ALIGN ( CNTL_2P2Z_1 , 64 );
CNTRL_2p2zInit(&CNTL_2P2Z_1
     ,REF                       // Ref
     ,A2,A3                     // a1,a2
     ,B0,B1,B2                  // b0,b1,b2
     ,_IQ23(1.0)                // K
     ,_IQ23(0.0),_IQ23(0.9999)  // min, max
     );
```

6.2.8 CNTRL_2p2z

void CNTRL_2p2z(CNTRL_2p2zData* Ptr)

where:
Ptr - Pointer to a 2p2z control structure.

6.2.8.1 Description

Performs the 2 pole 2 zero (2p2z) control algorithm using the information stored within the 2p2z control structure that is passed as a pointer to this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_2p2zInit().

The feedback value from the system being controlled must be read and stored within the 2p2z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the Out.m_Int property.

This function is a "C" wrapper around an assembly function. This gives faster execution time without requiring any assembly knowledge.

6.2.8.2 Examples

Reads the feedback value from the ADC, which will be >=0.0 and < 1.0 and
calls the 2p2z control algorithm. The ePWM module 1 duty for channel A is updated using the output of the control algorithm.

```
// Control
CNTL_2P2Z_1.Fdbk.m_Int = ADC_getValue(ADC_MOD_1,3);  // Read feedback
CNTRL_2p2z(&CNTL_2P2Z_1 );                           // Run algorithm
PWM_setDutyA(PWM_MOD_1, CNTL_2P2Z_1.Out.m_Int );     // Set new output
```

6.2.9 CNTRL_2p2zSoftStartConfig

void CNTRL_2p2zSoftStartConfig(CNTRL_2p2zData* Ptr,uint32_t RampMs,uint32_t UpdatePeriodNs)

where:
Ptr -
RampMs -
UpdatePeriodNs -

6.2.9.1 Description

Configures and enables a soft start for the 2p2z control code.

The 'RampMs' argument is the time in milli-seconds for the reference to reach its steady state value. The period of execution for the update function, CNTRL_2p2zSoftStartUpdate(), is specified by the argument 'UpdatePeriodNs'.

After configuring the soft start using this function the soft start is executed by calling the update function CNTRL_2p2zSoftStartUpdate() at the frequency determined by the period argument 'UpdatePeriodNs'. The update function should preferably be called from inside an idle loop.

6.2.9.2 Examples

This sets the soft start for 2 seconds with an update rate of 200kHz (T=5000ns).

```
CNTRL_2p2zSoftStartConfig( &Cntrl2p2z, 2000, 5000 );
```

6.2.10 CNTRL_2p2zSoftStartUpdate

void CNTRL_2p2zSoftStartUpdate(CNTRL_2p2zData* Ptr)

where:
Ptr -

6.2.10.1 Description

Performs an update of the 2p2z reference value according to the soft start parameters set using CNTRL_2p2zSoftStartConfig().

The reference value is updated with a value initially calculated within the function CNTRL_2p2zSoftStartConfig().

This function must be called at the frequency determined by the period argument of the CNTRL_2p2zSoftStartConfig() function call. The update function should be called from within an idle loop.

6.2.10.2 Examples

Updates the current 2p2z reference with the soft ramp delta value from within the main idle loop. A delay is generated which lasts for the period specified in the configuration parameters.

```
while ( 1 )
{
   CNTRL_2p2zSoftStartUpdate( &Cntrl2p2z );
   SYS_usDelay( 5 ); // Delay for 5000ns
}
```

6.2.11 CNTRL_2p2zSoftStartDirection

void CNTRL_2p2zSoftStartDirection(CNTRL_2p2zData* Ptr,int PowerUp)

where:
Ptr -
PowerUp -

6.2.11.1 Description

Converts the soft start to a soft stop or vice versa.

After a soft start has been configured the user may require a soft stop. This function will reverse the ramp value allowing for the CNTRL_2p2zSoftStartUpdate() function to generate a soft stop.

The ramp value may be reversed again to generate a soft start. The soft start or soft stop is determined by the 'PowerUp' argument. If this is true the update function will generate a soft start. If this is false the update function will generate a soft stop. This parameter could be read from an input pin allowing the end user to generate a soft start or soft stop.

6.2.11.2 Examples

This configures the controller to perform a soft stop.

```
CNTRL_2p2zSoftStartDirection( &Cntrl2p2z, false );
```

6.2.11.3 Notes

6.2.12 CNTRL_3p3zFloatInit

void CNTRL_3p3zFloatInit(CNTRL_3p3zDataFloat* Ptr,uint16_t Ref,float a1,float a2,float a3,float b0,float b1,float b2,float b3,float k)

where:
Ptr -
Ref -
a1 -
a2 -
a3 -
b0 -
b1 -
b2 -
b3 -
k -

6.2.12.1 Description

Initializes the 3 pole 3 zero (3p3z) structure with the required coefficients.

A structure, of type CNTRL_3p3zDataFloat, must be declared and passed as a reference to this function. This is the location where the function will
store the parameters. It will be used later on by the CNTRL_3p3zFloat() function within the control loop.

The coefficients A1-A3 and B0-B3 are passed as floats numbers.

The argument REF is stored as a uint16_t value that is compared to the feedback value within the structure. This feedback value will most likely
come from the ADC, which returns a value between 0 and 0xFFF (i.e. a 12 bit value).

REF is also stored as a uint16_t value. Therefore it is recommended that the REF argument is set with the desired return value of the ADC.

For example, if a 3.5V output value is required, then using an ADC that has a range of 0 (0V) to 4095 (3.3V) and a 1/2x prescaler (a potential divider)
on the input to the ADC pin,

```
Ref = ( (REFval * Prescaler) * (ADCmax / ADCmaxV)
Ref = ( (3.5    * 0.5      ) * (4095   / 3.3    ) ) = 2172
```

Therefore the argument REF can be passed as 2172.

The control algorithm will attempt to keep the output of the ADC feedback value around 2172 (out of the 4095 range in this case).

The m_Min and m_Max elements of the 3p3z structure are set to -FLT_MAX and +FLT_MAX respectively. These limit the output of the controller and can be changed by manually overriding these values within the structure.

The parameter K is the scaling factor. It is determined using the following equation,

```
K = ( 1 / Prescaler ) * ( ADCmaxV / ADCmax ) * ( PWMperiod )
  = ( 1 /    0.5    ) * (   3.3   /  4095  ) * (    500    )
  = 0.80586
```

Where Prescaler is the potential divider scaling factor on the input to the ADC pin. PWMperiod is the period of the PWM signal as a number of PWM ticks. This can be obtained from the function PWM_freqToTicks(). The value of K is float.

6.2.12.2 Examples

Initializes the CNTL_3P3Z_f structure with A1..A3, B0..B3, reference, min and max values.

```
CNTRL_3p3zFloatInit(&CNTL_3P3Z_f
     ,REF                        // Ref
     ,A1,A2,A3                   // a1,a2,a3
     ,B0,B1,B2,B3                // b0,b1,b2,b3
     ,1.0                        // K
     );
```

6.2.13 CNTRL_3p3zFloat

void CNTRL_3p3zFloat(CNTRL_3p3zDataFloat* Ptr)

where:
Ptr -

6.2.13.1 Description

Performs the 3 pole 3 zero (3p3z) control algorithm using the information stored within the 3p3z control structure that is passed as a pointer to this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_3p3zFloatInit().

The feedback value from the system being controlled must be read and stored within the 3p3z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the m_U[0] property.

6.2.13.2 Examples

Reads the feedback value from the ADC, which will be 0 and 0xFFF and
calls the 3p3z control algorithm. The ePWM module 1 duty for channel A is updated using the output of the control algorithm.

```
// Control
CNTL_3P3Z_f.m_Ref = ADC_getValue(ADC_MOD_1,3);  // Read feedback
CNTRL_3p3z(&CNTL_3P3Z_f );                      // Run algorithm
PWM_setDutyA(PWM_MOD_1, CNTL_3P3Z_f.Out );      // Set new output
```

6.2.14 CNTRL_2p2zFloatInit

void CNTRL_2p2zFloatInit(CNTRL_2p2zDataFloat* Ptr,uint16_t Ref,float a1,float a2,float b0,float b1,float b2,float k)

where:
Ptr -
Ref -
a1 -
a2 -
b0 -
b1 -
b2 -
k -

6.2.14.1 Description

Initializes the 2 pole 2 zero (2p2z) structure with the required coefficients.

6.2.14.2 Examples

Initializes the CNTL_2P2Z_f structure with A1..A3, B0.., reference, min and max values.

```
CNTRL_2p2zFloatInit(&CNTL_2P2Z);
```

6.2.15 CNTRL_2p2zFloat

void CNTRL_2p2zFloat(CNTRL_2p2zDataFloat* Ptr)

where:
Ptr -

6.2.15.1 Description

Performs the 2 pole 2 zero (2p2z) control algorithm using the information stored within the 2p2z control structure that is passed as a pointer to this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_2p2zFloatInit().

The feedback value from the system being controlled must be read and stored within the 2p2z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the m_U[0] property.

The m_Min and m_Max elements of the 2p2z structure are used as anti-integral windup limits. The controller output will be capped by these bounds and fed back in the form of the previous output state to prevent integral windup.

6.2.15.2 Examples

Reads the feedback value from the ADC, which will be between 0 and 0xFFF and calls the 2p2z control algorithm. The ePWM module 1 duty for channel A is updated using the output of the control algorithm.

```
// Control
CNTL_2P2Z_f.m_Ref = ADC_getValue(ADC_MOD_1,3);  // Read feedback
CNTRL_2p2z(&CNTL_2P2Z_f );                      // Run algorithm
PWM_setDutyA(PWM_MOD_1, CNTL_2P2Z_f.Out );      // Set new output
```

6.2.16 CNTRL_RampFloatConfig

void CNTRL_RampFloatConfig(CNTRL_RampFloat* Ptr,float Min,float Max,uint32_t RampMs,uint32_t UpdateFreqHz)

where:
Ptr -
Min -
Max -
RampMs -
UpdateFreqHz -

6.2.16.1 Description

Configures a ramp which can be used for soft start in other control code.

The 'RampMs' argument is the time in milli-seconds for the reference to reach its steady state value. The period of execution for the update function, CNTRL_RampUpdate(), is specified by the argument 'UpdatePeriodNs'.

After configuring the ramp using this function the ramp is executed by calling the update function CNTRL_RampUpdate() at the frequency determined by the period argument 'UpdatePeriodNs'. The update function should preferably be called from inside an idle loop.

6.2.16.2 Examples

This sets the ramp from 0 to 5.6 for 2 seconds with an update rate of 200kHz

```
CNTRL_RampFloatConfig( &Ramp, 5.6, 2000, 200000 );
```

6.2.17 CNTRL_RampUpdate

void CNTRL_RampUpdate(CNTRL_RampFloat* Ptr)

where:
Ptr -

6.2.17.1 Description

Performs an update of the ramp value according to the ramp parameters set using CNTRL_RampFloatConfig().

The reference value is updated with a value initially calculated within the function CNTRL_RampFloatConfig().

This function must be called at the frequency determined by the period argument of the CNTRL_RampFloatConfig() function call. The update function should be called from within an idle loop.

6.2.17.2 Examples

Updates the current 2p2z reference with the ramp delta value from within the main idle loop. A delay is generated which lasts for the period specified in the configuration parameters.

```
while ( 1 )
{
   CNTRL_RampUpdate( &Ramp );
   Cntrl2p2z.m_Ref = Ramp.m_Out;
   SYS_usDelay( 5 ); // Delay for 5000ns
}
```

6.2.18 CNTRL_RampSetDirection

void CNTRL_RampSetDirection(CNTRL_RampFloat* Ptr,int PowerUp)

where:
Ptr -
PowerUp -

6.2.18.1 Description

Converts the ramp to ramp up or down.

After a ramp has been configured the user may require a ramp down for soft stop. This function will reverse the ramp value allowing for the CNTRL_RampUpdate() function to generate a soft stop.

The ramp value may be reversed again to generate a soft start. The soft start or soft stop is determined by the 'PowerUp' argument. If this is true the update function will generate a soft start. If this is false the update function will generate a soft stop. This parameter could be read from an input pin allowing the end user to generate a soft start or soft stop.

6.2.18.2 Examples

This configures the controller to perform a soft stop.

```
   CNTRL_RampSetDirection( &Ramp, false );
```

6.2.18.3 Notes

6.2.19 CNTRL_RampSetMin

void CNTRL_RampSetMin(CNTRL_RampFloat* Ptr)

where:
Ptr -

6.2.19.1 Description

6.2.20 CNTRL_RampSetMax

void CNTRL_RampSetMax(CNTRL_RampFloat* Ptr)

where:
Ptr -

6.2.20.1 Description

6.2.21 CNTRL_2p2zFloatAsm

void CNTRL_2p2zFloatAsm(CNTRL_2p2zDataFloat* Ptr)

where:
Ptr -

6.2.21.1 Description

6.3 Types

6.3.1 CNTRL_ARG

```
union CNTRL_ARG
{
      _iq15 m_IQ;
      int   m_Int;
};
```

6.3.1.1 Description

Allows a int to be written directly in to a _iq variable.

6.3.2 CNTRL_3p3zData

```
struct CNTRL_3p3zData
{
      CNTRL_ARG   Ref;  /* +0 This is a range of +1 */
      CNTRL_ARG   Fdbk; /* +2 This is a range of +1 */
      CNTRL_ARG   Out;  /* +4 This is a range of +1 */
      long  temp; /* +6 */
      _iq24 m_U1; /* +8 */
      _iq24 m_U2; /* +10 */
      _iq24 m_U3; /* +12 */
      _iq31 m_E0; /* +14 */
      _iq31 m_E1; /* +16 */
      _iq31 m_E2; /* +18 */
      _iq31 m_E3; /* +20 */
      _iq26 m_B3; /* +22 */
      _iq26 m_B2; /* +24 */
      _iq26 m_B1; /* +26 */
      _iq26 m_B0; /* +28 */
      _iq26 m_A3; /* +30 */
      _iq26 m_A2; /* +32 */
      _iq26 m_A1; /* +34 */
      _iq23 m_K;  /* +36 */
      _iq15 m_max;      /* +38 */
      _iq15 m_min;      /* +40 */
      int   m_PeriodCount;
      long  m_SoftRamp;
      long  m_SoftRef;
      long  m_SoftMax;
};
```

6.3.2.1 Description

This is the 3 pole 3 zero control structure.

6.3.3 CNTRL_inlineContextSave

```
#if 1
#define CNTRL_inlineContextSave() \
asm("        PUSH    XAR7"\
    "\t\n    PUSH    XT"\
    "\t\n    PUSH    ACC"\
    )
#endif
```

6.3.3.1 Description

Stores the registers used by the 3p2z/2p2z inline function.

6.3.4 CNTRL_inlineContextRestore

```
#if 1
#define CNTRL_inlineContextRestore() \
asm("        POP    ACC"\
    "\t\n    POP    XT"\
    "\t\n    POP    XAR7"\
    )
#endif
```

6.3.4.1 Description

Restores the registers used by the 3p2z/2p2z inline function.

6.3.5 CNTRL_3p3zInline

```
#if 1
#define CNTRL_3p3zInline(x) \
asm("        MOVW    DP, #_"#x"+0        ;CNTRL_3p3z"\
    "\t\n    MOVL    XAR7,#_"#x"+22      ;(COEFF) Local coefficient pointer
(XAR7)"\
\
    "\t\n    SETC    SXM,OVM"\
    "\t\n    MOV     ACC,@0              ;(Ref)Q15"\
    "\t\n    SUB     ACC,@2              ;(Fdbk)Q15"\
    "\t\n    LSL     ACC,#16             ;Q31"\
\
    "\t\n    ; Diff equation"\
    "\t\n    MOVL    @8+6,ACC            ;(DBUFF+6)"\
    "\t\n    MOVL    XT,@8+12            ;(DBUFF+12) XT=e(n-3),Q31"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; b3*e(n-3),Q26*Q31(64-bit result)"\
\
    "\t\n    MOVDL   XT,@8+10            ;(DBUFF+10) XT=e(n-2), e(n-3)=e(n-2)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b3*e(n-3)+b2*e(n-2) P=b1*e(n-
1),Q26*Q31(64-bit result)"\
    "\t\n    ADDL    ACC,P               ; 64-bit result in Q57, So ACC is in
Q25"\
\
    "\t\n    MOVDL   XT,@8+8             ;(DBUFF+10) XT=e(n-1), e(n-2)=e(n-1)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b2*e(n-2) P=b1*e(n-1),Q26*Q31(64-
bit result)"\
    "\t\n    ADDL    ACC,P               ; 64-bit result in Q57, So ACC is in
Q25"\
\
    "\t\n    MOVDL   XT,@8+6             ;(DBUFF+6) XT=e(n), e(n-1)=e(n)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b3*e(n-3)+b2*e(n-2)+b1*e(n-1),
P=b0*e(n),Q26*Q31(64-bit result)"\
    "\t\n                                ; 64-bit result in Q57, So ACC is in
Q25"\
    "\t\n    ADDL    ACC,P               ; ACC=b3*e(n-3)+b2*e(n-2)+b1*e(n-
1)+b0*e(n), Q25"\
    "\t\n    SFR     ACC,#1"\
    "\t\n    MOVL    @6,ACC              ;(temp) Q24"\
\
    "\t\n    MOVL    XT,@8+4             ;(DBUFF+4) XT=u(n-3),Q24"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; P=a3*u(n-3), Q26*Q24(64-bit result)"\
\
```

```
    "\t\n    MOVDL   XT,@8+2             ;(DBUFF+2) XT=u(n-2), u(n-3)=u(n-
2),Q24"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; ACC=a2*u(n-2)"\
    "\t\n                                ; 64-bit result in Q50, So ACC is in
Q18"\
    "\t\n    ADDL    ACC,P               ; ACC=a1*u(n-1)+a2*u(n-2)+a3*u(n-3),ACC
in Q18"\
\
    "\t\n    MOVDL   XT,@8+0             ;(DBUFF+0) XT=u(n-1), u(n-2)=u(n-
1),Q24"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; P=a2*u(n-2)"\
    "\t\n                                ; 64-bit result in Q50, So ACC is in
Q18"\
    "\t\n    ADDL    ACC,P               ; ACC=a1*u(n-1)+a2*u(n-2)+a3*u(n-3),ACC
in Q18"\
\
    "\t\n    LSL     ACC,#5              ; Q23"\
    "\t\n    ADDL    ACC,ACC             ; Q24"\
    "\t\n    ADDL    ACC,@6              ;(temp) Q24,ACC=a1*u(n-1)+a2*u(n-
2)+b2*e(n-2)+b1*e(n-1)+b0*e(n)"\
    "\t\n    MOVL    @8+0,ACC            ; (DBUFF+0) ACC=u(n)(Q24)"\
\
    "\t\n    MOVL    XT,ACC              ; XT = ACC iq24"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; ACC = XT * K(23) >> 32 => iq15"\
\
    "\t\n    MINL    ACC,*XAR7++         ; Saturate the result [0,1]"\
    "\t\n    MAXL    ACC,*XAR7++"\
\
    "\t\n    MOV     @4, AL ;(Out)")

/*end of code macro*/
#endif
```

6.3.5.1 Description

Performs the 3 pole 3 zero (3p3z) control algorithm using the information stored within the 2p2z control structure that is passed as a structure to
this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_3p3zInit().

The feedback value from the system being controlled must be read and stored within the 3p3z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the Out.m_Int property.

This function is inline assembly code and it is the responsibility of the user to make sure the "C" context is saved between calls.

There are CNTRL_inlineContextSave() and CNTRL_inlineContextRestore() which saves/restores the required context.

6.3.6 CNTRL_2p2zData

```
struct CNTRL_2p2zData
{
      CNTRL_ARG   Ref;  /* +0 This is a range of +1 */
      CNTRL_ARG   Fdbk; /* +2 This is a range of +1 */
      CNTRL_ARG   Out;  /* +4 This is a range of +1 */
      long  temp; /* +6 */
      _iq24 m_U1; /* +8 */
      _iq24 m_U2; /* +10 */
      _iq31 m_E0; /* +12 */
      _iq31 m_E1; /* +14 */
      _iq31 m_E2; /* +16 */
      _iq26 m_B2; /* +18 */
      _iq26 m_B1; /* +20 */
      _iq26 m_B0; /* +22 */
      _iq26 m_A2; /* +24 */
      _iq26 m_A1; /* +26 */
      _iq23 m_K;  /* +28 */
      _iq15 m_max;      /* +30 */
      _iq15 m_min;      /* +32 */
      int   m_PeriodCount;
      long  m_SoftRamp;
      long  m_SoftRef;
      long  m_SoftMax;
};
```

6.3.6.1 Description

This is the 2 pole 2 zero control structure.

6.3.7 CNTRL_3p3zDataFloat

```
struct CNTRL_3p3zDataFloat
{
      uint16_t    m_Ref;
      uint16_t    m_Fdbk;
      float m_A1;
      float m_A2;
      float m_A3;
      float m_B0;
      float m_B1;
      float m_B2;
      float m_B3;
      float m_E[4];
      float m_U[4];
      float m_K;
      uint16_t    m_Out;
      float m_Min;
      float m_Max;
};
```

6.3.7.1 Description

6.3.8 CNTRL_2p2zDataFloat

```
struct CNTRL_2p2zDataFloat
{
      float m_Ref;      /* 0 */
      float m_Fdbk;     /* 2 */
```

```
      float m_Out;      /* 4 */
      float m_B2; /* 6 */
      float m_B1; /* 8 */
      float m_B0; /* 10 */
      float m_A2; /* 12 */
      float m_A1; /* 14 */
      float m_Max;      /* 16 */
      float m_Min;      /* 18 */
      float m_U[2];     /* 20, 22 */
      float m_E[3];     /* 24 */
};
```

6.3.8.1 Description

6.3.9 CNTRL_RampFloat

```
struct CNTRL_RampFloat
{
      float m_Max;      /* max value */
      float m_Min;      /* Min value */
      float m_Out;      /* current value */
      float m_Step;     /* step direction, pos=ramp up, neg= ramp down */
};
```

6.3.9.1 Description

6.3.10 CNTRL_2p2zInline

```
#if 1
#define CNTRL_2p2zInline(x) \
asm("        MOVW    DP, #_"#x"+0        ;CNTRL_2p2z"\
    "\t\n    MOVL    XAR7,#_"#x"+18      ;(COEFF) Local coefficient pointer
(XAR7)"\
\
    "\t\n    SETC    SXM,OVM"\
    "\t\n    MOV     ACC,@0              ;(Ref)Q15"\
    "\t\n    SUB     ACC,@2              ;(Fdbk)Q15"\
    "\t\n    LSL     ACC,#16             ;Q31"\
\
    "\t\n    ; Diff equation"\
    "\t\n    MOVL    @8+4,ACC            ;(DBUFF+4)"\
    "\t\n    MOVL    XT,@8+8             ;(DBUFF+8) XT=e(n-2),Q31"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; b2*e(n-2),Q26*Q31(64-bit result)"\
\
    "\t\n    MOVDL   XT,@8+6             ;(DBUFF+6) XT=e(n-1), e(n-2)=e(n-1)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b3*e(n-3)+b2*e(n-2) P=b1*e(n-
1),Q26*Q31(64-bit result)"\
    "\t\n    ADDL    ACC,P               ; 64-bit result in Q57, So ACC is in
Q25"\
\
    "\t\n    MOVDL   XT,@8+4             ;(DBUFF+10) XT=e(n-0), e(n-1)=e(n-0)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b2*e(n-2) P=b1*e(n-1),Q26*Q31(64-
bit result)"\
\
    "\t\n    ADDL    ACC,P               ; ACC=b2*e(n-2)+b1*e(n-1)+b0*e(n-0),
Q25"\
    "\t\n    SFR     ACC,#1"\
    "\t\n    MOVL    @6,ACC              ;(temp) Q24"\
\
```

```
    "\t\n    MOVL    XT,@8+2             ;(DBUFF+2) XT=u(n-2),Q24"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; ACC=a2*u(n-2), Q26*Q24(64-bit
result)"\
\
    "\t\n    MOVDL   XT,@8+0             ;(DBUFF+0) XT=u(n-1), u(n-2)=u(n-
1),Q24"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; P=a1*u(n-1)"\
    "\t\n                                ; 64-bit result in Q50, So ACC is in
Q18"\
    "\t\n    ADDL    ACC,P               ; ACC=a1*u(n-1)+a2*u(n-2),ACC in Q18"\
\
    "\t\n    LSL     ACC,#5              ; Q23"\
    "\t\n    ADDL    ACC,ACC             ; Q24"\
    "\t\n    ADDL    ACC,@6              ;(temp) Q24,ACC=a1*u(n-1)+a2*u(n-
2)+b2*e(n-2)+b1*e(n-1)+b0*e(n)"\
    "\t\n    MOVL    @8+0,ACC            ; (DBUFF+0) ACC=u(n)(Q24)"\
\
    "\t\n    MOVL    XT,ACC              ; XT = ACC iq24"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; ACC = XT * K(23) >> 32 => iq15"\
\
    "\t\n    MINL    ACC,*XAR7++         ; Saturate the result [0,1]"\
    "\t\n    MAXL    ACC,*XAR7++"\
\
    "\t\n    MOV     @4, AL ;(Out)")

/*end of code macro*/
#endif
```

6.3.10.1 Description

Performs the 2 pole 2 zero (2p2z) control algorithm using the information stored within the 2p2z control structure that is passed as a structure to
this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_2p2zInit().

The feedback value from the system being controlled must be read and stored within the 2p2z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the Out.m_Int property.

This function is inline assembly code and it is the responsibility of the user to make sure the "C" context is saved between calls.

There are CNTRL_inlineContextSave() and CNTRL_inlineContextRestore() which saves/restores the required context.

6.3.11 CNTRL_2p2zLimitInline

```
#if 1
#define CNTRL_2p2zLimitInline(x) \
asm("       MOVW    DP, #_"#x"+0        ;CNTRL_2p2z"\
    "\t\n    MOVL    XAR7,#_"#x"+18      ;(COEFF) Local coefficient pointer
(XAR7)"\
```

```
\
    "\t\n    SETC    SXM,OVM"\
    "\t\n    MOV     ACC,@0              ;(Ref)Q15"\
    "\t\n    SUB     ACC,@2              ;(Fdbk)Q15"\
    "\t\n    LSL     ACC,#6              ;Q21 <>"\
\
    "\t\n    ; Diff equation"\
    "\t\n    MOVL    @8+4,ACC            ;(DBUFF+4)"\
    "\t\n    MOVL    XT,@8+8             ;(DBUFF+8) XT=e(n-2),Q21"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; b2*e(n-2),Q26*Q21(64-bit result)"\
\
    "\t\n    MOVDL   XT,@8+6             ;(DBUFF+6) XT=e(n-1), e(n-2)=e(n-1)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b3*e(n-3)+b2*e(n-2) P=b1*e(n-
1),Q26*Q21(64-bit result)"\
    "\t\n    ADDL    ACC,P               ; 64-bit result in Q47, So ACC is in
Q15"\
\
    "\t\n    MOVDL   XT,@8+4             ;(DBUFF+10) XT=e(n-0), e(n-1)=e(n-0)"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; ACC=b2*e(n-2) P=b1*e(n-1),Q26*Q21(64-
bit result)"\
\
    "\t\n    ADDL    ACC,P               ; ACC=b2*e(n-2)+b1*e(n-1)+b0*e(n-0),
Q15"\
    "\t\n    NOP     "\
    "\t\n    MOVL    @6,ACC              ;(temp) Q15"\
\
    "\t\n    MOVL    XT,@8+2             ;(DBUFF+2) XT=u(n-2),Q15"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; ACC=a2*u(n-2), Q26*Q15(64-bit
result)"\
\
    "\t\n    MOVDL   XT,@8+0             ;(DBUFF+0) XT=u(n-1), u(n-2)=u(n-
1),Q15"\
    "\t\n    QMPYL   P,XT,*XAR7++        ; P=a1*u(n-1)"\
    "\t\n                                ; 64-bit result in Q41, So ACC is in
Q9"\
    "\t\n    ADDL    ACC,P               ; ACC=a1*u(n-1)+a2*u(n-2),ACC in Q9"\
\
    "\t\n    LSL     ACC,#5              ; Q14"\
    "\t\n    ADDL    ACC,ACC             ; Q15"\
    "\t\n    ADDL    ACC,@6              ;(temp) Q15,ACC=a1*u(n-1)+a2*u(n-
2)+b2*e(n-2)+b1*e(n-1)+b0*e(n)"\
    "\t\n    MOVL    @8+0,ACC            ; (DBUFF+0) ACC=u(n)(Q15)"\
\
    "\t\n    MINL    ACC,@30             ; Saturate the result"\
    "\t\n    MAXL    ACC,@32"\
\
    "\t\n    MOVL    XT,ACC              ; XT = ACC iq15"\
    "\t\n    QMPYL   ACC,XT,*XAR7++      ; ACC = XT * K(30) >> 32 => iq13"\
    "\t\n    LSL     ACC,#2              ; Q15"\
\
    "\t\n    MOV     @4, AL ;(Out)")

/*end of code macro*/
#endif
```

6.3.11.1 Description

Performs the 2 pole 2 zero (2p2z) control algorithm using the information stored within the 2p2z control structure that is passed as a structure to
this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_2p2zInit().

The feedback value from the system being controlled must be read and stored within the 2p2z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the Out.m_Int property.

This function is inline assembly code and it is the responsibility of the user to make sure the "C" context is saved between calls.

There are CNTRL_inlineContextSave() and CNTRL_inlineContextRestore() which saves/restores the required context.

7 dpwr_filt_

7.1.1 Description

Contains a description

7.1.2 Examples

Initializes.

7.2 Api

FILT_2p2zFloatInit()
FILT_3p3zFloatInit()
FILT_2p2zFloatInline()
FILT_3p3zFloatInline()

7.2.1 FILT_2p2zFloatInit

void FILT_2p2zFloatInit(FILT_2p2zDataFloat* Ptr,float a1,float a2,float b0,float b1,float b2)

where:
Ptr -
a1 -
a2 -
b0 -
b1 -
b2 -

7.2.1.1 Description

7.2.2 FILT_3p3zFloatInit

void FILT_3p3zFloatInit(FILT_3p3zDataFloat* Ptr,uint16_t Ref,float a1,float a2,float a3,float b0,float b1,float b2,float b3)

where:
Ptr -
Ref -
a1 -
a2 -
a3 -
b0 -
b1 -
b2 -
b3 -

7.2.2.1 Description

Initializes the 3 pole 3 zero (3p3z) structure with the required coefficients.

7.2.2.2 Examples

Initializes the CNTL_3P3Z_f structure with A1..A3, B0..B3, reference, min and max values.

```
FILT_3p3zFloatInit(&CNTL_3P3Z_f
     ,REF                        // Ref
     ,A1,A2,A3                   // a1,a2,a3
     ,B0,B1,B2,B3                // b0,b1,b2,b3
     );
```

7.2.3 FILT_2p2zFloatInline

void FILT_2p2zFloatInline(FILT_2p2zDataFloat* Ptr)

where:
Ptr -

7.2.3.1 Description

Performs the 2 pole 2 zero (2p2z) control algorithm using the information stored within the 2p2z control structure that is passed as a structure to
this function.

The structure should already have been declared and populated with coefficients using the function CNTRL_2p2zInit().

The feedback value from the system being controlled must be read and stored within the 2p2z structure before this function is called.

The result of the control algorithm is also stored within the structure
in the Out.m_Int property.

This function is inline assembly code and it is the responsibility of the user to make sure the "C" context is saved between calls.

There are CNTRL_inlineContextSave() and CNTRL_inlineContextRestore() which saves/restores the required context.

7.2.3.2 Examples

Reads the feedback value from the ADC, which will be >=0.0 and < 1.0 and
calls the 2p2z control algorithm. The ePWM module 1 duty for channel A is updated using the output of the control algorithm.

```
// Control
CNTL_2P2Z_1.Fdbk.m_Int = ADC_getValue(ADC_MOD_1,3);  // Read feedback
CNTRL_inlineContextSave();
CNTRL_2p2zInline(CNTL_2P2Z_1 );                      // Run algorithm
CNTRL_inlineContextRestore();
PWM_setDutyA(PWM_MOD_1, CNTL_2P2Z_1.Out.m_Int );     // Set new output
```

7.2.4 FILT_3p3zFloatInline

void FILT_3p3zFloatInline(FILT_3p3zDataFloat* Ptr)

where:
Ptr -

7.2.4.1 Description

Performs the 3 pole 3 zero (3p3z) control algorithm using the information stored within the 3p3z control structure that is passed as a pointer to this function.

7.2.4.2 Examples

Reads the feedback value from the ADC, which will be 0 and 0xFFF and
calls the 3p3z control algorithm. The ePWM module 1 duty for channel A is updated using the output of the control algorithm.

```
// Control
CNTL_3P3Z_f.m_Ref = ADC_getValue(ADC_MOD_1,3);  // Read feedback
FILT_3p3z(&CNTL_3P3Z_f );                      // Run algorithm
PWM_setDutyA(PWM_MOD_1, CNTL_3P3Z_f.Out );      // Set new output
```

7.3 Types

7.3.1 FILT_3p3zDataFloat

```
struct FILT_3p3zDataFloat
{
	float m_In;
	float m_A1;
	float m_A2;
	float m_A3;
	float m_B0;
	float m_B1;
	float m_B2;
	float m_B3;
	float m_E[4];
	float m_U[4];
	float m_Out;
};
```

7.3.1.1 Description

7.3.2 FILT_2p2zDataFloat

```
struct FILT_2p2zDataFloat
{
	float m_In;
	float m_A1;
	float m_A2;
	float m_B0;
	float m_B1;
	float m_B2;
	float m_E[3];
	float m_U[3];
	float m_Out;
};
```

7.3.2.1 Description

8 dpwr_pfc_

8.1.1 Description

Contains functions to implement digital power factor correction.

This library contains PFC functions named PFCXX_ where XX describes the type of PFC converter the function has been written for:

```
01 = CCM Boost
```

More will be added in later versions

8.1.2 Examples

See c2806x/PFCproject_returnsense for a full example of a Digital CCM Boost project.

8.1.3 Links

file:///C:/TI/controlSUITE/libs/csl/latest/doc/CSL_C280x.pdf

8.2 Api

PFC01_configFloat()
PFC01_iLoopFloatInit()
PFC01_vffFloatInit()
PFC01_vLoopFloatInit()
PFC01_softStartDirectionFloat()
PFC01_vLoopFloat()
PFC01_iLoopFloat()
PFC01_getImoutFloat()
PFC01_enablePhaseFloat()

8.2.1 PFC01_configFloat

void PFC01_configFloat(PFC_DataFloat* Ptr,uint32_t FastLoopHz,uint32_t SlowLoopHz,uint32_t RampMs)

where:
Ptr - A structure, of type PFC_DataFloat, must be declared and passed as a reference to this function. This is the location where the function will store the parameters.
FastLoopHz - The frequency of the fast loop, current loop, in Hz.
SlowLoopHz - The frequency of the slow loop, voltage loop, in Hz.
RampMs -

8.2.1.1 Description

Configures the master PFC structure with the parameters passed as arguments

This function must be called before any of the loops are initialized. This function is responsible for setting the timing information for the fast and slow loops.

Stores the frequency of the current loop ISR, the execution frequency of the slow voltage loop and the desired soft start period.

8.2.1.2 Examples

```
#define SWITCHING_FREQ_HZ 200000
#define VOLTAGE_LOOP_HZ 6000
#define SOFT_START_MS 1000

PFC01_configFloat( &Pfc,
                   SWITCHING_FREQ_HZ, VOLTAGE_LOOP_HZ,
                   SOFT_START_MS );
```

8.2.2 PFC01_iLoopFloatInit

void PFC01_iLoopFloatInit(PFC_DataFloat* Ptr,CNTRL_2p2zDataFloat* loopData,float a1,float a2,float b0,float b1,float b2,float KIin,float MaxDuty)

where:
Ptr - A structure, of type CNTRL_2p2zDataFloat, must be declared and passed as a reference to this function along with the master PFC structure. The CNTRL_2p2zDataFloat structure will store the controller parameters.
loopData -
a1 -
a2 -
b0 -
b1 -
b2 -
KIin - Gain term is used to scale the output of the controller.
MaxDuty - Sets the maximum duty as a floating point number between 0.0 and 1.0.

8.2.2.1 Description

Initializes the 2 pole 2 zero (2p2z) current loop structure with the required coefficients.

The structure, CNTRL_2p2zDataFloat, must be aligned to 64 words,

```
// Structure is aligned to 64 words
#pragma DATA_ALIGN ( iLoopPhase0 , 64 );
CNTRL_2p2zDataFloat Pfc;
```

a1, a2, b0, b1, b2 - Controller coefficients. These must be analytically calculated.

Each phase of the PFC converter must have its own structure declared and this function should be called for each phase. All current loops must be initialized before the voltage loop.

This function sets the coefficients for the 2p2z current loop controller.
The coefficients are passed as arguments to the function. These coefficients must be analytically calculated according to the converter specification as taught in the Biricha PFC workshop.

Each time this function is called a counter, the element m_PhaseCount of the PFC structure, is incremented which counts the total number of phases in this PFC system. Therefore, this function should be called for each PFC phase within this system. This allows easy set up of multiple phases for an interleaved converter.

The floating point controller used with the current loop has anti-integral windup which prevents the controller from saturating. The lower limit is set to 0.0 and the upper limit is set such that, after scaling by KIin, the output will be equal to the maximum duty. Although not recommended, these can be manually overridden by adjusting the m_Max and m_Min properties of the CNTRL_2p2zDataFloat structure.

8.2.2.2 Examples

Example 1 - Automatic interleaved set up using two controllers. Initializing the current loops of a two phase CCM Boost converter:

```
// Structure is aligned to 64 words
#pragma DATA_ALIGN ( iLoopPhase0 , 64 );
#pragma DATA_ALIGN ( iLoopPhase1 , 64 );
CNTRL_2p2zDataFloat iLoopPhase0;
CNTRL_2p2zDataFloat iLoopPhase1;

// Initialize the controller in the current loop
PFC01_iLoopFloatInit( &Pfc, &iLoopPhase0,
                      I0A1, I0A2,
                      I0B0, I0B1, I0B2,
                      KIloop, 1.0 );

PFC01_iLoopFloatInit( &Pfc, &iLoopPhase1,
                      I1A1, I1A2,
                      I1B0, I1B1, I1B2,
                      KIloop, 1.0 );
```

Example 2 - Manual interleaved set up using only one controller. The output of which sets the duty for both phases of this two phase CCM Boost converter.

```
// Structure is aligned to 64 words
#pragma DATA_ALIGN ( iLoopPhase0 , 64 );
CNTRL_2p2zDataFloat iLoopPhase0;

// Initialize the controller in the current loop
PFC01_iLoopFloatInit( &Pfc, &iLoopPhase0,
                      I0A1, I0A2,
                      I0B0, I0B1, I0B2,
                      KIloop, 1.0 );

// Set up the second phase to use the same controller
// as the first. Duty of this is set in the same ISR
// as the first.
Pfc.m_PhaseCount++;
PFC01_enablePhaseFloat( &Pfc, 1, true );  // Index 1 meaning 2nd phase
```

8.2.3 PFC01_vffFloatInit

void PFC01_vffFloatInit(PFC_DataFloat* Ptr,float a1,float a2,float b0,float b1,float b2)

where:
Ptr - The controller coefficients are stored within the master PFC structure first initialized using PFC01_configFloat(). This master structure must be passed to this function as a reference.
a1 -
a2 -
b0 -
b1 -
b2 -

8.2.3.1 Description

Initializes the 2 pole 2 zero (2p2z) filter structure with the required coefficients.

a1, a2, b0, b1, b2 - The filter coefficients which must be analytically calculated.

The correct calculation method for these coefficients is discussed in the Biricha PFC workshop.

8.2.3.2 Examples

The VFF filter is initialized:

```
// Initialize the VFF filter
PFC01_vffFloatInit( &Pfc, VinA1,
                    VinA2, VinB0, VinB1, VinB2 );
```

8.2.4 PFC01_vLoopFloatInit

void PFC01_vLoopFloatInit(PFC_DataFloat* Ptr,uint16_t Ref,uint16_t UnderRef,uint16_t OverRef,float a1,float a2,float b0,float b1,float b2,float K,float Min,float Max)

where:
Ptr - The controller coefficients are stored within the master PFC structure first initialized using PFC01_configFloat(). This master structure must be passed to this function as a reference.
Ref - Output voltage reference. The output voltage after conversion by the ADC is subtracted from this reference to find the error value. Therefore this reference must be scaled accordingly to be within the same range as the ADC outputs.
UnderRef - Output voltage under voltage reference. This sets the level at which the under voltage protection occurs. This reference must be scaled to be within the same range as the ADC outputs.
OverRef - Output voltage over voltage reference. This sets the level at which the over voltage protection occurs. This reference must be scaled to be within the same range as the ADC outputs.
a1 -
a2 -
b0 -
b1 -
b2 -
K - This gain is used to scale the output of the voltage loop.
Min - This sets the minimum output of the controller. This anti-windup limit prevents the controller from saturating.
Max - This sets the maximum output of the controller. This anti-windup limit prevents the controller from saturating.

8.2.4.1 Description

Initializes the 2 pole 2 zero (2p2z) voltage loop controller structure with the required coefficients and scaling factor, KVaout.

a1, a2, b0, b1, b2 - The controller coefficients which must be analytically calculated.

This voltage loop initialization function requires the ADC reference values for nominal output voltage (Vout), under voltage (UVP) and over voltage (OVP) conditions. These should be passed as a number within the range of the ADC output. Thus, the function VoutToAdcValue() can be used to convert from Volts to an ADC value.

If Vout > OVP then the controller stops PWM but does not reset soft start (SS). This means that as soon as Vout falls below OVP, the PWM are restarted without soft starting.

If Vout < UVP then the controller turns off the PWM and resets SS. This means that as soon as Vout goes above the UVP level, the PWM is restarted by means of a soft start. This functionality is almost identical to the analog UCC3817 part.

The floating point controller used with the voltage loop has anti-integral windup which prevents the controller from saturating. The lower limit is set by m_Min and the upper limit is by m_Max. These limits are applied after the controller output has been scaled by K.

8.2.4.2 Examples

Initialize the voltage loop:

```
PFC01_vLoopFloatInit( &Pfc,
                      VoutToAdcValue(Vout),
                      VoutToAdcValue(20.0),
                      VoutToAdcValue(55.0),
                      VA1, VA2,
                      VB0, VB1, VB2,
                      KVaout, VloopMin, VloopMax );
```

8.2.5 PFC01_softStartDirectionFloat

void PFC01_softStartDirectionFloat(PFC_DataFloat* Ptr,uint16_t PowerUp)

where:
Ptr - The master PFC structure.
PowerUp - An integer which determines the soft-start state.

8.2.5.1 Description

Sets the direction of the soft-start or soft-stop.

If the PowerUp argument is non-zero the converter will soft-start. If PowerUp is zero the converter will soft-stop.

The duration of the soft-start/stop is set with PFC01_configFloat().

This function should be called within the voltage/slow loop.

8.2.5.2 Examples

Reads the value of a switch to determine soft-start or soft-stop.

```
PFC01_softStartDirectionFloat( &Pfc, GPIO_get( GPI_SWITCH ) );
```

8.2.5.3 Notes

8.2.6 PFC01_vLoopFloat

void PFC01_vLoopFloat(PFC_DataFloat* Pfc)

where:
Pfc - The master PFC structure.

8.2.6.1 Description

Executes the voltage loop. Therefore this function should only be called within the voltage/slow loop.

This function executes the voltage feed-forward filter, executes the 2p2z controller of the voltage loop (which is scaled by KVaout and the number of phases enabled), calculates VAOUT/VRMS^2, updates the soft start and finally calculates the current reference scaling factor, ImoutScale, for use within the current loop.

The output of this functions is later used in the ISR by the current controller to calculate the demand value of current and hence the new duty.
Note that the scaling factor takes in to account the number of phases, if there are any, for interleaved converters. This allows easy set up for interleaved operation.

The number of interleaved phases is automatically detected from the number of user defined current controllers and thus, the user does not need to calculate any additional scaling factors.

Phases are enabled and disabled using PFC01_enablePhaseFloat(). Calling PFC01_enablePhaseFloat() to enable/disable phases will automatically update the scaling factor accordingly to share the current between the phases.

8.2.6.2 Examples

Executes the voltage loop filter and controller:

```
PFC01_vLoopFloat(&Pfc);
```

8.2.6.3 Notes

8.2.7 PFC01_iLoopFloat

void PFC01_iLoopFloat(void Pfc,void PhaseIndex)

where:
Pfc - The master PFC structure.
PhaseIndex - The index of the phase whose controller is to be executed.

8.2.7.1 Description

Executes the current loop.

This function calculates and sets the current reference value of the 2p2z controller and then executes the controller.

This function calls PFC01_getImoutFloat() to calculate the current reference for this phase. The value of VAOUT/VRMS^2 which was calculated in the slow idle loop is multiplied by the sampled rectified mains voltage; this is the demand value of the current.

The current controller is then executed with the demand value of the current and the sampled (i.e actual) value current as inputs; the output is the new value of duty.

The PhaseIndex argument is the phase number of the current loop being executed. Note that 0 means the first phase, 1 would mean the 2nd phase of an interleaved converter, 2 would mean the 3rd phase and so on.

This function is normally called within the ADC interrupt service routine.

8.2.7.2 Examples

Executes the current loop for the first phase:

```
PFC01_iLoopFloat( &Pfc, 0 );
```

8.2.7.3 Notes

8.2.8 PFC01_getImoutFloat

float PFC01_getImoutFloat(void Pfc)

where:
Pfc - The master PFC structure.

8.2.8.1 Description

Calculates the current reference for use with the current loop controller.

The value of VAOUT/VRMS^2 which was calculated in the slow idle loop is multiplied by the sampled rectified mains voltage; this is the demand value of the current.

This function would not normally need to be called separately as it is already used within the current loop controller.

8.2.8.2 Examples

Calculates the current reference:

```
float ref = PFC01_getImoutFloat(Pfc);
```

8.2.8.3 Notes

8.2.9 PFC01_enablePhaseFloat

void PFC01_enablePhaseFloat(PFC_DataFloat* Pfc,uint16_t Index,bool Enabled)

where:
Pfc - The master PFC structure.
Index - The phase to be enabled/disabled.
Enabled - A Boolean value, true to enable or false to disable.

8.2.9.1 Description

Enables and disables current loop phases.

This function sets the enable mask for the specified phase, x. The mask "m_DutyMask[x]" should be bitwise AND'd with the PWM duty to enable or
disable the PWM output according to this function and the OVP/UVP overrides.
See the example below.

If the phase is enabled and there are no OVP/UVP overrides then:
.m_DutyMask[x] = 0xfff

If the phase is enabled and there is an OVP/UVP condition then:
.m_DutyMask[x] = 0x0;

If the phase is disabled then:
.m_DutyMask[x] = 0x0

A scaling factor is calculated in this function which is used in the voltage loop to share the demand current equally between the number of phases enabled.
The number of enabled phases must be counted and thus this function should be executed in slow loop idle time.

8.2.9.2 Examples

Enables phase 1:

```
// Enable phase 1
PFC01_enablePhaseFloat( &myPfc, 1, true );

// Set the duty in the interrupt routine:
PWM_setDutyA(PWM_MOD_4, (uint16_t)iLoopPhase1.m_Out/2 &
             myPfc.m_DutyMask[1]);

// The property m_DutyMask[1] = 0xfff, assuming no OVP or UVP overrides.
```

8.2.9.3 Notes

8.3 Types

8.3.1 MAX_PHASES

```
#define MAX_PHASES 8
```

8.3.1.1 Description

8.3.2 PFC_DataFloat

```
struct PFC_DataFloat
{
      uint16_t    m_PhaseCount;     /* The number of phases initialised */
      uint16_t    m_PhaseMaskEnabled;     /* a bit mask of phases enabled */
      float m_PhaseScale;     /* This changes depending on the number of phases
enabled */
      uint16_t    m_FastPeriodTicks;
      uint32_t    m_SlowLoopHz;
      CNTRL_2p2zDataFloat     Vloop2p2z;
      CNTRL_RampFloat   VloopVref;
      FILT_2p2zDataFloat      VffFilter;
      float m_ImoutScale;
      bool  m_SoftStartReset; /* When true the softstart is reset to zero */
      uint32_t    m_RampMs;
      uint16_t    m_VloopUnderRef;
      uint16_t    m_VloopOverRef;
      bool  m_VloopIsFault;   /* this is set when Vloop is overvoltage or
undervoltage */
      uint16_t    m_DutyMask[MAX_PHASES];
      CNTRL_2p2zDataFloat*    iLoop2p2z[MAX_PHASES];
};
```

8.3.2.1 Description