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.

- **Technical Specifications:**

Minimum System Requirements:

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

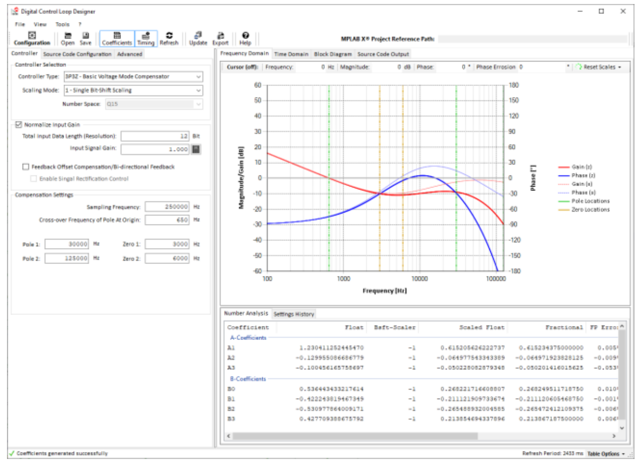


Figure 1: z-Domain Configuration Window of the Digital Control Loop Designer SDK

- **Recommended Literature:**

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

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

(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).

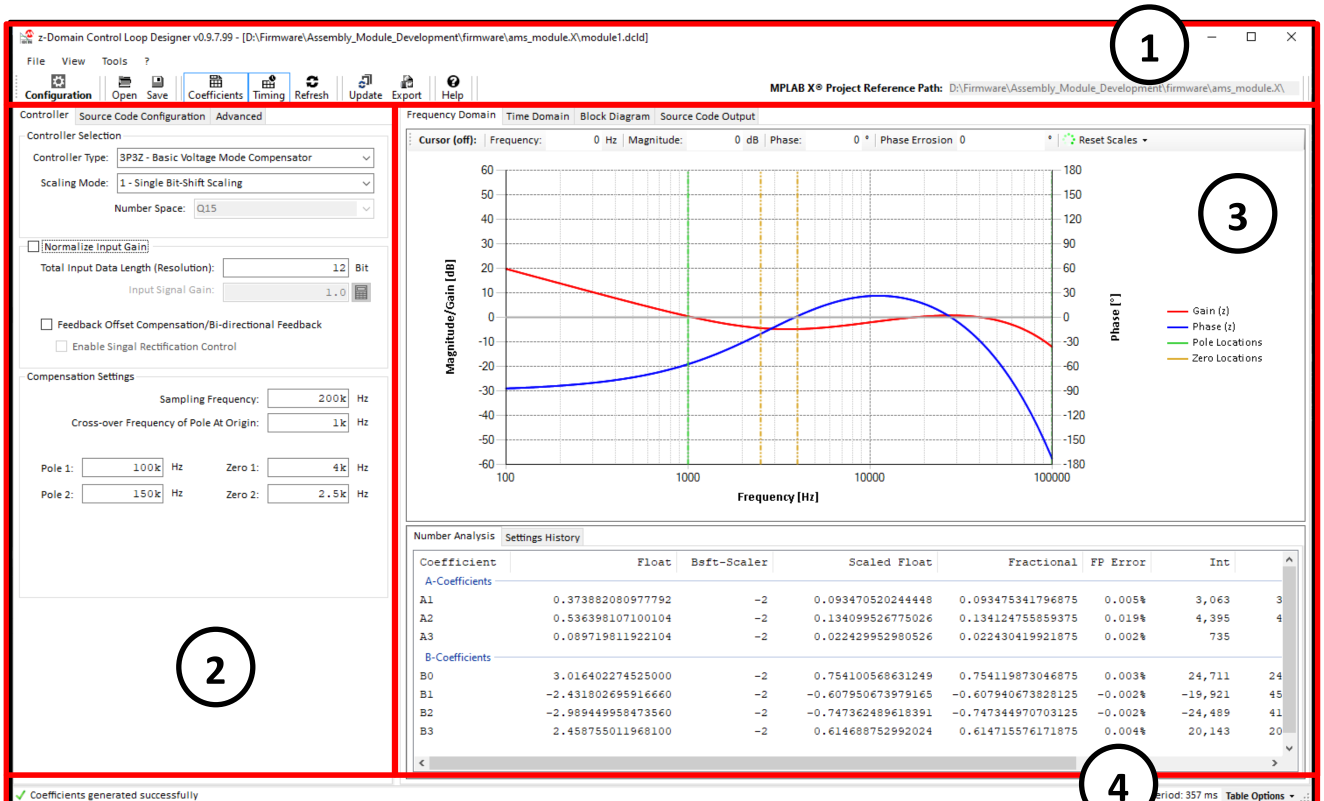


Figure 2: Digital Control Loop Designer z-Domain Controller Main Window Overview

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)

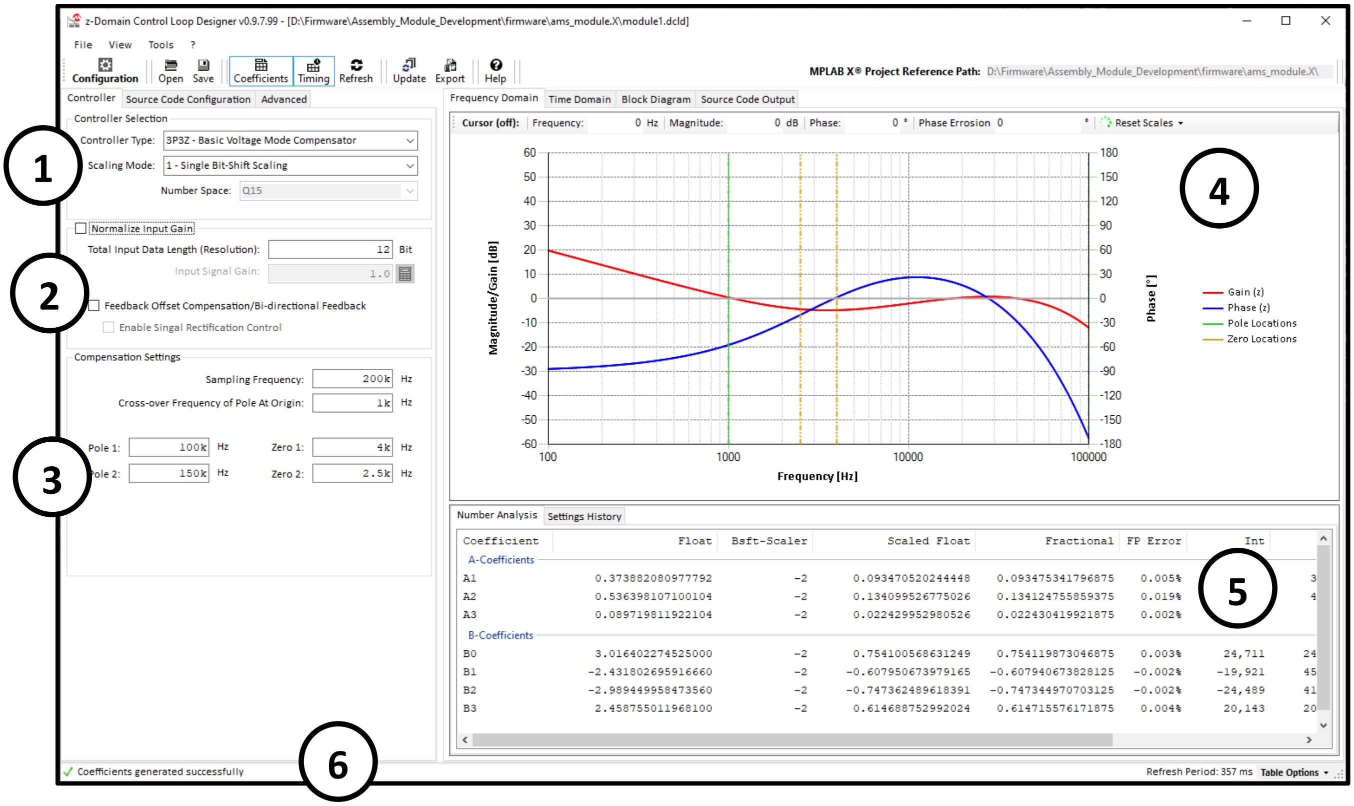


Figure 3: Digital Control Loop Designer Frequency Domain View

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)

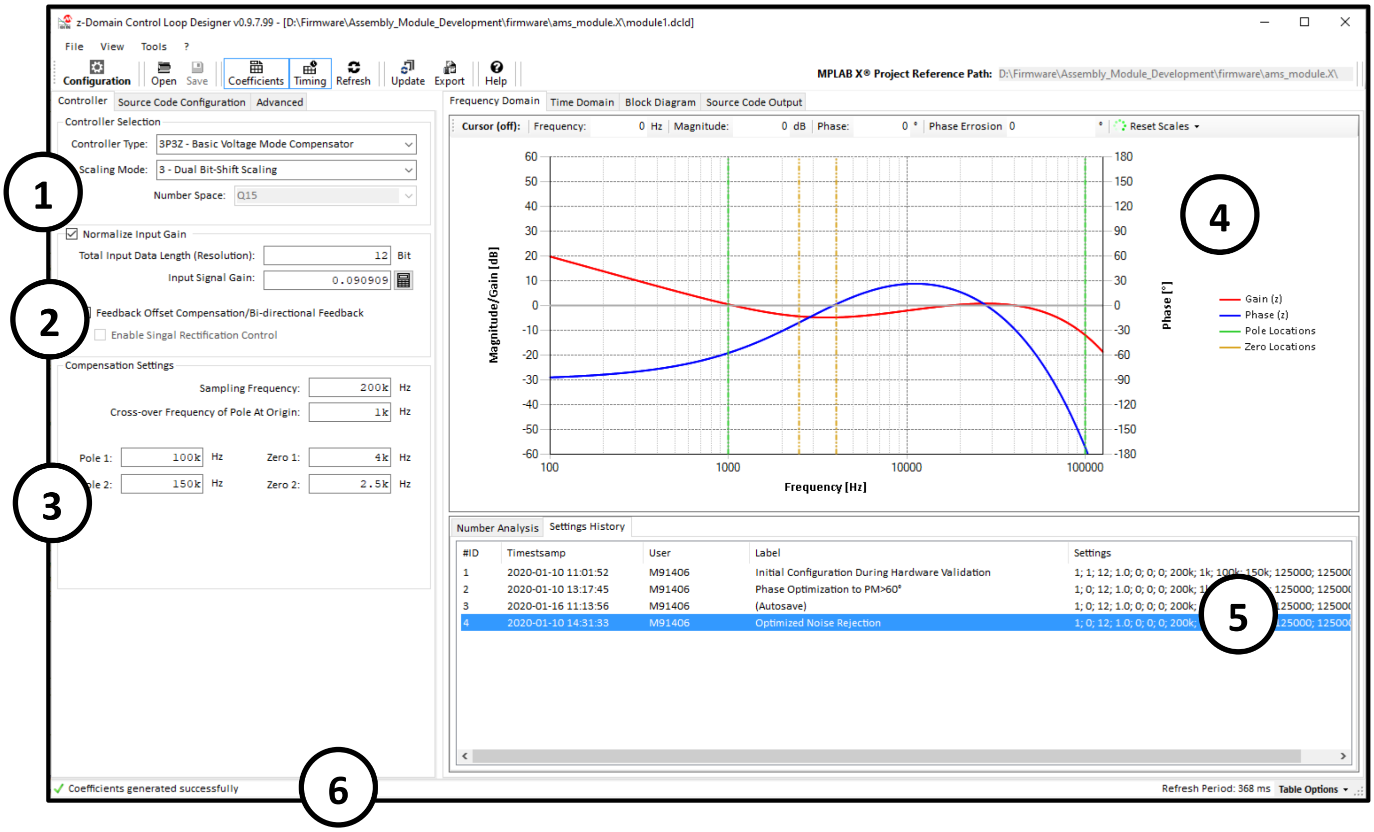


Figure 4: Digital Control Loop Designer Frequency Domain View with Workflow History

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: (Autosave); can be renamed by user by hitting key F2 on the keyboard or right-click an entry to open the 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 Nyquist-Shannon 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

$$G_{VD} = \frac{R2}{R1+R2} = \frac{1.1k\Omega}{8.2k\Omega + 1.1k\Omega} = 0.1183; \quad \text{Equation 2-1}$$

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.

$$G_{CS} = R_S \times G_{AMP} = 0.01\Omega \times \frac{20V}{V} = 0.200; \qquad \text{Equation 2-2}$$

The input gain to enter is 0.200.

- **Input Signal Gain Calculation Tool**

DCLD also offers a calculation tool for most common feedback circuits such as voltage dividers, current sense amplifiers, current sense transformers and digital sources such as input capture modules.

Please note:

The feedback gain derived by using these calculation tools is only used to normalize the feedback loop gain and does not consider frequency domain characteristics of components.

- **Voltage Divider Gain Calculator**

  Figure 5 shows the voltage divider gain calculator where the Analog-to-Digital converter reference voltage and resolution as well as upper and lower resistor value can be entered. In addition, this calculator offers an additional option for adding the gain of an operational amplifier. If no operational amplifier is used, its gain must be set to ‘1’.

  This feedback gain is calculated in V_{SENSE} / V_{SOURCE}

  The effective feedback gain is calculated and displayed at the lower right of this window. This value will be taken over into the main window by clicking the ***OK*** button.

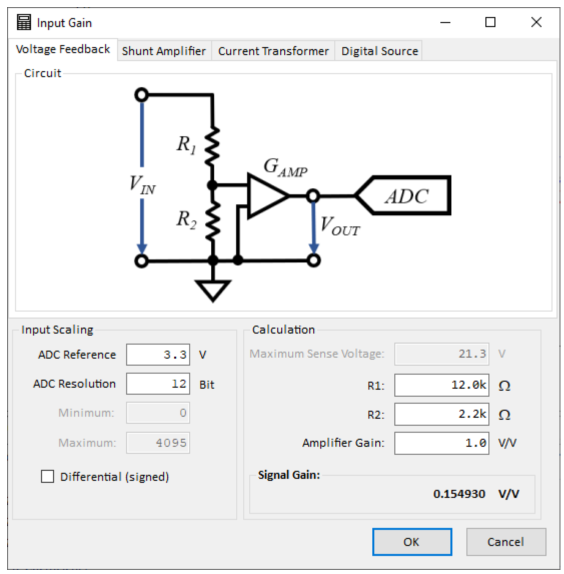


Figure 5: Voltage Divider Gain Calculator

- **Shunt Amplifier / Current Sense Amplifier**

  Figure 6 shows the shunt amplifier gain calculator. This input mask offers fields for the shunt resistor value and amplifier gain.

  This feedback gain is calculated in [V_{SENSE} / A_{SOURCE}]

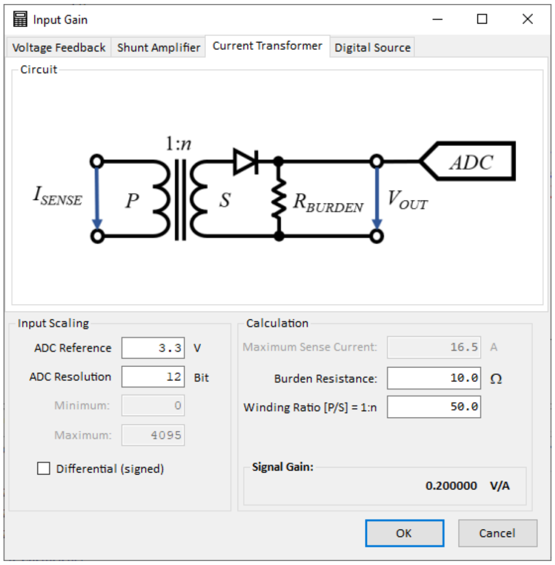


Figure 6: Shunt Amplifier Gain Calculator

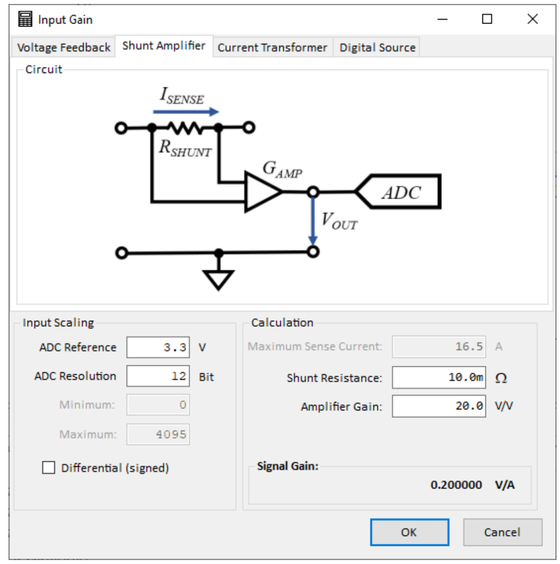


Figure 7: Current Sense Transformer Gain Calculator

- ### Current Sense Transformer

  Figure 7 shows the current sense transformer gain calculator. This input mask offers fields for the winding ratio of the transformer and the burden resistor used to transform the secondary current into voltage. This voltage level may be sampled by the ADC or being applied to the input of an analog comparator.

  This feedback gain is calculated in V_{SENSE} / A_{SOURCE}

- ### Digital Source

  Figure 8 shows the input mask for digital feedback sources. This mask supports a wide variety of sources such as input capture modules, which are often used to measure digital signals like output frequencies of a V/f converter but also applies to bare digital values received through communication interfaces.

  The only aspect of interest with any of these sources is the effective bit resolution of the maximum available range.

  Example:

  A Voltage-to-Frequency converter is used to measure a voltage signal on the other side of a galvanic barrier. The output frequency is transmitted through a digital opto-coupler and received and measured through an input capture module. The output frequency range of the V/f converter can vary between 1 kHz up to 1 MHz
  (= 1 ms to 1 µs period)
  The input capture clock is running at 50 MHz providing an

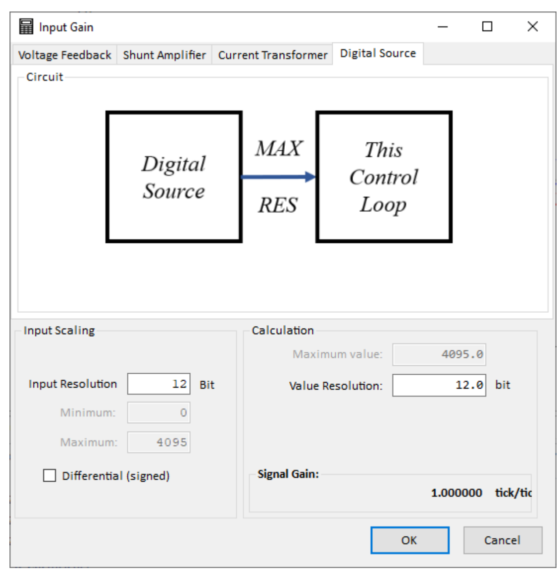


Figure 8: Digital Feedback Source Gain Calculator

effective resolution of 20 ns. The required resolution to be entered is calculated by dividing the source frequency by the time-base resolution.

In our example, the frequency of interest is the V/f converter output frequency at nominal output voltage:

$$f_{V/f} = 250\text{ kHz} \Leftrightarrow T_{V/f} = 4\text{ µs};\ T_{IC} = 20\text{ ns}$$

$$\frac{T_{V/f}}{T_{IC}} = \frac{4\text{ µ}s}{20\ ns} = 200 \approx 7.64\ bit;$$

To calculate the required gain value, this effective signal resolution must be referenced against the maximum input resolution, which is represented by the time-base range of the input capture module of 16 bit (= 65,535).

Hence, the effective feedback gain is

$$\frac{2^{Resolution\ Feedback}}{2^{Resolution\ Input\ Capture}} = 2^{(Resolution\ Feedback - Resolution\ Input\ Capture)} = 2^{7.64-16} = 2^{-8.36} \approx 0.003$$

- **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 Nyquist-Shannon 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 Nyquist-Shannon limit.

3.0 TIME DOMAIN WINDOW

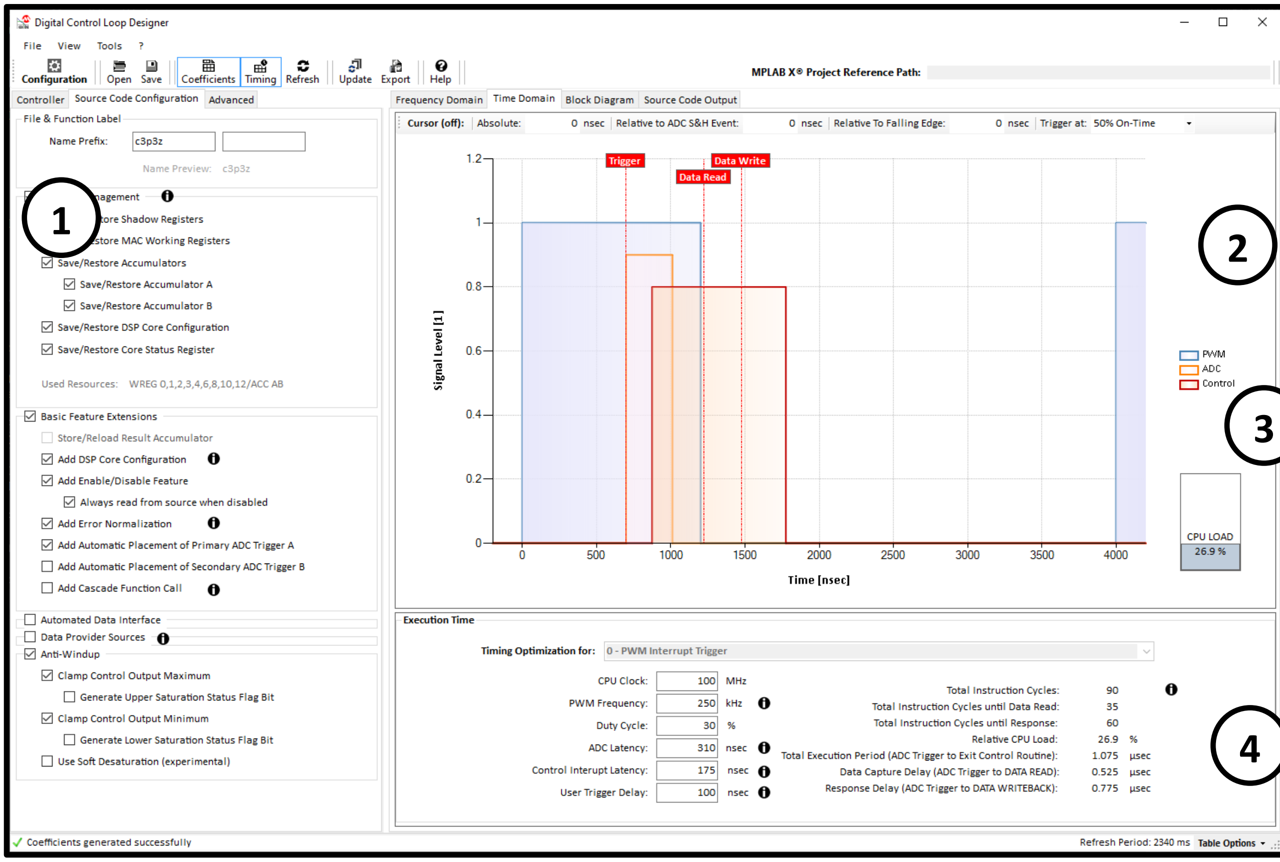


Figure 9: Code Generator Configuration and Timing Diagram

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
  - User Trigger Delay: This generic delay is usually set in software to account for signal chain delays such as propagation delays of MOSFET drivers. This signal delays require a placement optimization of the ADC trigger. As both, control loop execution and ADC trigger are in most cases controlled by the PWM time base, these delays can be added to the timing view to gain a better representation of the real runtime relations.

- **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

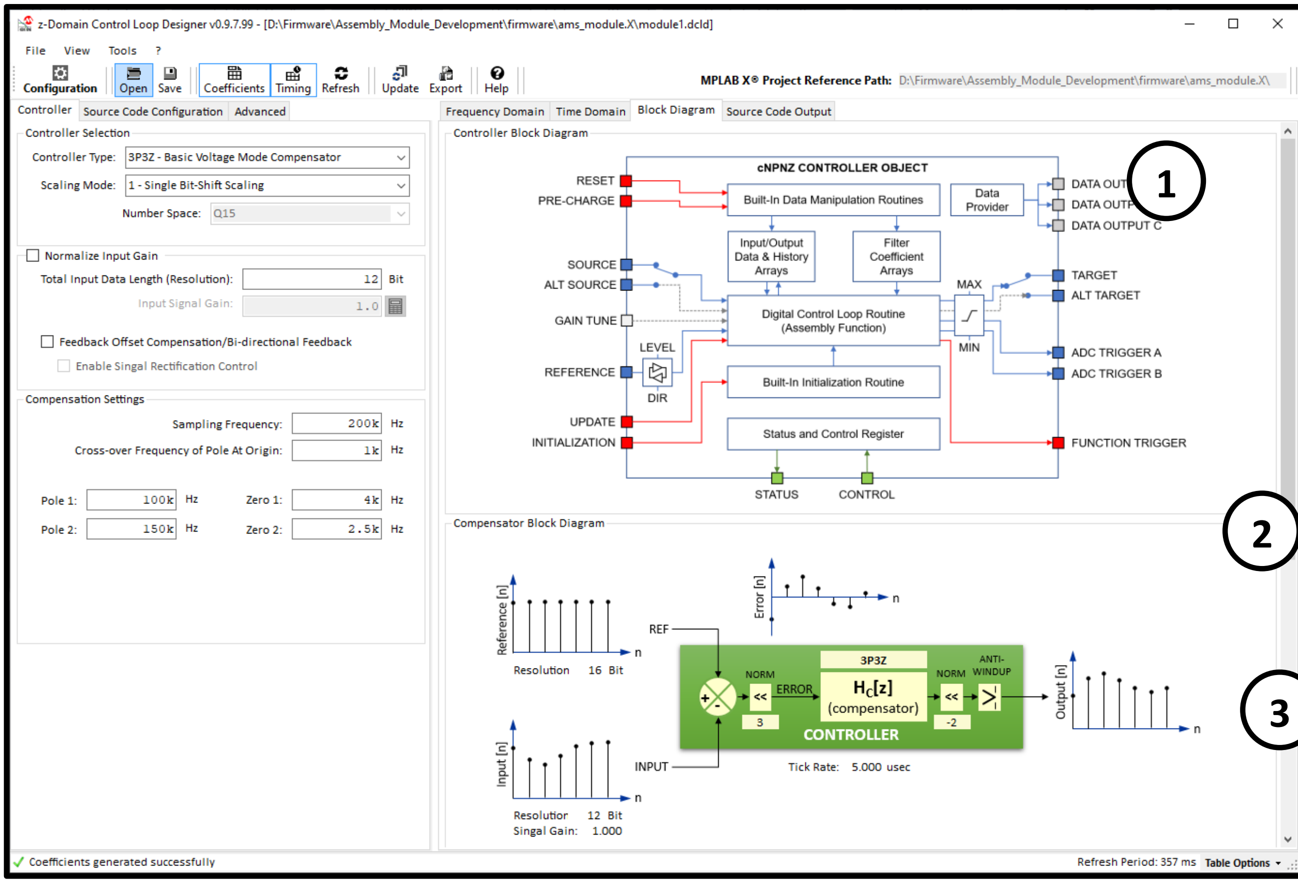


Figure 10: Block Diagram Overview

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.

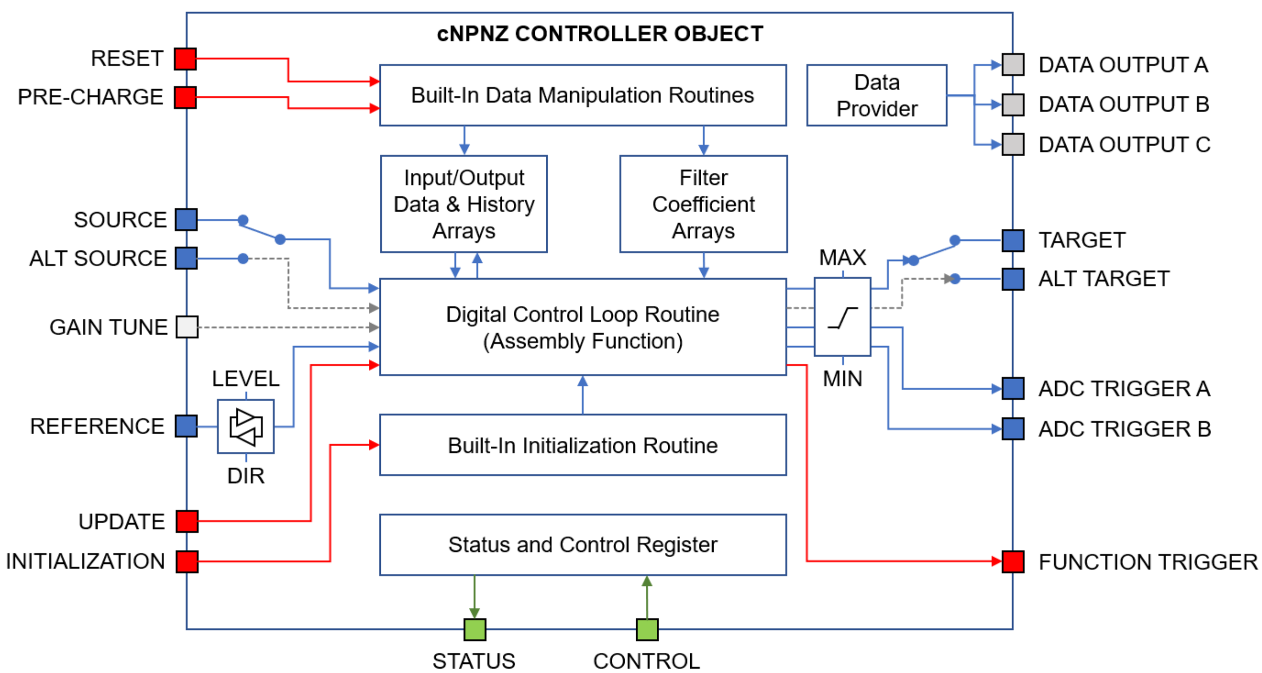


Figure 11: cNPNZ Controller Object Block Diagram

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 WINDOW**

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 11 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`	☒
`agc_enable`[1]	Bit [11]: Adaptive Gain Control Algorithm Execution Control Bit 1 = AGC factor is multiplied with feedback coefficients 0 = AGC factor is not multiplied with feedback coefficients	☒
	Bit [10:2]: (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 (Ports)		
`Source`	Primary data input register/variable	
`ptrAddress`	Pointer to data input register/variable	☒
`NormScaler`	Input signal normalization scaler (integer)	☒
`NormFactor`	Input signal normalization factor (fractional)	☒
`Offset`	Input signal offset	☒
`AltSource`[1]	Alternate data input register/variable	
`ptrAddress`	Pointer to data input register/variable	☒
`NormScaler`	Input signal normalization scaler (integer)	☒
`NormFactor`	Input signal normalization factor (fractional)	☒
`Offset`	Input signal offset	☒
`Target`	Primary data output register/variable	
`ptrAddress`	Pointer to data input register/variable	☒
`NormScaler`	Output signal/value normalization scaler (integer)	☒
`NormFactor`	Output signal/value normalization factor (fractional)	☒
`Offset`	Output signal/value offset	☒
`AltTarget`[1]	Alternate data output register/variable	
`ptrAddress`	Pointer to data input register/variable	☒
`NormScaler`	Output signal/value normalization scaler (integer)	☒
`NormFactor`	Output signal/value normalization factor (fractional)	☒
`Offset`	Output signal/value offset	☒
`ptrControlReference`	Pointer to control reference variable	☒
Compensation Filter Data Objects		
`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	☐
`ACoefficientsArraySize`	Size of the A coefficients array in X-space	☐
`BCoefficientsArraySize`	Size of the B coefficients array in X-space	☐
`ControlHistoryArraySize`	Size of the control history array in Y-space	☐
`ErrorHistoryArraySize`	Size of the error history array in Y-space	☐
`normPreShift`	Data input normalization scaler	☐
`normPostShiftA`	Data output normalization scaler A	☐
`normPostShiftB`	Data output normalization scaler B	☐
`normPostScaler`	Data output normalization factor	☐

TABLE 5: NPNZ PROPERTIES CONFIGURATION (CONTINUED)

Output Limits		
MinOutput[1]	Control output minimum value	☒
MaxOutput[1]	Control output maximum value	☒
AltMinOutput[1]	Alternate Control output minimum value	☒
AltMaxOutput[1]	Alternate Control output maximum value	☒
ADC Trigger Control		
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	☒
Cascade Trigger		
CascadedFunction[1]	Pointer to user-defined function	☒
CascadedFunParam[1]	Pointer to one 16-bit wide user-function parameter	☒
Feedback Gain Control		
agcScaler[1]	Bit-shift scaler of Adaptive Gain Modulation factor	☒
agcFactor[1]	Q15 value of Adaptive Gain Modulation factor	☒
agcMedian[1]	Q15 value of Adaptive Gain Modulation nominal operating	☒
ptrAgcObserverFunction[1]	Function Pointer to Observer update function	☒
Advanced Feature Parameters		
advParam1[1]	generic 16-bit wide, user-defined parameter #1	☒
advParam2[1]	generic 16-bit wide, user-defined parameter #2	☒
advParam3[1]	generic 16-bit wide, user-defined parameter #3	☒
advParam4[1]	generic 16-bit wide, user-defined parameter #4	☒

(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

TABLE 7: NPNZ DEFAULT FUNCTIONS (CONTINUED)

Function	Parameters	Description
`npnz_PTermUpdate`(1)		**Execute P-Term Control Loop** This routine is optional and will only be available when the advanced code option "*Use P-Term Controller for Plant Measurements*" is enabled. **Please note:** This variant of an integrator-free, proportional controller is used for plant measurement only. It is highly unstable and should never be used for regulating a power supply under normal operation. The code generator will generate a control loop using the very same code generation options selected for the main control loop, such as context management, basic feature extensions, automated data interfaces, data provider sources and anti-windup settings. Hence, this control loop can directly replace the main control loop without affecting other software instances.
	`cNPNZ16b_t* controller`	NPNZ controller data structure holding system and user configuration

(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

4.0 CODE GENERATOR OUTPUT WINDOW

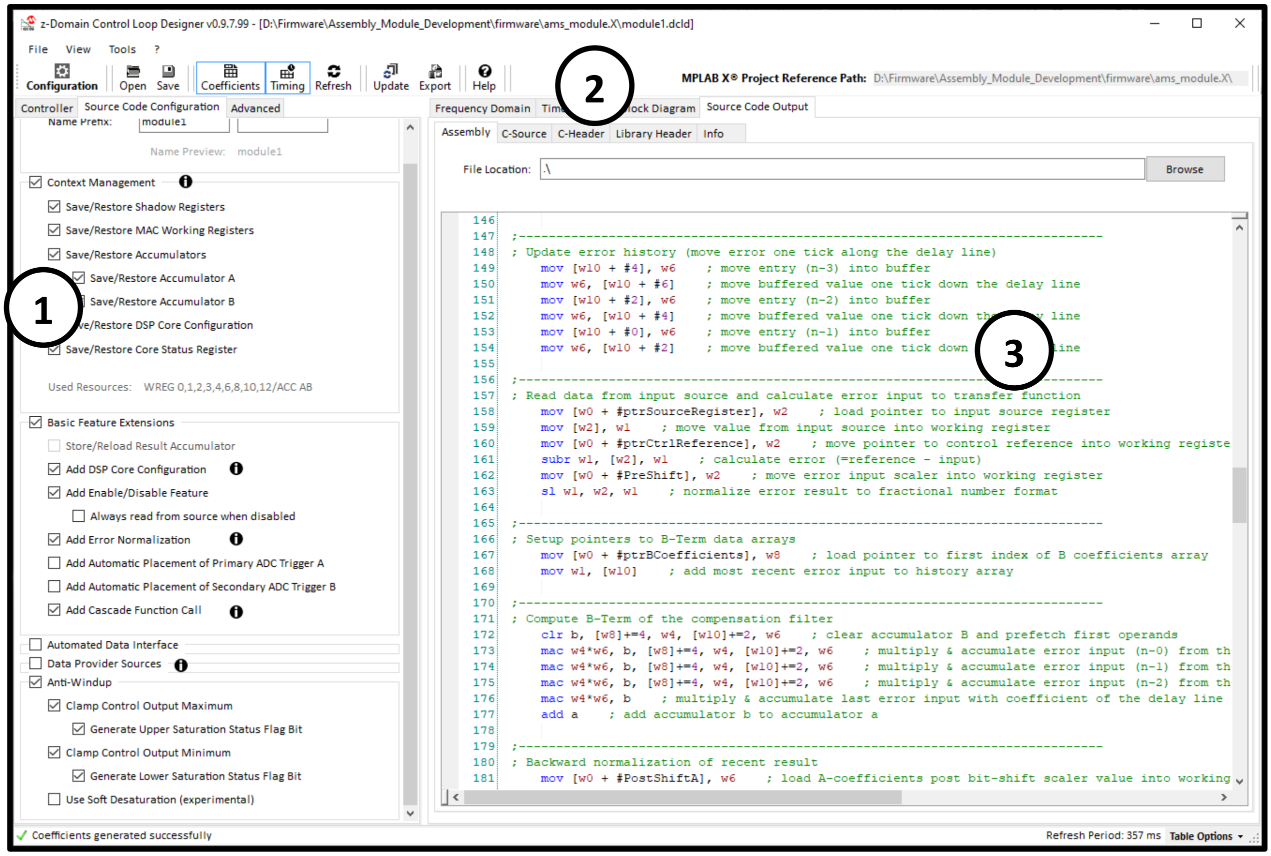


Figure 12: Code Generator Output View

TABLE 8: 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].

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

---

[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

- **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

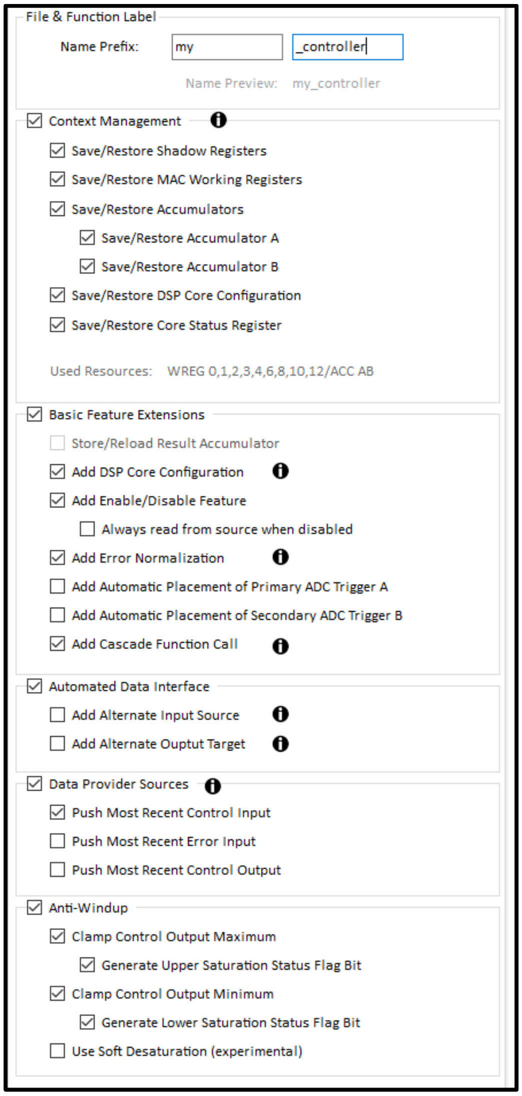


Figure 13: Code Generator Options

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 unified Control Loop Label Prefixes and Labels specified here. DCLD initially provides default labels for controller names 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.

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 variables 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.

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.

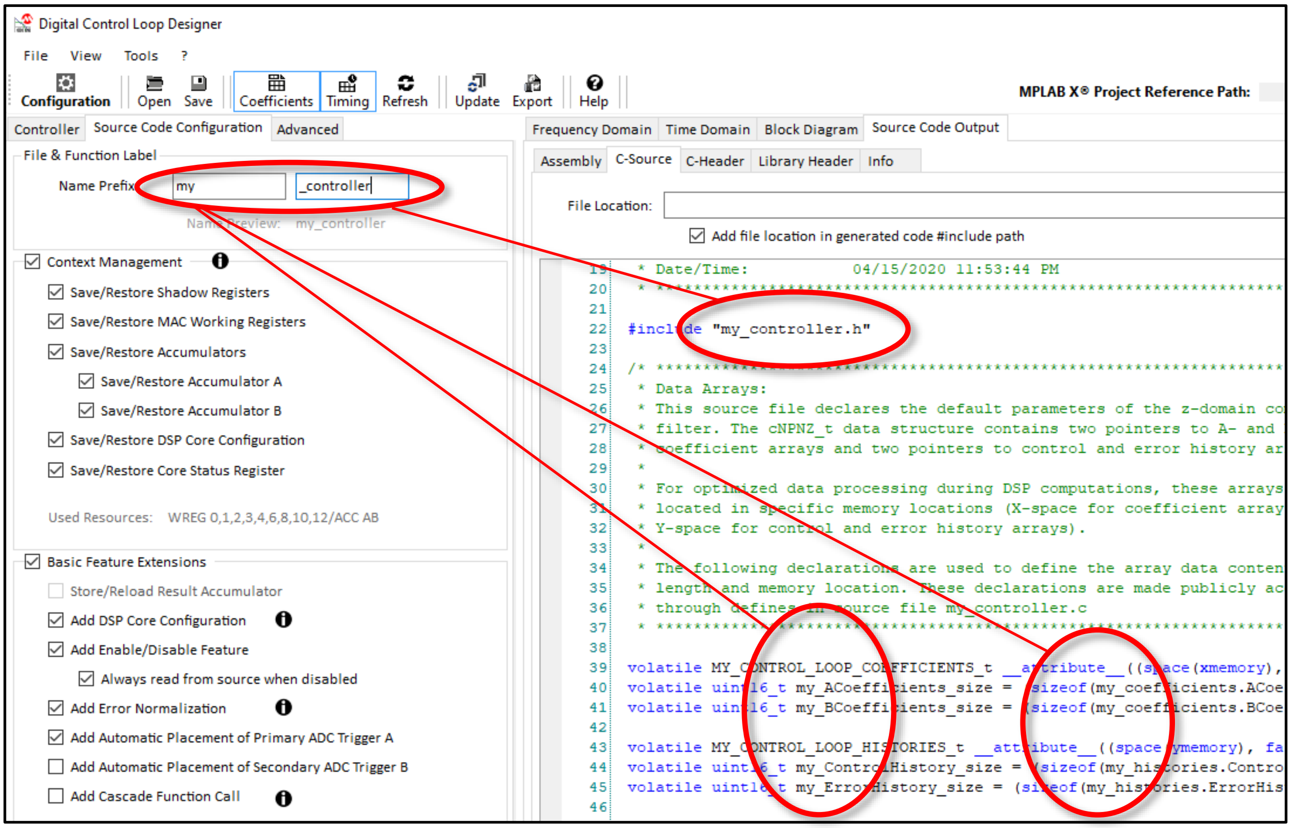


Figure 14: Assigning user-specific names for variables and objects

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 <u>*will not be affected*</u> 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.

  <u>**Please note:**</u>
  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.

  <u>**Please note:**</u>
  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 with enabled ***Data Provider Sources*** (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.DataProvider.ptrControlInput = &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 ADVANCED CODE GENERATOR OPTIONS

In addition to standard control loops DCLD also allows adding special functions and advanced control features to the generated code modules, which can be found on the *Advanced* tab of the controller configuration on the left of the main window.

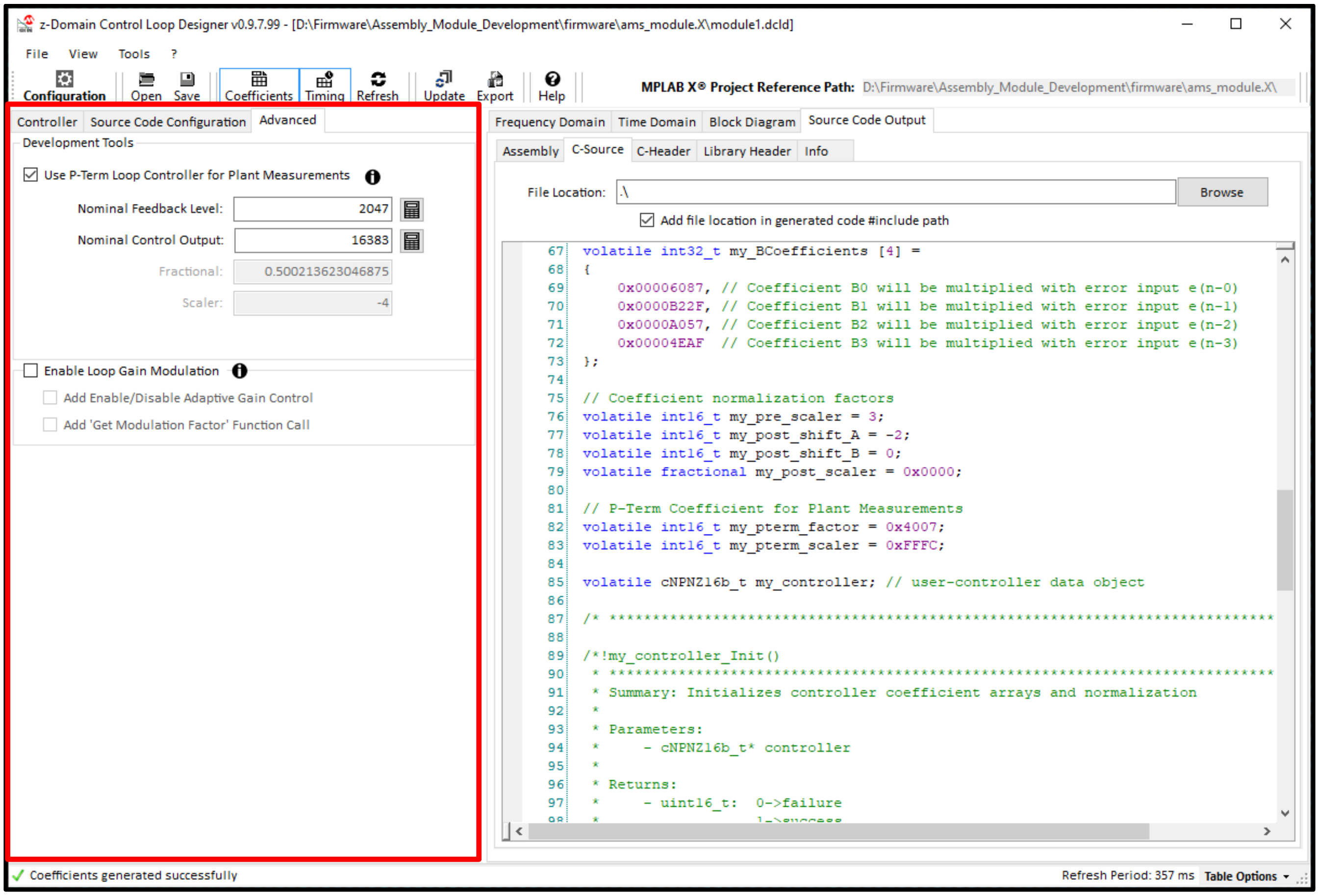


Figure 15: Advanced Code Generator Options

These advanced features fall into two categories:

- **Development Tools**

  This group contains useful features and functions supporting common procedures during development. These features are usually not meant to be present in common application code.

- **Advanced Control Features**

  Advanced control features are additional main control loop extensions meant to be used in application code.

6.1 Plant Measurement Support

The compensation filter of a switched-mode power supply feedback loops serves the sole purpose of compensating non-linear gain characteristics of the power filter (plant) to stabilize its operation and prevent undesired instabilities such as oscillations or noise transfer. This is achieved by compensating all poles in the power plant transfer function with zeros in the compensation filter transfer function and all zeros of the power plant transfer function with poles in the compensation filter transfer function. In general compensation poles and zeros are placed close to / at the frequency locations of their power plant counterpart.

Hence, every design process of any feedback loop starts with the compensation of the power plant and essentially requires knowledge about its frequency domain characteristic represented by its transfer function. This transfer function is generally derived by using:

- Analytical equations, incorporating major component values into equations of specific equations for known power filter topologies
- Polynomial equations defined by coefficients derived through numerical approximation
- Analog circuit simulation
- Direct measurement using Vector Network Analyzers (VNA)

Each of these approaches has its individual advances and drawbacks, which will not be further discussed in this user guide. The only method highlighted here is the direct measurement of the power plant transfer function using a Vector Network Analyzer.

- **Theoretical Background**

The frequency domain design approach of switched-mode power supply feedback loops is based on the small signal model. This approach defines the system as combinations of interconnected transfer functions. The simplest approach only defines two transfer functions, one defining the power plant as transfer function $G(s)$ and the second defining the control stage as transfer function $H(s)$.

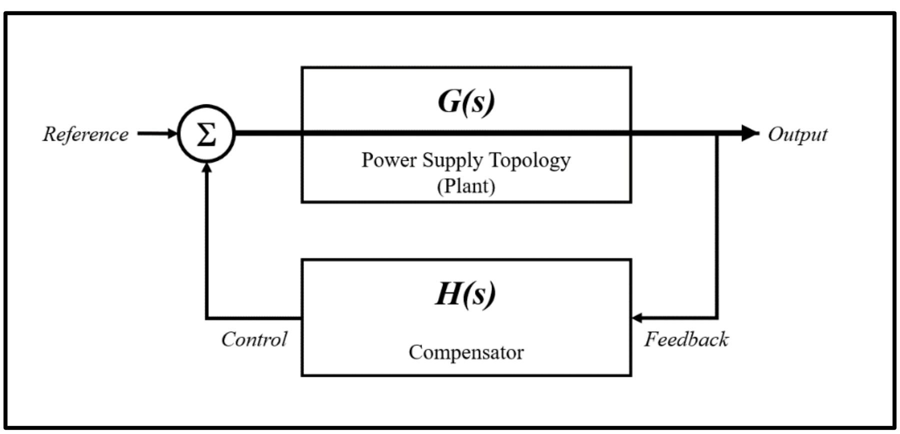


Figure 16: Closed Loop Model of Switched-Mode Power Supply Control System

In this model block diagram, the compensator receives the feedback signal from the system output and turns this into a control signal. This control signal is then summed with the reference and applied to the power plant.

It is important to notice that this is not an exact representation of a real-world feedback loop, where the feedback is directly summed with the reference signal and only the remaining error signal is processed by the compensator. However, this difference does not have any mathematical impact.

According to this model, the transfer function of the closed loop system is defined as

$$G_{CL}(s) = \frac{G(s)}{1 + G(s) \times H(s)} \quad \text{Equation 6-3}$$

The closed loop transfer function equation shows only one unstable / undefined condition, when *G(s) x H(s)* approaches a total value of *-1*. Translated to real-world systems, this condition would lead to even smallest changes of the feedback signal would result in a very large response at the output, making the change on the feedback signal difference even greater, resulting in an even greater output, eventually forcing the system into high gain oscillation. The aspect of interest for stability analysis therefore comes down to determining the value of the Open Loop Gain in s Closed Loop System (red)

$$G(s) \times H(s) > -1$$

The so-called Sole Point of Instability at *G(s) x H(s) = -1* can be found in a frequency domain analysis by searching for the point where the phase crosses -180°. At this point the loop gain must be negative to prevent the system from oscillating.

Common design practices ensure this requirement is met by tuning the loop gain as close as possible to a first-order system with a continuously falling gain of -45° (resp. -20dB/dec) over frequency and if the cross-over frequency of the gain (point where the gain cross the 0dB line) lies within a region where the phase is still greater than -180°.

Instability close to the Sole Pont of Instability is not a suddenly occurring condition but a gradual process. Hence, instability (increasing difficulties in maintaining a stable power output and suppressing tendencies to oscillate) can already be observed when the open loop gain approaches the -180° point. In a practical design, the phase at the cross-over frequency of the gain needs some safety margin called the Phase Margin. This stability criteria is one of three major reference points used to stability analysis:

- Phase Margin: phase-level at the point where the gain crosses 0dB
- Gain Margin: negative gain level at the point where the phase crosses -180°
- Gain slope at Cross-Over: Slope of the gain at the point where it crosses 0dB

Only if all three criteria are meeting the parameters given in design specifications, the power supply can be assumed to be stable and reliable.

- **Open Loop Gain Linearization and Regulation**

As stated above, the common frequency domain design approach splits the design of a feedback loop control stage into two major aspects:

- **Compensation:**
  Compensation is the linearization of the power plant transfer function frequency domain characteristic. The combination of inductive and capacitive components introduces different and very specific poles and zeros at various locations, making the total transfer function highly non-linear

and in many cases physically impossible to apply proper regulation. Hence, the main aspect of compensation is to linearize the non-linear characteristics of the power plant by eliminating the effects of poles and zeros of the power plant.

Effects of Poles on the Frequency Domain Transfer Function:
In the frequency domain a pole introduces a negative gain slope of -20dB/dec, starting at the pole location, and a change in phase of -90°, starting one decade below the pole location and ending one decade above.

Effects of Zeros on the Frequency Domain Transfer Function:
Every zero introduces a positive gain slope of +20dB/dec, starting at the pole location, and a change in phase of +90°, starting one decade below the pole location and ending one decade above.

Fundamental Compensation Approach:
As poles and zeros have complementary effects on gain and phase across frequency, placing a pole at the frequency location of a zero and vice versa is sufficient to eliminate the non-linear effects of one by the other.

Plant Pole & Zero Locations:
Power filters topologies are all, without exceptions, low-pass LC filters. All low-pass LC filters have a similar characteristic where gain and phase run flat up to the resonant frequency of the LC filter, at which point both drop sharply, effectively cutting off higher frequencies. This point where the drop in gain and phase can be observed is usually the point where the power supply filter circuit starts to approach the Sole Point of Instability. Stabilizing the power filter is therefore done by compensating every pole and zero found in the power plant transfer function $G(s)$ by a correcting pole or zero counterpart in the compensation filter $H(s)$, eventually linearizing the system, turning its frequency domain into a flat gain and phase profile.

- **Regulation:**
  Applying the compensation method described above, however, only stabilizes the operation of the power plant. However, there is still no stable output regulation yet and no additional measures for high frequency noise rejection and output impedance stabilization have been applied. This is done by introducing a so-called Pole At The Origin, which introduces a continuous negative gain slope of -45° (resp. -20dB/dec), bringing back the desired low-pass filter characteristic. This Pole At The Origin is created by adding an error integrator to the feedback loop. This integration gain can be used to adjust the total loop gain level. DCLD allows adjustments of the integrator gain by specifying the cross-over frequency of the gain slope introduced by the Pole At The Origin.

  High Frequency Rejection is achieved by adding one more pole into the compensation filter, which is not countered by a zero in the plant transfer function and which is placed at or near below the Nyquist-Shannon frequency of the power supply or control sampling frequency (whatever is lower). Adding an additional pole at high frequencies further reduces the gain level at frequencies above the Sole Point of Instability, helping to improve the gain margin and thus making the power supply robust against fast noise transients.

- **Deriving Pole and Zero Locations through measurements of the Plant Transfer Function**

As stated above, there are multiple ways to derive the transfer function of a power filter design, ranging from analytical models, numerical approximation, circuit simulation and bench test measurements. In general, it is safe to state that any theoretical approach will have the ultimate weakness of requiring validation against the real hardware under test and thus all approaches therefore require a bench test measurement at some point.

For this purpose, the DCLD code generator allows the generation of a specific, Proportional controller without error integrator, which has a continuous gain of 1 and can therefore be used to close the feedback loop without influencing the final plant transfer function. The continuous gain of 1 will make the feedback loop transparent to the measurement but also does not stabilize the power plant.

PLEASE NOTE
Proportional controllers are physically incapable of stabilizing a switched-mode power supply and therefore not suited to be used for common regulation purposes. This particular controller type only serves the purpose of allowing measurements of the power plant under defined and stable operation conditions. **Do not use this controller for any regulation tasks in your final application!**

- **Proportional Controller without Integrator**

This specific proportional controller without integrator reads the feedback signal and determines the error with regards to the applied reference. This error is multiplied with a single gain factor and applied to the power plant (e.g. through a PWM duty cycle). The desired regulation point is reached when the minimum error produces its related output value (e.g. $error \rightarrow 0$ when the recent duty ratio produces the nominal output voltage matching the reference). To make sure the loop ends up at this desired regulation point, the gain factor P is determined by the ratio of the nominal output value U_N over the reference signal REF.

$$U = P \times e \qquad \text{Equation 6-4}$$

with

U = most recent control output
P = Proportional gain factor
e = most recent error

$$P = \frac{U_N}{REF}; \Rightarrow U = \frac{U_N}{REF} \times e \qquad \text{Equation 6-5}$$

with

U = most recent control output
U_N = expected, nominal output value
REF = nominal reference
e = most recent error

This control loop can regulate the output close to the desired, nominal regulation point but will still show some observable DC-offset. This offset may be manually adjusted to get a better representation of the PWM gain in during measurements. However, approaching the nominal operation point too close commonly significantly increases the risk of sudden instability. Hence, the measurement result will properly represent the power plant's transfer function but will be translated down by some small amount of DC gain error.

- **DCLD Proportional Controller Design Aid**

By enabling the function *Use P-Term Loop Controller for Plant Measurements*, the DCLD code generator will add two more variables to the C-source and header files (see Figure 17). These two variables are forming a bit-shift scaled fixed-point number of the form:

HIGH WORD	LOW WORD	
SCALER	SIGN	FACTOR
xxxxxxx xxxxxxxx	x	xxxxxxxx xxxxxxxx
Bit [31:16]	Bit [15]	Bit [14:0]

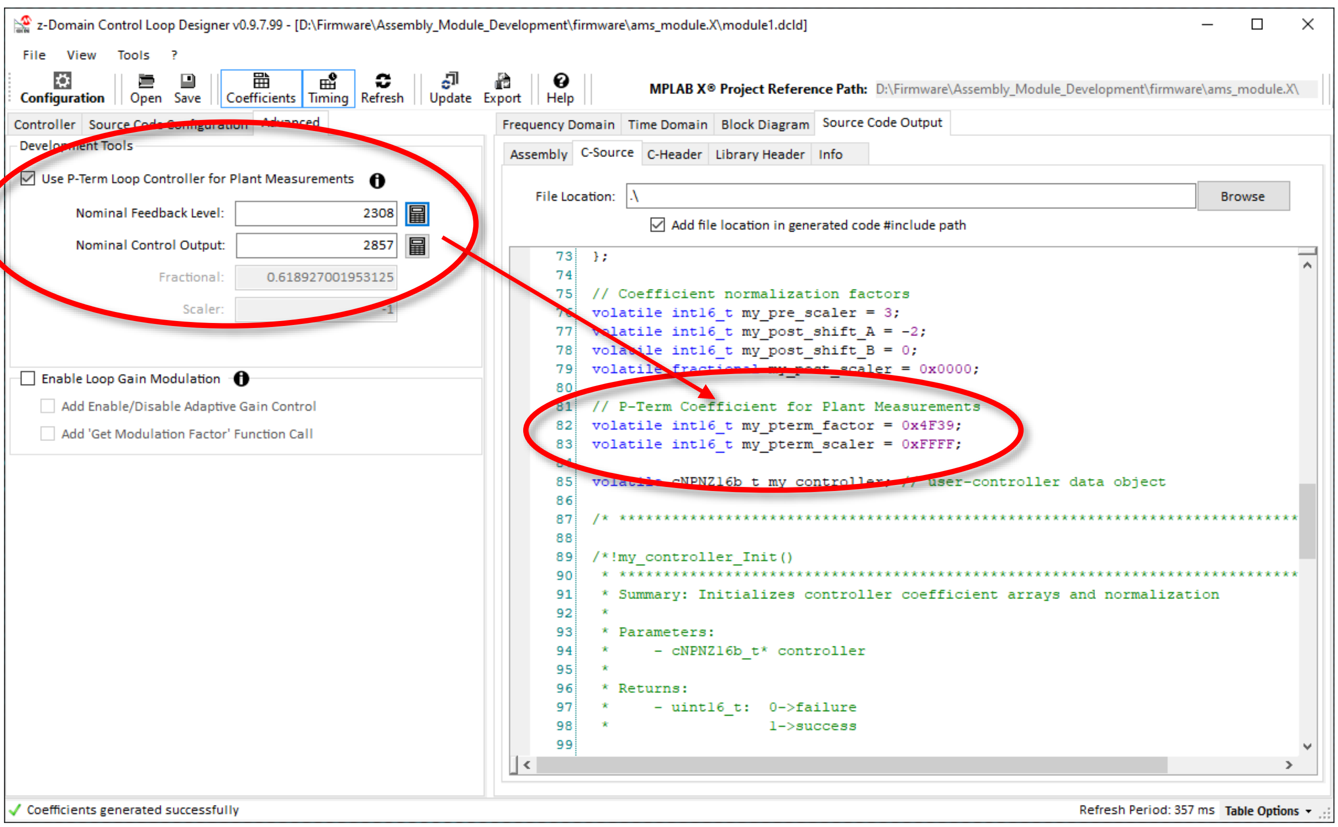


Figure 17: Enabled P-Term Control Loop Generation

DCLD offers two field for entering the nominal reference and nominal output value required for the P-factor calculation. As an additional help, calculation tools for both values are provided. By clicking on the calculator buttons next to the value field will open the respective

- **Nominal Feedback Level Calculation Design Aid**

  Figure 18 shows the Nominal Feedback Level Calculator window. It offers the same options as the Input Gain Calculator window, with the difference of allowing users to enter a user-defined sense signal.

  The example loaded in Figure 18 shows the output voltage divider window. By entering the nominal output voltage in the marked field, this tool will calculate the related reference value used by the controller.

  By clicking OK this value is automatically entered into the Nominal feedback level field of the main window.

  This calculation too offers calculation support for

  - Voltage Divider
  - Shunt Amplifier/Current Sense Amplifier
  - Current Sense Transformer
  - Digital Source

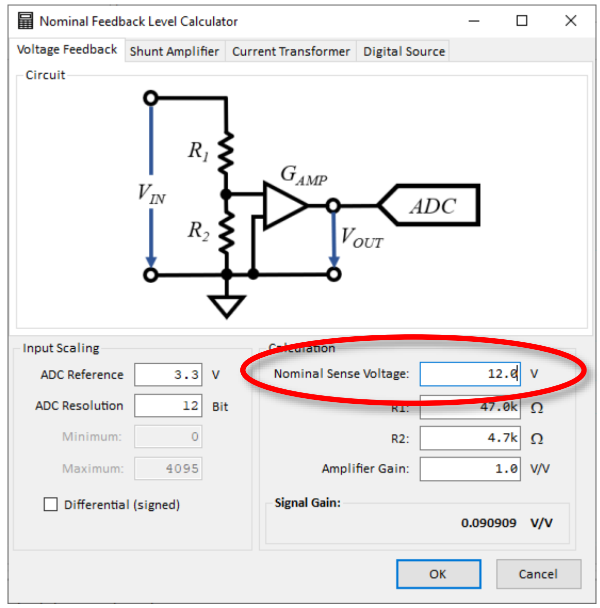


Figure 18: Nominal Feedback Level Calculator

- **Nominal Control Output Level Calculation Design Aid**

  Figure 19 shows the Nominal Control Output Level Calculator window. It offers calculations of duty cycles, phase shifts and frequencies of switching signals and can be configured by specifying PWM module parameters such as time base resolution and switching frequency.

  The example loaded in Figure 19 shows the fixed frequency, duty cycle control output window. By entering the nominal switching frequency and PWM time-base resolution and duty ratio in their respective fields, the tool will calculate the nominal control output value.

  By clicking OK this value is automatically entered into the Nominal feedback level field of the main window.

  This calculation too offers calculation support for

  - Fixed Frequency / Duty Ratio
  - Variable Frequency
  - Phase Shift PWM

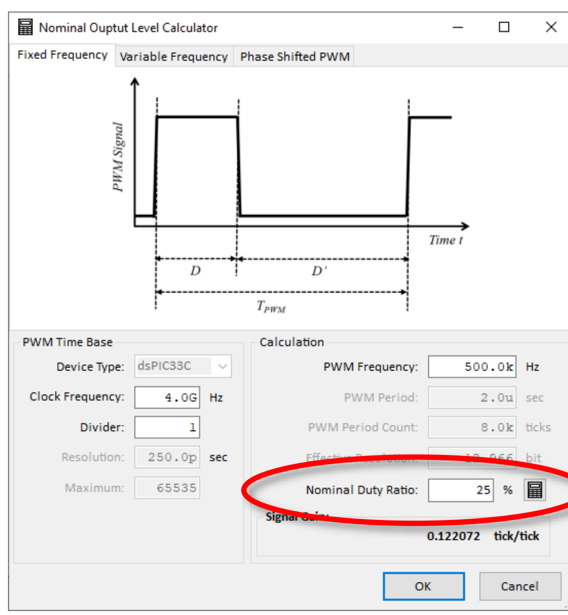


Figure 19: Nominal Control Output Level Calculator

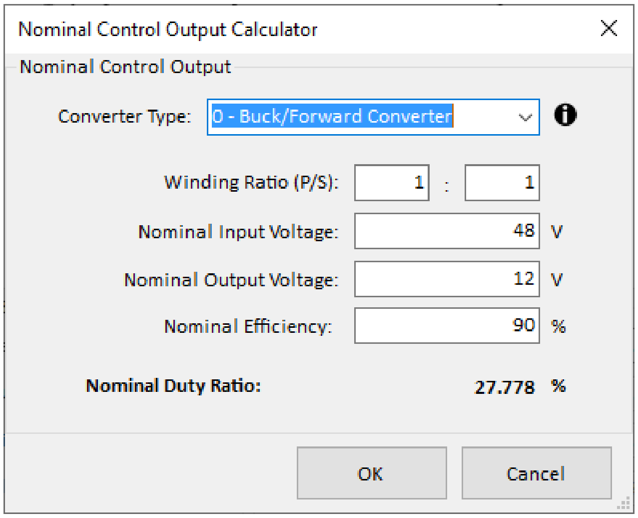


Figure 20: Duty Ratio Calculator

- **Duty Ratio Calculator Window**

For duty ratio calculations an additional calculation tool can be opened by clicking on the calculator button next to the duty ratio field of the fixed frequency view marked in Figure 19.

This calculator allows calculation of ideal duty ratios of power converters of type

- Forward / Buck Converters
  (e.g. Buck, Forward, 2-Switch Forward, Active Clamp Forward, Half-Bridge, Full Bridge)
- Boost Converter
  (e.g. Boost, PFC)
- Buck/Boost Converter
  (e.g. Flyback, SEPIC, 4-Switch Buck/Boost, Non-Isolated Buck/Boost)

- **P-Term Controller Firmware Implementation**

The P-Term controller uses the same API data structure as the main loop configured by DCLD including all customizations provided by the code generator option catalog, such as Enable/Disable switch or Anti-Windup. Hence, although DCLD generates an independent P-Term Controller routine inside the assembly library, by using the same configuration as the main controller, the P-Term controller does not have to be configured in user code. The only change users have to make is to replace the function call of the main controller by the function call of the P-Term controller routine:

```
my_controller_PTermUpdate(&my_controller);
```

- **P-Term Controller Application Tips:**

As stated above multiple times and is also pointed out by warnings created by DCLD, the Proportional controller is an unstable feedback loop. During a measurement it is therefore recommended to follow these guidelines:

- ***Only operate the power supply under stable conditions***
  Make sure input voltage and load does not change while running in Proportional control mode. Any transient introduced bares the risk of destabilizing the power supply.
- ***Adjust the amplitude of the injected error signal to low levels***
  VNAs usually support functions to adjust the maximum amplitude of the injected error signal. It is recommended to start from low levels first and increase the signal size slowly and carefully until the noise content of the measurement result becomes clearly visible. For further noise reduction, use averaging modes (if available) and run continuously averaged measurements
- ***Protect the Device Under Test***
  Protection of the power supply during the measurement is mandatory as the unstable nature of the Proportional controller may cause oscillations with unpredictable outcome at any time. In addition to setting current limits of power sources and enabling additional protection features on the control chip,

such as hardware comparators shutting down the PWM in case of a threshold violation, You can use the Anti-Windup feature to limit the allowed range of control outputs to further protect the system.

- ***Soft Start is mandatory***
  The Proportional controller, as the name implies, will create a proportionally large response to large errors. Hence, it is recommended to write a soft-start routine, which slowly increments the reference up to the value determined during the configuration of the P-Term gain coefficient. Thus, the control loop will never see large errors and therefore also only create small responses, limiting the risk of driving the power supply into unpredictable oscillations.

- **Bench Test Measurement Setup**

This P-Term Controller implementation can be applied to any kind of loop-level, such as on outer voltage loop level as well as for inner current loops. It also works for loops where parts of the control loop implementation become part of the plant such as the analog comparator in peak current mode control.

The following example (Figure 21) shows the measurement setup for a single voltage mode plant transfer function of a single, non-isolated buck converter:

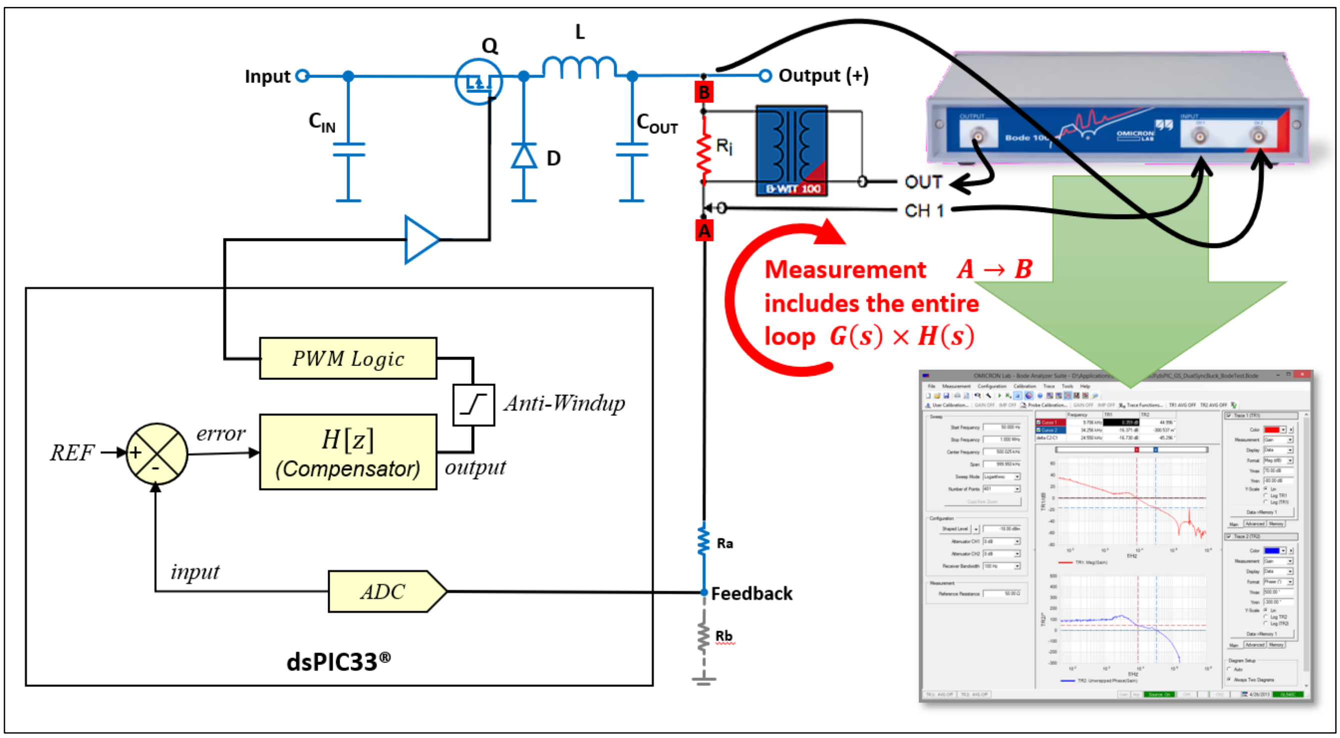


Figure 21: Loop Measurement Setup

The device for measuring the loop response used in this example is a stand-alone VNA alled Bode 100 from OMICRON Lab. Further information on this device, its features and functional operating data can be found here: https://www.omicron-lab.com/products/vector-network-analysis/bode-100/

- **Bench Test Measurement Results**

Following the process guidelines described above, the following results were derived:

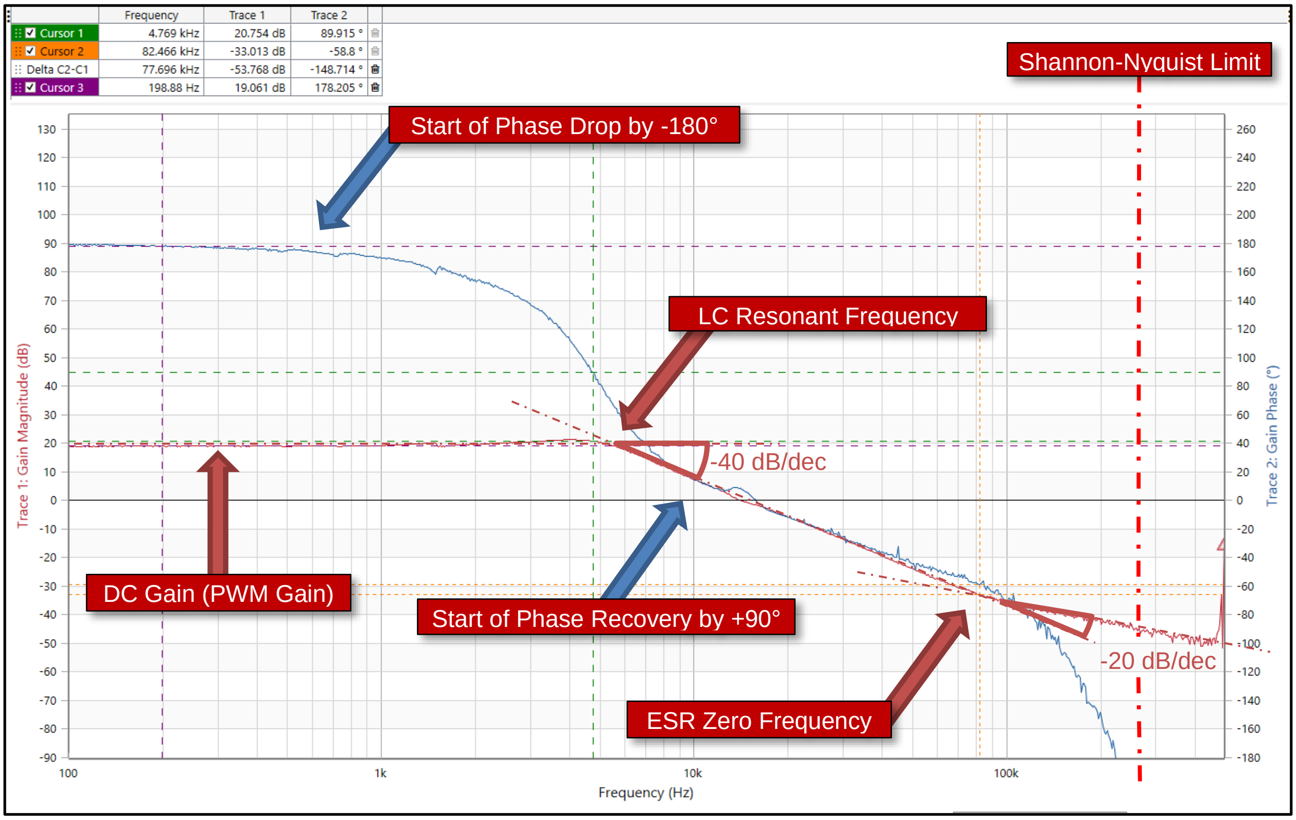


Figure 22: Power Plant Measurement Result of Voltage Mode Buck Converter

As stated above, by using Proportional control without error integration during the measurement, the control loop becomes transparent (constant gain of *=1*) and the recorded frequency response reproduces the transfer function of the power plant.

PLEASE NOTE
In VNA Measurements the phase range of the result is shifted by +180°. The small signal model defines the location at which transients are injected into the loop at the reference (see Figure 16) while the location chosen to inject the transients in a real measurement is at the top of the feedback path (Figure 21), hence, half way around the loop. As a result, The Sole Point of Instability, defined by the small signal model to be found at $\Phi = 180°$ is represented in the measurement results at $\Phi = 0°$ This phase shift is usually appreciated in engineering as the measurement directly reflects the phase margin.

- **Verifying Bench Test Measurement Results**

The Device Under Test (DUT) in this example is a non-isolated, synchronous buck converter. The measurement conditions were set at ~50% load, where the converter is operating in Continuous Conduction Mode (CCM).

Test Conditions:

V_{IN}	=	9.0 V	L	=	10 µH
V_{OUT} (nominal)	=	3.30 V	C	=	100 µF
V_{OUT} (real)	=	2.94 V	ESR	=	18 mΩ
I_{OUT} (nominal)	=	1.25 A	f_R	=	5.03 kHz
f_{SW}	=	500 kHz	f_{ESR}	=	88.4 KhZ
f_{SAMP}	=	500 kHz	G_{DC}	=	19.1 dB

When using Proportional controllers to measure the plant transfer function, it is important to be aware of the error margin of the results, especially when these results are used to verify theoretical models.

In this example we use a very simple, first approximation approach to verify how well the results meet the estimated/expected frequency response.

The values for converter components listed in under the test conditions above have been taken from the data sheets of the used components. We use these values to quickly verify how accurate the measured pole and zero locations and the DC-gain match.

- **LC Resonant Frequency**

  The location of the complex-conjugate pole at the resonance of an LC filter is calculated by using the equation

$$f_R = \frac{1}{2\pi\, RC} = \frac{1}{2\pi\, ESR\ \times C} \qquad \text{Equation 6-6}$$

with

f_R = Natural Resonant Frequency of the LC Filter
L = Inductance of the main inductor
C = Total output capacity of the LC filter

Using the values listed under Test Conditions above, we get

$$f_R = \frac{1}{2\pi\,\sqrt{LC}} = \frac{1}{2\pi\,\sqrt{10\times 10^{-6}H\ \times 100\times 10^{-6}F}} = 5{,}033\ Hz$$

This equation gives the corner frequency of an LC filter, hence, the point where the gain has changed by 3 dB relative to the previous, continuous section. By using the Cursor 1 of the VNA software, this point is located at ~5,600 Hz (see Figure 22: Power Plant Measurement Result of Voltage Mode Buck ConverterFigure 22).
After the resonant point, the gain drops with a negative slope of ~-40 dB/dec. One decade below the resonant frequency the phase also starts to drop with a characteristic of a -180° drop. These two

indicators point to the presence of two poles (resp. one complex conjugate pole) at the resonant frequency. This double-pole needs later to be compensated by two zeros in the compensation filter.

- **ESR Frequency**
  The location of the RC zero introduced by the internal Equivalent Series Resistance (ESR) of the output capacitors is calculated by using the equation:

$$f_{ESR} = \frac{1}{2\pi\sqrt{ESR \times C}} \quad \text{Equation 6-7}$$

with

f_{ESR} = Corner frequency of the RC filter formed by the internal capacity and equivalent series resistance of the output capacitor
ESR = Equivalent Series Resistance of the output capacitor
C = Total output capacity of the LC filter

Using the values listed under Test Conditions above, we get

$$f_{ESR} = \frac{1}{2\pi\ ESR\ \times C} = \frac{1}{2\pi\ \times 18 \times 10^{-3}\Omega\ \times 100 \times 10^{-6}F} = 88{,}419\ Hz$$

This equation gives the corner frequency of an RC filter, hence, the point where the gain has changed by 3 dB relative to the previous, continuous section. By using Cursor 2 of the VNA software, this point is located at ~82,466 Hz (see Figure 22: Power Plant Measurement Result of Voltage Mode Buck ConverterFigure 22).

Finding the ESR zero requires finding a change in slope by 3 dB and a related change in phase one decade below. The change in gain is clear to identify. The change in phase, however, is partially overlapping with the phase change introduced by the complex-conjugate pole at the resonance. The indicator here is that the slope of the phase drop at the resonant frequency is shallowing out at/after 8 kHz. This is the counter-effect introduced by the +90° change of the ESR zero located at ~80 kHz.

In a passive filter the phase would eventually recover to 90° at f_{ESR}. However, in a switched-mode power supply this is not the case. As the power transfer of a switched-mode power supply is “chopped” in cycles of charging and discharging processes between inductive and capacitive components, the operating power supply filter shows characteristics of a discrete time domain system. Hence, the total deterministic range is limited by the Nyquist-Shannon limit f_N at half of the switching frequency. Although the gain does not show any significant drop at f_N, the phase is eventually dropping to negative infinity. This effect is amplified by also using a discrete time domain feedback loop (a.k.a. digital controller), which only responds once per switching cycle.

- **DC-Gain / PWM Gain**
  At low frequencies, the gain of a passive low-pass filter always runs flat at 0dB with an effective gain of $=1$. This unity gain region allows frequencies to pass the filter without damping. In a power supply, however, this gain level is higher and shows a stable, constant offset up to the point where filter components start to affect the passing transients.

This gain offset is usually referred to as DC Gain or PWM Gain.

The switching cell of any PWM controller has a major impact on the plant transfer function and can fundamentally change its order and overall characteristic. This is why, by definition, the switching cell of the PWM controller device is associated with the plant transfer function rather than the feedback loop. While there are other techniques available to measure the corner frequency of passive filters and filter components, it is the influence of the PWM generation during operation, which only can be made visible by operating the power supply at / close to its nominal operating point.

To verify the DC Gain measurement result, we can use the equation:

$$G_{DC} = 20\log_{10}\left(\frac{V_{IN}}{V_{RAMP}}\right) \qquad \text{Equation 6-8}$$

with

G_{DC} = Low Frequency Gain Offset (PWM Gain)
V_{IN} = Converter Input Voltage in [V]
V_{RAMP} = Analog PWM Ramp Generator Peak Voltage (always $= 1$ with digital PWM logic)

In analog PWM generators the peak voltage of the PWM ramp generator goes int the equation as input divider. When the PWM is generated by digital logic, this dividing effect disappears. Thus, for any PWM signal generated by digital logic the value V_{RAMP} is set $=1\ V$, changing the equation to:

$$G_{DC} = 20\log_{10}\left(\frac{V_{IN}}{V_{RAMP}}\right) = 20\log_{10}\left(\frac{V_{IN}}{1V}\right) = 20\log_{10}([V_{IN}])$$

Using the parameters from the bench test measurement conditions:

$$G_{DC} = 20\log_{10}([V_{IN}]) = 20\ \log_{10}([9V]) = 20\ \log_{10}(9) = 19.085\ dB$$

The Proportional controller used by the P-Term control loop option provided by DCLD suffers, like any Proportional controller, from a noticeable DC error. In this example this DC error becomes visible as ~10% deviation of the output voltage measured during the bench test from the given reference (2.94 V measured output voltage vs. 3.30 V desired).
The DC Gain result, however, is not affected by this mismatch. By using Cursor 3 of the VNA software, the DC Gain is measured at ~19,061 Hz (see Figure 22: Power Plant Measurement Result of Voltage Mode Buck ConverterFigure 22) and thus almost perfectly matching the theoretical estimate.

However, if is important to keep in mind that this accuracy can only be met when all influencing factors are considered, which influence the nominal output value.

In this example of a fixed frequency, duty-cycle controlled buck converter, the duty cycle is the control output value. The buck converter topology divides its input voltage by the given duty cycle to produce the desired output voltage. While driving a real-world power stage, the control loop is always a fraction bigger than its ideal estimate as it needs to compensate for losses.
The influencing factors are:

- Input Voltage
- Output Voltage
- Efficiency

In our case the efficiency of the power supply is estimated with 90%. The measured DC gain value only turns out to be accurate if this additional parameter is considered.

6.2 Adaptive Gain Control

In general, adaptive control mechanisms are commonly used to compensate for effects of undesired parameter changes during runtime, tuning the system for specific performance criteria, enhance efficiency or higher reliability. Each of these approaches have their individual parameter matrix determining which additional parameters must be considered to tune one or more system parameters to achieve the desired outcome. Algorithms and parameter matrices can be very different depending on purpose and scope of the adaptation but may also be highly hardware dependent.

- **Adaptive Control Fundamentals**
  Because the variety of adaptive measures is huge, basic principles of adaptive control can only be defined on a higher, unspecific level. However, there are three major aspects applying for any system:

  - *Parameter Matrix*
    The ultimate starting point is the determination of dependencies of the parameter P_X we intend to modulate. When these dependencies are known and understood, we (ideally) end up with an equation where the new value of P_X is calculated based on its dependencies, which are determined by other parameters P_1, P_2, …, P_n. Dependent on the complexity of the model, parameters P_n may be observable (e.g. through direct measurements) allowing a system identification at runtime. This type of adaptive control is called Model Identification Adaptive Control (MIAC). More complex dependencies may require basing the adaptive control block on reference models, where values of P_X are commonly estimated based on characterization. This type of adaptive control is called Model Reference Adaptive Control (MRAC).

  - *Observer / System Identification*
    Once the parameter matrix is known, so-called *Observers* are installed monitoring the change of relevant/influencing parameters P_n, detecting relevant changes and triggering the modulation of P_X. *Relevance* is highly system dependent and eventually determines the *Granularity of Adaptive Steps* across a certain modulation range.
    Adjusting the granularity of adaptive steps is an important part of the design process as low granularity may introduce undesired side-effects such as noise.

  - *Modulator*
    The Modulator is the mechanism which eventually changes the parameter P_X at runtime. Yet again, implementations may differ widely, however, important design aspects can be narrowed down to the modulation frequency and maximum change rate per modulation step. In a power supply controller, the point in time relative to switching cycles may also be relevant to ensure reliable operation

- **Adaptive Gain Control (AGC) Introduction**
  AGC is a specific method for tuning the overall feedback loop gain during runtime. It falls into the classification of Nondual Adaptive Controllers and can be implemented as Model Identification Adaptive Control (MIAC) as well as Model Reference Adaptive Control (MRAC).

  With other words, the fundamental main control loop is deterministic and is working satisfactorily in a specific region but may need optimization to do so across the entire operating range. In a digital power supply control feedback loop this translates into having a static controller with one, static set of coefficients optimized for a defined state of operation (e.g. at a specific input voltage while under a specific load condition), just like any analog feedback loop where poles and zeros of the compensation network as well as the integrator gain are fixed by component values of resistors and capacitors in the feedback loop of the error amplifier. In systems like these, corner cases such as no-load, light load or when operating under high- or low-line conditions, the system's transient response will divert from the desired optimum and must be verified to lie within the safe operating area of stability margins specified for each individual design.

For many designs, this approach is sufficient and corner cases at the extremes can be covered by designing for wider tolerance margins as needed. However, there is also a wide range of applications, where large deviations are not acceptable, like in Point-Of-Load (POL) or Voltage Regulator Modules (VRM) powering sensitive and demanding high speed digital loads, or when the range over which fundamental system parameters change is too wide to be covered by one static feedback loop, such as in Power Factor Correction (PFC).

For these cases, Adaptive Gain Control offers a generic, adaptive feedback loop gain tuning option with a high degree of flexibility and scalability.

- **Adaptive Gain Control (AGC) Use Case Example**
  Figure 23 shows the implementation of AGC as Feed Forward controller, adapting the loop gain of the compensator $H_C(z)$ dependent on changes in input voltage V_{IN}.
  It consists of

  - a second data source input (ADC) for the converter input voltage V_{IN}
  - an observer, triggering on relevant changes requiring compensation of plant gain variations
  - a modulator tuning the feedback compensation loop gain

  Depending on the gain modulation should be only respond to changes in input voltage, one input port is sufficient to compensate plant gain variations. However, more precise gain variation compensation can be achieved by additionally include the most recent output voltage and load into the modulation scheme.

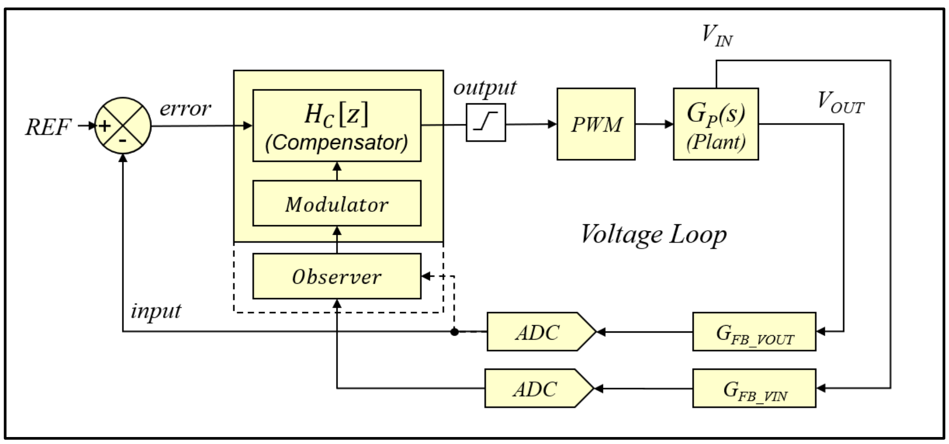


Figure 23: Adaptive Gain Control (Feed Forward) Block Diagram

- **AGC Modulator**
The AGC modulator increases or decreases the total feedback loop gain by adjusting the cross-over frequency of the gain of the pole at the origin (integrator gain) at runtime. The analysis of the transfer function equation of an analog derivative of the digital z-domain filter, which may be used as prototype filter for a bi-linear transform, is used to derive the dependencies of digital filter coefficients and to determine and establish an appropriate modulation scheme.

  Although the bi-linear transform of an analog prototype filter is not essentially required to derive filter coefficients of a digital control loop, it is used here to mathematically derive the physical dependencies of each coefficient. Starting from a common transfer function of a type III compensator, we find that the equation determining the feedback loop controller consists of two major blocks:

$$H_c(s) = \underbrace{\frac{\omega_{P0}}{s}}_{\text{Integrator Gain}} \times \underbrace{\frac{\left(\frac{s}{\omega_{Z1}}+1\right)\left(\frac{s}{\omega_{Z2}}+1\right)}{\left(\frac{s}{\omega_{P1}}+1\right)\left(\frac{s}{\omega_{P2}}+1\right)}}_{\text{Lead-Lag Compensator}} \quad \text{Equation 6-9}$$

  by substituting the Laplace operator *s* by the term (Tustin or Trapezoidal Substitution) $\frac{2}{T_s}\frac{(1 - z^{-1})}{(1 + z^{-1})}$

  We can derive equations for each individual coefficient A_1, A_2, A_3 and B_0, B_1, B_2, B_3, which will be used within the Linear Difference Equation (LDE) computation determining the next control output $u(n)$.

$$u_n = +A_3u_{n-3} + A_2u_{n-2} + A_1u_{n-1} + B_3e_{n-3} + B_2e_{n-2} + B_1e_{n-1} + B_0e_n \quad \text{Equation 6-10}$$

  Observing the results of each individual coefficient equation, we find that the integrator gain introduced into the control system by the presence of the pole at the origin ω_{P0} only influences the B-coefficients while all A-coefficients remain unchanged. Furthermore, the cross-over-frequency of the gain introduced by the pole at the origin ω_{P0} only appears as simple factor to the rest of the coefficient equation.

Digital Type III (3P3Z) Coefficient Equations:

$$A_1 = -\frac{\left(-12 + T_S^2\omega_{P1}\omega_{P2} - 2T_S(\omega_{P1} + \omega_{P2})\right)}{(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-11}$$

$$A_2 = \frac{\left(-12 + T_S^2\omega_{P1}\omega_{P2} + 2T_S(\omega_{P1} + \omega_{P2})\right)}{(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-12}$$

$$A_3 = \frac{(-2 + T_S\omega_{P1})(-2 + T_S\omega_{P2})}{(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-13}$$

$$B_0 = \omega_{P0}\,\frac{\omega_{P1}\omega_{P2}T_S(2 + T_S\omega_{Z1})(2 + T_S\omega_{Z2})}{2\omega_{Z1}\omega_{Z2}(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-14}$$

$$B_1 = \omega_{P0}\,\frac{\omega_{P1}\omega_{P2}T_S\left(-4 + 3T_S^2\omega_{Z1}\omega_{Z2} + 2T_S(\omega_{Z1} + \omega_{Z2})\right)}{2\omega_{Z1}\omega_{Z2}(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-15}$$

$$B_2 = \omega_{P0}\,\frac{\omega_{P1}\omega_{P2}T_S\left(-4 + 3T_S^2\omega_{Z1}\omega_{Z2} - 2T_S(\omega_{Z1} + \omega_{Z2})\right)}{2\omega_{Z1}\omega_{Z2}(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-16}$$

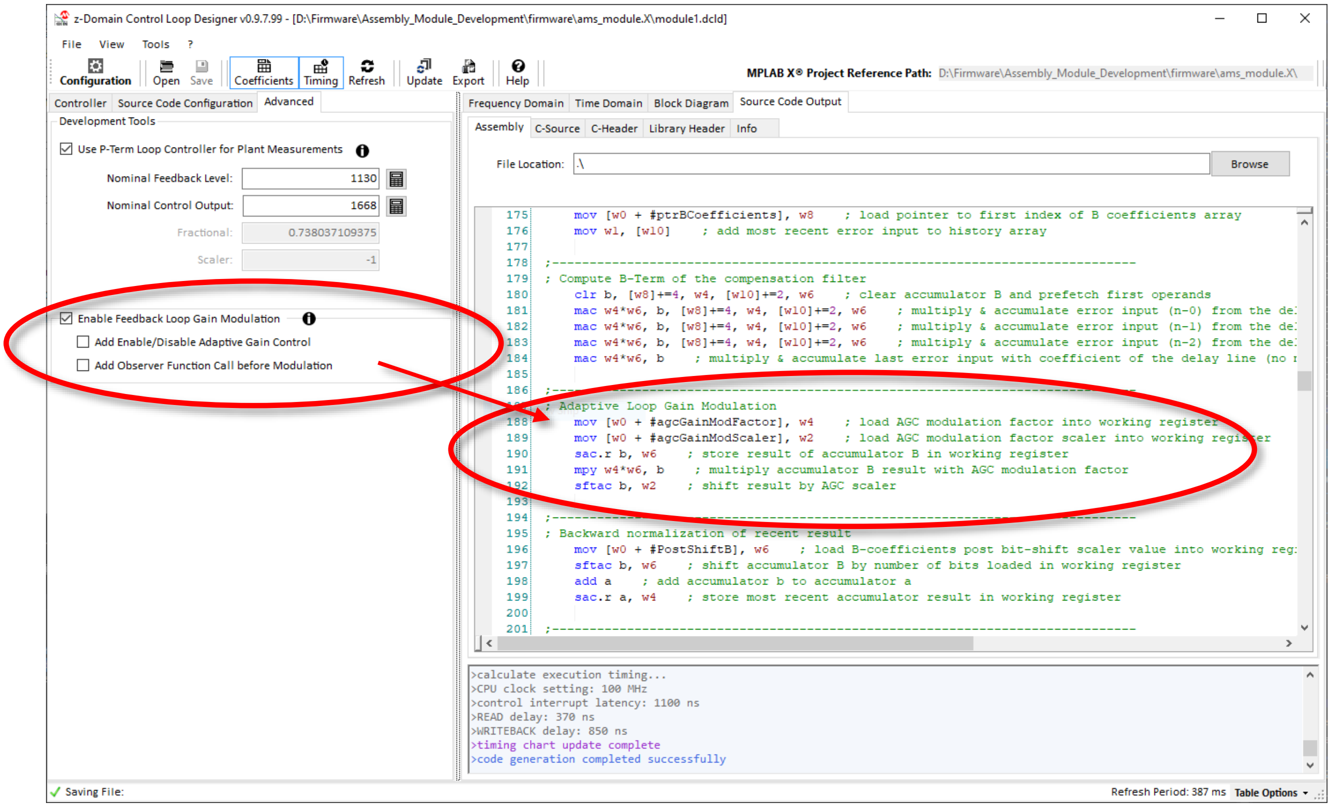
$$B_3 = \omega_{P0}\,\frac{\omega_{P1}\omega_{P2}T_S(-2 + T_S\omega_{Z1})(-2 + T_S\omega_{Z2})}{2\omega_{Z1}\omega_{Z2}(2 + T_S\omega_{P1})(2 + T_S\omega_{P2})} \tag{Equation 6-17}$$

PLEASE NOTE
This relationship is true for *all* lead-lag compensators and is independent from the control order. Hence, this modulation technique is applicable to all control orders and to all feedback loop such as outer voltage loops or inner current loops.

Modulating the loop gain can therefore be performed by multiplying a modulation factor k_{AGC} with all B-coefficients during runtime to achieve active loop gain modulation, effectively changing the LDE (Equation 6-10) into:

$$u_n = +A_3u_{n-3} + A_2u_{n-2} + A_1u_{n-1} + k_{AGC}\left(B_3e_{n-3} + B_2e_{n-2} + B_1e_{n-1} + B_0e_n\right) \tag{Equation 6-18}$$

- **AGC Modulator Implementation**
  When enabling the option *Enable Loop Gain Modulation* in DCLD, the code generator adds the additional multiplication of a user-specified factor `agcFactor` with the intermediate result after the B-term computation and before A-term and B-term of the LDE get added to form the final control output result.



The modulation factor is of the following format of a 32-bit wide, bit-shift scaled fractional number:

HIGH WORD	LOW WORD	
SCALER	SIGN	FACTOR
XXXXXXX XXXXXXXX	X	XXXXXXXX XXXXXXXX
Bit [31:16]	Bit [15]	Bit [14:0]

Scaler and factor are declared as two, separate 16-bit values for improved execution performance.

DCLD adds default values to the C-Source file of the generated code. These default values will be loaded automatically into the cNPNZ16b_t data structure during initialization of the main loop coefficients. Both values are also provided as global variable in the C-Header and are available to be changed to different default values in user code before the initialization routine is called.

PLEASE NOTE
This factor is always initiated by default with a value of *=1* to allow transparent operation (without modulation) of the main loop until active modulation is applied.

- **AGC Modulator Options**
  When the AGC modulator function is enabled, DCLD offers two more options to tailor its use inside the Assembly code and throughout the user software:

  - *Add Enable/Disable Adaptive Gain Control*
    In some applications it may not be desired to apply feedback gain modulation continuously but to restrict the execution of the modulation to certain operating conditions such as when adding a defined gain boost when the converter drops in discontinuous conduction mode, limiting the feedback gain during startup or adjusting feedback gain at the extremes of the rail voltages. During validation, this option may also be useful to run comparison measurements between an AGC modulated control loop versus a static loop.

    To cover these application specific aspects, Enable/Disable control can be added to the assembly code. When disabled, the feedback gain modulation will be bypassed, and the compensation filter will be executed in the common way (B-coefficients will remain unchanged).

    When *Enable/Disable Adaptive Gain Control* has been added, the `agc_enabled` bit in the `status` word of the `cNPNZ16b_t` data structure of the main controller can be used to manually turn AGC modulation on and off in user code.

  - *Add Observer Function Call before Modulation*
    Observers need to be tailored to design-specific values and feedback sources may differ widely from design to design. Purpose and scope of the feedback gain modulation determines what parameter matrix is appropriate for determining the modulation rate as well as the call rate (call frequency) with which the modulation is executed. Further, the implementation of deriving the modulation may also differ significantly. An MIAC implementation may take up significant CPU resources to calculate the new modulation factor based on real-time data, while MRAC implementations may provide data points of the parameter matrix in pre-defined tables. Depending on available resources and control frequency (available time to execute this calculation), calculations might have to be replaced by simpler look-up tables.

    To support any of these approaches and provide the user with an approach to create the most effective implementation possible, a function pointer can be added to the `cNPNZ16b_t` data structure, pointing to a user-defined function in which the appropriate steps can be taken deriving the modulation factor. This function will be called just before the new modulation factor is applied to B-coefficients on a cycle-by-cycle basis.

PLEASE NOTE

When the *Add Observer Function Call before Modulation* option is enabled, users **must** declare a pointer to a publicly accessible function. If this option is enabled and no function pointer is declared (`Null`-pointer) or the control loop is executed before the function pointer got initialized, an *Address Error Trap* will be thrown.

This example shows how this function pointer is declared:

```
my_controller.GainControl.ptrAgcObserverFunction = &my_function;
```

- **AGC Observers**
  As stated above, observers determining/updating the modulation factor during runtime may be very different depending on the scope and purpose of the feedback gain modulation. Hence. generation of observer code is currently not supported by DCLD.

  Future versions may offer default implementations for the most common use-cases such as feed-forward control and output impedance tuning. However, at the point of this release, these are only available as code example.

7.0 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 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 include paths. The compiler always starts in the main project directory (location of the Makefile), but users can also specify other include paths for common and special C-sources as well as assembly include paths. 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.

    Project associations are specified in the project configuration window. By opening a MPLAB® X project file, DCLD will read all required information from the MPLAB® X project XML file.

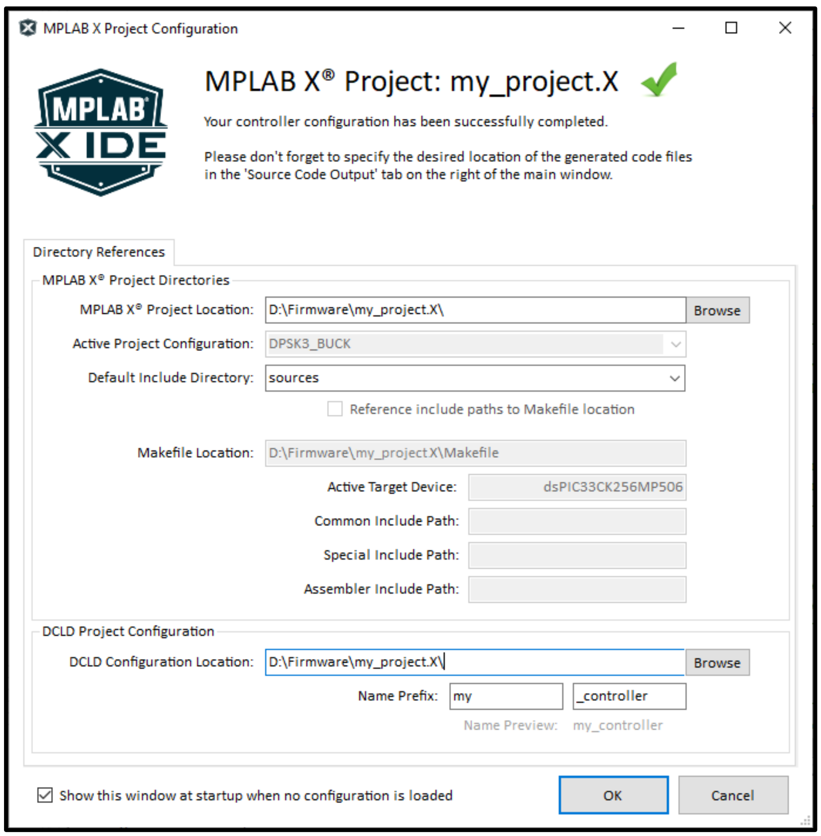


Figure 24: MPLAB® X Project Root Directory Declaration

This project configuration window will automatically open when DCLD detects conflicts with path declarations or if the project has not been saved yet.

As soon as the configuration file locations and MPLAB® X project location are known, DCLD is able to generate and export source code files.

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

  Figure 25 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.

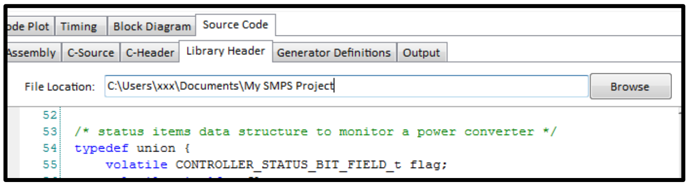


Figure 25: Source Code File Target Directory Declaration

- **Generate & Export Code Files**

Figure 26 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.

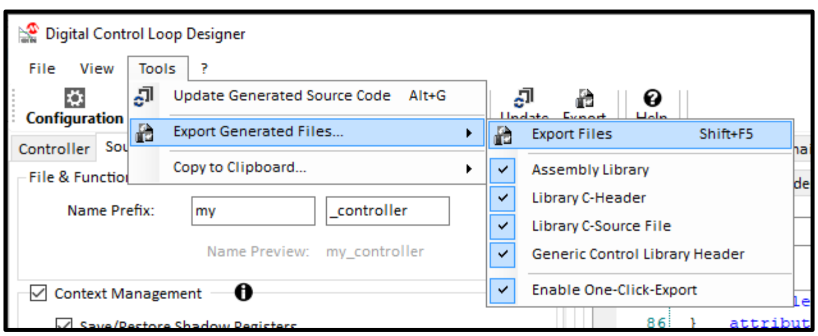


Figure 26: Code Generator Menu

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 26)

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.

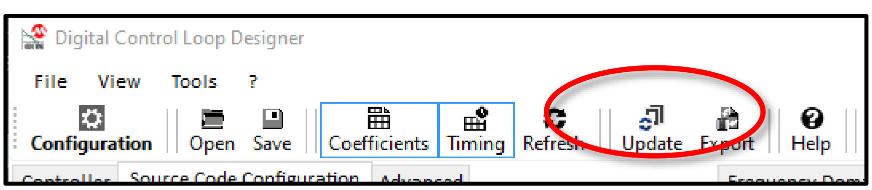


Figure 27: Code Generator Tool Bar

8.0 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.

Right-click on the *Important Files* folder in the project manager and select *Add Item to Important Files*. (see Figure 29)

From the File Browser dialog, select the DCLD configuration file which should be added to the project and click *Open*. (see Figure 28)

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 would 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.

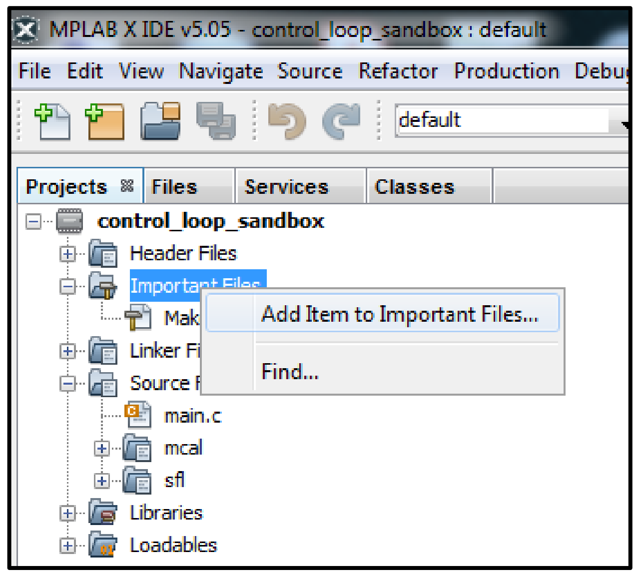


Figure 29: Adding DCLD Configuration Files to a Project

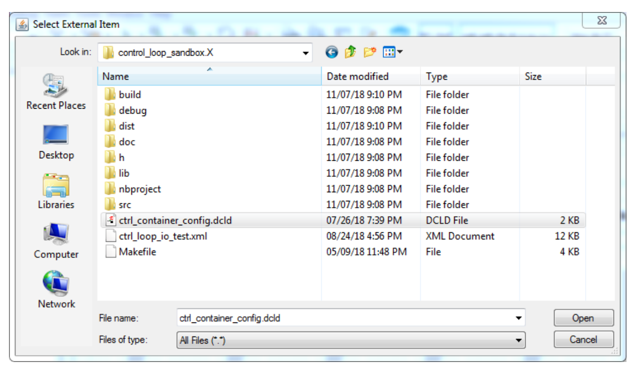


Figure 28: Select the DCLD Configuration File

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

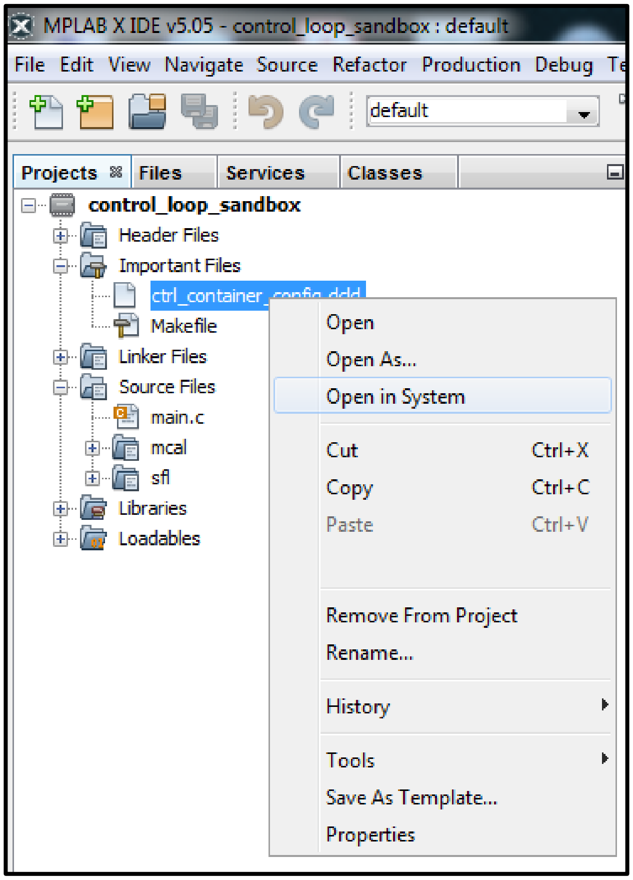


Figure 30: Opening DCLD from the MPLAB X IDE

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 30):

○ 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.

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 `#include` 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 `#include` 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 31)

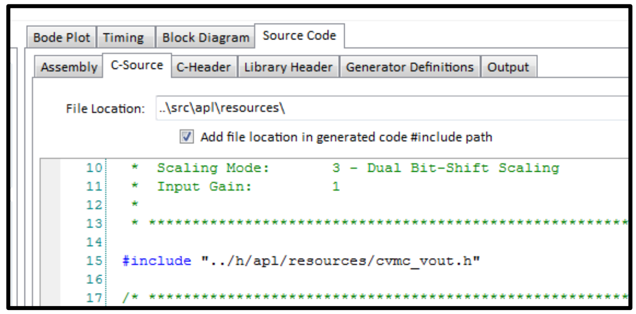


Figure 31: Optimizing file locations

Please review section *7.0 Code Generation* of this user guide for more information of file references.

9.0 APPLICATION INFORMATION / TROUBLESHOOTING

9.1 Application Information Window

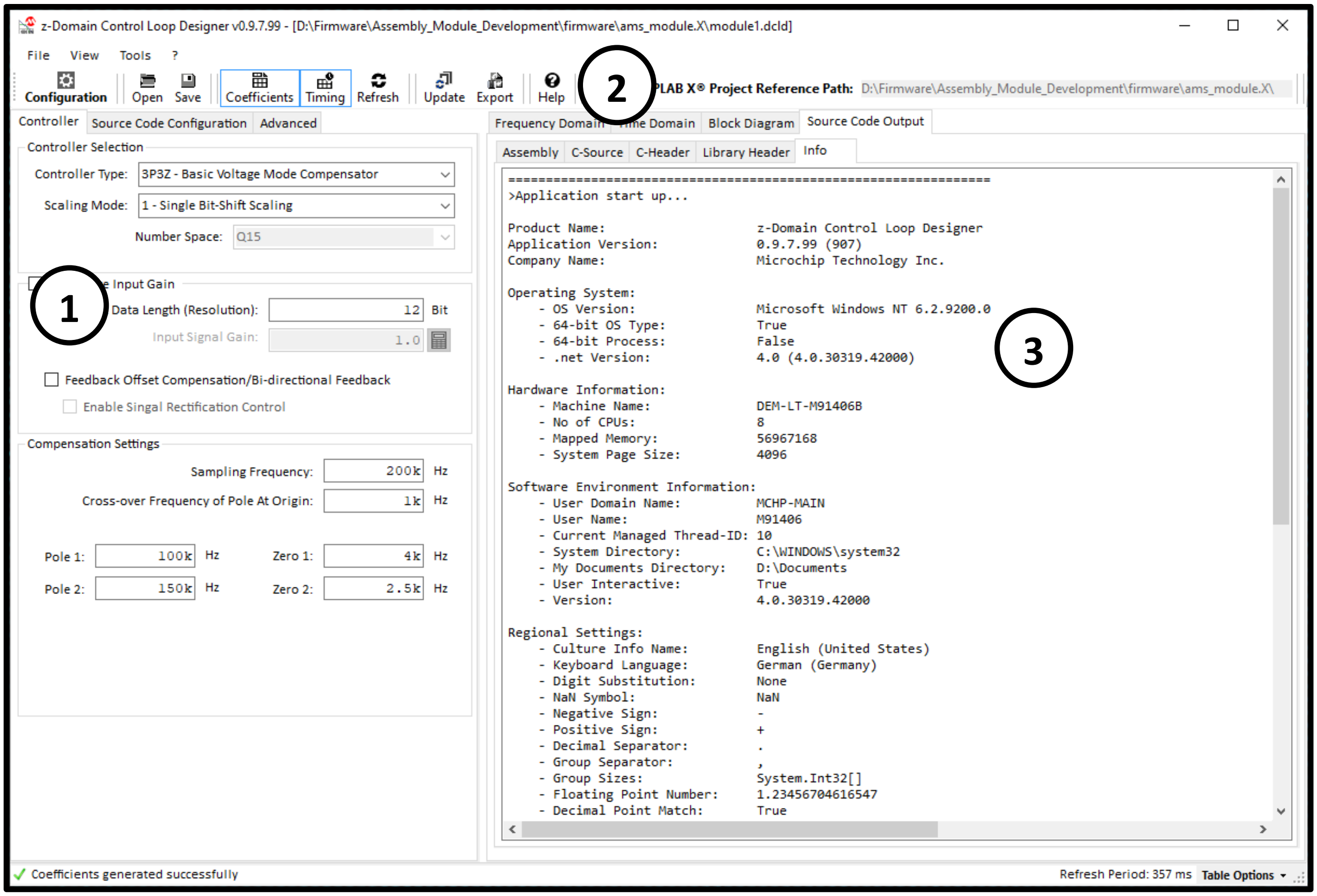


Figure 32: Output Window

TABLE 9: 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 and application startup information.

9.2 Process Output Window

More detailed information can be found in the process output window. It is located at the bottom of the Source Code Output tab on the right side of the main window. (see Figure 34). If the process output window is not visible, it can be opened from the View menu like shown in Figure 33.

The process output window displays internal process data generated during the update of coefficients, transfer function calculations, timing chart calculations, code generation and file export. In case of exceptions during any of these processes, detailed error messages will be generated, which can be used for further troubleshooting.

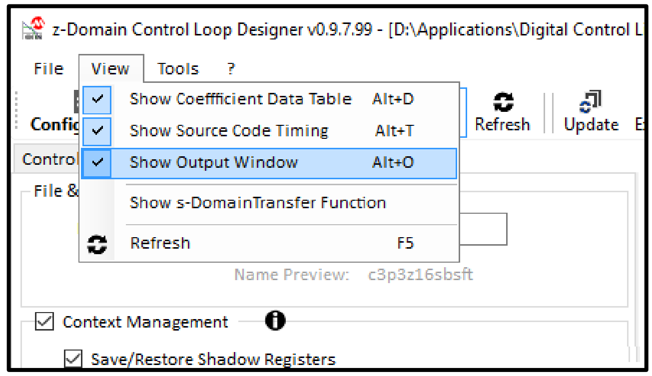


Figure 33: Open the Output Window from the View menu

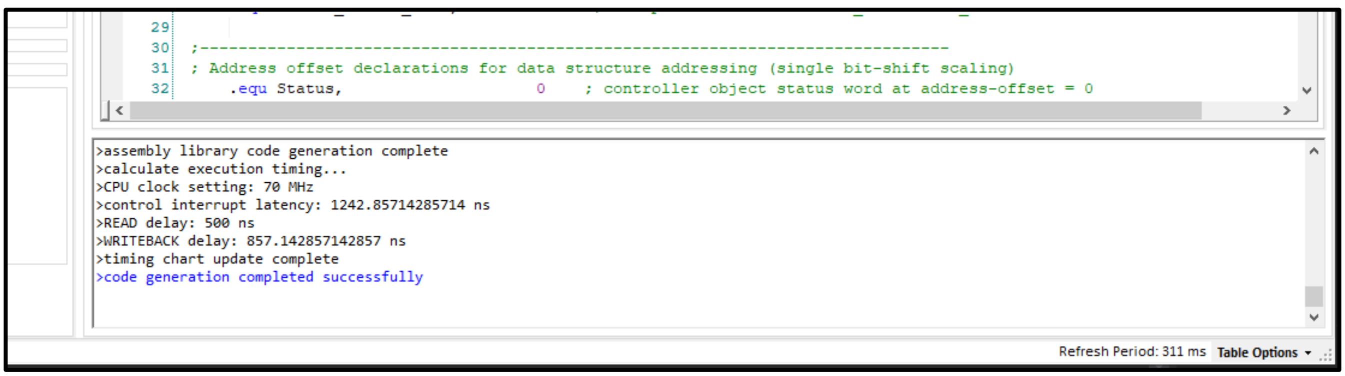


Figure 34: Process Output Window

10.0 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.

10.1 Multiple Controllers using the same Assembly Code

The DCLD code generator is creating the following four essential library files by default: (see Chapter 7.0 Code Generation)

- Assembly Library File
- Library C-Source File
- Library C-Header Files
- 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 35)

Figure 35: 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. *<u>Example:</u>* `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.

10.2 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 36).

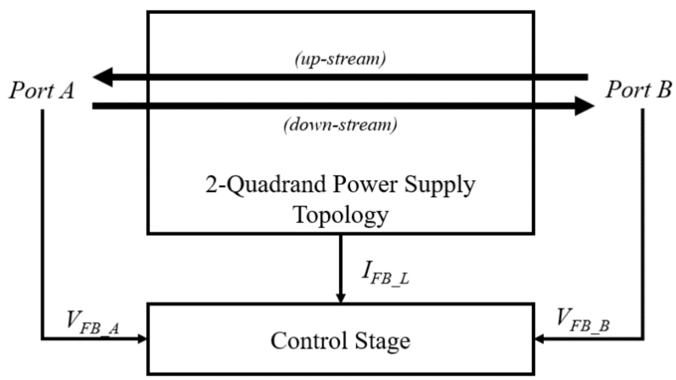


Figure 36: Bi-Directional Power Converter Block Diagram

- **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 37)

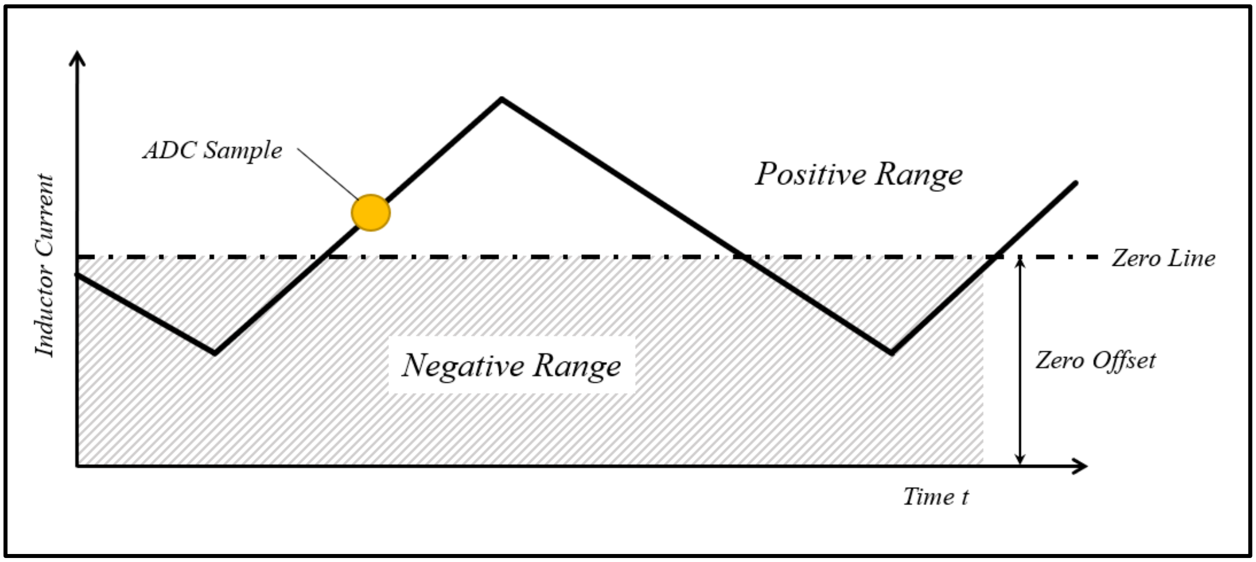


Figure 37: Bi-Directional Current Feedback Signal with Offset

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 38)

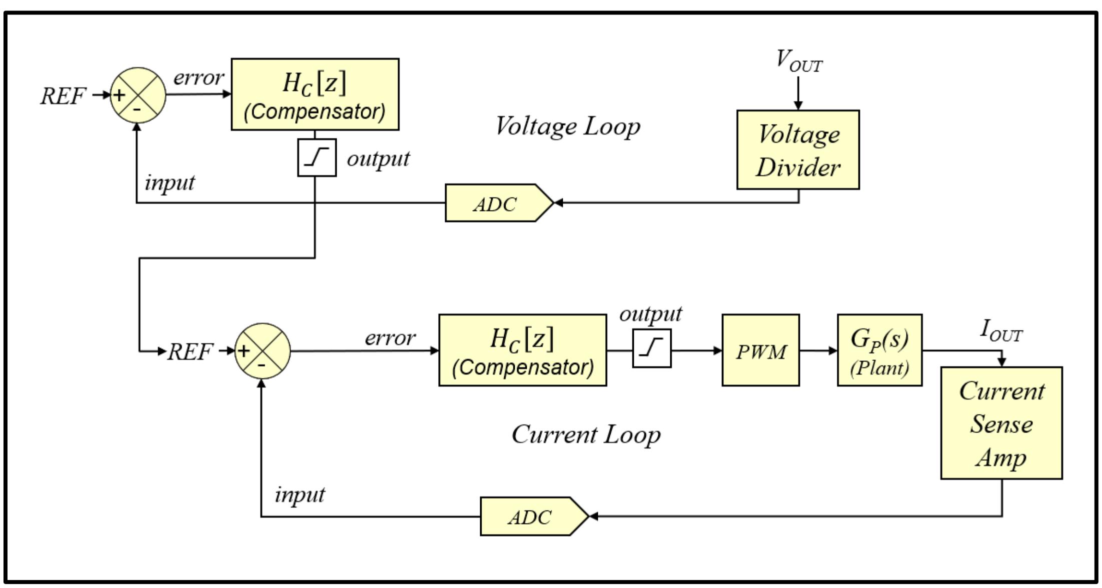


Figure 38: Standard Average Current Mode Control Feedback Loop Block Diagram

As shown in Figure 38, 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.Ports.Source.ptrAddress = &ADCBUF4; // ADC buffer #4
                                               // (output voltage)
    ```
  - Set data output target:
    ```
    V_LOOP.Ports.Target.ptrAddress = &I_REF; // V_LOOP output writes to
                                             // current reference
    ```
  - Set Current clamping values:
    ```
    V_LOOP.Limits.MinOutput = -200; // ADC ticks representing min current
    V_LOOP.Limits.MaxOutput = 1600; // ADC ticks representing max current
                                    // without offset (!!!)
    ```
  - Set voltage reference source:
    ```
    V_LOOP.Ports.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.Ports.Source.Offset` 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.Ports.Source.ptrAddress = &ADCBUF0; // ADC buffer #0
                                               // (inductor current)
    ```
  - Set data output target:
    ```
    I_LOOP.Ports.Target.ptrAddress = &PG1DC; // I_LOOP output writes to PWM1
                                             // Duty Cycle
    ```
  - Set Current clamping values:
    ```
    I_LOOP.Limits.MinOutput = 400; // PWM ticks representing min duty ratio
    I_LOOP.Limits.MaxOutput = 8000; // PWM ticks representing max duty ratio
    ```
  - Set current reference source:
    ```
    I_LOOP.Ports.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.CascadeTrigger.ptrCascadedFunction= (uint16_t)&i_loop_Update;
V_LOOP.CascadeTrigger.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 38 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 39).

The second problem we must 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 39.

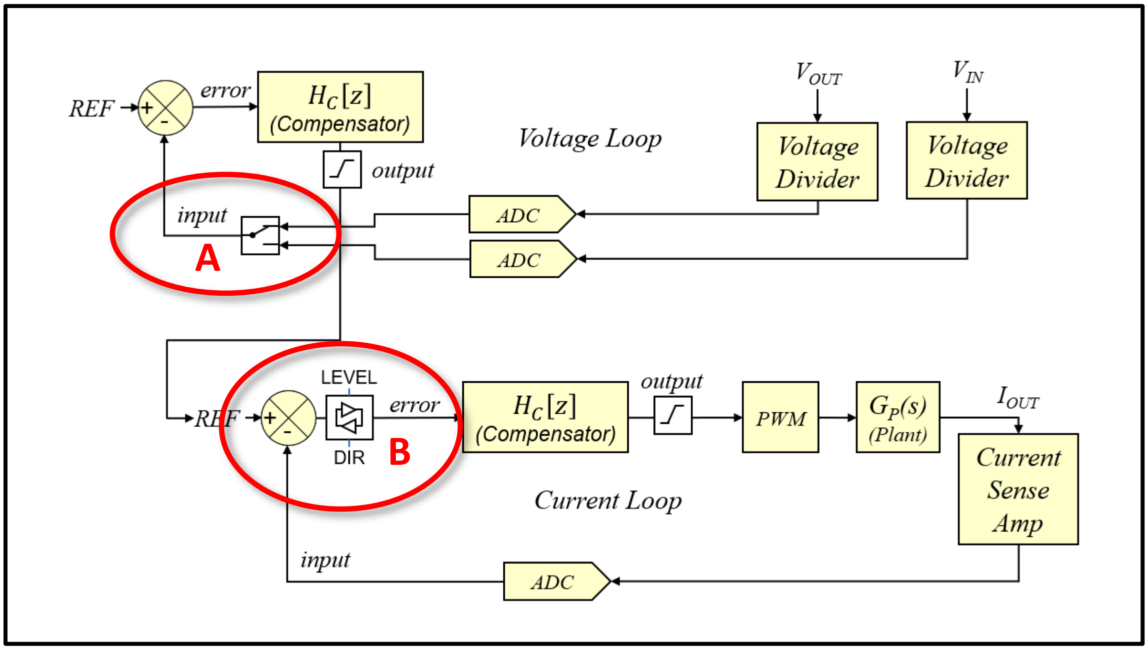


Figure 39: Bi-Directional Average Current Mode Control Feedback Loop Block Diagram

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
```


11.0 TABLE OF FIGURES

12.0 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.

SOFTWARE LICENSE AGREEMENT

Microchip licenses to you the right to use, modify, copy and distribute Software only when embedded on a Microchip microcontroller or digital signal controller, which is integrated into your product or third party product (pursuant to the sublicense terms in the accompanying license agreement). You should refer to the license agreement accompanying this Software for additional information regarding your rights and obligations. SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═**

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

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: