Digital Control Loop Designer SDK User Guide

z-Domain Configuration Window

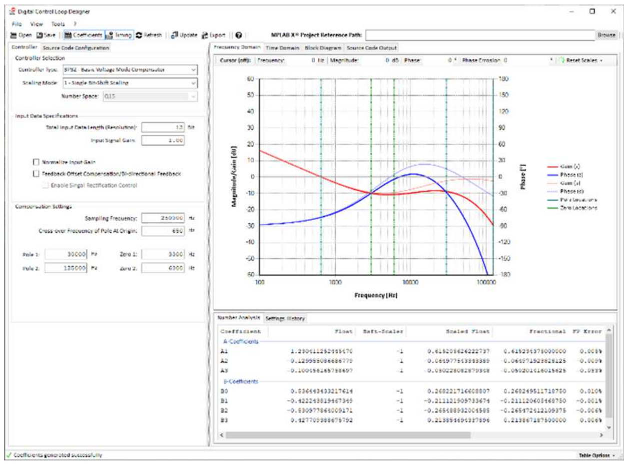
INTRODUCTION

The Digital Control Loop Designer SDK is a Software Development Kit (SDK) consisting of individual tools covering system definition, system modeling, code generation, control system fine tuning and real-time debugging of fully digital control systems for Switched-Mode Power Supplies (SMPS) for dsPIC® Digital Signal Controllers (DSC).

This user guide section is meant to be a quick start guide to the z-Domain Controller Configuration tool (DCLD.exe), which can be used to select and configure discrete time domain control systems, tailor their features to the specific controller device used and generate control libraries with a generic application programming interface (API) to allow fast and seamless integration of the generated source code in custom firmware projects.

PLEASE NOTE

This software is still in a preliminary, experimental state with limited support.
All features and functions are subject to change at any time without further notice.
Please always refer to the most recent readme.txt file to get updates on features and functions and to review release notes and history.


Digital Control Loop Designer
File View Tools ?
Open Save Coefficients Timing Refresh Update Export
MPLAB X® Project Reference Path: Browse
Controller Source Code Configuration
Frequency Domain Time Domain Block Diagram Source Code Output
Magnitude/Gain [dB]
Phase [°]
Frequency [Hz]
Number Analysis Settings History
Table Options

Controller Selection
Controller Type: 3P3Z - Basic Voltage Mode Compensator
Scaling Mode: 1 - Single Bit-Shift Scaling
Number Space: Q15

Input Data Specifications
Total Input Data Length (Resolution): 12 bit
Input Signal Gain: 1.00
Normalize Input Gain
Feedback Offset Compensation/Bi-directional Feedback
Enable Singal Rectification Control

Compensation Settings
Sampling Frequency: 250000 Hz
Cross-over Frequency of Pole At Origin: 650 Hz
Pole 1: 30000 Hz Zero 1: 3000 Hz
Pole 2: 125000 Hz Zero 2: 6000 Hz

Coefficient	Float	Bit-Scaler	Scaled Float	Fractional	FP Error
A-Coefficients					
A1	1.230411252445470	-1	0.615205626222737	0.615234375000000	0.005%
A2	-0.129955086686775	-1	-0.064977543343389	-0.064971923828125	-0.009%
A3	-0.100456165758697	-1	-0.050228082879348	-0.050201416015625	-0.053%
B-Coefficients					
B0	0.536443433217614	-1	0.268221716608807	0.268249511718750	0.010%
B1	-0.422243819467349	-1	-0.211121909733674	-0.211120605468750	-0.001%
B2	-0.530977864009171	-1	-0.265488932004585	-0.265472412109375	-0.006%
B3	0.427709388675792	-1	0.213854694337896	0.213867187500000	0.006%

Coefficients generated successfully

Figure 1: z-Domain Configuration Window of the Digital Control Loop Designer SDK

- **Recommended Literature:**

Data sheets and reference manuals are available on http://www.microchip.com.

- dsPIC33EP128GS806 data sheet
- dsPIC33CH128MP506 data sheet
- dsPIC33CK256MP506 data sheet
-

- **Technical Specifications:**

Minimum System Requirements:

- Microsoft Windows® 7 32-bit Operating System
- 1 GB RAM
- 24 MB of free hard drive space

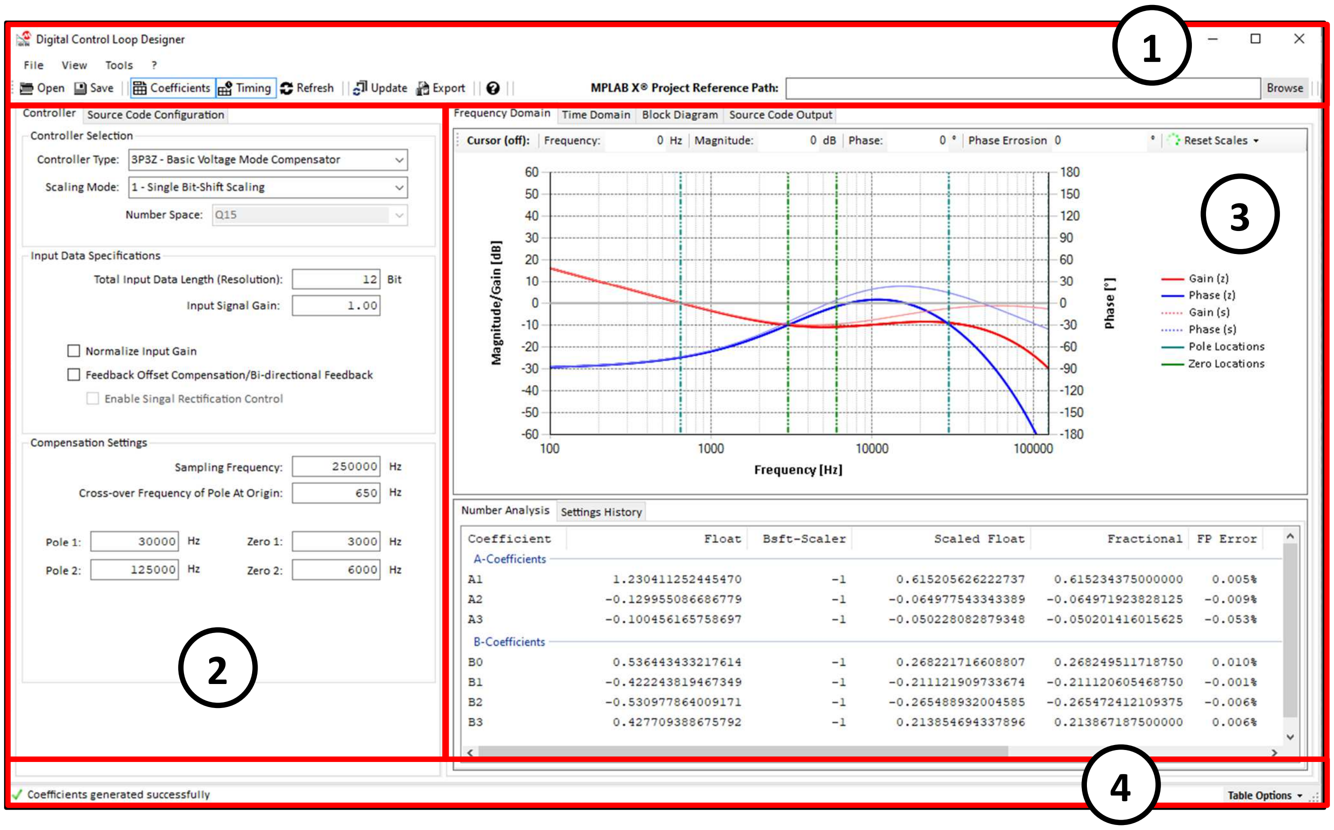
(this page was left blank intentionally)

TABLE OF CONTENTS

1.0 GRAPHICAL USER INTERFACE OVERVIEW

The main application window is divided into four sections a shown in Figure 2 below. The graphical user interface (GUI) has been designed following Microsoft Win32 UX Guidelines to make it most intuitive to use. On the top of the Window you find menus giving access to files and application functions. A command bar has been added to give quick access to most common functions of the application (1).

The main section of the window is divided into a User Configuration section (2) on the left. This is where all user settings are made. On the right, an Application Output section (3) shows the results of the most recent user configuration. Due to the complexity of digital compensator design, the results are split into multiple sub-sections grouped by topics (Frequency Domain, Time Domain, Block Diagram and Source Code).



Controller Selection
Controller Type: 3P3Z - Basic Voltage Mode Compensator
Scaling Mode: 1 - Single Bit-Shift Scaling
Number Space: Q15

Input Data Specifications
Total Input Data Length (Resolution): 12 Bit
Input Signal Gain: 1.00
☐ Normalize Input Gain
☐ Feedback Offset Compensation/Bi-directional Feedback
☐ Enable Singal Rectification Control

Compensation Settings
Sampling Frequency: 250000 Hz
Cross-over Frequency of Pole At Origin: 650 Hz
Pole 1: 30000 Hz
Zero 1: 3000 Hz
Pole 2: 125000 Hz
Zero 2: 6000 Hz

Coefficient	Float	Bsft-Scaler	Scaled Float	Fractional	FP Error
A-Coefficients					
A1	1.230411252445470	-1	0.615205626222737	0.615234375000000	0.005%
A2	-0.129955086686779	-1	-0.064977543343389	-0.064971923828125	-0.009%
A3	-0.100456165758697	-1	-0.050228082879348	-0.050201416015625	-0.053%
B-Coefficients					
B0	0.536443433217614	-1	0.268221716608807	0.268249511718750	0.010%
B1	-0.422243819467349	-1	-0.211121909733674	-0.211120605468750	-0.001%
B2	-0.530977864009171	-1	-0.265488932004585	-0.265472412109375	-0.006%
B3	0.427709388675792	-1	0.213854694337896	0.213867187500000	0.006%

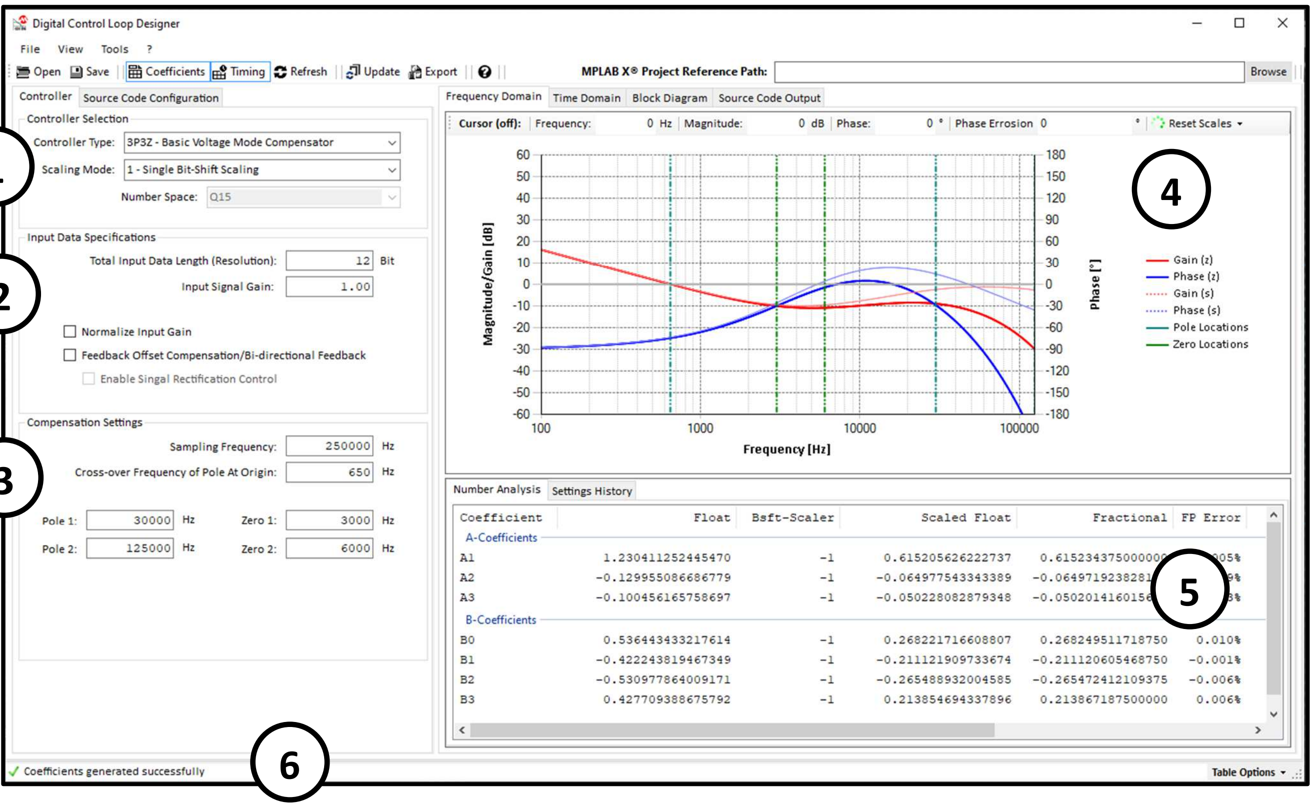
✓ Coefficients generated successfully

Figure 2: Digital Control Loop Designer z-Domain Controller Main Window Overview

TABLE 1: MAIN WINDOW DESCRIPTION

No	Description
1	Main Menu and Control Bar
2	User Configuration Panel
3	Application Output Panel
4	Application Status Information

2.0 FREQUENCY DOMAIN CONFIGURATION (BODE PLOT)



Controller Selection

- Controller Type: 3P3Z - Basic Voltage Mode Compensator
- Scaling Mode: 1 - Single Bit-Shift Scaling
- Number Space: Q15

Input Data Specifications

- Total Input Data Length (Resolution): 12 Bit
- Input Signal Gain: 1.00
- ☐ Normalize Input Gain
- ☐ Feedback Offset Compensation/Bi-directional Feedback
- ☐ Enable Singal Rectification Control

Compensation Settings

- Sampling Frequency: 250000 Hz
- Cross-over Frequency of Pole At Origin: 650 Hz
- Pole 1: 30000 Hz
- Zero 1: 3000 Hz
- Pole 2: 125000 Hz
- Zero 2: 6000 Hz

Coefficient	Float	Bsft-Scaler	Scaled Float	Fractional	FP Error
A-Coefficients					
A1	1.230411252445470	-1	0.615205626222737	0.61523437500000	005%
A2	-0.129955086686779	-1	-0.064977543343389	-0.0649719238281	9%
A3	-0.100456165758697	-1	-0.050228082879348	-0.0502014160156	3%
B-Coefficients					
B0	0.536443433217614	-1	0.268221716608807	0.268249511718750	0.010%
B1	-0.422243819467349	-1	-0.211121909733674	-0.211120605468750	-0.001%
B2	-0.530977864009171	-1	-0.265488932004585	-0.265472412109375	-0.006%
B3	0.427709388675792	-1	0.213854694337896	0.213867187500000	0.006%

✓ Coefficients generated successfully

Figure 3: Digital Control Loop Designer Frequency Domain View

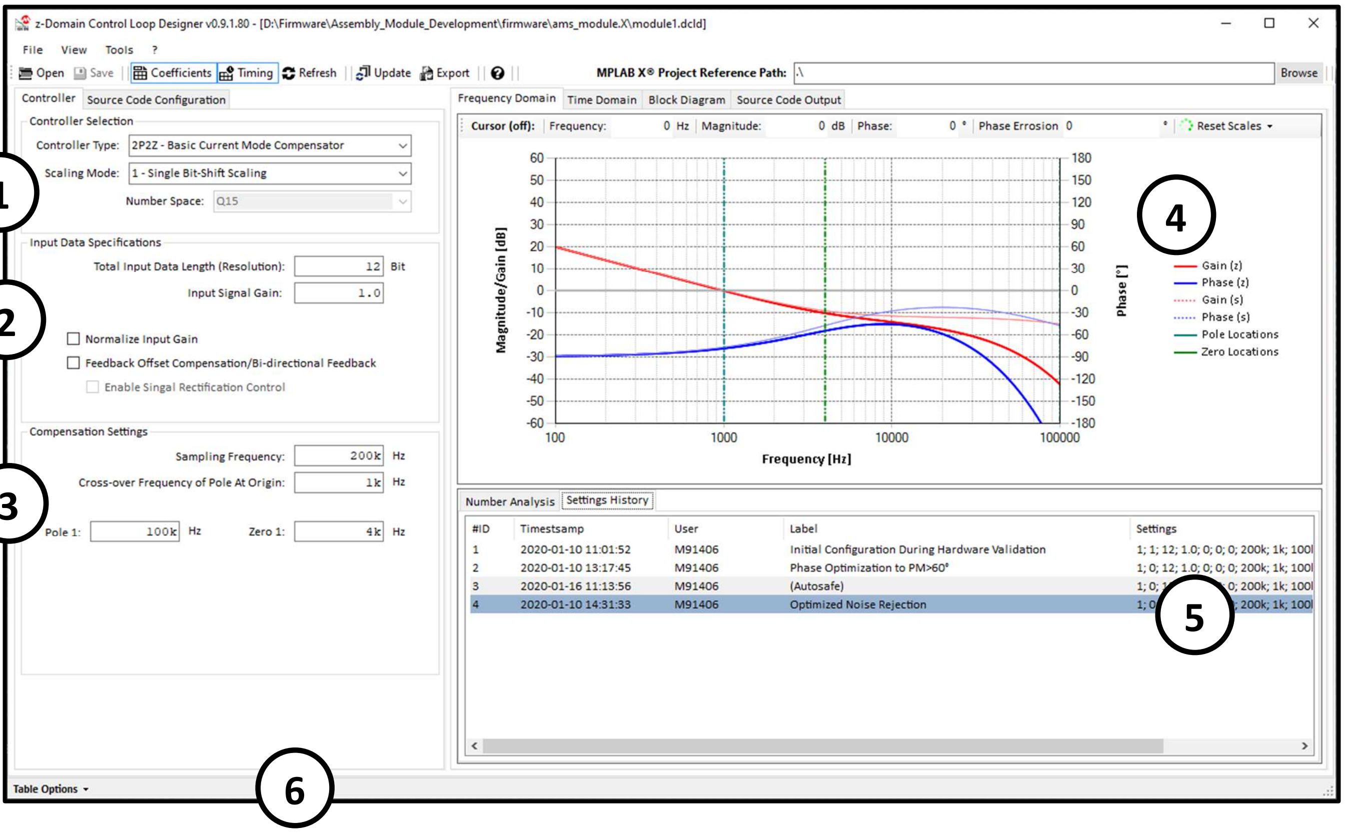
TABLE 2: MAIN WINDOW DESCRIPTION

No	Description
1	Controller Order and Number Format Selection
2	Input Data Specification
3	Compensator Configuration
4	Frequency Response (Bode Plot) of s-Domain and z-Domain transfer function
5	Digital Filter Coefficients derivation transcript with accuracy analysis of final values
6	Status bar indicating background activity and output messages

The z-Domain Controller configuration window is ordered into a left configuration plane and a right plane showing the results based on recent settings. Both planes are separated in individual sub-planes (tabs) offering access to settings of individual, functional blocks.

The default view starts with the controller selection and frequency domain configuration on the left and the Bode Plot graph of the transfer function on the right. Below the Bode plot a data table shows the derivation transcript of the calculation result. This table is also used to display warnings of the number accuracy analyzer.

2.0 Frequency Domain Configuration (Bode Plot) (CONTINUED)


z-Domain Control Loop Designer v0.9.1.80 - [D:\Firmware\Assembly_Module_Development\firmware\ams_module.X\module1.dcld]
File View Tools ?
Open Save Coefficients Timing Refresh Update Export
Browse
Controller Source Code Configuration
Frequency Domain Time Domain Block Diagram Source Code Output
Reset Scales
Magnitude/Gain [dB]
Phase [°]
Frequency [Hz]
Gain (z)
Phase (z)
Gain (s)
Phase (s)
Pole Locations
Zero Locations
Number Analysis Settings History
Table Options
1
2
3
4
5
6

MPLAB X® Project Reference Path: .\

Controller Selection
Controller Type: 2P2Z - Basic Current Mode Compensator
Scaling Mode: 1 - Single Bit-Shift Scaling
Number Space: Q15

Input Data Specifications
Total Input Data Length (Resolution): 12 Bit
Input Signal Gain: 1.0
☐ Normalize Input Gain
☐ Feedback Offset Compensation/Bi-directional Feedback
☐ Enable Singal Rectification Control

Compensation Settings
Sampling Frequency: 200k Hz
Cross-over Frequency of Pole At Origin: 1k Hz
Pole 1: 100k Hz Zero 1: 4k Hz

Cursor (off): Frequency: 0 Hz Magnitude: 0 dB Phase: 0 ° Phase Errosion 0 °

#ID	Timestsamp	User	Label	Settings
1	2020-01-10 11:01:52	M91406	Initial Configuration During Hardware Validation	1; 1; 12; 1.0; 0; 0; 0; 200k; 1k; 100l
2	2020-01-10 13:17:45	M91406	Phase Optimization to PM>60°	1; 0; 12; 1.0; 0; 0; 0; 200k; 1k; 100l
3	2020-01-16 11:13:56	M91406	(Autosafe)	1; 0; 1 [illegible] 0; 200k; 1k; 100l
4	2020-01-10 14:31:33	M91406	Optimized Noise Rejection	1; 0 [illegible]; 200k; 1k; 100l

Figure 4: Digital Control Loop Designer Frequency Domain View with Workflow History

Loop tuning is a major step in the design process of a power supply. System optimizations might require to frequently modify filter settings to solve design tradeoffs.

To simplify the management of optimization iterations, Section (5) of the Frequency Domain View also provides access to the workflow history of the filter design process. This history table captures filter settings when code is generated, assuming generated code will be programmed into a device and measurements/bench-tests are performed.

History items cover:

• ID:	continuously incrementing number of history entries
• Time Stamp:	date and time of capture event / code generation event
• User Name:	user who last updated the filter parameters
• Label:	default: (Autosafe); can be renamed by user by hitting key F2 on the keyboard or right-click entry to open context menu selecting ‘Rename’
• Settings:	encoded list of settings used

Please note:

Settings can be recalled by Double-Click or selecting an entry and hitting ENTER on the keyboard.
This history only captures the filter configuration. Code generator options are ***not*** saved and restored.

2.1 Controller Selection

After application start the main window is starting in the frequency domain view and controller configuration. All digital controllers supported by this tool are based on discrete time domain transfer functions, which have been derived from continuous time domain transfer function prototypes using the bi-linear transform (BLT).

The continuous time domain prototype transfer functions are based on conventional type I, II, III compensation circuits used in SMPS control systems the industry since the early 1980s. All higher order transfer functions used by this tool, are mathematically up-scaled versions of the same control approach.

These controllers consist of lead-lag compensators of the n-th order, multiplied with a simple integrator term incorporating the gain influence over frequency of an additionally introcuced pole at the origin. The only difference between these transfer functions is the order of the lead-lag compensator term, which may include no pole/zero pair at all (1st order) or up to n pole/zero pairs (n-th order).

2.2 Why using s-Domain Prototype Filters?

Creating discrete time domain controllers would not necessarily require the deviation of their transfer function from a continuous time domain prototype filter. However, this deviation path allows to establish relations between continuous and discrete time domain control systems, which allow us to use the very same, well understood design procedures and techniques common in the power supply industry. This also includes using tools for both domains while still being able to consider, incorporate or compensate for the differences between them.

The z-Domain Configuration Window of the Digital Control Loop Designer SDK takes pole and zero frequency locations to characterize the compensation filter and total feedback loop gain and can therefore directly be applied to system-level frequency domain models including filters, feedback conditioning circuits and the plant by merging continuous and discrete time domain blocks.

2.3 Selecting the Right Controller

The selection of the right controller for an application exclusively depends on the power supply plant circuit characteristics and applied control mode. Based on the knowledge of the frequency domain characteristic of a specific design, the controller selection mainly comes down to compensating of the power supply plant by compensating each system pole with a zero in the compensator and each system zero with a pole in the compensator. This method is basically a linearization approach where the non-linearity of the plant frequency domain is made linear. By introducing a pole at the origin, the now flat, linear open loop transfer function is rotated by 45°, turning the system into a 1st order control low-pass filter.

Although this sounds easy and straight forward, the physics of a power supply circuit leaves us with plenty of system aspects which are not covered by this linearization approach, such as filter resonance frequencies or time constants of the exchange of charges between components defined by their parasitic parameters. So eventually a compensator does *not* result in a truly linear system, but this compensation technique is the fundamental basis to stabilize a by-default unstable power supply filter circuit.

A comprehensive description of plant circuits and influencing factors is beyond the scope of this user guide. However, the following, general guidelines may help.

The frequency domain design process essentially requires knowledge of the frequency domain characteristic of the power supply circuit. Once this is known and pole and zero locations have been derived, a controller is chosen which has at least as many poles and/or zeros as the plant.

Further, it is necessary to keep in mind that a switch-mode power supply circuit is not able to continuously provide power to the output as the power path between input and output is broken and re-connected in every switching cycle while continuous output power is only provided by the output capacitor. In fact, a switch-mode power supply is more like a filtered z-domain system, which cannot respond to transients faster than half of its switching frequency (z-domain Shannon-Nyquist limit). To prevent fast transients/high frequency noise affects the performance of the power supply and its feedback loop, a good, high-frequency noise rejection is required. This is usually achieved by placing an additional pole at the edge of the maximum frequency band (either half of the switching frequency or half of the sampling frequency, whatever is lower)

DCLD provides compensator filters up to the 6th order by selecting one of the following ***Controller Type*** options:

- **1P1Z:**
  1st order with integrator and no pole/zero pair
- **2P2Z:**
  2nd order with integrator and one pole/zero pair
- **3P3Z:**
  3nd order with integrator and two pole/zero pairs
- **4P4Z:**
  4nd order with integrator and three pole/zero pairs
- **5P5Z:**
  5nd order with integrator and four pole/zero pairs
- **6P6Z:**
  6nd order with integrator and five pole/zero pairs

Every pole and zero location can be set by either moving the respective pole or zero indicator in the Bode plot using the mouse pointer or edit the frequency in the respective entry boxes below the controller selection.

2.4 Scaling Mode Selection

Each controller design is a tradeoff between performance and accuracy. With increased number space the result accuracy can be enhanced. Enhanced accuracy, however, requires more CPU cycles for the computation. Luckily, filter coefficients are getting smaller with increased frequency so that more efficient scaling methods can be used when CPU load constraints are getting tougher.

To solve these tradeoffs easily and individually for every loop and on system-level, the z-Domain Configuration Window offers four different number scaling modes for each controller type:

- **Single Bit-Shift Scaling**
  Highest performance is achieved by directly utilizing the fixed-point DSP core of the dsPIC® DSC by scaling all filter coefficients with the very same scaling factor. The factor scaling is implemented by shifting the number bit

code to the right (divide by power of 2) or left (multiply by power of 2). This scaling method is sufficient for a wide variety of applications with standard topologies.

- **Single Bit-Shift with Output Factor Scaling**
  Occasionally coefficients with single fixed scalers may be affected by accuracy limitations, which, in the worst case, could corrupt the convolution process of the digital filter and negatively affect the error integration.

  In this scaling mode one additional factor is added and all coefficients are rescaled to minimize the rounding error of coefficients.
- **Dual Bit-Shift Scaling**
  The single bit-shifting mode with output factor scaling may come short, when coefficients of filter terms A and B vary significantly in size. For these conditions the Dual Bit Shift Mode was introduced, which applies *two* different scalers, one for A and one for the B term coefficients. The performance impact is very similar to the Single Bit-Shift with Output Factor Scaling.
- **Fast Floating Point Coefficient Scaling**
  In Fast Floating Point mode each coefficient gets its individual bit-shift scaler to maximize number accuracy. *This number format is different from conventional IEEE 754 floating point numbers*. Fast Floating Point numbers have re-ordered binary encoding to optimize the computation process on fixed-point DSP cores. This number format is the most accurate but also most intensive in terms of CPU cycles.

Fast Floating Point (ffloat16) Number Encoding

HIGH WORD		**LOW WORD**
SIGN	FRACTIONAL	SCALER
x	xxxxxxx xxxxxxxx	xxxxxxxx xxxxxxxx
Bit [31]	Bit [30:16]	Bit [15:0]

Example:

The number 7.965702247619620 needs to be encoded as Fast Floating Point (`ffloat16`) number. As the fractional portion of `ffloat16` is a signed Q15 fractional number, the available number range is limited to:

- maximum positive number: $(2^{15} - 1) \times 2^{-15}$ = 0.999969482421875
  (= Hexadecimal 0x7FFF)
- maximum negative number: $-2^{15} \times 2^{-15}$ = -1.000000000000000
  (= Hexadecimal 0x8000)

Obviously, 7.965… is greater than the maximum specified number and therefore needs to be scaled into the valid number range. B applying bit-shift scaling of the integer representation of the fractional number, we perform fast multiplies and divides by powers of 2 which can be executed in a single CPU cycle. One bit-shift to the right is therefore equivalent to a divide by 2. One bit-shift to the left is equivalent to a multiply by 2.

To scale the number 7.965… into the available range between -1.000… and +0.999… we need to shift the integer representation of this number three times to the right (divide by 8), giving us the number 0.995712780952453. This number is now less than 0.999969482421875 and fits into the Q15 number range. Each bit shift performed is stored in the `SCALER` of the `ffloat16` number and can therefore be decoded correctly later.

The recommended process of determining the best scaling mode is to start from single bit-shifting while observing the coefficient output window below the Bode plot. Should one or more coefficients exceed 0.5% error, it will be marked in yellow (warning level), if the inaccuracy exceeds 1.0%, it will be marked red (error level).

Should any of these warnings appear, increase the scaling option until all warnings disappear. Observe the timing diagram (tab Time Domain) on the right, to keep track on the CPU load, execution time and overall timing alignment.

2.5 Input Data Specification

This section is used to normalize the input data to the computation engine. The input data range needs to meet the number format of the selected controller. Input data needs to meet the following requirements to prevent gain mismatches between model and desired control output:

- Only positive 16-bit wide numbers are allowed (0 to 65535)
- Maximum bit resolution needs to be scaled using the ***Total Input Data Length*** setting
- Static offsets should be compensated using the ***Feedback Offset Compensation*** option

Additionally, the Input Data Specification offers three additional options accounting for the physical scale or the feedback signal:

- **Input Signal Gain**
  Assuming the input data is coming from an analog-to-digital converter (ADC) reading a pre-conditioned analog signal, the gain of the signal conditioning circuit can be entered here (e.g. reading voltage from a voltage divider). As a result, the Bode plot graph on the right will show the impact on the frequency response of the controller (gains < 1 will drop the gain, gains > 1 will increase the gain)

Example 1: Voltage Divider Gain
The output voltage of a power converter is conditioned by voltage divider providing a feedback voltage VFB to the ADC input, where the upper resistor R1 is 8.2kW and the lower resistor R2 is 1.1kW. The divider ratio represents the gain G and is calculated using the equation

$$\boldsymbol{G = \frac{R2}{R1 + R2} = \frac{1.1k\Omega}{8.2k\Omega\ + 1.1k\Omega} = 0.1183;}$$

Example 2: Shunt Amplifier Gain
The output current of a power converter is sensed across a shunt resistor RS of 10mW. The sense voltage is amplified by a shunt amplifier IC with an output gain GAMP of 20 V/V without signal offset. At an output current IOUT of 1A the amplifier would produce a feedback voltage VFB of 200mV.
The gain G is defined as ratio of V/A.

$$G = R_S \times G_{AMP} = 0.01\Omega \times 20V/V = 0.200;$$

The input gain to enter is 0.200.

- **Input Signal Gain Compensation (Option Normalize Input Gain)**

  In case the input data gain is different from 1, this option will automatically increase/decrease the gain of the controller to compensate for the physical signal gain deviation.

  If this option is not selected, gain variations have to be compensated by manually adjusting the Cross-Over Frequency Of The Pole At The Origin (a.k.a Zero-Pole) to achieve the desired results.

- **Compensating Input Data Offsets**

  - **Option *Feedback Offset Compensation***
    This option has been added to auto-correct simple, static signal offsets. As these offset values often need to be calibrated under specific test or operating conditions (e.g. while converter is off) or may even change during runtime, this offset value needs to be specified *in user code*.

    By enabling this option, the assembly code generator engine will add the item `InputOffset` to the assembly code module. This offset compensation is basically a control reference level shifter which will automatically add the user-defined offset value `InputOffset` to the most recent control reference value `ptrControlReference` on a cycle-by-cycle basis. Shifting the reference prevents potential gain distortions between outer and inner loop in cascaded control loop systems, such as average-current mode control, as well as accidental control loop inversions by feeding negative numbers into the unsigned number interface of the computation.

    The optional input offset value needs to be a signed 16-bit integer number ranging between
    -32,768 to +32,767.

  - **Signal Rectification Control**
    This option has been added to better support the control of bi-directional power supplies (2-Quadrant supplies). In these types of converters, the zero point of the current feedback is often lifted to half of the ADC range. Positive currents are represented by numbers greater than the zero-offset while negative currents are represented by numbers less than the zero-offset value. The catch with this signal conditioning is, that negative currents become indirect proportionally represented, which bares the risk of accidentally inverting the inverting feedback loop resulting in a non-inverting feedback loop. In this condition the power converter would go unstable instantly.
    While the converter is operating in one, defined direction, it might still be necessary to interpret numbers less than the zero offset as negative currents to allow the power supply to operate properly at/near zero load. Only when the power transfer is reversed, the current feedback polarity needs to be inverted intentionally to provide the expected, direct proportional representation of the current feedback to the control loop.

    By selecting the option *Enable Signal Rectification Control* adds a control bit to the `status` word of the `cNPNZ` data structure allowing to turn on/off the signal rectification manually *in user code*.

2.6 Compensation Filter Parameters

- **Sampling Frequency**

  Coefficients of z-domain filters need to incorporate the time-information between two samples to be able to 'reconstruct' the analog signal and give the accurate, desired response. Therefore, the sampling period T_S is the most vital and sensitive parameter of the digital compensation filter.

  This parameter is defined by entering the sampling frequency in Hz here.

- **Pole & Zero Locations**

  - **Cross-Over Frequency of Pole at Origin (Zero-Pole Frequency ZPF)**
    The Pole At The Origin is the major parameter which distinguishes the power supply compensator from a normal lead-lag compensator. It is this parameter which eventually introduces a steady falling gain slope of -20 dB/dec over frequency, turning the power supply into an active low-pass filter with good regulation.

    As this pole is indeed located in the origin of the frequency domain, as its name suggests, its effect on the frequency domain is adjusted by placing the cross-over frequency location (point where gain crosses 0 dB) at a specific point rather than the pole itself.

    By *increasing* the Cross-Over Frequency of the Pole At The Origin (Zero-Pole Frequency = ZPF), the absolute gain level of the feedback loop is *increased* without changing/affecting any of the other pole and zero locations.

    By *decreasing* the ZPF, the absolute gain level of the feedback loop is *decreased*

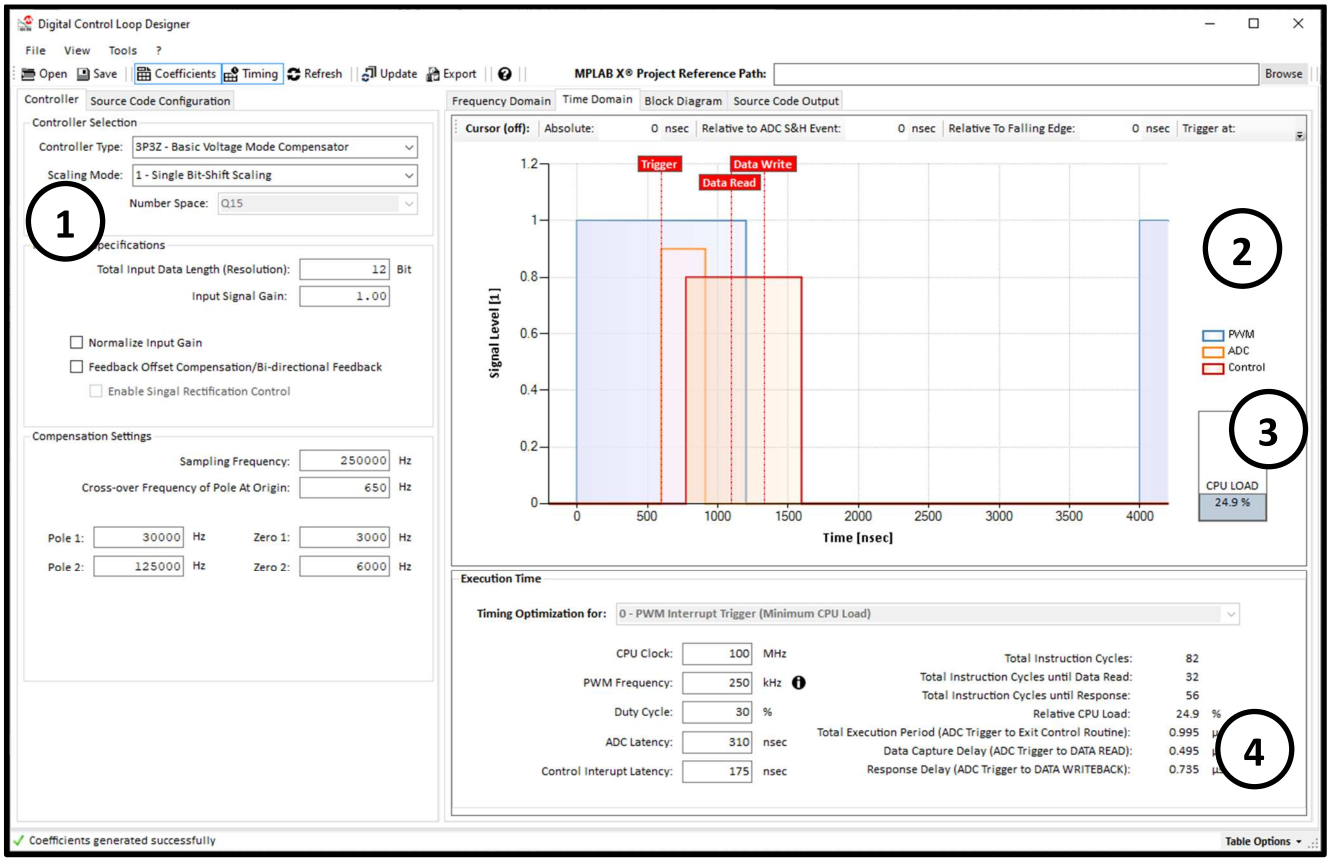
    Lag Compensator Pole & Zero Locations
    As explained under *2.3 Selecting the Right Controller*, pole and zero locations of the compensator need to be matched with pole and zero locations of the power plant. However, beyond this very basic compensation approach, some pole & zero locations may be repositioned slightly from their strict compensation locations to improve the frequency response of the power supply in accordance to design criteria and application requirements.

    Enter the derived pole & zero locations in the fields provided.

    Please note:
    In a digital controller, the sampling frequency determines the maximum continuous time domain (s-plane) frequency range, which can be mapped into the valid range of the discrete time domain (z-plane), limited by the Shannon-Nyquist limit at half of the sampling frequency. Beyond this point the z-domain filter stops working as the incoming data becomes to fractured to allow a proper signal representation of the real, analog signal.

    Any frequency entered *should not exceed* the Shannon-Nyquist limit.

3.0 TIME DOMAIN WINDOW



Controller Selection

Controller Type: 3P3Z - Basic Voltage Mode Compensator

Scaling Mode: 1 - Single Bit-Shift Scaling

Number Space: Q15

pecifications

Total Input Data Length (Resolution): 12 Bit

Input Signal Gain: 1.00

☐ Normalize Input Gain

☐ Feedback Offset Compensation/Bi-directional Feedback

☐ Enable Singal Rectification Control

Compensation Settings

Sampling Frequency: 250000 Hz

Cross-over Frequency of Pole At Origin: 650 Hz

Pole 1: 30000 Hz Zero 1: 3000 Hz

Pole 2: 125000 Hz Zero 2: 6000 Hz

Cursor (off): Absolute: 0 nsec | Relative to ADC S&H Event: 0 nsec | Relative To Falling Edge: 0 nsec | Trigger at:

Execution Time

Timing Optimization for: 0 - PWM Interrupt Trigger (Minimum CPU Load)

CPU Clock: 100 MHz

PWM Frequency: 250 kHz

Duty Cycle: 30 %

ADC Latency: 310 nsec

Control Interupt Latency: 175 nsec

Total Instruction Cycles:	82	
Total Instruction Cycles until Data Read:	32	
Total Instruction Cycles until Response:	56	
Relative CPU Load:	24.9	%
Total Execution Period (ADC Trigger to Exit Control Routine):	0.995	µ
Data Capture Delay (ADC Trigger to DATA READ):	0.495	µ
Response Delay (ADC Trigger to DATA WRITEBACK):	0.735	µ

✓ Coefficients generated successfully

Figure 5: Code Generator Configuration and Timing Diagram

TABLE 3: TIMING DIAGRAM OVERVIEW DESCRIPTION

No	Description
1	Code generator configuration option catalog
2	Timing diagram of control loop execution vs. SMPS switching waveform
3	Relative CPU Load bar
4	Timing calculation output and target device parameter configuration

A robust control loop needs to be executed with a fixed frequency and minimum time delay between an ADC sampling point and the related controller response write-back. Considering the time required to execute the control loop and its repetition rate, each control loop consumes a certain amount of the total available CPU bandwidth. Solving the trade-off between available CPU resources, control features and control accuracy is one of the major design objectives.

In this context a proper timing analysis is vital to prevent timing conflicts and CPU load bottlenecks which will both inevitably bare the risk of major system failures. The chart provided shows the PWM signal (main PWM pulse only), ADC trigger event and ADC conversion delay, control loop execution time, controller data read event and controller response write-back event.

- **Time Domain Chart Configuration**

  The parameters listed in Section 4 of the Time Domain view can be used to adjust the chart to represent specific application characteristics.

  These parameters are:

• CPU Clock:	Sets the effective CPU core clock / execution clock this parameter is used to calculate the generated control code
• PWM Frequency:	Set up the time scale of the chart and set the frequency of the displayed main PWM signal
• Duty Cycle:	Sets the pulse-width of the displayed main PWM signal
• ADC Latency:	Sets the pulse width of the ADC conversion time indicator
• Control Interrupt Latency:	Sets the time delay from trigger source to start of the control loop execution

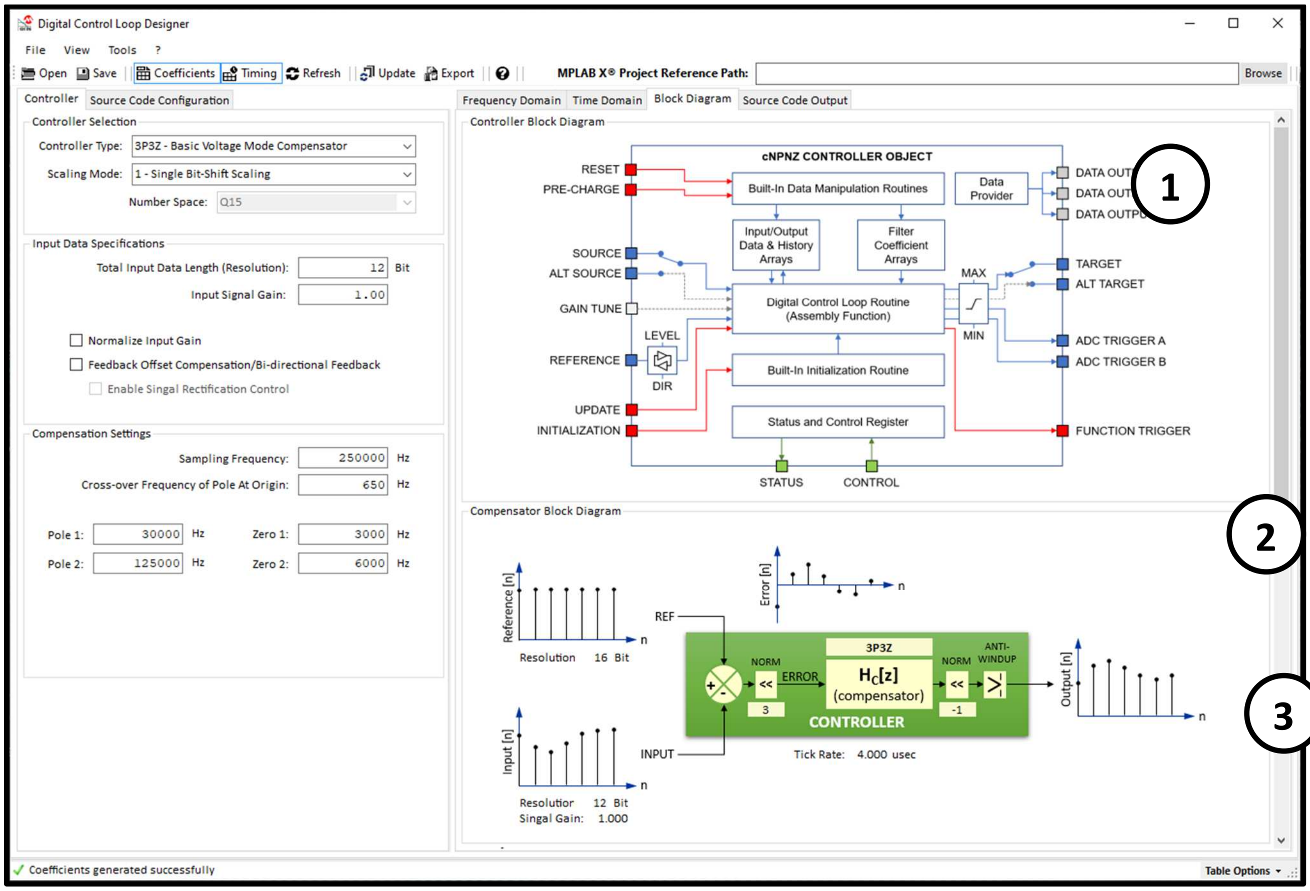
- **Time Domain Chart Output Table**

  The DCLD engine is calculating various timing relations depending on controller type and options selected. The calculation results are shown in Section 4 of the Time Domain view

  These parameters are:

• Total Instruction Cycles:	Number of instruction cycles required by this control loop (from start to end or `npnz_Update` controller function)
• Instruction Cycles until Data Read	Number of instruction cycles required from first instruction up to until the DATA READ instruction is executed
• Instruction Cycles until Response	Number of instruction cycles required from first instruction up to until the response writeback instruction is executed
• Instruction Cycles until Response	Number of instruction cycles required from first instruction up to until the response writeback instruction is executed
• Relative CPU Load:	This value represents the relative CPU load in [%] consumed by executing the configured control loop
• Total Execution Period:	Total execution time of one control step determined by the delay from ADC sampling to Exit of control routine
• Response Delay:	Total Zero-Order Hold (ZOH) of one control step determined by the time delay from ADC sampling to control loop data write back event

3.1 CONTROLLER BLOCK DIAGRAM WINDOW



Controller Selection

Controller Type: 3P3Z - Basic Voltage Mode Compensator

Scaling Mode: 1 - Single Bit-Shift Scaling

Number Space: Q15

Input Data Specifications

Total Input Data Length (Resolution): 12 Bit

Input Signal Gain: 1.00

☐ Normalize Input Gain

☐ Feedback Offset Compensation/Bi-directional Feedback

☐ Enable Singal Rectification Control

Compensation Settings

Sampling Frequency: 250000 Hz

Cross-over Frequency of Pole At Origin: 650 Hz

Pole 1: 30000 Hz Zero 1: 3000 Hz

Pole 2: 125000 Hz Zero 2: 6000 Hz

✓ Coefficients generated successfully

Figure 6: Block Diagram Overview

TABLE 4: BLOCK DIAGRAM OVERVIEW DESCRIPTION

No	Description
1	Controller block diagram with discrete time domain signal waveforms
2	Generic format of s- and z-domain compensator transfer function equivalents
3	Firmware module implementation block diagram and flow-chart

The block diagram overview shows four different block diagrams:

- NPNZ Controller Block
- Compensator Block (core block of NPNZ Controller Block)
- z-Domain transfer function
- Compensator processing workflow block diagram

The following sections provide more information about the firmware integration of the generated controller block and intended use cases.

- **Controller Module Firmware Integration**

DCLD generates code modules providing a “black box” controller with one, unified Application Programming Interface (API). The look-and-feel of the generated code blocks is like working with hardware peripherals on any MCU where the user sets the configuration and then enables the module. Once these code modules have been added to a project and the user configuration has been added to the firmware, user settings will remain valid even if controller options change, filter settings are modified or even compensators of different order or different number scaling types are selected.

The following section gives a high-level overview about the various settings made available by the API and shows which one of them are managed by the control code itself and which require user configuration.

3.2 Block Diagram

The generated control code library is a generic block with defined input and output ports.

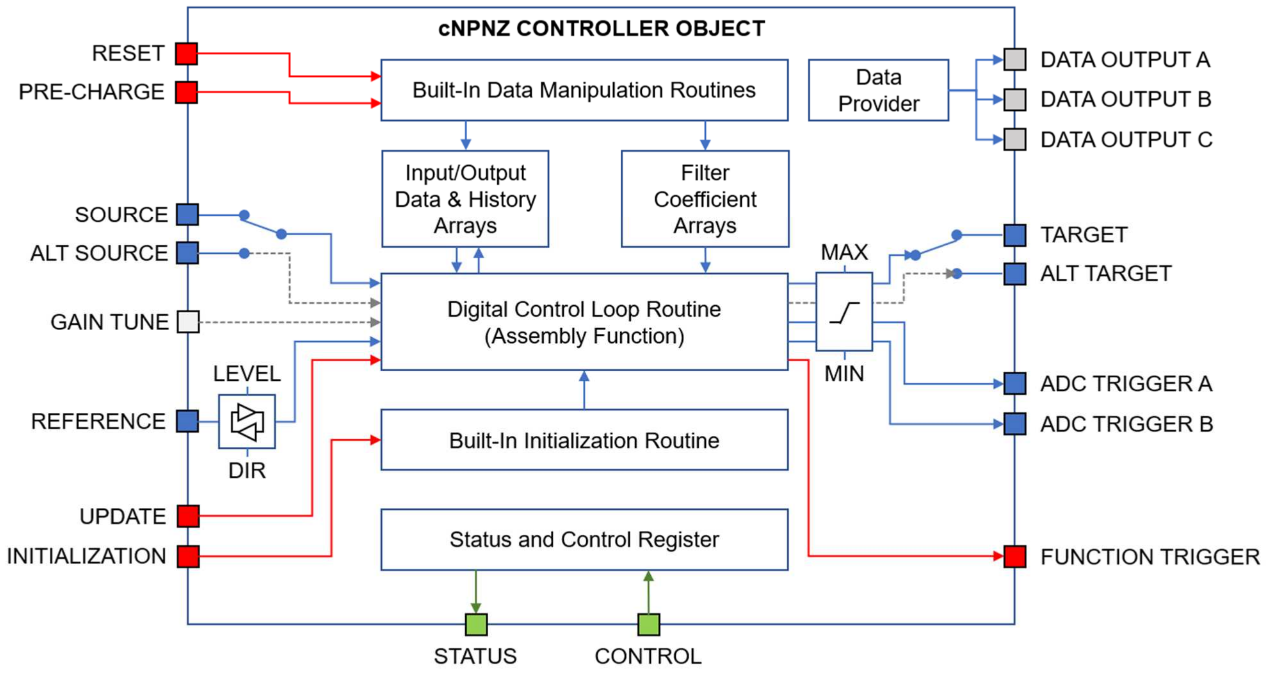


Figure 7: cNPNZ Controller Object Block Diagram

The control object API offers **DATA PATH (BLUE)**, **FUNCTION CALL PORTS (RED)** and **STATUS & CONTROL PORTS (GREEN)**. Some of these ports can be enabled/disabled/added/removed by selecting their option in the Code Generator Configuration.

These options are described in chapter **4.0 Code Generator Output W**

3.3 The NPNZ Data Structure

The generic NPNZ data structure introduces a data type called cNPNZ16_t, which covers all properties of all supported controller types. In addition, the code generator generates functions for initialization, reset and execution of the controller object.

3.4 NPNZ Object Configuration

As shown by Figure 7 above, the NPNZ object manages data paths between coefficient, input data and history data arrays internally. These 'connections' are initialized by the controller initialization routine, generated by the Assembly and C-Source code generator.

However, these are just a fraction of the required configurations required to operate the NPNZ controller in user code. Users must specify the following data paths and related optional parameters before the control loop can be enabled:

TABLE 5: NPNZ PROPERTIES CONFIGURATION

Property	**Description**	**User Config.**
Status & Control		
`status`	Controller status word with status and control bits	
`enable`	Bit [15]: Controller Enable/Disable Control Bit 1 = Execution of the compensation filter is enabled 0 = Execution of the controller is bypassed	☒
`invert_input`(1)	Bit [14]: Data Input Rectification Control Bit 1 = Error input will be inverted 0 = Error input will not be inverted	☒
`swap_source`(1)	Bit [13]: Data Input Source Control Bit 1 = Controller reads input data from `ptrAltSource` 0 = Controller reads input data from `ptrSource`	☒
`swap_target`(1)	Bit [12]: Data Output Target Control Bit 1 = Controller writes output data to `ptrAltTarget` 0 = Controller writes output data to `ptrTarget`	☒
	Bit [11:2]: (reserved)	
`flt_clamp_max`(1)	Bit [1]: Output Clamping Maximum Status 1 = Control output is greater than `MaxOutput` (got clamped) 0)= Control output is less than `MaxOutput`	
`flt_clamp_min`(1)	Bit [0]: Output Clamping Minimum Status 1 = Control output is less than `MinOutput` (got clamped) 0)= Control output is greater than `MinOutput`	

TABLE 5: NPNZ PROPERTIES CONFIGURATION (CONTINUED)

Property	Description	User Config.
I/O Data Interface		
`ptrSource`	Pointer to data input register/variable	☒
`ptrAltSource`(1)	Pointer to alternative data input register/variable	☒
`ptrTarget`	Pointer to data input register/variable	☒
`ptrAltTarget`(1)	Pointer to alternative data input register/variable	☒
`ptrControlReference`	Pointer to control reference variable	☒
`InputOffset`(1)	Static offset of data provided by `Source` or `AltSource`	☒
`MinOutput`(1)	Control output minimum value	☒
`MaxOutput`(1)	Control output maximum value	☒
Internal Data Sources and Normalization		
`ptrACoefficients`	Pointer to compensation filter A-coefficient array	☐
`ptrBCoefficients`	Pointer to compensation filter B-coefficient array	☐
`ptrControlHistory`	Pointer to compensation filter control output history array	☐
`ptrErrorHistory`	Pointer to compensation filter error input history array	☐
`normPreShift`	Data input normalization scaler	☐
`normPostShiftA`	Data output normalization scaler A	☐
`normPostShiftB`	Data output normalization scaler B	☐
`normPostScaler`	Data output normalization factor	☐
`ptrADCTriggerARegister`	Pointer to ADC trigger register of primary ADC trigger	☒
`ADCTriggerAOffset`	Constant primary ADC trigger delay value	☒
`ptrADCTriggerBRegister`	Pointer to ADC trigger register of secondary ADC trigger	☒
`ADCTriggerBOffset`	Constant secondary ADC trigger delay value	☒
Data Provider Sources		
`ptrDataProviderControlInput`(1)	Pointer to user variable receiving most recent input value	☒
`ptrDataProviderControlError`(1)	Pointer to user variable receiving most recent error	☒
`ptrDataProviderControlOutput`(1)	Pointer to user variable receiving most recent control output	☒
Data Provider Sources		
`CascadedFunction`(1)	Pointer to user-defined function	☒
`CascadedFunParam`(1)	Pointer to one 16-bit wide user-function parameter	☒

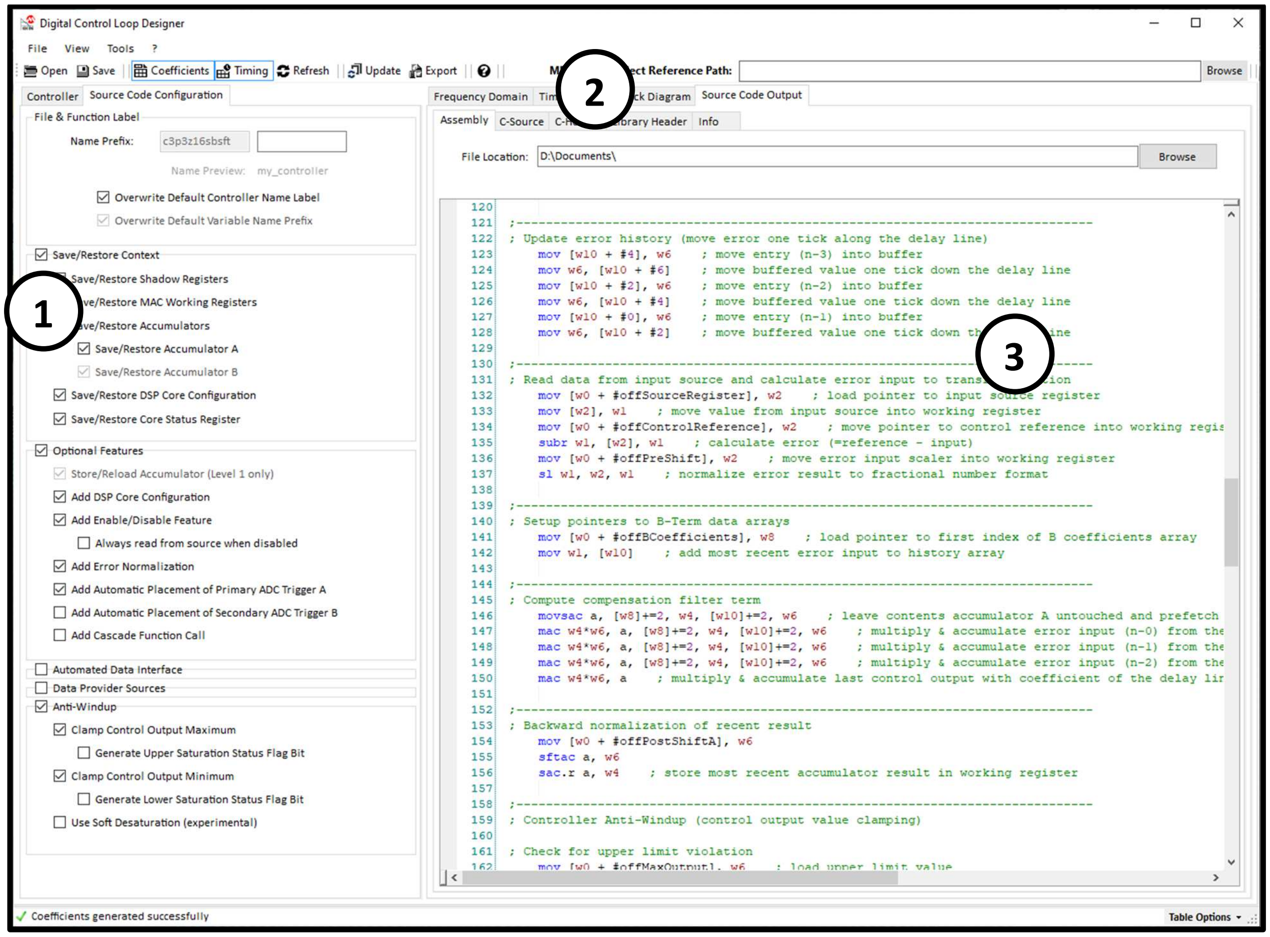
(1): This is an optional property which will only be available when the related controller option is selected. Parameters for selected options must be configured in user code

TABLE 6: NPNZ DEFAULT FUNCTIONS

The code generator also generates additional functions for extended controller block control.

Function	Parameters	Description
`npnz_Init`		**Controller Initialization Routine** Calling this routine will configure all controller properties in TABLE 5 which are marked with the symbol ☐
	`cNPNZ16b_t* controller`	NPNZ controller data structure holding system and user configuration
`npnz_Reset`		**Control History Reset** This routine is clearing all existing error and control output history values within the respective history arrays. All values will be set to zero.
	`cNPNZ16b_t* controller`	NPNZ controller data structure holding system and user configuration
`npnz_Precharge`		**Control History Precharge** This routine is loading user defined error and control output history values into their respective history array.
	`cNPNZ16b_t* controller`	NPNZ controller data structure holding system and user configuration
	`fractional ctrl_input`	signed 16-bit number representing a static error value which should be loaded into the error history
	`fractional ctrl_input`	signed 16-bit number representing a static control output which should be loaded into the control output history
`npnz_Update`		**Execute Control Loop** This routine is calling the NPNZ controller. Each function call will execute one single control step. This routine has to be called frequently to execute continuous control (e.g. from a PWM synchronized ADC interrupt service routine or PWM interrupt) Once configured, the controller module is fully self-sustained and does not need further instructions. However, to take advantage of the digital control layer and the various options provided by DCLD, controller runtime manipulation should be performed from a higher control layer (e.g. firmware state machine)
	`cNPNZ16b_t* controller`	NPNZ controller data structure holding system and user configuration

4.0 CODE GENERATOR OUTPUT WINDOW



File & Function Label

Name Prefix: c3p3z16sbsft

Name Preview: my_controller

- [✓] Overwrite Default Controller Name Label
- [✓] Overwrite Default Variable Name Prefix

- [✓] Save/Restore Context
  - Save/Restore Shadow Registers
  - ve/Restore MAC Working Registers
  - ve/Restore Accumulators
    - [✓] Save/Restore Accumulator A
    - [✓] Save/Restore Accumulator B
  - [✓] Save/Restore DSP Core Configuration
  - [✓] Save/Restore Core Status Register
- [✓] Optional Features
  - [✓] Store/Reload Accumulator (Level 1 only)
  - [✓] Add DSP Core Configuration
  - [✓] Add Enable/Disable Feature
    - [ ] Always read from source when disabled
  - [✓] Add Error Normalization
  - [✓] Add Automatic Placement of Primary ADC Trigger A
  - [ ] Add Automatic Placement of Secondary ADC Trigger B
  - [ ] Add Cascade Function Call
- [ ] Automated Data Interface
- [ ] Data Provider Sources
- [✓] Anti-Windup
  - [✓] Clamp Control Output Maximum
    - [ ] Generate Upper Saturation Status Flag Bit
  - [✓] Clamp Control Output Minimum
    - [ ] Generate Lower Saturation Status Flag Bit
  - [ ] Use Soft Desaturation (experimental)

File Location: D:\Documents\

```

;------------------------------------------------------------------------------
; Update error history (move error one tick along the delay line)
    mov [w10 + #4], w6      ; move entry (n-3) into buffer
    mov w6, [w10 + #6]      ; move buffered value one tick down the delay line
    mov [w10 + #2], w6      ; move entry (n-2) into buffer
    mov w6, [w10 + #4]      ; move buffered value one tick down the delay line
    mov [w10 + #0], w6      ; move entry (n-1) into buffer
    mov w6, [w10 + #2]      ; move buffered value one tick down the delay line

;------------------------------------------------------------------------------
; Read data from input source and calculate error input to transfer function
    mov [w0 + #offSourceRegister], w2     ; load pointer to input source register
    mov [w2], w1     ; move value from input source into working register
    mov [w0 + #offControlReference], w2     ; move pointer to control reference into working regis
    subr w1, [w2], w1     ; calculate error (=reference - input)
    mov [w0 + #offPreShift], w2     ; move error input scaler into working register
    sl w1, w2, w1     ; normalize error result to fractional number format

;------------------------------------------------------------------------------
; Setup pointers to B-Term data arrays
    mov [w0 + #offBCoefficients], w8     ; load pointer to first index of B coefficients array
    mov w1, [w10]     ; add most recent error input to history array

;------------------------------------------------------------------------------
; Compute compensation filter term
    movsac a, [w8]+=2, w4, [w10]+=2, w6     ; leave contents accumulator A untouched and prefetch
    mac w4*w6, a, [w8]+=2, w4, [w10]+=2, w6     ; multiply & accumulate error input (n-0) from the
    mac w4*w6, a, [w8]+=2, w4, [w10]+=2, w6     ; multiply & accumulate error input (n-1) from the
    mac w4*w6, a, [w8]+=2, w4, [w10]+=2, w6     ; multiply & accumulate error input (n-2) from the
    mac w4*w6, a     ; multiply & accumulate last control output with coefficient of the delay lir

;------------------------------------------------------------------------------
; Backward normalization of recent result
    mov [w0 + #offPostShiftA], w6
    sftac a, w6
    sac.r a, w4     ; store most recent accumulator result in working register

;------------------------------------------------------------------------------
; Controller Anti-Windup (control output value clamping)

; Check for upper limit violation
    mov [w0 + #offMaxOutput], w6     ; load upper limit value
```

✓ Coefficients generated successfully

Figure 8: Code Generator Output View

TABLE 7: CODE GENERATOR OUTPUT VIEW DESCRIPTION

No	Description
1	Control loop / Code generator configuration option catalog
2	Source code output tab controls (access to output windows of assembly and c-code modules)
3	Source code output window

The built-in code generator of DCLD updates the generated source code in real time while the user makes changes to configurations. The generated code is displayed in individual, separated output windows for assembly and C-code modules, where the code can be reviewed and edited[1][2].

[1] changes made by the user may get overwritten by the generator without warning (see Code Generator Settings)
[2] code editor has no compiler support

The Source Code View covers multiple sub windows for every generated code module. The generated control library source code provides four different files:

- **Optimized Assembly Code**
  All runtime functions are generated as optimized assembly routines. These routines read data from and write data to a data structure (`cNPNZ16b_t`), which holds all parameters and pointers to Special Function Registers (SFRs) and user defined variables used by the library. This data is loaded into the data structure by the C-domain initialization function. Depending on code generator options selected, additional information will be written to the data structure, from which C-domain application code can gain access (e.g. status bits, most recent calculation results, etc.)
- **C-Source File**
  The C-source file contains the static default set of filter coefficients, number scaling constants and the data structure initialization function of this individual controller.

PLEASE NOTE

The C source initialization routine only initializes the digital filter coefficients and number scaling settings.
Controller/system-specific parameters like anti-windup thresholds, source and target registers, ADC trigger registers and offsets must be set in user code.

Please review chapter 3.4 NPNZ Object Configuration for more information.

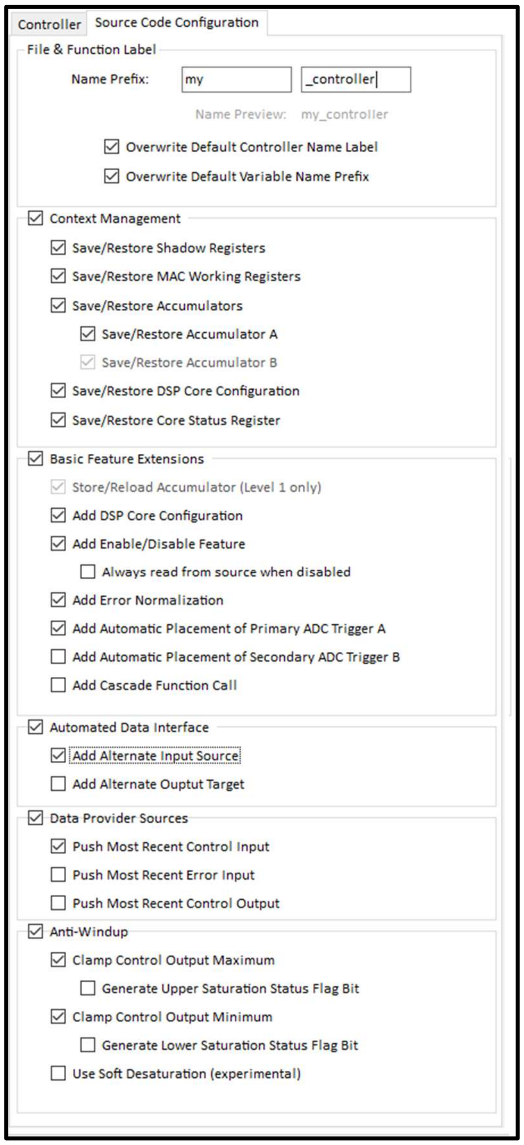
- **C-Header File**
  The C-header file holds all public variable and function declarations of this individual controller, making them accessible from throughout the user firmware.
- **C-Library Header File**
  The library header contains all generic declarations of the `cNPNZ16b_t` data structure, status bits and related global defines. This file only needs be added once per project. All declarations will be used by all individually configured controllers.

PLEASE NOTE

The `cNPNZ16b_t` data structure can be used for all 16-bit based coefficient types and number scaling methods. When a fast floating point (`ffloat`) format is used to scale coefficients, its recommended to declare controllers of the type `cNPNZ3216b_t` instead.

Both data structures have the exact same elements. Thus, switching scaling modes from simpler bit-shift scaling modes to fast floating point won't impact other sections of the user code-

5.0 CODE GENERATOR OPTIONS



Controller | Source Code Configuration

File & Function Label

Name Prefix: my _controller

Name Preview: my_controller

[✓] Overwrite Default Controller Name Label
[✓] Overwrite Default Variable Name Prefix

[✓] Context Management
- [✓] Save/Restore Shadow Registers
- [✓] Save/Restore MAC Working Registers
- [✓] Save/Restore Accumulators
  - [✓] Save/Restore Accumulator A
  - [✓] Save/Restore Accumulator B
- [✓] Save/Restore DSP Core Configuration
- [✓] Save/Restore Core Status Register

[✓] Basic Feature Extensions
- [✓] Store/Reload Accumulator (Level 1 only)
- [✓] Add DSP Core Configuration
- [✓] Add Enable/Disable Feature
  - [ ] Always read from source when disabled
- [✓] Add Error Normalization
- [✓] Add Automatic Placement of Primary ADC Trigger A
- [ ] Add Automatic Placement of Secondary ADC Trigger B
- [ ] Add Cascade Function Call

[✓] Automated Data Interface
- [✓] Add Alternate Input Source
- [ ] Add Alternate Ouptut Target

[✓] Data Provider Sources
- [✓] Push Most Recent Control Input
- [ ] Push Most Recent Error Input
- [ ] Push Most Recent Control Output

[✓] Anti-Windup
- [✓] Clamp Control Output Maximum
  - [ ] Generate Upper Saturation Status Flag Bit
- [✓] Clamp Control Output Minimum
  - [ ] Generate Lower Saturation Status Flag Bit
- [ ] Use Soft Desaturation (experimental)

Figure 9: Code Generator Options

The code generator offers several options helping to tailor the way code is implemented in the firmware, add/remove required features and optimize the overall timing to application specific needs.

The available code generation options are grouped in six major categories:

- **File & Function Label**
  customize names of objects, variables and functions in multi-loop systems allowing multiple controllers to coexist in firmware
- **Context Management / Save/Restore Context**
  optimize interrupt latency
- **Basic Feature Extensions**
  add/remove standard features
- **Automated Data Interface**
  reassign/swap inputs and outputs during runtime
- **Data Provider Sources**
  automate distributing of data across firmware
- **Anti-Windup Configuration**
  add/adjust controller output limits

Options, like the Context Management or some of the Basic Feature Extensions, allow designers to optimize the generated code library for different dsPIC® device generations or to account for XC16 compiler features or custom device configurations used. Other options like the Basic Feature Extensions, Automated Data Interface, Data Provider Sources and Anti-Windup are generic and can be applied across all devices as needed/desired.

The following sections provide more detailed information on individual options and information of possible use cases.

5.1 FILE & FUNCTION LABEL

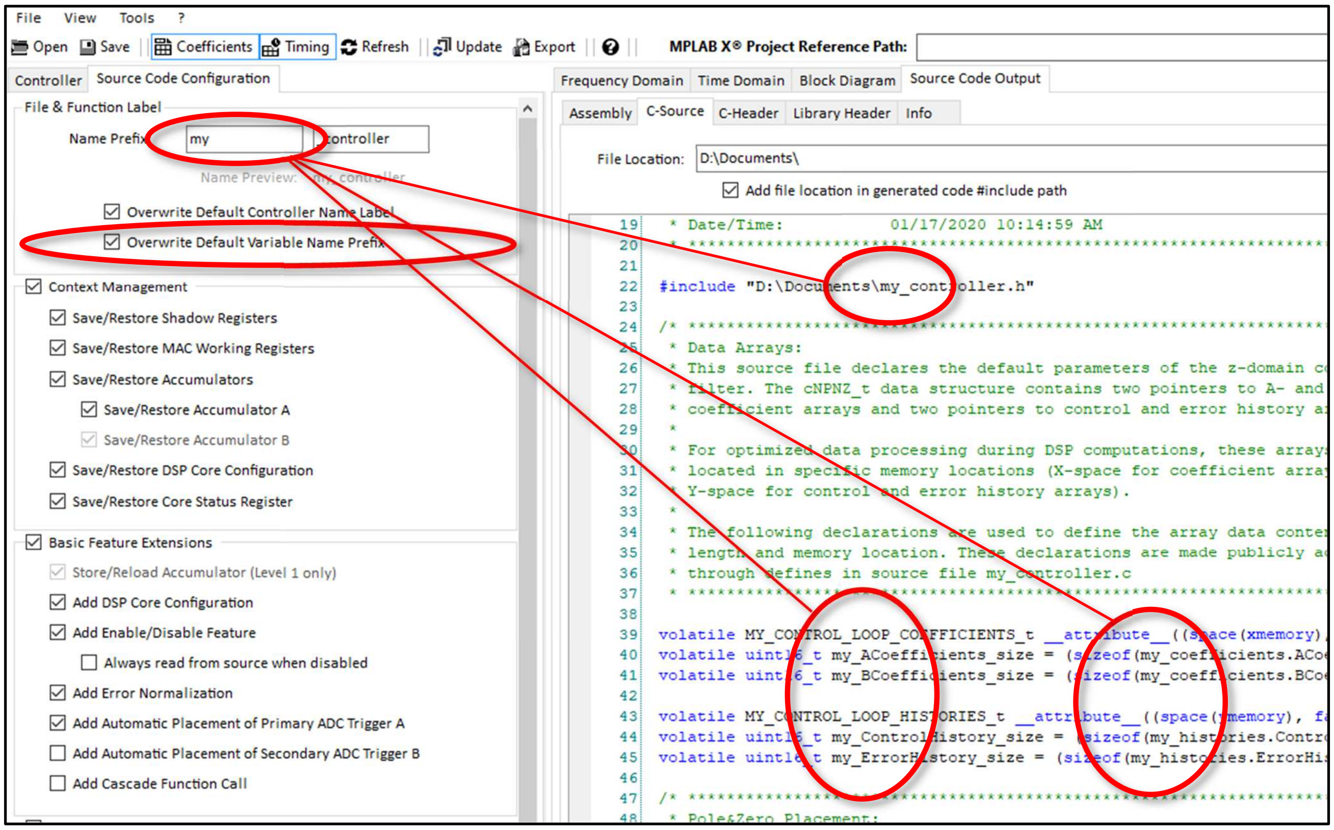
The control loop object, related variables, function call labels and file names are using a unified Control Loop Label Prefixes and Labels specified here. DCLD initially provides default labels for controller name and variable declarations when a new configuration is created. However, these labels may not be unique or may not reflect the function the control loop serves within the application code.

Therefore, it's recommended to customize these labels to improve readability and compatibility with user application code. Two custom prefixes are provided of which the first (Variable Name Prefix) will be added to all variable and data type declarations as well as to the controller name and file name. The second (Controller Name Label) will be added to the controller name and file name only.

- **Overwrite Default Variable Name Prefix**
  When selected, this option will replace the name prefix of all global variable and data type declarations and will be added to the controller name and file names, assigning them to a specific controller block configured by this application.

• PLEASE NOTE

User-defined labels are mandatory when building multi-loop systems to prevent naming conflicts between file names, variables, function calls and data objects.



File & Function Label

Name Prefix: my | controller

Name Preview: my_controller

- [✓] Overwrite Default Controller Name Label
- [✓] Overwrite Default Variable Name Prefix

[✓] Context Management

- [✓] Save/Restore Shadow Registers
- [✓] Save/Restore MAC Working Registers
- [✓] Save/Restore Accumulators
  - [✓] Save/Restore Accumulator A
  - [✓] Save/Restore Accumulator B
- [✓] Save/Restore DSP Core Configuration
- [✓] Save/Restore Core Status Register

[✓] Basic Feature Extensions

- [✓] Store/Reload Accumulator (Level 1 only)
- [✓] Add DSP Core Configuration
- [✓] Add Enable/Disable Feature
  - [ ] Always read from source when disabled
- [✓] Add Error Normalization
- [✓] Add Automatic Placement of Primary ADC Trigger A
- [ ] Add Automatic Placement of Secondary ADC Trigger B
- [ ] Add Cascade Function Call

File Location: D:\Documents\

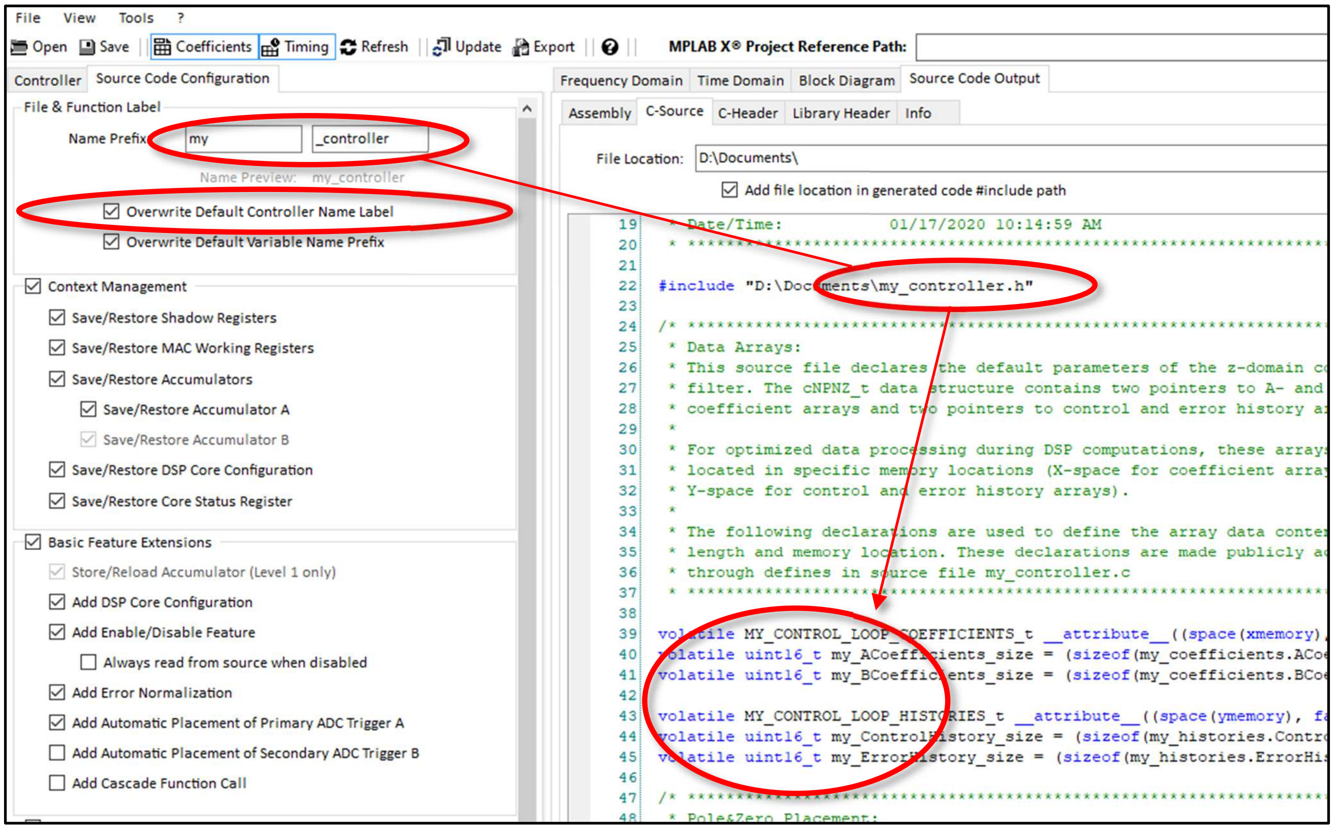
[✓] Add file location in generated code #include path

```
 * Date/Time:           01/17/2020 10:14:59 AM
 * ******************************************************************

#include "D:\Documents\my_controller.h"

/* ******************************************************************
 * Data Arrays:
 * This source file declares the default parameters of the z-domain c
 * filter. The cNPNZ_t data structure contains two pointers to A- and
 * coefficient arrays and two pointers to control and error history a
 *
 * For optimized data processing during DSP computations, these array
 * located in specific memory locations (X-space for coefficient arra
 * Y-space for control and error history arrays).
 *
 * The following declarations are used to define the array data conte
 * length and memory location. These declarations are made publicly a
 * through defines in source file my_controller.c
 * ******************************************************************

volatile MY_CONTROL_LOOP_COEFFICIENTS_t __attribute__((space(xmemory)
volatile uint16_t my_ACoefficients_size = (sizeof(my_coefficients.ACo
volatile uint16_t my_BCoefficients_size = (sizeof(my_coefficients.BCo

volatile MY_CONTROL_LOOP_HISTORIES_t __attribute__((space(ymemory), f
volatile uint16_t my_ControlHistory_size = (sizeof(my_histories.Contr
volatile uint16_t my_ErrorHistory_size = (sizeof(my_histories.ErrorHi

/* ******************************************************************
 * Pole&Zero Placement:
```

Figure 10: Assigning user-specific names for variables and objects

- **Overwrite Default Controller Name Label**
  When selected, this option will combine the Variable Name Prefix and Controller Name Label and replace the automatically generated label prefix used for file names, controller object and function calls.

  This label is always available and can be used with the automatically created default Variable Name Prefix. However, it is recommended to exclusively use user defined labels to prevent naming conflicts with user application code modules.



File & Function Label

Name Prefix: my _controller

Name Preview: my_controller

☑ Overwrite Default Controller Name Label

☑ Overwrite Default Variable Name Prefix

☑ Context Management

- ☑ Save/Restore Shadow Registers
- ☑ Save/Restore MAC Working Registers
- ☑ Save/Restore Accumulators
  - ☑ Save/Restore Accumulator A
  - ☑ Save/Restore Accumulator B
- ☑ Save/Restore DSP Core Configuration
- ☑ Save/Restore Core Status Register

☑ Basic Feature Extensions

- ☑ Store/Reload Accumulator (Level 1 only)
- ☑ Add DSP Core Configuration
- ☑ Add Enable/Disable Feature
  - ☐ Always read from source when disabled
- ☑ Add Error Normalization
- ☑ Add Automatic Placement of Primary ADC Trigger A
- ☐ Add Automatic Placement of Secondary ADC Trigger B
- ☐ Add Cascade Function Call

File Location: D:\Documents\

☑ Add file location in generated code #include path

```
 * Date/Time:          01/17/2020 10:14:59 AM
 * ****************************************************************

#include "D:\Documents\my_controller.h"

/* ****************************************************************
 * Data Arrays:
 * This source file declares the default parameters of the z-domain c
 * filter. The cNPNZ_t data structure contains two pointers to A- and
 * coefficient arrays and two pointers to control and error history a
 *
 * For optimized data processing during DSP computations, these array
 * located in specific memory locations (X-space for coefficient arra
 * Y-space for control and error history arrays).
 *
 * The following declarations are used to define the array data conte
 * length and memory location. These declarations are made publicly a
 * through defines in source file my_controller.c
 * ****************************************************************

volatile MY_CONTROL_LOOP_COEFFICIENTS_t __attribute__((space(xmemory),
volatile uint16_t my_ACoefficients_size = (sizeof(my_coefficients.ACo
volatile uint16_t my_BCoefficients_size = (sizeof(my_coefficients.BCo

volatile MY_CONTROL_LOOP_HISTORIES_t __attribute__((space(ymemory), f
volatile uint16_t my_ControlHistory_size = (sizeof(my_histories.Contr
volatile uint16_t my_ErrorHistory_size = (sizeof(my_histories.ErrorHi

/* ****************************************************************
 * Pole&Zero Placement:
```

Figure 11: Assigning user-specific name for controller and files

Please observe the C-source file generator output to see how name prefixes change.

5.2 Save/Restore Context

High speed control systems in SMPS are purely interrupt driven, triggered by PWM, ADC or timer peripherals several 100 thousand times per second. Thus, interrupt latency is a major performance aspect. Optimizing interrupt latency mainly depends on the amount of context information which needs to be saved and restored by the CPU when an interrupt occurs. As context management vary over dsPIC® device generations and compiler versions, these options allow the manual management of context registers used by the controller. For example, dsPIC33FJ DSCs only have one set of shadow registers where working registers (WREGs) need to be pushed to and popped from RAM individually. dsPIC33EP, in comparison, offer additional sets of alternate working registers where the recent context can be swapped out in a single CPU cycle. dsPIC33CH and dsPIC33CK have even further extended features including DSP accumulators and the core configuration register.

Depending on basic device configurations, such as `#pragma config CTXTn` configuration bits, and compiler features used, such as interrupt service routine attributes `context` or `naked`, the need for saving and restoring context information can be reduced or turned off entirely. By selecting specific options from this option list, code for saving and restoring will be added/removed from the generated assembly code file.

PLEASE NOTE

All these settings need to be configured individually using dedicated SFRs and configuration bits. These configuration steps are not covered by this code generator.

Please refer to the specific device data sheet and available code examples for details on how to configure your individually generated code module.

- **Shadow Registers**
  covering working registers `WREG0` to `WREG3` only. These registers usually hold function parameters like the start address of the **`cNPNZ16b_t`** data structure, intermediate results of calculation steps and pointers to memory addresses to access data in the **`cNPNZ16b_t`** data structure and related settings.
- **MAC Working Registers**
  covering working registers WREG4 through WREG10 used for the filter computation.
- **Accumulators**
  The generated controller library will always start from a cleared accumulator and leave the result in the accumulator after the filter computation. The control loop therefore *will not be affected* by other code modules changing the contents the accumulators. In return, however, control libraries will inevitably override contents of one or both accumulators.

  If the DSP core is used by other application code modules which rely on keeping the contents of the DSP accumulators alive, it is necessary to save DSP accumulator contents before and restore the contents after the loop filter computation by enabling ***Save/Restore Accumulators*** options.

  Please note:
  The usage of accumulators depends on the controller type and scaling mode selected. Accumulator save/restore options will only be available for accumulators used by the generated code.

- **CPU and DSP Core Status Register (SR)**
  The core status register may hold information about active calculation status bits and may therefore be affected by computations run by the control library. If any additional application code module may rely on this information, this register needs to be saved and restored before and after the control code is executed.

  The generated control code does not rely on specific contents of the core status register.
- **CPU and DSP Core Configuration Register (CORCON)**
  The control library computation requires a specific DSP configuration to run the control code most efficiently. If the DSP is used by any other application code module, which may use a different core configuration, this register needs to be saved, changed and restored.

  Please note:
  When the core configuration register is used with different settings by multiple application code modules and this register is saved and restored, please also enable option ***Add Core Configuration*** in ***5.3 Basic Feature Extensions***.

  If no other application code uses the DSP, the core can be configured only once during device startup and this context management option can be turned off.

5.3 Basic Feature Extensions

This section can be used to add specific, generic features to the control code, which will be embedded in the assembly code for most efficient execution.

- **Add Core Configuration (CORCON)**
  This option may need to be selected if other application code modules are using the DSP with different configurations (see *5.2 Save/Restore Context, CPU and DSP Core Configuration Register*).

  Please note:
  If the DSP is only used by the control loop libraries or the DSP configuration can be shared across the entire application, it is recommended to configure the core in a separated initialization routine executed during startup rather than changing the contents of the core configuration in every control loop execution call.
- **Add Enable/Disable Switch**
  When enabled, this option will add a control bit to the status word of the **`cNPNZ16b_t`** data structure used to enable and disable the execution of the controller code.

  This enable bit will be checked before every execution of the controller update library function. When disabled, the control code will be bypassed, and no data will be read nor written. When disabled, the histories will be frozen to their latest state, the last control output will remain as a constant, no ADC buffer reads and no output anti-windup (if selected) will be performed.

  Please Note:
  When this option is selected, the controller needs to be manually turned on (enabled) in user code by setting the control bit `status.enable = 1`.

- **Always read form source when disabled**
  As stated above, by adding the Enable/Disable feature and the controller is in a disabled state, the control code is bypassed, and no reads of the source register will be performed by default. However, if continuous sampling of the source *is* required even if the controller is turned off, this additional option will enforce reading the source register in off state while still bypassing the controller code.

  This feature is most useful in conjunction when ***Data Provider Sources*** are enabled (see below).
- **Add Input Normalization**
  dsPIC33FJ, dsPIC33EP, dsPIC33CH and dsPIC33CK ADC converters offer different data format options. Further, data format may be different when multiple compensators are coupled in multi-loop systems. To deal with these differences in number resolution, the input data may or may not have to be normalized during the execution of the control loop code.

  Please read the specific device data sheet for details on ADC converter data format configurations for more details.
- **Add ADC Trigger Placement**
  Control loops, which depend on a precise ADC trigger point which needs to be synchronized to varying duty cycles or periods, will need the ADC trigger to be repositioned with the control output.

  By selecting this option, the ADC trigger is placed automatically at 50% of the calculated value. Referencing to this trigger point, users can define a static offset from this relative trigger placement by using the `ADCTriggerAOffset` in the `cNPNZ16b_t` data structure.

5.4 Automated Data Interface

This option has been designed to better support complex, non-standard control tasks such as bi-directional control or the PWM output management of complex topologies like interleaved LLC resonant converters with synchronous rectification. These are only two examples of a variety of use-cases where the runtime management of multiple sources and targets is required.

By selecting one of these options, an additional control bits `swap_source` and/or `swap_target` are added to the `cNPNZ16b_t` data structure `status` word, allowing to switch between `ptrSource` and `ptrAltSource` (resp. `ptrTarget` and `ptrAltTarget`).

5.5 Data Provider Sources

These options have been added to allow other application code modules to track and monitor data only accessed by the control loop and which will either not be accessible from external code or would have to be read twice, such as voltage or current information. Instead of reading the ADC registers again, the control loop can be configured to push the most recent values automatically to user-defined variables during execution of the control code.

The control code configuration offers three data provider options:

- Most recent ADC sample
- Most recent error

- Most recent control output

These three data sources can be enabled individually. Their data receiver target needs to be configured by declaring a pointer to a user defined, global variable in the `cNPNZ16b_t` data structure

Example:

A single controller has been created to regulate the output of a power converter in voltage mode. The firmware requires access to the most output voltage for fault handling and to send the most recent value over a communication interface. The ADC trigger for the voltage loop analog input is triggered by the PWM module. The controller is called from the ADC interrupt service routine (ISR) of this analog input. To keep the total interrupt time as low as possible, it is desired to let the control loop collect and distribute the output voltage information instead of reading the ADC buffer register again.

This can be accomplished by following these steps:

- Create a global user-defined variable for the output voltage in user code (e.g. `my_vout`)
- Enable option ***Data Provider Sources*** → ***Push Most Recent Control Input***
- Assign the user variable `my_vout` as target for the newly added data provider channel by adding the following code line in user code:
  :

  ```
  my_controller.ptrDataProviderControlInput = &my_vout;
  ```

  Please Note:
  This pointer assignment needs to be executed before the controller update routine is called for the first time or an address error trap will occur.

- **Optional:**
  If the controller also uses the Enable/Disable feature but the output voltage should be sampled continuously, even if the control loop is not active, enable option
  Basic Feature Extensions → ***Always read form source when disabled***

As soon as the ADC is triggered and the control loop is called, valiable `my_vout` will automatically be updated by the control loop in the background.

5.6 Anti-Windup

Digital controllers can clamp the control output to a user defined level by overwriting the most recent computation result with user defined limits. When using number clamping, the control history will also be clamped at the defined maximum value without saturation effects known from analog control systems. When a digital controller with proper anti-windup clamping is used and the control loop hits output limits (minimum or maximum), it will be clamped there. When the system recovers, the control loop will start to respond immediately and without desaturation delay.

- **Clamp Control Output Maximum**
  The control output will be monitored and clamped to a user defined maximum value when exceeded.
- **Generate Upper Saturation Status Flag Bit**
  When the control output gets overwritten by the defined maximum value, a status bit will be set within the

status word of the controller to allow external application code modules to detect the saturation condition and respond to it accordingly. This status bit set and cleared automatically by the controller.

- **Clamp Control Output Minimum**
  The control output will be monitored and clamped to a minimum value when underrun.
- **Generate Lower Saturation Status Flag Bit**
  When the control output gets overwritten by the defined minimum value, a status bit will be set within the status word of the controller to allow external application code modules to detect the saturation condition and respond to it accordingly. This status bit set and cleared automatically by the controller.
- **Use Soft Desaturation (experimental)**
  This feature has been added to emulate desaturation the behavior or analog compensation circuits. Desaturation occurs when the compensation network and error amplifier of an analog feedback loop got saturated. This saturation usually occurs when the feedback is significantly off the reference (e.g. during short circuit conditions). Saturated analog feedback loops usually require some time to recover while digital feedback loops will only be clamped at precise, defined maximum and recover immediately.

  Although the digital anti-windup is usually superior, there might be use cases where the analog, soft desaturation characteristic may be preferred. Until these use cases have been identified, this option will be labeled (experimental).

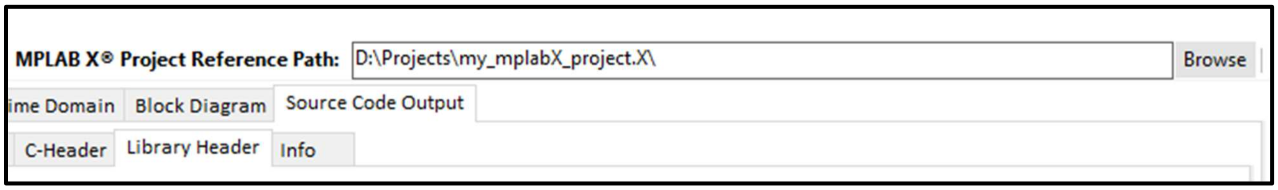
6.0 CODE GENERATION

Although the code generator is generating code in real time when configurations in DCLD are modified, it does not generate output files by default. This process must be deliberately executed by the user following these steps:

- **Specifying File Destinations**
  - **Save the Most Recent Configuration**

    DCLD is using relative file paths by default. Absolute file paths are only used when no reference directory is available or file locations are non-local (e.g. file destinations on network drives or cloud servers). To allow DCLD creating the correct relative path references, it is recommended that you save the most recent configuration to a known location, ideally, but not necessarily, inside the code project directory.
  - **Specify a reference to the MPLAB® X Project**

    DCLD has been developed an add-on tool for the MPLAB® X Integrated Development Environment (IDE). When code files are added to a project, file locations and especially header file inclusions need to be specified correctly to prevent conflicts with the C-Compiler search paths. The compiler always starts in the main project directory. All include paths are referenced to this root directory. To ensure DCLD is generating the correct include paths in the C-source and header files, the location of the MPLAB® X project root directory needs to be known.



MPLAB X® Project Reference Path: D:\Projects\my_mplabX_project.X\

Figure 12: MPLAB® X Project Root Directory Declaration

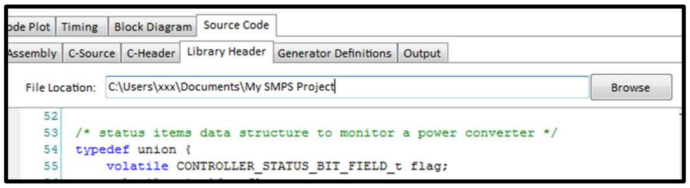
As soon as the configuration file locations and MPLAB® X project location are known, DCLD will only display relative paths.

PLEASE NOTE

Don't get confused. *File path declarations* for the code generator reference to the location of the *DCLD configuration file* while *Include Path declarations* in source files reference to the *MPLAB® X project location*

  - **Specify the Target Directory for *Every* Code File**

    Figure 13 shows the file location entry on top of every generated code file (assembly source, C-source, C-header and library header). Use this entry text box to declare the path to the directory in which the generated code file should be located.



File Location: C:\Users\xxx\Documents\My SMPS Project

```

/* status items data structure to monitor a power converter */
typedef union {
    volatile CONTROLLER_STATUS_BIT_FIELD_t flag;
```

Figure 13: Source Code File Target Directory Declaration

- **Generate & Export Code Files**

  Figure 14 below shows the code generator menu of DCLD. Once all file locations have been declared, code can be generated. The code generation process consists of two generation steps:

  - Update Generated Source Code (source code refresh)
  - Export Generated Files (actual file generation in specified locations)

  These two steps can be executed either one-by-one by first clicking on *Tools → Update Generated Source Code* and then on *Tools → Export Generated Files → Export Files* or can be executed in one single step when the option *Enable One-Click Export* is enabled.

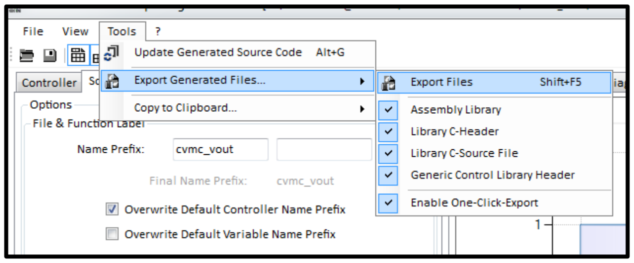


Figure 14: Code Generator Menu

If you'd like to restrict the generation of files to individual items, use the check box list within the same menu to determine which files should be generated. (see Figure 14)

The DCLD Tool Bar also allows quick access to the two generation steps Update and Export. When option *Enable One-Click Export* is enabled, one single click on *Export* will perform Update and Export sequentially.

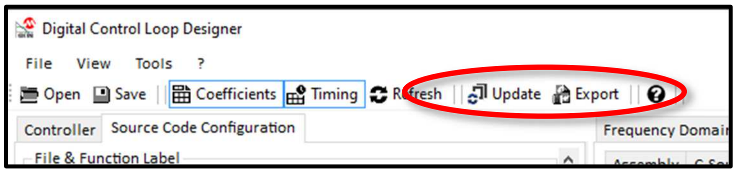


Figure 15: Code Generator Tool Bar

PLEASE NOTE

The file location declarations also support relative file paths. The root path to which relative target paths are referred to is the physical file location of the recently opened DCLD configuration file.

7.0 USING DCLD WITH MPLAB® X IDE

When installing DCLD on a Windows® computer, the setup program will associate the file type *.dcld with the Digital Control Loop Designer application executable DCLD.exe.

When you use this tool to create a control library for your dsPIC® project, the DCLD configuration file can be included in your MPLAB® X project files to ease access by allowing to open the tool from inside the MPLAB® X Integrated Development Environment (IDE).

- **Adding DCLD Configuration Files to X Project**

  The recommended procedure to add DCLD configuration files to your project is to place them in the *Important Files* folder, which is automatically created with the new project. This folder is also the home of the *Makefile* used by compiler and linker to build the project.

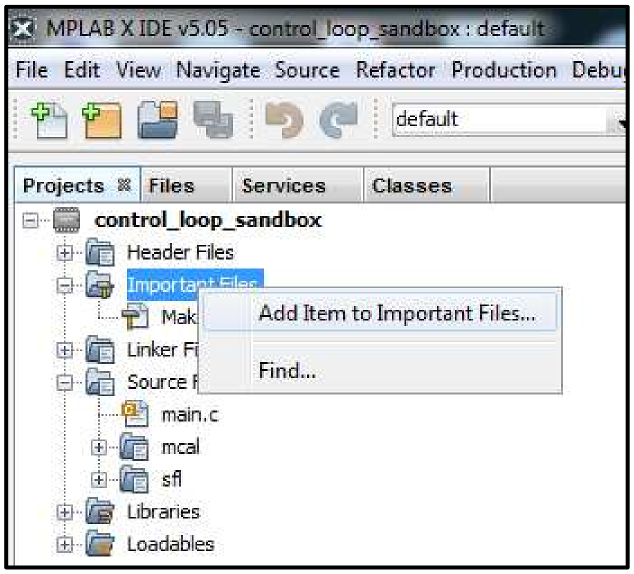
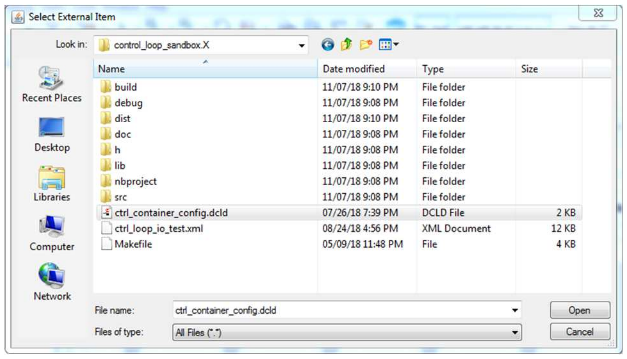

MPLAB X IDE v5.05 - control_loop_sandbox : default
File Edit View Navigate Source Refactor Production Debu
default
Projects Files Services Classes
control_loop_sandbox
Header Files
Important Files
Mak
Linker Fi
Source F
main.c
mcal
sfl
Libraries
Loadables
Add Item to Important Files...
Find...

Figure 16: Adding DCLD Configuration Files to a Project

Right-click on the *Important Files* folder in the project manager and select *Add Item to Important Files*.



Look in: control_loop_sandbox.X

Name	Date modified	Type	Size
build	11/07/18 9:10 PM	File folder	
debug	11/07/18 9:08 PM	File folder	
dist	11/07/18 9:10 PM	File folder	
doc	11/07/18 9:08 PM	File folder	
h	11/07/18 9:08 PM	File folder	
lib	11/07/18 9:08 PM	File folder	
nbproject	11/07/18 9:08 PM	File folder	
src	11/07/18 9:08 PM	File folder	
ctrl_container_config.dcld	07/26/18 7:39 PM	DCLD File	2 KB
ctrl_loop_io_test.xml	08/24/18 4:56 PM	XML Document	12 KB
Makefile	05/09/18 11:48 PM	File	4 KB

File name: ctrl_container_config.dcld
Files of type: All Files (*.*)

Figure 17: Select the DCLD Configuration File

From the File Browser dialog, select the DCLD configuration file which should be added to the project and click *Open*.

The selected DCLD configuration file will now be shown in the Important Files folder in the Project Manager. You can now open and access DCLD from the Project Manager view in MPLAB® X.

If you'd like to add multiple configurations for more than one control loop, repeat the described process until all control loop configurations for this project have been added to the project.

- **Opening DCLD from MPLAB® X Project Manager**

  When a control loop needs to be reconfigured during the development process, you can open the DCLD GUI directly from MPLAB® X Project Manager by following these steps (see Figure 18):

  - Right-click on the DCLD configuration file in the MPLAB® X Project Manager
  - Click *Open in System*
    This will open your saved DCLD configuration in the DCLD GUI where you can modify your configuration.

  When your edits to the settings are complete, click on *Export* to update the control loop project files. MPLAB® X will immediately recognize the externally changed files and refresh them inside the editor window. The project can then be immediately built without further steps. The DCLD GUI can remain open to make further adjustments, if necessary.

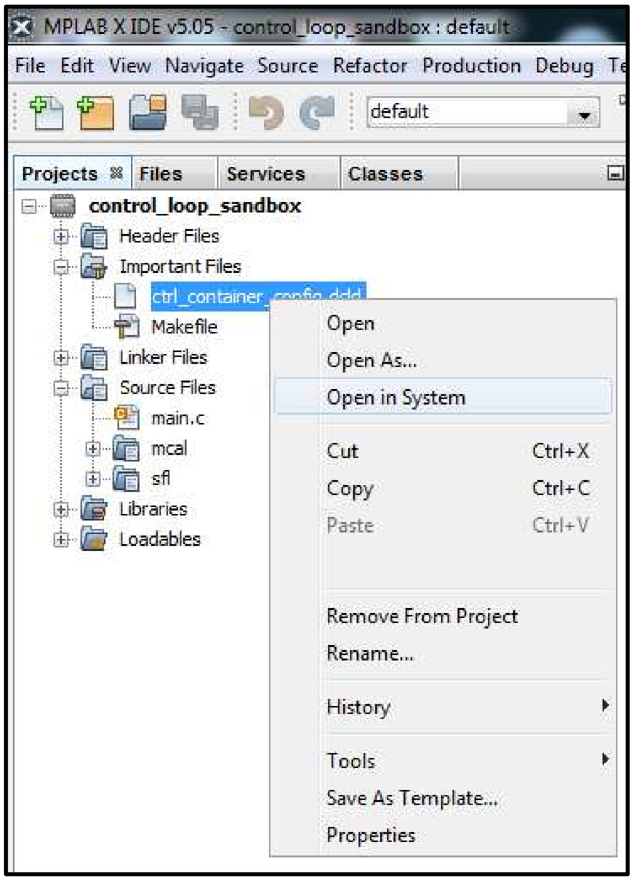


Figure 18: Opening DCLD from the MPLAB X IDE

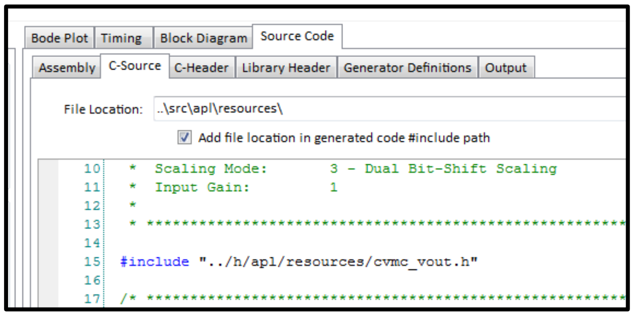
PLEASE NOTE

When DCLD generates and exports code files, previous version will be overwritten without warning. Any manual changes you may have made to these files will be lost. However, the MPLAB® X Editor offers a History feature, which can be used to restore previous code sections if files got overwritten accidentally.

- **File Locations and Include Paths**

  Generated header files are associated by `#inlude` pre-compiler directives at the beginning of the C-source file and C-header file. In some projects it may be required to include the user-specified file location to this `#inlude` pre-compiler directive.

  This is achieved by selecting the *Add file location in generated code #include path* option right below the File Location text box to be found on top of each code generator output tab. (see Figure 19)



File Location: ..\src\apl\resources\

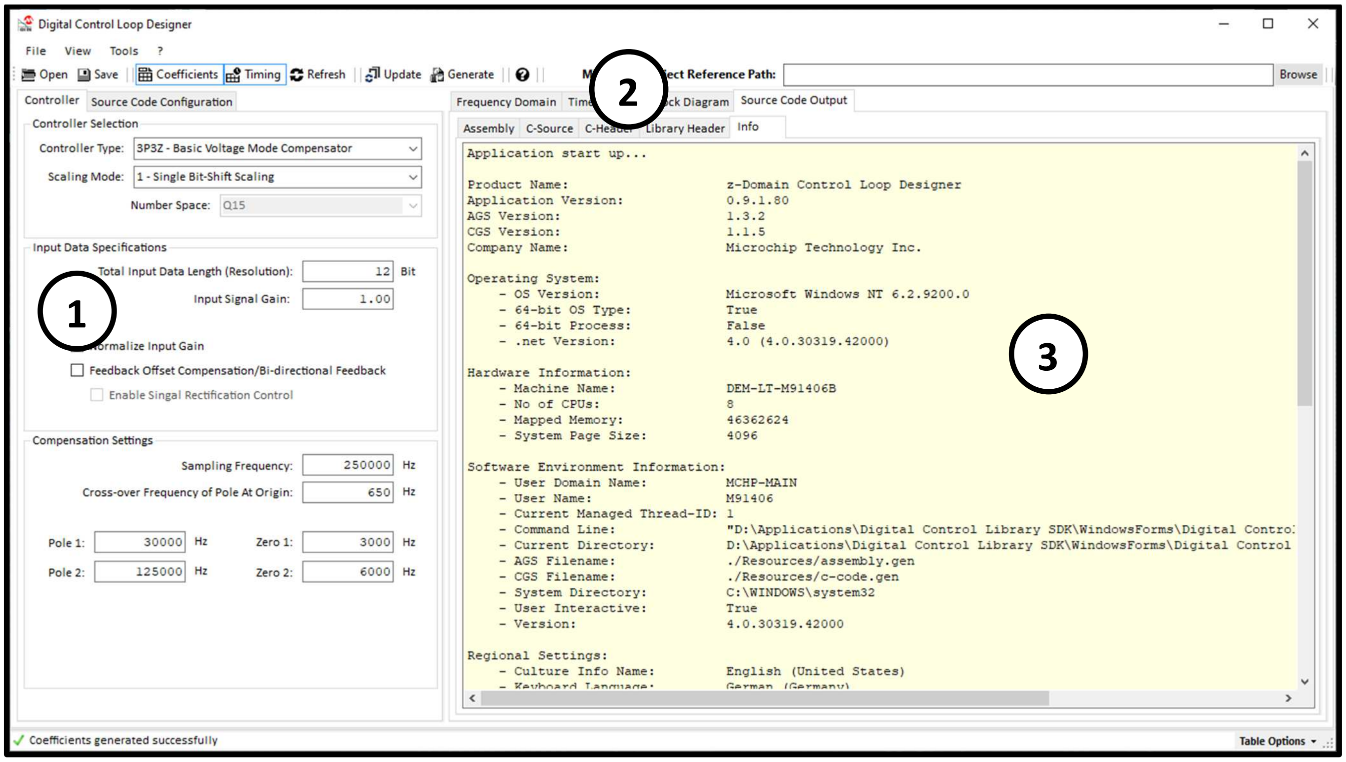
☑ Add file location in generated code #include path

```
 *  Scaling Mode:       3 - Dual Bit-Shift Scaling
 *  Input Gain:         1
 *
 * ********************************************************

#include "../h/apl/resources/cvmc_vout.h"

/* ********************************************************
```

Figure 19: Optimizing file locations

Please review section *6.0 Code Generation* of this user guide for more information of file references.

8.0 APPLICATION INFORMATION / TROUBLESHOOTING



Controller Selection

Controller Type: 3P3Z - Basic Voltage Mode Compensator
Scaling Mode: 1 - Single Bit-Shift Scaling
Number Space: Q15

Input Data Specifications

Total Input Data Length (Resolution): 12 Bit
Input Signal Gain: 1.00
ormalize Input Gain
☐ Feedback Offset Compensation/Bi-directional Feedback
☐ Enable Singal Rectification Control

Compensation Settings

Sampling Frequency: 250000 Hz
Cross-over Frequency of Pole At Origin: 650 Hz
Pole 1: 30000 Hz
Zero 1: 3000 Hz
Pole 2: 125000 Hz
Zero 2: 6000 Hz

```
Application start up...

Product Name:                  z-Domain Control Loop Designer
Application Version:           0.9.1.80
AGS Version:                   1.3.2
CGS Version:                   1.1.5
Company Name:                  Microchip Technology Inc.

Operating System:
    - OS Version:              Microsoft Windows NT 6.2.9200.0
    - 64-bit OS Type:          True
    - 64-bit Process:          False
    - .net Version:            4.0 (4.0.30319.42000)

Hardware Information:
    - Machine Name:            DEM-LT-M91406B
    - No of CPUs:              8
    - Mapped Memory:           46362624
    - System Page Size:        4096

Software Environment Information:
    - User Domain Name:        MCHP-MAIN
    - User Name:               M91406
    - Current Managed Thread-ID: 1
    - Command Line:            "D:\Applications\Digital Control Library SDK\WindowsForms\Digital Contro
    - Current Directory:       D:\Applications\Digital Control Library SDK\WindowsForms\Digital Control
    - AGS Filename:            ./Resources/assembly.gen
    - CGS Filename:            ./Resources/c-code.gen
    - System Directory:        C:\WINDOWS\system32
    - User Interactive:        True
    - Version:                 4.0.30319.42000

Regional Settings:
    - Culture Info Name:       English (United States)
    - Keyboard Language:       German (Germany)
```

✓ Coefficients generated successfully

Figure 20: Output Window

TABLE 8: APPLICATION INFORMATION OUTPUT WINDOW DESCRIPTION

No	Description
1	Control loop / Code generator configuration option catalog
2	Source code output tab controls (access to output windows of assembly and c-code modules)
3	Output Window view

The output window is an additional software debugging tool helping to verify proper and reliable output results and offers additional information for troubleshooting software and platform issues. It lists system information, folder settings, user inputs and internal messages during execution.

9.0 COMMON USE CASES AND APPLICATION GUIDANCE

Code file organization in complex control system may require have to be very different from project to project. DCLD is offering multiple options to tailor the way code files are generated and named to give designers a much flexibility as possible.

9.1 MULTIPLE CONTROLLERS USING THE SAME ASSEMBLY CODE

The DCLD code generator is creating the following four essential library files by default: (see Chapter 6.0 Code Generation)

- o Assembly Library File
- o Library C-Source File
- o Library C-Header Files
- o Generic Library Header File

When creating a single-loop control system, all four files are required for a proper implementation of the control loop library code. In multi-loop systems, however, it might be beneficial to limit the number of generated files to save memory space in the target device.

A most common use-case are average current controllers consisting of an outer voltage loop and an inner current loop. Usually both controllers are of Type II (2P2Z in digital) so that only one assembly library but multiple sets of coefficients as well as controller history arrays would be required.

In this case it's possible to create two independent DCLD configuration files (one for the voltage loop and one for the current loop).

One of both configuration files needs to use the default setting, which will export all four library files. For every additional loop based on the same compensator type, using the same number scaling method and code feature options, it's only required to export the C-Source and C-Header file containing the coefficient and controller object declarations.

This can be configured individually by clicking on menu Tools → Export Generated Files. The File Export context menu offers options for every individual file to be exported or not (see Figure 21)

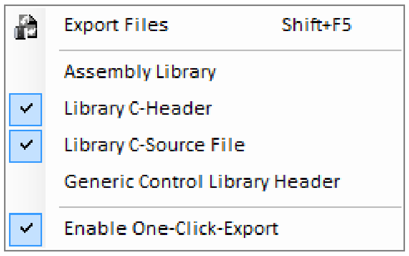


Figure 21: Excluding Files from Generation

PLEASE NOTE

Function name labels of the initially generated assembly code will be determined by the DCLD configuration which was used to generate them. When assembly files are used for multiple controllers, make sure the function calls placed in user code use the correct function name labels and hand over the correct pointer to the individual controller object.

Example:

```
my_controller_Update(&controller_A);
my_controller_Update(&controller_B);
my_controller_Update(&controller_C);
```

- **Limitations**

  Using generated code for multiple loops based on the same assembly library introduces some limitation which need to be kept in mind to prevent address errors and other undesired conflicts. Preventing these conflicts is the full responsibility of the user!

  - **Main Filter Type Implementation**
    The selected compensator type (e.g. 2P2Z, 3P3Z, 4P4Z, etc.) will be used as template to determine how many filter iterations will be executed by the assembly code library block. To make this most efficient in terms of execution time, no dynamic adjustment to different filter types is made. Thus, the generated assembly library only supports the filter type selected.
  - **Scaling Options**
    Different scaling options will equally result in incompatible code when used by different controller objects, which are not using the same number scaling format. Scaling factors, number normalization and resolution differ significantly depending on the selection made. Using controller objects configured for different scaling options therefore cannot use the same assembly library.
  - **Code Features**
    Code feature selection will have an equally vital impact on the code integrity but does not necessarily exclude multiple controller object from using the same assembly library.

    Assuming multiple controller objects are built using the same controller/filter type and number-scaling method, but one controller needs anti-windup clamping while the other controller doesn't.

    In this case it is still possible to use the same assembly library, which, however, will always execute the anti-windup code block. Thus, the second controller needs to hold reasonable thresholds in its respective data structure spaces to not get cut off by accidentally being clamped to zero.
  - **Context Save/Restore**
    As long as all controller objects are based on the same compensator/filter type and using the same number scaling method, context save/restore options should be consistent. Nevertheless, if alternate working registers (`ALTWREG`) on dsPIC33EP, dsPIC33CH or dsPIC33CK are used, it is important to verify that all control library function calls like `xxx_Update(yyy)` are called on the same interrupt priority with a properly associated `ALWREG` set. These `ALTWREG` sets can be different but must be accessible and changes to the working registers must not result in conflicts with other tasks.

9.2 ESTABISHING BI-DIRECTIONAL CONTROL SYSTEMS

Control of bi-directional converter (a.k.a. 2-Quadrant Power Supplies) is a very common application for digitally controlled power converters and are widely used in the industry in applications such as renewable energy storage devices, automotive 48V-to-12V and 400V-to-12V bus converters or smaller consumer products like USB Power Delivery source/sink devices.

Developing bi-directional power supplies require the selection of specific topologies supporting the reversal of the power transfer. Some of them may have the same transfer function in both directions, such as Phase-Shifted Full Bridge (PSFB) converters or 4-Switch Buck/Boost (4SWBB) converters. Others may have fundamentally different transfer functions in each direction such as Synchronous Buck converters, which will be turned into a Synchronous Boost converter when power transfer is reversed.

Power converter types with identical transfer functions in both directions can be controlled by one and the same control block where only minor changes may have to be made, such as assignment of alternative feedback inputs, PWM-control outputs and references. Power converter types with different transfer functions may require new sets or coefficients or maybe even entirely different compensator types with different features.

Providing detailed design guidance for each of these applications is beyond the scope of this user guide. The major focus of this section is to provide some high-level guidance on certain, dedicated feature provided by DCLD, which might be useful to solve specific design challenges in a convenient and robust way.

- **Feedback Structure and Characteristics**

  2-Quadrant Power Supplies usually offer three fundamental feedback signals:

  - Input Voltage $V_{IN\,(Port\,A)}$
  - Output Voltage $V_{OUT\,(Port\,B)}$
  - Inductor Current I_L

  When the power transfer is reversed, V_{IN} and V_{OUT} swap positions and I_L becomes negative (see Figure 22).

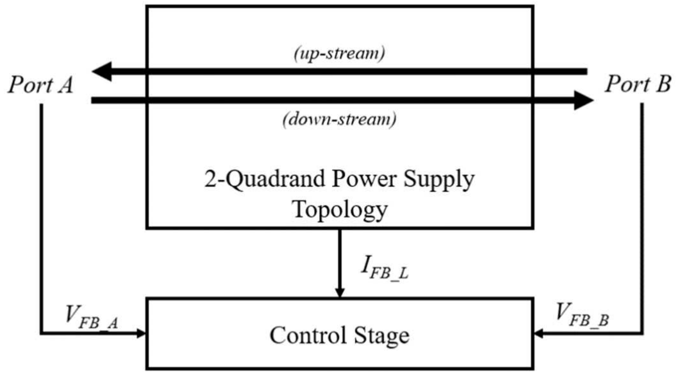


Figure 22: Bi-Directional Power Converter Block Diagram

- **Signal Offset**
  Processing bi-directional feedback signal through single-ended Analog-To-Digital Converters usually requires signal pre-conditioning lifting the zero point of the feedback signal above VSS. Typical offsets added by signal conditioning ICs, for example, are 1.65V for 3.3V devices or 2.5V for 5V devices but offsets might differ widely when discrete signal conditioning circuits are used. (see Figure 23)

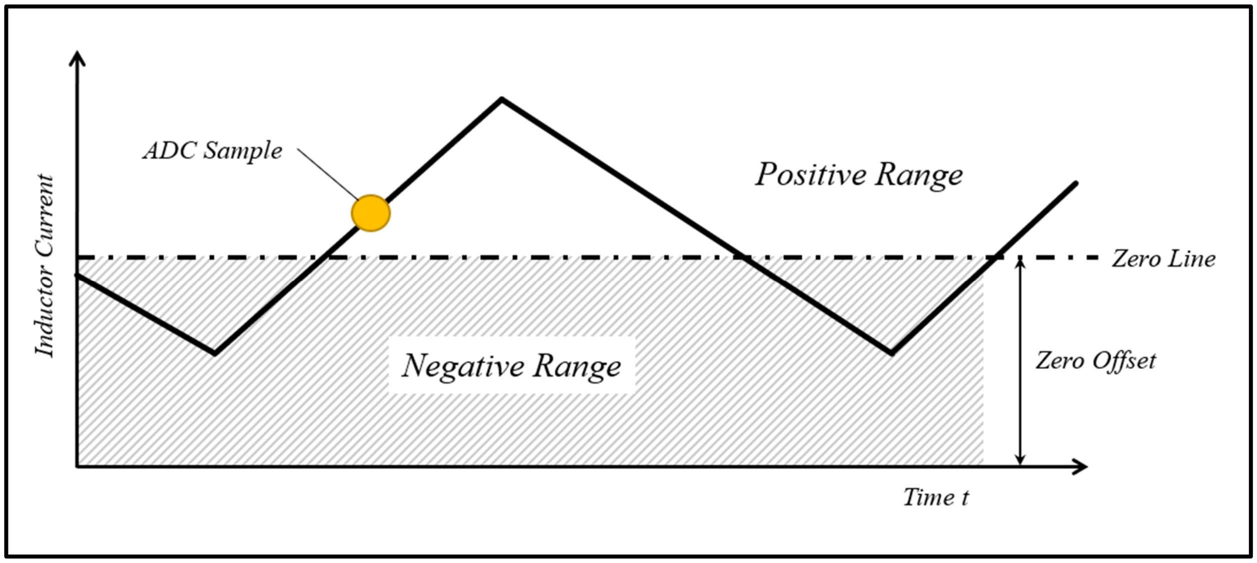


Figure 23: Bi-Directional Current Feedback Signal with Offset

Using ADCs to track analog high-speed signals is highly sensitive to the ADC trigger point location. When the trigger is not generated in perfect synchronization with the PWM signal the trigger might not occur at 50% of the rising (or falling) slope of the current and the taken sample will not represent the most recent average current (resp. will be affected by some tolerance).

By enabling option *Feedback Offset Compensation*, code will be added to the assembly routine adding the zero-point offset to the reference before processing the most recent ADC sample. Thus, the incoming data will not become negative when its value is less than the specified zero-point which would bare the risk of potentially destabilizing the control loop by feeding negative numbers into the control system effectively inverting the inverting feedback loop, turning it into a non-inverting loop.

By adding the offset to the reference, negative errors will remain negative even if the ADC value is less than the defined zero-point effectively preventing accidental inversion of the feedback loop. In cascaded control systems where an outer loop determines the reference for the inner loop, the outer loop gain is not affected by the feedback offset compensation of the inner loop. However, it’s recommended to allow the reference produced by the outer loop to become slightly negative to account for “negative glitches” caused by very likely, minor signal drifts.

- **Reversing current direction**
  The catch for the calculation engine with this signal conditioning is the resulting inversion of proportions above and below the zero line. While operating in the positive range, increasing positive currents are represented by increasing number values (direct proportional representation). While operating in the negative range, increasing negative currents are represented by decreasing number values. Although the

number representation of the voltage level of the feedback is still direct proportional, the representation of the physical domain of the absolute current level is inverted (indirect proportional representation).

As the power supply controller is based on an inverting feedback loop, inverting the proportional representation of its data input would inevitably result in an inversion of the inverting feedback loop, effectively flipping it over into a non-inverting feedback loop. As a result, the feedback loop would start amplifying instead of suppressing transients and the power supply would go unstable instantly.

This undesired behavior needs to be prevented by introducing a signal rectification at the data input of the controller.

- **Current Feedback Rectification**
  As mentioned above, when a power converter needs to change its power transfer direction, the feedback source of the outer loop needs to be swapped from what was the previous output to what was the previous input and vice versa. If the switch-over process should be seamless and smooth, the current direction also needs to be inverted at the very same moment.

  For this purpose, DCLD offers an extension to the *Feedback Offset Compensation* option called *Enable Signal Rectification Control* which allows to invert the most recent error by the `invert_input` control bit in the `cNPNZ` status word.

The following example provides some high-level guidelines of how these features supported by DCLD can be used to build a bi-directional multi-loop Average Current Mode Control (ACMC) feedback loop.

- **Designing an Average Current Mode Control (ACMC) Feedback Loop**

The most generic approach to establish a bi-directional control stage is by using Average Current Mode Control (ACMC). This control mode uses a multi-loop system consisting of one outer voltage loop and one inner current loop. The outer voltage loop is regulating for a constant output voltage by providing a dynamic current reference to the inner current loop. The output of the inner current loop then adjusts the PWM. (see Figure 24)

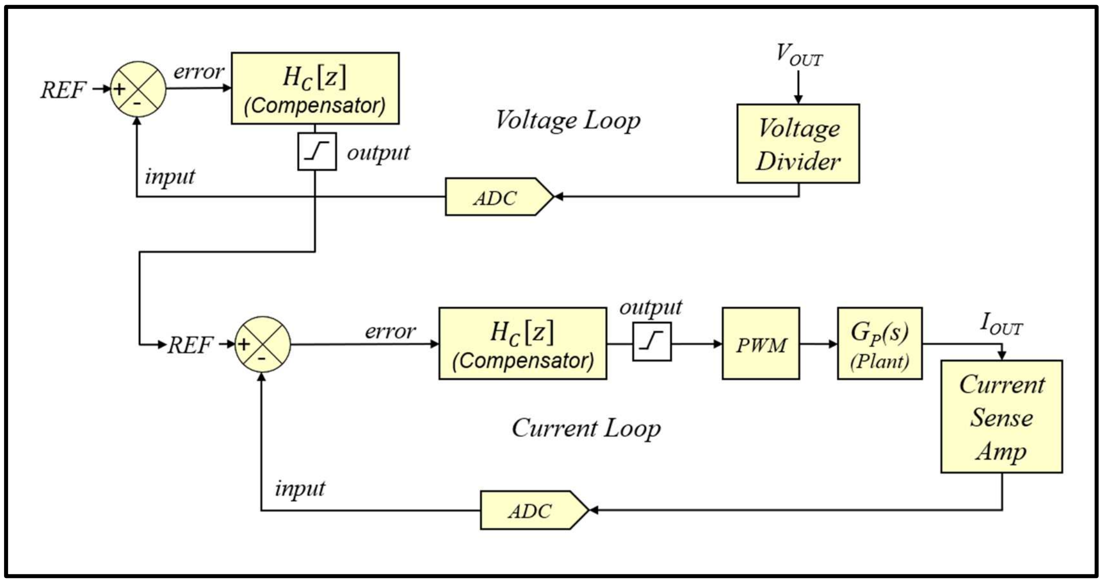


Figure 24: Standard Average Current Mode Control Feedback Loop Block Diagram

As shown in Figure 24, an ACMC feedback loop consists of two independent feedback loops. The following example provides a high-level guidance of the steps necessary to establish a dual loop ACMC feedback loop by using DCLD. However, many important aspects like device specific peripheral configuration, frequency domain design aspects or state machine design with soft-start and protection features are not covered in this example.

Using DCLD this control system would be established following these steps:

- Open DCLD and create the 2P2Z voltage loop controller `V_LOOP`, which reads the output voltage, compares it to a user defined reference variable `V_REF` and produces an output value, which is stored in the user-defined variable `I_REF`.
- Enable Anti-Windup for both, minimum and maximum output levels
- Input data scaling needs to be set to a resolution of 12-bit (ADC data width) with option *Add Error Normalization* enabled.
- Input data gain is set to the voltage divider gain calculated by $G = \frac{R2}{R1+R2}$
- Go to *Source Code Configuration* and enable option *Basic Feature Extensions → Add Enable/Disable Feature*
- Save the configuration and generate the voltage loop control code
- In user code, add code initializing the voltage loop:
  - Set data source:
    ```
    V_LOOP.ptrSource = &ADCBUF4; // ADC buffer #4 (output voltage)
    ```
  - Set data output target:
    ```
    V_LOOP.ptrTarget = &I_REF; // V_LOOP output writes to current reference
    ```
  - Set Current clamping values:
    ```
    V_LOOP.MinOutput = -200; // ADC ticks representing minimum current
    V_LOOP.MaxOutput = 1600; // ADC ticks representing maximum current
                             // without offset (!!!)
    ```
  - Set voltage reference source:
    ```
    V_LOOP.ptrControlReference = &V_REF; // Assign V_REF variable
    ```
  - Call the controller initialization routine to initialize data arrays and number scaling factors
    ```
    vloop_Init(&V_LOOP); // Call controller initialization
    ```
- Open DCLD and create the 2P2Z current loop controller `I_LOOP`, which reads the inductor current and compares it to the reference variable `I_REF` defined previously, which is continuously updated by the voltage loop as soon as the entire control loop is enabled. The output of this control loop is written to the dedicated PWM registers (e.g. Duty Cycle)
- Input data scaling needs to be set to a resolution of 12-bit (ADC data width) with option *Add Error Normalization* enabled.
- Enable option *Feedback Offset Compensation* and specify the offset `I_LOOP.InputOffset` in user code
- Input data gain is set to the current sense gain calculated by $G = R_{SHUNT} \times G_{AMP}$

- Go to *Source Code Configuration* and enable option *Basic Feature Extensions → Add Enable/Disable Feature*
- Save the configuration and generate the current loop control code
- In user code, add code initializing the current loop:
  - Set data source:
    ```
    I_LOOP.ptrSource = &ADCBUF0; // ADC buffer #0 (inductor current)
    ```
  - Set data output target:
    ```
    I_LOOP.ptrTarget = &PG1DC; // I_LOOP output writes to PWM1 Duty Cycle
    ```
  - Set Current clamping values:
    ```
    I_LOOP.MinOutput = 400; // PWM ticks representing minimum duty ratio
    I_LOOP.MaxOutput = 10000; // PWM ticks representing maximum duty ratio
    ```
  - Set current reference source:
    ```
    I_LOOP.ptrControlReference = &I_REF; // Assign I_REF variable
    ```
  - Call the controller initialization routine to initialize data arrays and number scaling factors
    ```
    i_loop_Init(&I_LOOP); // Call controller initialization
    ```
  - Go to the interrupt service routine of the output voltage ADC Channel and add the function calls of voltage and current loop controllers:
    ```
    v_loop_Update(&V_LOOP); // Call voltage loop controller
    i_loop_Update(&I_LOOP); // Call Current loop controller
    ```
  - Enable control
    ```
    V_LOOP.status.enable = true; // Enable voltage loop
    I_LOOP.status.enable = true; // Enable current loop
    ```

Note 1:

When working with current feedback signals with offset, it should be considered that the zero-line of the current amplifier device might be affected by tolerances and that the accuracy of the ADC samples are highly dependent on the accuracy of the ADC trigger placement. Both effects might result in deviations from the data sheet-value of the zero-line feedback voltage.

To ensure the feedback loops work correctly even at no load conditions, it is highly recommended to the adjust the voltage loop anti-windup minimum with some tolerance, allowing negative currents.

Note 2:

In this example voltage and current loop are daisy-chained inside the output voltage ADC interrupt service routine. When both controllers are executed at the same frequency, it is important to keep in mind that the maximum allowed frequency of current reference perturbations should be at least 6-10 times slower than the response time of the current loop. This can be accomplished by adjusting the open loop cross-over frequency of the voltage loop at least one magnitude below the open loop cross-over frequency of the current loop.

Note 3:

Daisy-chaining control loops can be simplified by adding the following features to the control loop library:

- Open the voltage loop configuration in DCLD
- Go to *Source Code Configuration* and enable option *Basic Feature Extensions → Add Cascade Function Call*
- Save the configuration and re-generate the voltage loop control code
- Add the following lines to the controller configuration of the voltage loop controller:

```
V_LOOP.CascadedFunction = (uint16_t)&i_loop_Update;
V_LOOP.CascadedFunParam= (uint16_t)&I_LOOP;
```

- Remove the current loop function call from the ADC ISR:

```
i_loop_Update(&I_LOOP); // Call Current loop controller
```

The current loop will now be automatically called by the voltage loop controller. All other settings remain untouched.

This control system is now suitable for operating the converter in one direction. However, there might be device-specific, additional parameters which have to be configured such as *Context Management Options*, *Basic Feature Extensions* and more, which are ignored for the sake of keeping this example focused on the creating and implementation process of the controller code.

- **Adding Bi-Directional Control Features**

As mentioned above, DCLD offers some features which can now be used to turn this uni-directional control system into a bi-directional feedback loop. The standard ACMC control system shown in Figure 24 only has one voltage feedback loop and only accepts positive currents.

Reversing power transfer would still be possible by re-assigning the input voltage ADC buffer as input source of the voltage loop. Although this is legitimate and would work without problems, it might be more elegant and convenient to build an input switch into the control library itself, which allows a simple switch over between the two sources using a simple control bit in the `cNPNZ` status word (See block ***A*** in Figure 25).

The second problem we have to solve is to reverse the current direction. We already established the level shifter for offset compensation by enabling the *Feedback Offset Compensation* option in *Controller → Input Data Specification*. To gain control over the current feedback signal polarity, this option is extended by enabling the second option *Enable Signal Rectification Control*. This will enable the direction control DIR shown in block ***B*** in Figure 25.

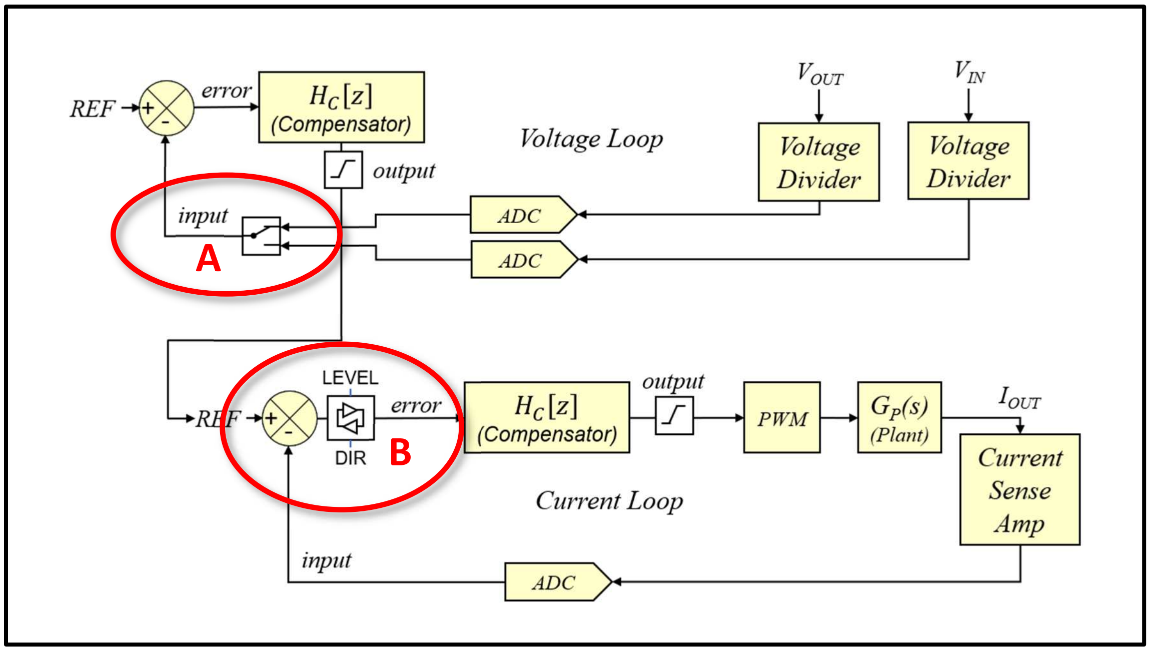


Figure 25: Bi-Directional Average Current Mode Control Feedback Loop Block Diagram

Enabling both features requires the following steps:

- Open the configuration of the voltage loop controller `V_LOOP`
- Go to *Source Code Configuration* and enable option *Automated Data Interface → Add Alternate Input Source*
- Save the configuration and generate the updated library file
- In user code, add the following code line to the controller initialization

```
V_LOOP.ptrAltSource = &ADCBUF6; // assign pointer to VIN ADC buffer
```

- Open the configuration of the current loop controller `I_LOOP`
- Go to *Controller* and enable option *Input Data Specification* → *Enable Signal Rectification Control*
- Save the configuration and generate the updated library file

Now the control loop is ready to perform a switch over power transfer directions by executing the following two code lines:

- Switch over from down-stream to up-stream operation:

```
V_LOOP.status.swap_source = true; // switch from output to input
I_LOOP.status.invert_input = true; // invert current feedback polarity
```

- Switch over from up-stream to down-stream operation:

```
V_LOOP.status.swap_source = false; // switch from output to input
I_LOOP.status.invert_input = false; // invert current feedback polarity
```

10.0 LEGAL TERMS FOR DEVELOPMENT BOARDS SOLD AND USED IN EUROPE REGARDING ZVEI REGULATIONS

IMPORTANT NOTICE TO CUSTOMERS

Boards marked as “NON-PUBLIC CONCEPT BOARD” are NOT part of Microchip’s usual development tool portfolio and no support is provided though Microchip’s common support channels. Changes may be applied without any further notice.

This specific hardware has been developed as proof-of-concept board or for training purposes for PROFESSIONAL USERS ONLY. You are not allowed to use this development board in any real application and others than professional lab environments. This board has not been certified or tested for any standards or any requirement such like EMC or safety.

CONTACT INFORMATION

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 ▪ Fax: 480-792-7277

Technical Support:	http://www.microchip.com/support
Web Address:	www.microchip.com

AMERICAS

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455
Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088
Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075
Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643
Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924
Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260
Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445
Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431
Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755
China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104
China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889
China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500
China - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189
China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431
China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470
China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205
China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066
China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393
China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760
China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118
China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256
China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130
China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123
India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632
India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513
Japan - Osaka
Tel: 81-66-152-7160
Fax: 81-66-152-9310
Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122
Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302
Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934
Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859
Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068
Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069
Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850
Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955
Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-330-9305
Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102
Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393
Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829
France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79
Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44
Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781
Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340
Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91
UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

NOTES: