Application Report
SPRABR2–September 2013

LED Lighting and DC-DC Conversion Control Integrated on One C2000 Microcontroller

Brett Larimore

ABSTRACT

This application report presents a solution to control an LED lighting system using TMS320F2803x microcontrollers. The Piccolo™ TMS320F2803x series of devices are part of the family of C2000 microcontrollers which enable cost-effective design of LED lighting systems. With these devices it is possible to control multiple strings of LEDs in an efficient and very accurate way. In addition, the speed of the C2000 microcontroller allows it to integrate many supplemental tasks that would, in a normal system, increase chip count and complexity. These tasks could include DC-DC conversion stages, AC-DC conversion stages with PFC, system management, and various communication protocols such as DALI, DMX512, KNX or even power line communication. On the board described in this application report a DC-DC SEPIC stage has been integrated in addition to the dimmable control of eight individual LED strings.

This application report covers the following:

- A brief overview of LED Lighting technology.
- The advantages C2000 can bring to a lighting system.
- How to run and get familiar with the Lighting_DCDC project.

Contents

List of Figures

Piccolo is a trademark of Texas Instruments.
I^{2}C is a registered trademark of Philips Electronics.
All other trademarks are the property of their respective owners.

List of Tables

1 LED Lighting Theory

1.1 The Benefits of LEDs

As a relatively new and developing technology, LEDs have already become a valid solution in many lighting applications. One major advantage LEDs bring to applications is their high efficiency. Today's high brightness LEDs have a luminous efficiency of up to 60 lm/W with claims of greater than 100 lm/W being made from various LED manufacturers. Another advantage LEDs have over other light sources is their extensive life (approximately 50,000 hours if designed correctly). In applications such as street lighting, bulb replacement can be quite expensive when considering labor and loss of service. Therefore, LEDs have an advantage in these spaces. LEDs also have excellent vibration resilience, provide directional lighting, and allow for the ability to almost fully dim. These features, and more, allow LEDs to be perceived as the future in lighting technology.

1.2 Light and LED Characteristics

All light has two major characteristics: luminous flux and chromaticity. Luminous flux is an attribute of visual perception in which a source appears to be radiating or reflecting light. The term *brightness* is often given to describe this characteristic; however, this term is often used in a physiological and nonquantitative way. A better term is *luminous flux* which, measured in lumens, is the light power measured multiplied with the V-M scaling function. This function compensates for the sensitivity of the human eye to different wavelengths.

Chromaticity, or color, is then an objective specification of the color regardless of the light's luminous flux. Figure 1, the 1931 CIE Chromaticity Diagram, is a 2-dimensional projection of the RGB color system for the visible range. The x,y coordinate system created is then used as a reference for light meters. For instance, white light is often specified as being at 0.3, 0.3 in the 1931 CIE coordinate system.

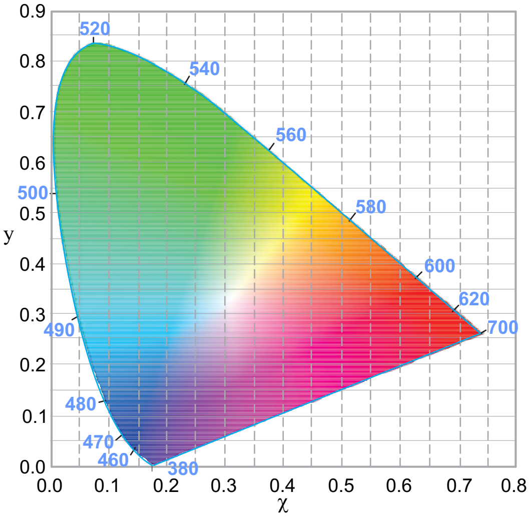


Figure 1. 1931 CIE Chromaticity Diagram

LEDs can be seen as a current-controlled device. The luminous flux and chromaticity are largely governed by the current that flows through the device. In Figure 2, taken from an LED data sheet, current is nearly proportional to the luminous flux output from an LED. Because of this fact, in many systems the average current is edited to dim an LED or LED string. The only other major variable that can affect luminous flux and chromaticity is temperature. Consequently, it is important to control temperature changes in an LED system.

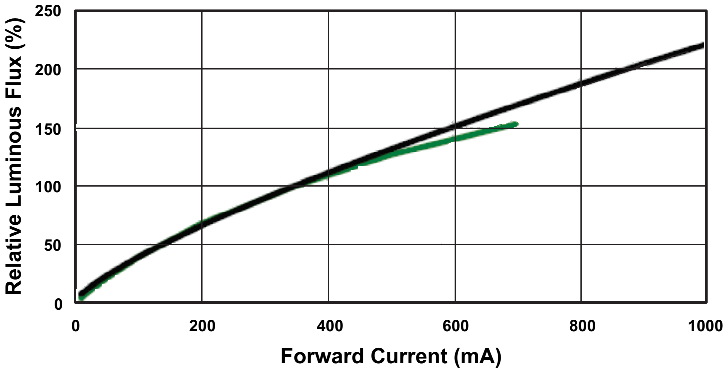


Figure 2. LED Current Versus Luminous Flux

Figure 3 shows the relationship between forward voltage and current in an LED.The LED behaves similarly to a diode in that it requires a certain threshold voltage before it begins conducting. Once the forward voltage becomes greater than the threshold voltage, the current increases exponentially until it reaches the maximum current specification of the LED device. For a string of six LEDs it is necessary to make the forward voltage larger than eight times the LED’s threshold voltage of the LED to make the LEDs light up.

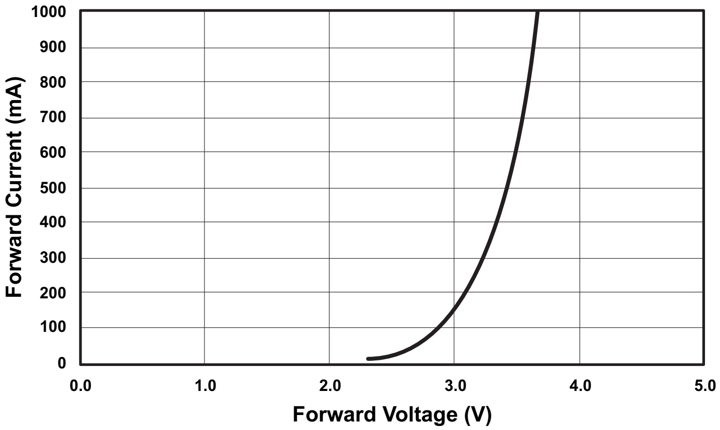


Figure 3. LED Forward Voltage Versus Current

Because of the relationship between LED voltage and LED current, it may seem reasonable to control the LED voltage to specify the LED current and therefore the luminous flux from the LED string. The problem here is that as temperature even slightly increases, Figure 3 keeps the same shape but shifts to the left. For instance, if one LED (with the specifications of Figure 3) was controlled at 3.0V, about 150mA of current will draw through the LED initially. However, as the LED warms up, the Figure 3 shifts to the right and the current increases slightly. This increased current increases the temperature. This cycle continues until the LED device fails. Because of this, current control of LED strings is highly preferred. It also helps to show the importance of thermal management in an LED system.

1.3 Control Techniques

There are various techniques for controlling the current for an LED string. In many cases a simple solution may be adequate, but for many other cases the number of LEDs in a string may be large or the total luminous flux needed is expected to be large. To maintain efficiency, a switched mode power supply(SMPS) will likely be needed. Figure 4 is one method of controlling one of these systems.

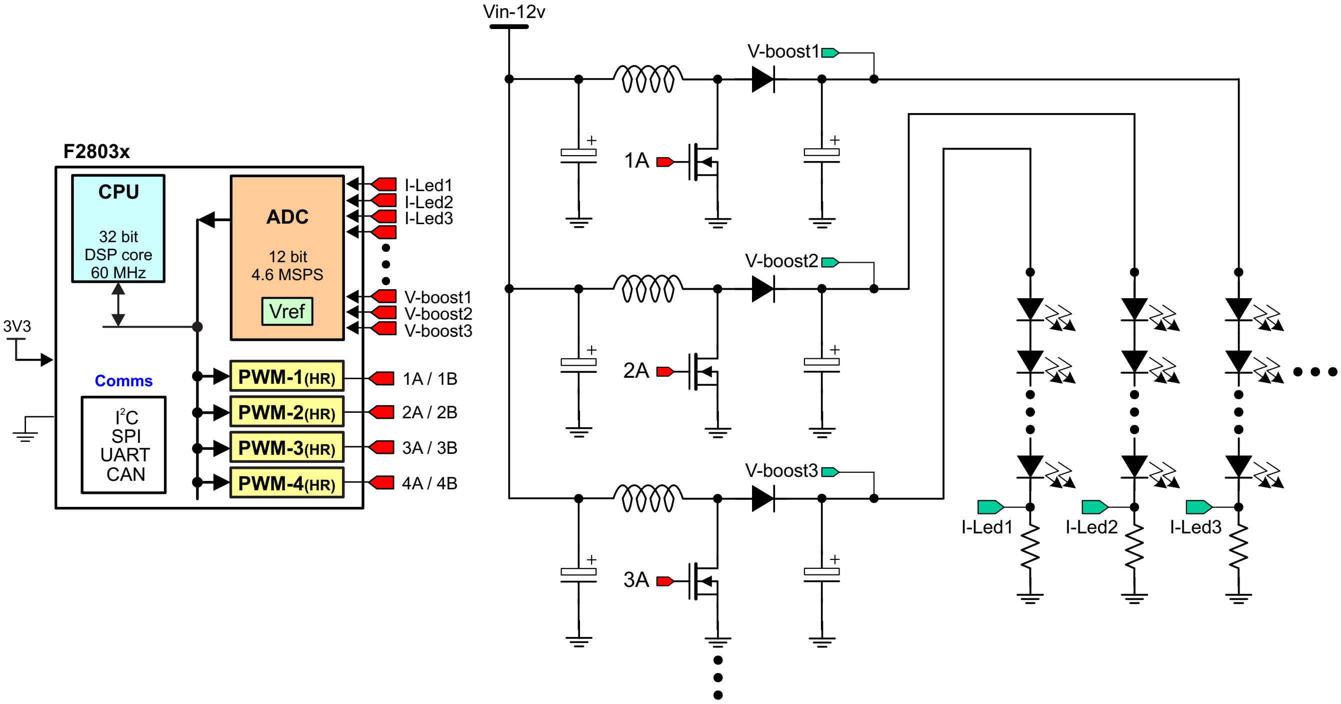


Figure 4. Individual Power Stage Per String

In Figure 4, each string is powered by a current-controlled boost stage. The Piccolo device provides the feedback loop by sampling each string's current from a resistor placed in series with the LED string. This sampled current is then compared to a reference within the controller and this helps to determine the next value of duty cycle sent to the MOSFETs in each individual DC-DC boost stage. In order to maintain the constant current needed, the PWM frequency of the FETs must be relatively large in order to prevent flickering.

The high-performance peripherals available on a Piccolo device can control the system shown in Figure 4 and still have a large amount of bandwidth to provide system management. The Piccolo peripherals, namely the on-chip comparator and the configurability of ADC sampling, allow the MCU to control the current through peak or average current mode, respectively.

Figure 5 shows a different technique. A single DC-DC stage provides the voltage that appears across the LED strings. This voltage corresponds to a single current that passes through an LED string. Each individual string is then switched on and off by a MOSFET so that the average power sent through an LED string is decreased. In this way, each individual LED string can be dimmed to a reference average current.

The DC-DC LED Lighting Kit implements this latter approach. The single DC-DC stage is a SEPIC converter and the Piccolo MCU also controls up to eight LED strings.

In either of these strategies, string interleaving can reduce the peak current and improve the overall efficiency. Piccolo PWMs allow for synchronization between individual PWM modules and with an external synchronization pulse.

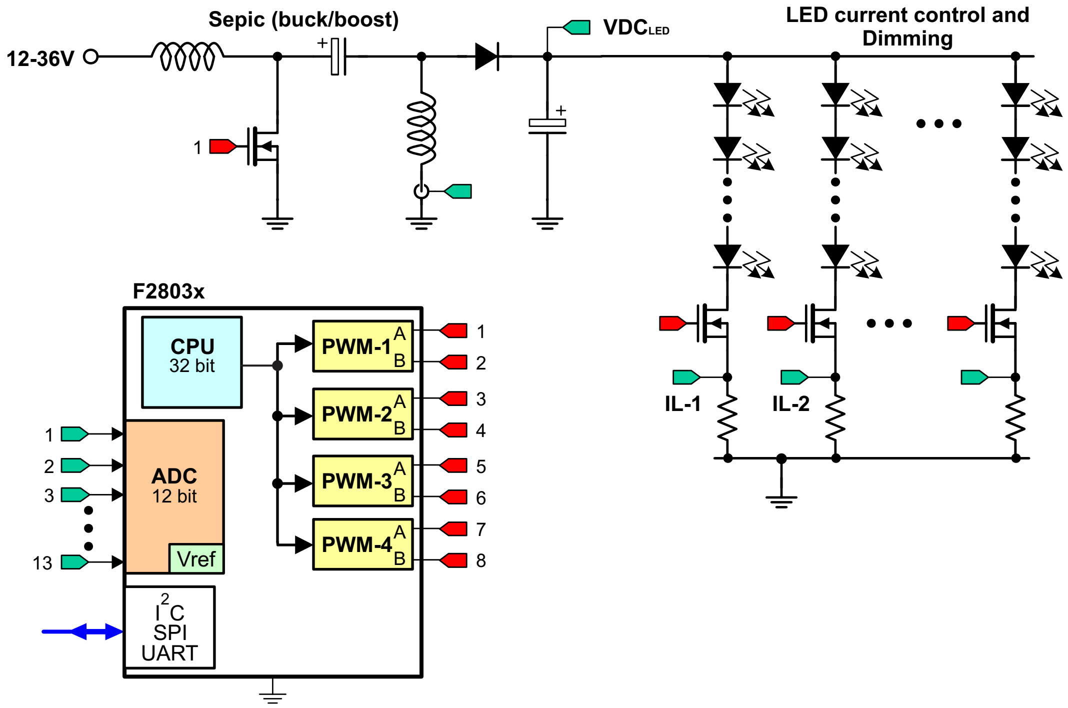


Figure 5. PWM Dimmed Strings With a Common Supply

1.4 An Efficient LED System

Any LED system must have power conversion stages to deliver the correct voltage and current to the LED strings. A typical system, like Figure 6, has an AC-DC rectifier, followed by a PFC boost circuit and then one or more parallel DC-DC stages drives one or more LED strings. To create an efficient system, each power stage must be designed to be efficient and the control of these power stages must be efficient.

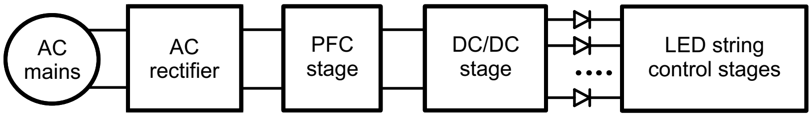


Figure 6. Lighting System

Communications also play a large role in the development of an intelligent, efficient system. A central area, or an application based on a mobile phone, could control the lighting in a home or office without the need to string control them individually. Many lighting standards, such as DALI, KNX, and DMX512, already exist. These standards, as well as others, exist as a twisted-pair, wireless, or power-line-communication (PLC) solutions.

With the advent of new and cheap MCUs like the C2000 Piccolo series and MSP430, an efficient system can be made not just by improving power stages and by using the most advanced LEDs. Intelligent lighting, in which the system is aware of its surroundings or is controlled by a networked host, can also play a large role in improving efficiency and reducing maintenance costs. Individual ballasts could use light sensors to sense ambient light and generate only the amount of light the area needs. An MCU could count the lifetime of an LED and compensate for the degeneration of light output with time and warn an external system if it has reached a predetermined lifetime in which the ballast could begin showing signs of failure.

With the high performance and software flexibility of the C2000 Piccolo series, string control, power stage control, communication, and intelligent lighting control can all be done on a single chip. Obviously, processor bandwidth will limit what can be done to some point, but in many systems the LED string controls, DC-DC stage, and some system control use less than 50% of the CPU bandwidth on the C2000 MCU.

2 Benefits of C2000 in LED Lighting

The C2000 family of devices posses the desired computation power to execute complex control algorithms and the right mix of peripherals needed in power stage control and LED string control. These peripherals have all the necessary hooks for implementing systems which meet safety requirements, like on-chip comparators and trip zones for the PWMs. Along with this, the C2000 ecosystem of software (libraries and application software) and hardware (application kits) help to reduce the time and effort needed to develop a lighting solution.

A C2000 lighting solution can provide the following benefits:

- Precise current matching between strings
- Accurate color matching
- Large range of dimmability; greater than 20,000:1 is possible
- Can perform PWM or constant current dimming
- Amount of LEDs in a string is not limited by the device. If a greater number of LEDs is needed in a particular product, only the MOSFET and MOSFET drivers would need to be checked and the software changes should be minor.
- Adaptive dimming based on LED use, aging, sensed external brightness, and more
- Easily synchronized to video clock through CAP and QEP peripherals
- Efficient control of PFC, AC-DC, DC-DC and more due to the power and flexibility of the C2000 peripheral set. Power supply control is a major strength of C2000 in the market.
- Large range of communication protocols such as I^{2}C®, SPI, and UART
- High-performance CPU allows the C2000 devices to do multiple tasks such as individual string control of multiple strings, DC-DC control, AC-DC control and communications on one chip.
- Because the device is programmed through software, creating products for multiple regions and multiple configurations often requires only minor changes in software.

3 System Overview

3.1 Hardware Overview

The DC-DC LED Lighting Developer's Kit takes in 12–36 V DC. This voltage is then regulated to a different level by a SEPIC converter. SEPIC, as a step-up and step-down converter topology, can increase or decrease the input voltage. For our application, we will set the SEPIC output voltage to approximately 20 V independent board input. The SEPIC output then connects to each LED string. To perform independent LED string dimming, a MOSFET is placed in series with each string and the on-time of this string's MOSFET will control the average current through an LED string. Because the brightness of an LED is roughly proportional to the LED current, we use the PWM duty cycle of each string to control the average current drawn. Figure 7 illustrates the hardware on the DC-DC LED Lighting Developer's Kit.

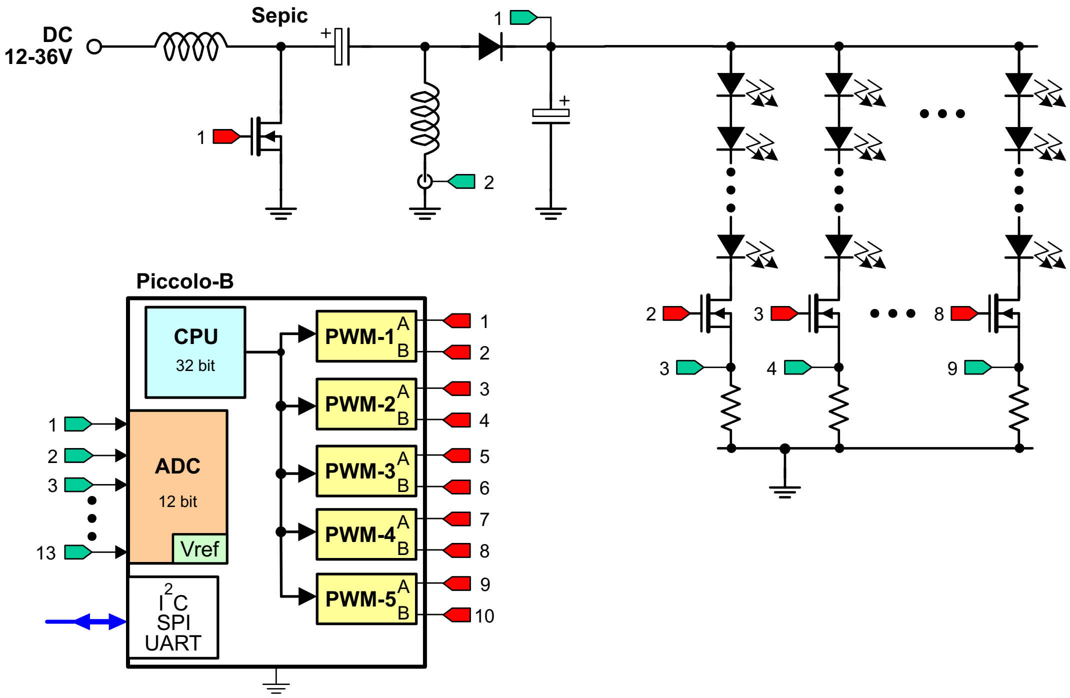


Figure 7. DC-DC LED Lighting Kit Hardware Block Diagram

Table 1 lists the key signal connections between the F28035 microcontroller and the DC-DC LED Lighting board.

Table 1. Key Peripherals Used

Signal Name	Description	Connection to Target MCU
EPWM-1A	SEPIC	GPIO-00
EPWM-1B	Not used	GPIO-01
EPWM-2A	LED string 1 (LED string 2A)	GPIO-02
EPWM-2B	LED string 2 (LED string 2B)	GPIO-03
EPWM-3A	LED string 3 (LED string 3A)	GPIO-04
EPWM-3B	LED string 4 (LED string 3B)	GPIO-05
EPWM-4A	LED string 5 (LED string 4A)	GPIO-06
EPWM-4B	LED string 6 (LED string 4B)	GPIO-07
ADC-A0	Current sense for LED string 2 (2B)	ADC-A0
ADC-A2	Current sense for SEPIC	ADC-A2
ADC-A3	Current sense for LED string 6 (4B)	ADC-A3
ADC-A4	Voltage sense for SEPIC	ADC-A4

Table 1. Key Peripherals Used (continued)

Signal Name	Description	Connection to Target MCU
ADC-A6	Voltage sense for Vin	ADC-A6
ADC-B0	Current sense for LED string 1 (2A)	ADC-B0
ADC-B1	Current sense for LED string 3 (3A)	ADC-B1
ADC-B2	Current sense for LED string 4 (3B)	ADC-B2
ADC-B3	Current sense for LED string 5 (4A)	ADC-B3

The Piccolo microcontroller, DC-DC LED Lighting board, and CCS project control two additional LED strings; however, the LED panel ships with six strings. Therefore, Table 1 contains only the hardware resources used for these six strings.

3.2 Software Overview

3.2.1 Build Options

In order for the user to slowly build up and understand the project, the project is divided into various builds separated by #if options in the Lighting_DCDC-Main.c and Lighting_DCDC-ISR.asm files. Which build is used is set by the variable INCR_BUILD in Lighting_DCDC-Settings.h.

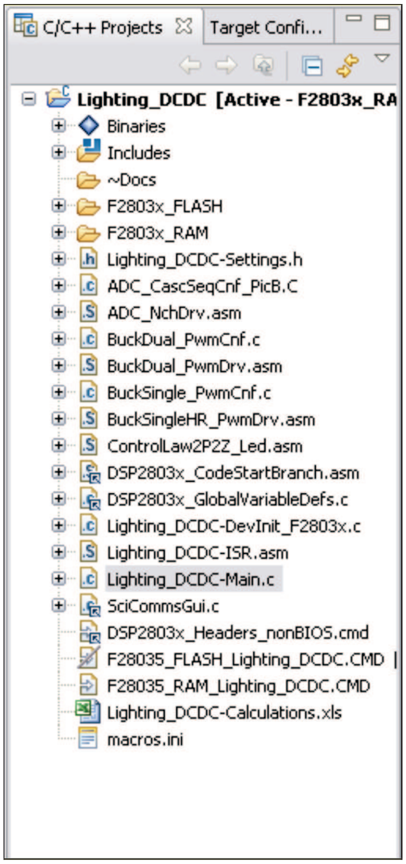
Following is a short description of the different builds available in the Lighting_DCDC project.

- Build 1: Open-Loop Test, check basic operation of the SEPIC and LED strings
- Build 2: Closed-Loop SEPIC, LED string run in open-loop test mode
- Build 3: Closed-Loop SEPIC (voltage) and closed-loop LED dimming control (current)

Table 2. Major Variables

Gui_Vin	The DC Input Voltage (0-66V), for Instrumentation purpose only
Gui_Vsepic	The Sepic Output Voltage (0-50V), for Instrumentation purpose only
Gui_ILed[1-8]	Each LED strings' Current (0-0.20A), for Instrumentation purpose only
Gui_VsetSepic	The Voltage Reference for the SEPIC Output Voltage (0-50V).
Pgain_Sepic	Proportional gain for the SEPIC; value adjustment: 0 – 1000
Igain_Sepic	Integral gain for the SEPIC; value adjustment: 0 – 1000
Dgain_Sepic	Derivative gain for the SEPIC; value adjustment: 0 – 1000
SlewStep	This value determines how fast the soft start mechanism would let the duty cycle reach the desired output of the SEPIC (approximately 1-50)
Gui_IsetLed[1-8]	This value determines the current reference for each LED string. (0 – 0.20A)
Dmax_LED	Maximum duty cycle allowed for each LED string
Pgain_LED	Proportional gain for each LED string; value adjustment: 0 – 1000
Igain_LED	Integral gain for each LED string; value adjustment: 0 – 1000
Dgain_LED	Derivative gain for each LED string; value adjustment: 0 – 1000
SlewStep_LED	This value determines how fast the soft start mechanism would let the current reach the desired LED string current – (1-50)
LEDs_linked	LEDs_linked = 0: LEDs are independently controlled. Gui_IsetLed[1-8] control strings 1-8 LEDs_linked = 1: LEDs are all controlled through the Gui_IsetLed[1] variable
Duty[0]	Sets the SEPIC duty cycle in Incremental Build 1
Duty[1-8]	Sets the duty cycle (and therefore the on-time) of LED string 1-8. This duty cycle corresponds to a dimming control.

3.2.2 Key Files in the Project



```
Lighting_DCDC [Active - F2803x_RA
    Binaries
    Includes
    ~Docs
    F2803x_FLASH
    F2803x_RAM
    Lighting_DCDC-Settings.h
    ADC_CascSeqCnf_PicB.C
    ADC_NchDrv.asm
    BuckDual_PwmCnf.c
    BuckDual_PwmDrv.asm
    BuckSingle_PwmCnf.c
    BuckSingleHR_PwmDrv.asm
    ControlLaw2P2Z_Led.asm
    DSP2803x_CodeStartBranch.asm
    DSP2803x_GlobalVariableDefs.c
    Lighting_DCDC-DevInit_F2803x.c
    Lighting_DCDC-ISR.asm
    Lighting_DCDC-Main.c
    SciCommsGui.c
    DSP2803x_Headers_nonBIOS.cmd
    F28035_FLASH_Lighting_DCDC.CMD
    F28035_RAM_Lighting_DCDC.CMD
    Lighting_DCDC-Calculations.xls
    macros.ini
```

Figure 8. Lighting_DCDC Project Files

The key framework files in this project are:

- Lighting_DCDC-Main.c: This file is used to initialize, run, and manage the application. Lighting_DCDC-Main.c is the primary file for the application.
- Lighting_DCDC-DevInit_F2803x.c: This file is responsible for a one-time initialization and configuration of the device (in this case, the F28035), and includes functions such as setting up the clocks, PLL, GPIO, and more.
- Lighting_DCDC-ISR.asm: This file contains all time critical “control type” code. This file has an initialization section that is executed one time by the C-callable assembly subroutine _ISR_Init. The file also contains the _ISR_Run routine, which executes at the same rate as the SEPIC PWM which is used to trigger it.
- ProjectSettings.h: This file is used to set global definitions (build options) for the project . This file is linked to both LedBacklight_Main.c and LedBacklight-ISR.asm.
- Lighting_DCDC-Calculations.xls: This is a spreadsheet file that calculates the values of the various scaling factors used in converting Q-value numbers, used by the MCU, to real world values. The variables K_Vin and iK_Vsepic are examples of scaling factors.

3.2.3 Macros Used

To reduce the effort for developing code each time, an approach of using Macro Blocks is used. These macro blocks are written in C-callable assembly and can have a C and assembly component. Table 3 lists the macros used in this project.

Table 3. Macros Used

C Configuration Function	ASM Initialization Macro	ASM Run Time Macro
BuckSingle_CNF	None	None
BuckDual_CNF	None	None
None	BuckSingle_DRV_INIT n	BuckSingle_DRV n
None	BuckDual_DRV_INIT n	BuckDual_DRV n
None	ControlLaw_2P2Z_INIT n	ControlLaw_2P2Z n
ADC_CascSeqCNF	ADC_NchDRV_INIT n	ADC_NchDRV n

As the configuration of the peripherals is done in C, it can be easily modified. The ASM drive macros provide the necessary compact code to run in real time. To better understand the role of each macro in the project, brief descriptions follow:

BuckSingle_CNF ()—Defined in BuckSingle_PwmCnf.c, this function configures the PWM channels as specified in the arguments, to drive the power stage. For details on how and what arguments are passed, see the source file. Changes may be needed to match the mappings of PWMs if using a different board.

BuckDual_CNF ()— Defined in BuckDual_PwmCnf.c, this function configures the PWM channels as specified in the arguments, to drive the power stage. For details on how and what arguments are passed, see the source file. Changes may be needed to match the mappings of PWMs if using a different board.

BuckSingle_DRV n—Defined in BuckSingle_PwmDrv.asm, this macro is used to update PWM[m]A in the ISR as configured by the CNF files.

BuckDual_DRV n— Defined in BuckDual_PwmDrv.asm, this macro is used to update PWM[m]A and PWM[m]B in the ISR as configured by the CNF files.

ControlLaw2P2Z n— This is a second-order compensator realized from an IIR filter structure. The five coefficients needed for this function are declared in the C background loop as an array of longs. This function is independent of any peripherals and therefore does not require a CNF function call. Defined in ControlLaw2P2Z.asm, this is a macro that is part of the TI PowerLib.

ADC_CascSeqCNF(), ADC_NchDRV n—Defined in ADC_CascSeqCnf_Pic.c and ADC_NchDrv.asm, This Macro abstracts the use of ADC module, all that nmust be done is call the configuration function with the correct array of channel numbers and enables. The driver macro copies the result from the ADCRegisters into a NetBus array variable, which can be used to connect to various library macros.

Table 4, Table 5, and Table 6 list the properties of the Lighting_DCDC project.

Table 4. Memory Usage of the Lighting_DCDC Project

Build Level	Program Memory Usage 2803x	Data Memory Usage 2803x[1]
Build 3 (RAM)	4489 words	719 words

[1] Excluding the stack size.

Table 5. CPU Use of Build 3 of the Lighting_DCDC Project

Name of Modules[1]	Number of Cycles
Context Save, and more	35
ADC_NchDRV 12	28
Timeslicing Overhead	10
SEPIC Control (updated every ISR)	
ControlLaw_2P2Z	30
BuckSingleHR_DRV	12
LED Control (2 strings updated each ISR)	
ControlLaw_2P2Z * 2	60
BuckDual_DRV	11
Total Number of Cycles	186
CPU Use @ 60 MHz	31.0%[2]
CPU Use @ 40 MHz	46.5%[2]

[1] The modules are defined in the header files as “macros”.

[2] At 100 kHz ISR frequency.

Table 6. System Features

Development / Emulation	Code Composer Studio v4.1 (or above) with real time debugging
Target Controller	TMS320F2803x
PWM Frequency	100-kHz PWM (Sepic); 20-kHz PWM (LED sting)
PWM Mode	Symmetrical with a programmable dead band
Interrupts	EPWM1 Time Base CNT_Zero – Implements 100 kHz ISR execution rate

4 Hardware Setup

In this guide, each component is named with its macro number followed by its reference name. For example, [M2]-J1 refers to the jumper J1 in the macro M2, and [Main]-J1 refers to the J1 on the board outside of the other defined macro blocks. Figure 9 shows some of the major connectors and features of the DC-DC LED Lighting board.

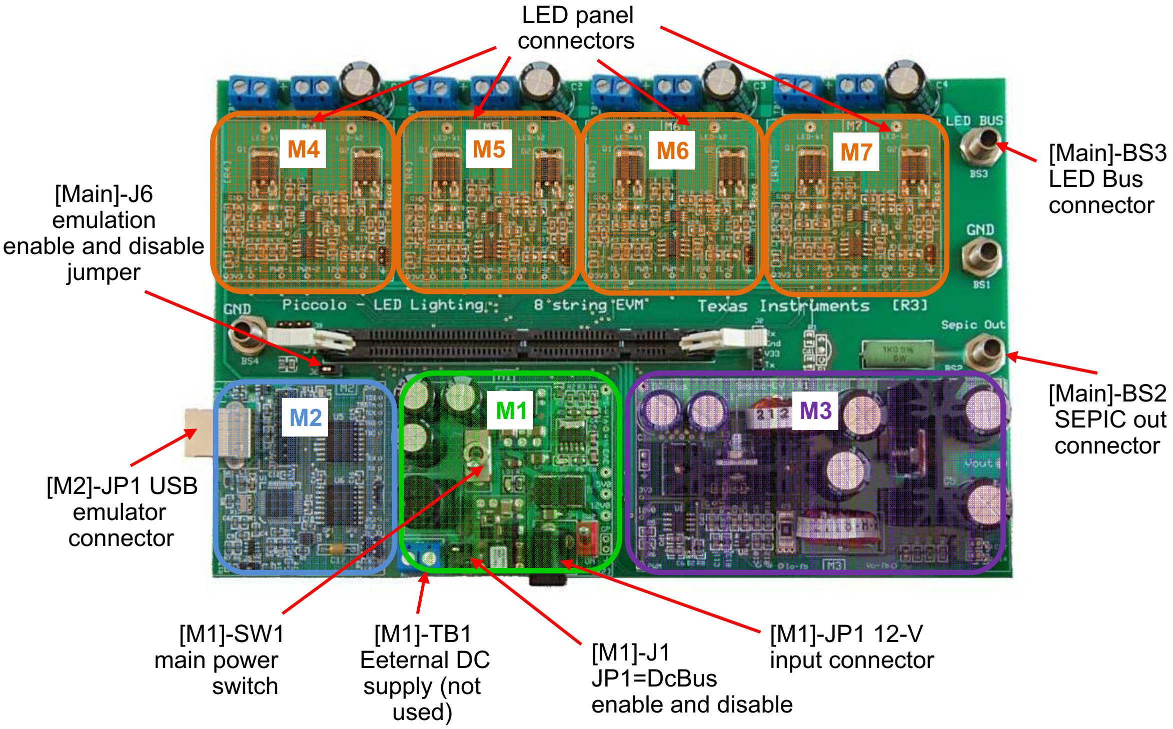
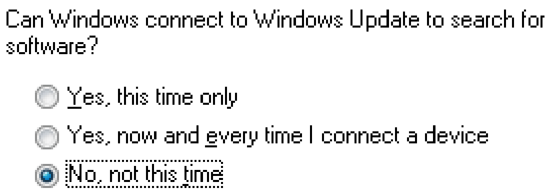

LED panel connectors
M4
M5
M6
M7
[Main]-BS3 LED Bus connector
[Main]-J6 emulation enable and disable jumper
Piccolo - LED Lighting 8 string EVM Texas Instruments [R3]
M2
M1
M3
[Main]-BS2 SEPIC out connector
[M2]-JP1 USB emulator connector
[M1]-SW1 main power switch
[M1]-TB1 Eeternal DC supply (not used)
[M1]-J1 JP1=DcBus enable and disable
[M1]-JP1 12-V input connector

Figure 9. Hardware Features

1. On the Piccolo F28035 controlCARD, check the following switches:
   - SW1: Ensure this switch is in the off (down) position.
   - SW2: Ensure position 1 and 2 are both in the on (up) position.
2. Insert an F28035 control card into the socket on the LED Lighting board.
3. Connect a cable from the USB connector on the board to the computer. [M2]-LD1, near the USB connector on the LED lighting board, should turn on.

   Note: If you have never installed Code Composer Studio, you may need to install drivers to ensure the board works correctly. If a window appears when the USB cable is connected from the board to the computer, use the install wizard to install drivers from the XDS100v1 directory of the USB drive included with this kit.

   (a) When Windows asks to search Windows Update, select “No, not this time” and click Next.



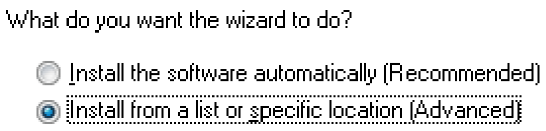
Can Windows connect to Windows Update to search for software?

○ Yes, this time only

○ Yes, now and every time I connect a device

● No, not this time

   (b) On the next screen, select “Install from a list or specific location” and click Next.

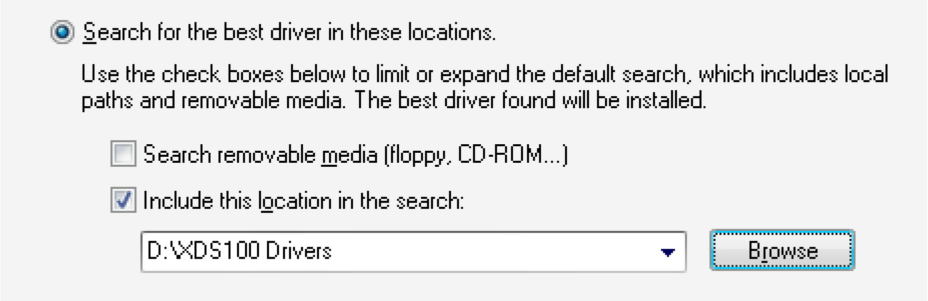


What do you want the wizard to do?

○ Install the software automatically (Recommended)

● Install from a list or specific location (Advanced)

(c) Select “Search for the best driver”, uncheck "Search removable media", and check "Include this location in the search" and browse to [USB Drive]:\XDS100 Drivers



● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

□ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

D:\XDS100 Drivers

(d) Click Next and the drivers will install. The driver install screen will appear three times. Repeat this procedure each time.

4. Connect the LED panel to the LED Lighting board through [Main]-TB1 to TB8.
5. Connect of verify the following:
   - Connect a jumper on [M1]-J1.
   - Connect a jumper on [M2]-J4.
   - Connect a jumper on [Main]-J6.
6. Connect the banana-to-banana plug wire that came with the kit between the SEPIC out Connector (BS2) and the LED bus connector (BS3).
7. Place [M1]-SW2 into the off position. This switch is not used in this demonstration.
8. Connect a 12-V power supply to power up to [M1]-JP1 of the board.
9. Turn [M1]-SW1 to the on position. When on, [M1]-LD1 and [M1]-LD2 should turn on.

5 Software Setup

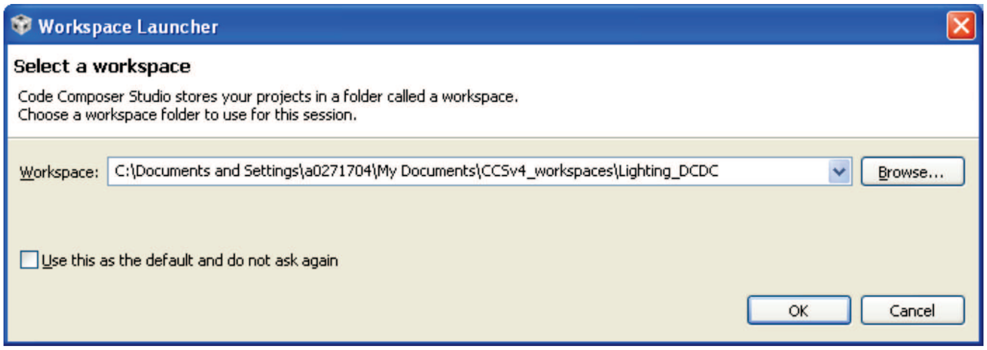
5.1 Installing Code Composer and controlSUITE

1. If not already installed, install Code Composer v4.x from the DVD included with the kit.
2. Download and run the controlSUITE installer from http://www.ti.com/controlsuite. Install the “DC/DC LED Lighting” software and allow the installer to also download all automatically checked software.

5.2 Setup Code Composer Studio to Work With the DC-DC LED Lighting Kit

1. Open Code Composer Studio v4.

   Once Code Composer Studio opens, the workspace launcher may appear and ask to select a workspace location. Note: The workspace is a location on the hard drive where all the user settings for the IDE are saved, such as which projects are open and which configuration is selected. This location can be anywhere on the disk. The location mentioned below is just for reference. Also note that if this is not your first time running Code Composer Studio this dialog may not appear.



Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

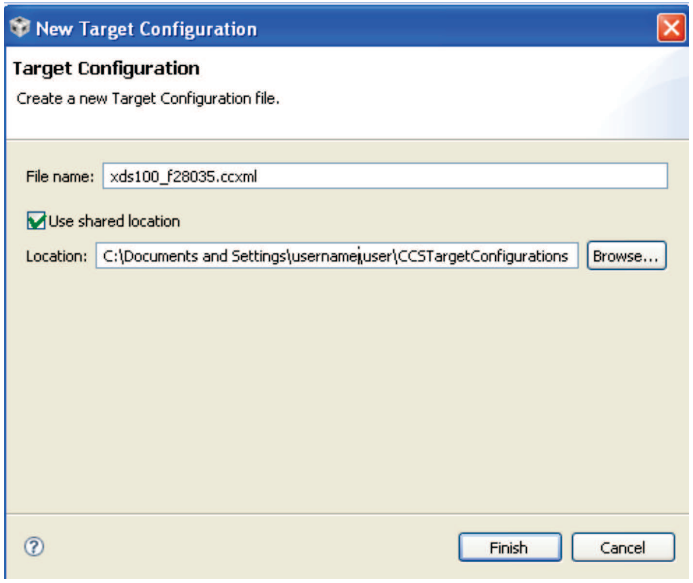
Workspace: C:\Documents and Settings\a0271704\My Documents\CCSv4_workspaces\Lighting_DCDC

□ Use this as the default and do not ask again

Figure 10. Workspace Launcher

2. Click the Browse… button.
3. Make new folders as necessary to create the following path: C:\Documents and Settings\My Documents\CCSv4_workspaces\Lighting_DCDC.
4. Uncheck the box that labeled “Use this as the default and do not ask again”.
5. Click OK.

   Configure Code Composer to determine to which MCU it will connect.
6. Click Target → New Target Configuration.
7. Name the new configuration xds100-f28035.ccxml. Ensure that the “Use shared location” checkbox is checked and then click Finish.



New Target Configuration

Target Configuration

Create a new Target Configuration file.

File name: xds100_f28035.ccxml

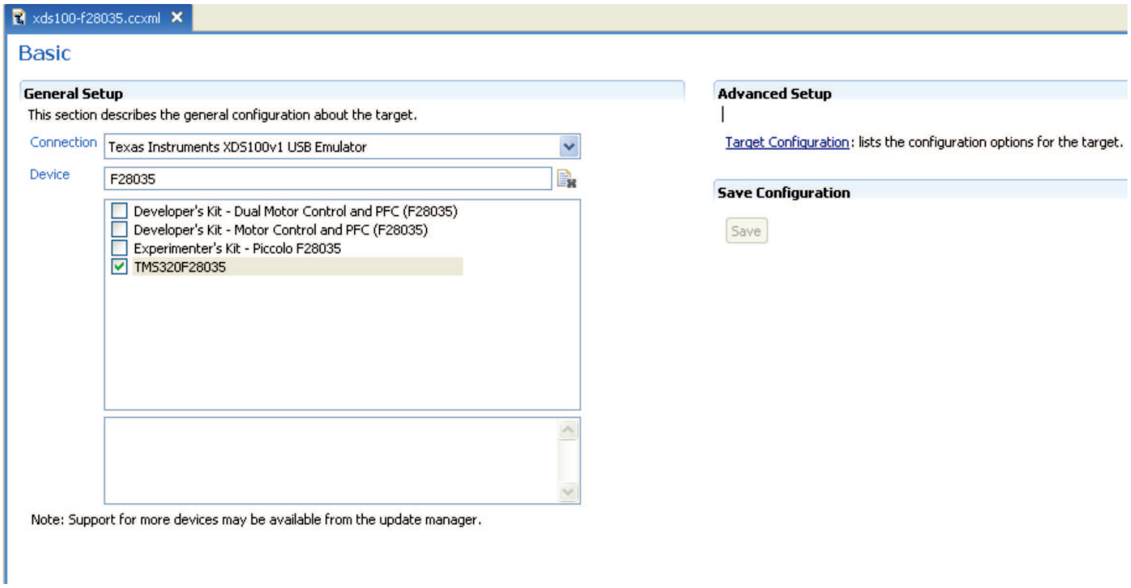
☑ Use shared location

Location: C:\Documents and Settings\username\user\CCSTargetConfigurations

Figure 11. Creating a Target Configuration

A new tab opens, seen in Figure 12.

8. Select and enter the options as shown:
   - Connection: Texas Instruments XDS100v1 USB Emulator
   - Device: TMS320F28035
9. Click Save.
10. Close the xds100-f28035.ccxml tab.



xds100-f28035.ccxml

Basic

General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS100v1 USB Emulator

Device: F28035

□ Developer's Kit - Dual Motor Control and PFC (F28035)
□ Developer's Kit - Motor Control and PFC (F28035)
□ Experimenter's Kit - Piccolo F28035
☑ TMS320F28035

Note: Support for more devices may be available from the update manager.

Advanced Setup

Target Configuration: lists the configuration options for the target.

Save Configuration

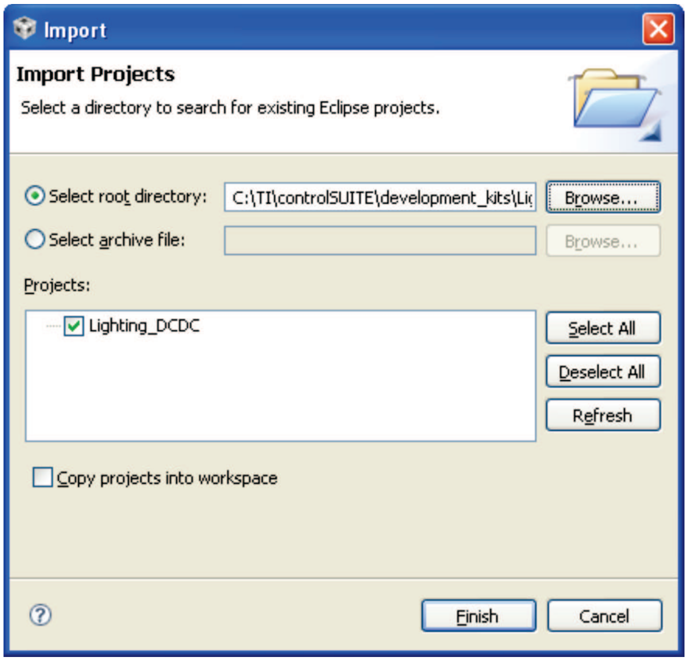
Figure 12. Configuring a New Target

Assuming this is your first time using Code Composer, the xds100-F28035 configuration is now set as the default target configuration for Code Composer.

11. Check the default configuration by selecting View →Target Configurations.
12. In the User Defined section, right-click on the xds100-F28035.ccxml file.
13. Select Set as Default.

This tab also lets you reuse existing target configurations and link them to specific projects.

14. Add the DC-DC LED Lighting project into your current workspace by clicking Project → Import Existing CCS/CCE Eclipse Project.
15. Select the root directory of the DC-DC LED Lighting Kit: C:\TI\controlSUITE\development_kits\Lighting_DCDC_vX.X\.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\TI\controlSUITE\development_kits\Li(

○ Select archive file:

Projects:

☑ Lighting_DCDC

□ Copy projects into workspace

Figure 13. Adding the DC-DC LED Lighting Kit Project to Your Workspace

16. Click Finish. The Lighting_DCDC project will copy into the workspace.

5.3 Configuring a Project

The Lighting_DCDC project should be the active project.

1. Right-click on the project name and click “Set as Active Project”.
2. Expand the file structure of the project.

   Each project can be configured to create code and run in either flash or RAM. You may select either of the two. However, for lab experiments we will use RAM configuration to simplify the debug process and then move to the FLASH configuration for production.
3. As shown in Figure 14, right-click on an individual project and select Active Build Configuration → F2803x_RAM configuration.

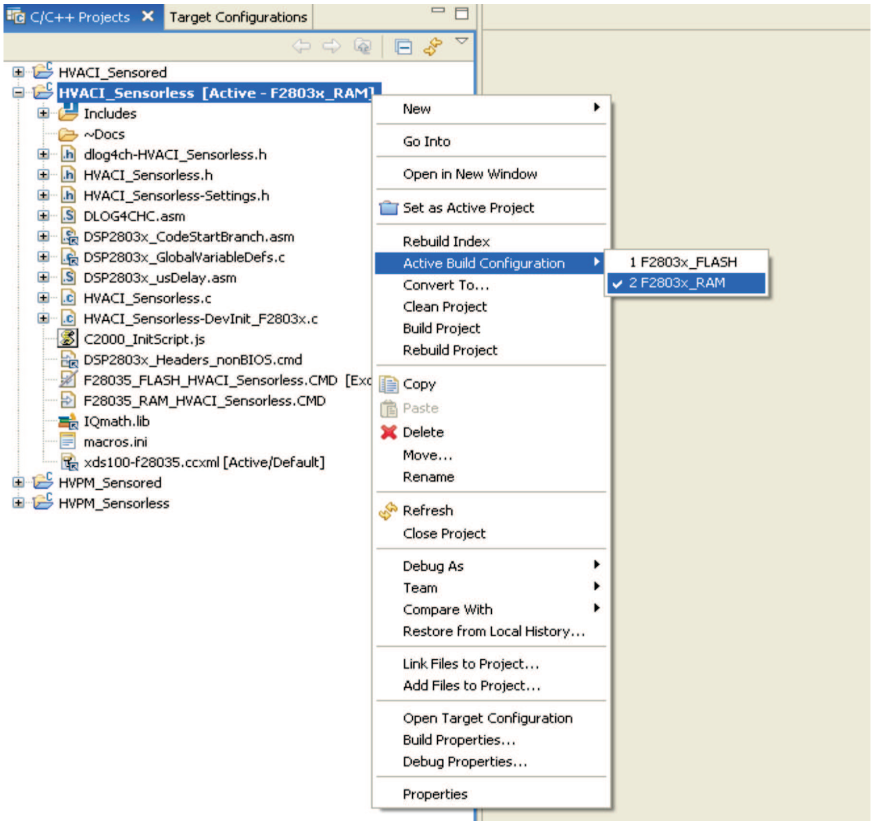


Figure 14. Selecting the F2803x_RAM configuration

5.4 Incremental System Build for Lighting_DCDC project

The lighting system is gradually built up in order for the final system to confidently operated. Three phases (system builds) are designed to verify the major software modules used in the system. A short description of each build follows:

- Build 1: Open-Loop Test, check basic operation of the SEPIC and LED strings
- Build 2: Closed-Loop SEPIC, LED string run in open-loop test mode
- Build 3: Closed-Loop SEPIC (voltage) and closed-loop LED dimming control (current)

Table 7 summarizes the macro modules tested and used in each incremental system build.

Table 7. Testing Modules in Each Incremental System Build

Software Module	Phase 1[1]	Phase 2[1]	Phase 3[1]
BuckSingleHR_DRV	√√	√	√
BuckDual_DRV	√√	√	√
ADC_NchDRV	√√	√	√
ControlLaw_2P2Z		√√	√√

[1] Note: the symbol √ means this module is using and the symbol √√ means this module is testing in this phase.

6 Build 1: Open Loop Sepic Control and Open Loop LED String Control

NOTE: This section assumes you have completed Section 4, *Hardware Setup*, and Section 5, *Software Setup*. Complete these sections before continuing.

The objective of this build is as follows:

- Verify that the power stages on the board are working correctly.
- Control the duty cycles of the various power stages on the board and evaluate their output.

Figure 15 describes the components of the system as used in the software.

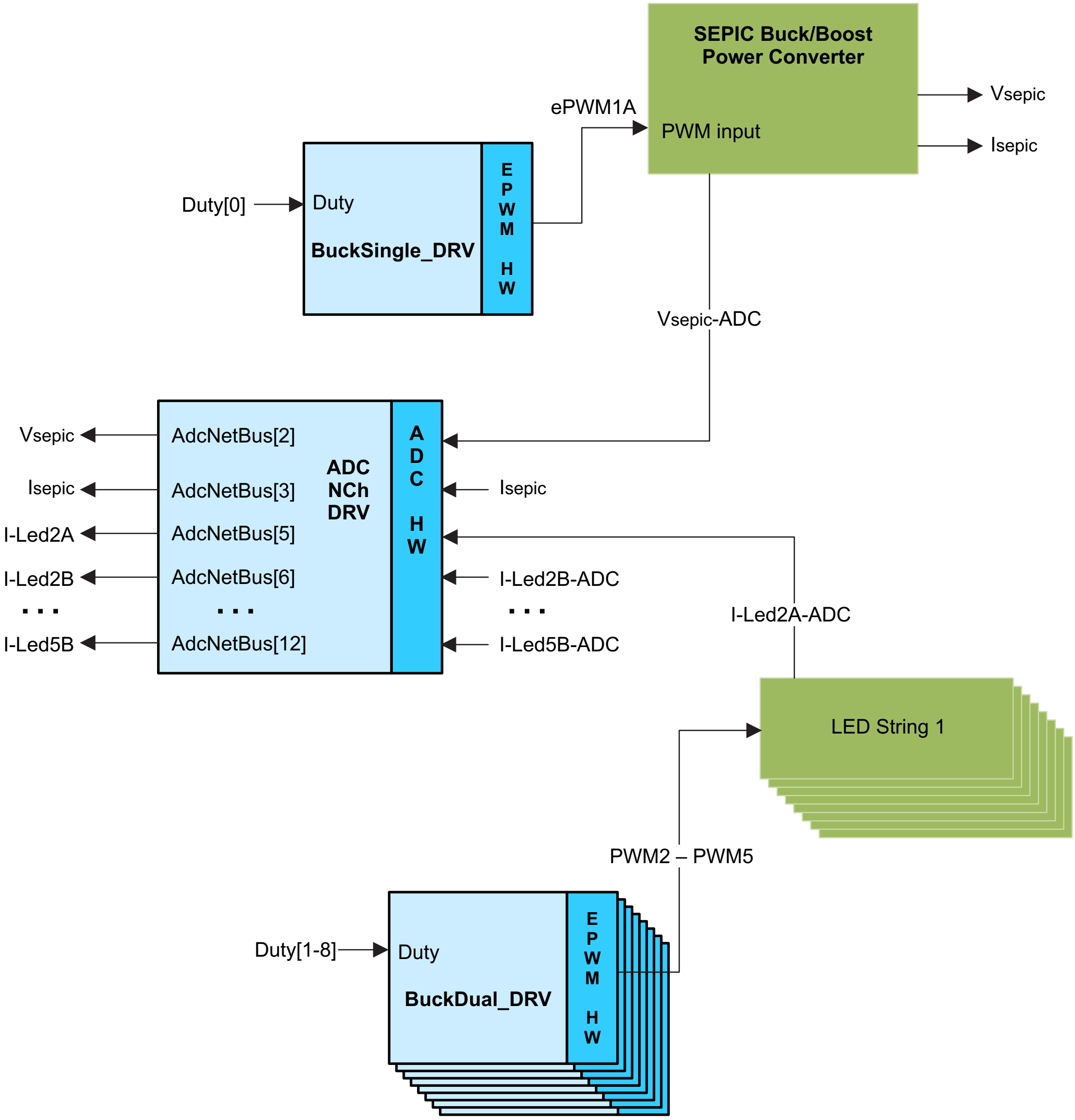


Figure 15. Build Level 1 Block Diagram

6.1 Inspect the Project

After the initial boot processes are complete, the software starts from the main function.

1. Open up Lighting_DCDC-Main.c and find the main() function on line 252.

   The first thing that the software does in the main() function is call a function called DeviceInit() found in Lighting_DCDC-DevInit_F2803x.c.

2. Open and inspect Lighting_DCDC-DevInit_F2803x.c by double-clicking on the filename in the project window.

   In this file, the various peripheral clocks are enabled and disabled, and the functional pinout, which configures which peripherals come out of which pins, is defined.

3. Confirm that the ADC and PWM1-5 peripheral clocks are enabled (lines 107–123). Also confirm that GPIO00 and GPIO02–GPIO09 are configured to be PWM outputs (lines 158–216).

   The Lighting_DCDC project is provided with incremental builds where different components and macro blocks of the system are pieced together one by one to form the entire system. These incremental builds help in step-by-step debug and understanding of the system.

4. From the C/C++ Project tab, open the Lighting_DCDC-Settings.h file.
5. Ensure that INCR_BUILD is set to 1 and save this file.

   After we test build 1, this variable will need to be redefined to move on to build 2, and so on, until all builds are complete.

6. Return to the Lighting_DCDC-Main.c file and notice the various incremental build's configuration code. The Build LEVEL1 configuration code begins at line 403 and continues to line 513. In it the ADC, PWM and macro block connections are configured. If INCR_BUILD is set to 1 in Lighting_DCDC-Settings.h lines 629-797 will be ignored by the compiler because they do not belong in the current build. A similar method of enabling or disabling code based on the value of INCR_BUILD is done in the Lighting_DCDC-ISR.asm file.

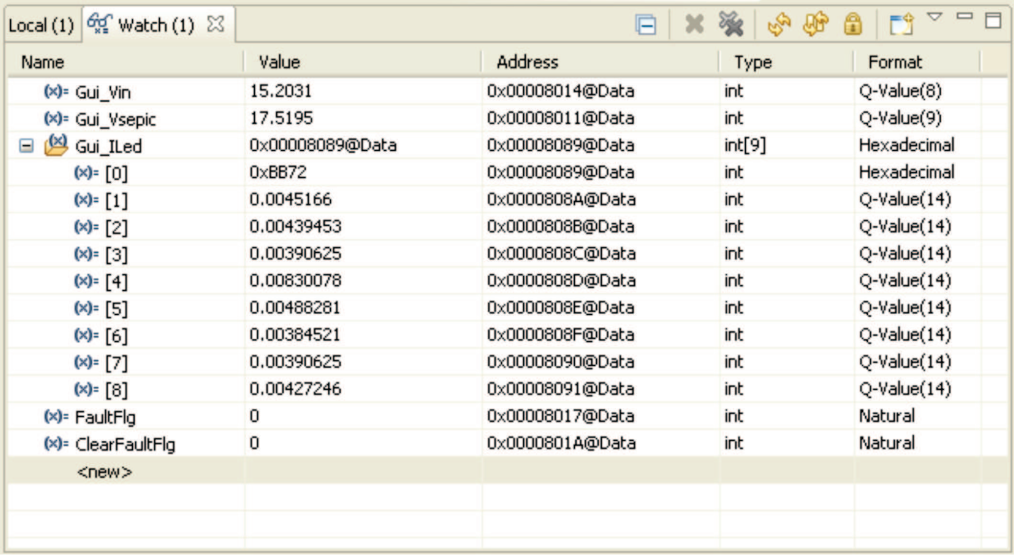
6.2 Build and Load the Project

1. Right-click on the Project Name and click on “Rebuild Project” and watch the Console window. Any errors in the project will be displayed in the Console window.
2. After the build successfully finishes, click the Debug button, located in the top-left side of the screen. The IDE automatically connects to the target, loads the output file into the device, and changes to the debug perspective.
3. Click the real-time mode button labeled “Enable silicon real-time mode”. This allows the user to edit and view variables in real time without halting the program.
4. If a message box appears, select YES to enable debug events. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a 0. The DGBM is the debug enable mask bit. When the DGBM bit is set to 0, memory and register values can be passed to the host processor for updating the debugger windows.

6.3 Set Up Watch Window and Graphs

1. Click View→Watch on the menu bar to open a watch window to view the variables being used in the project.
2. Add the variables listed in Figure 16 to the watch window.
3. Change the format of each variable to match Figure 16:
   - Right-click on a particular variable and select a Q-Value.
   - Use Shift-click to select multiple variables and elements, then right-click and change the format to affect all variables selected.

   The variables in this watch window are largely mathematically altered ADC samples and will be used to monitor the status of the board.

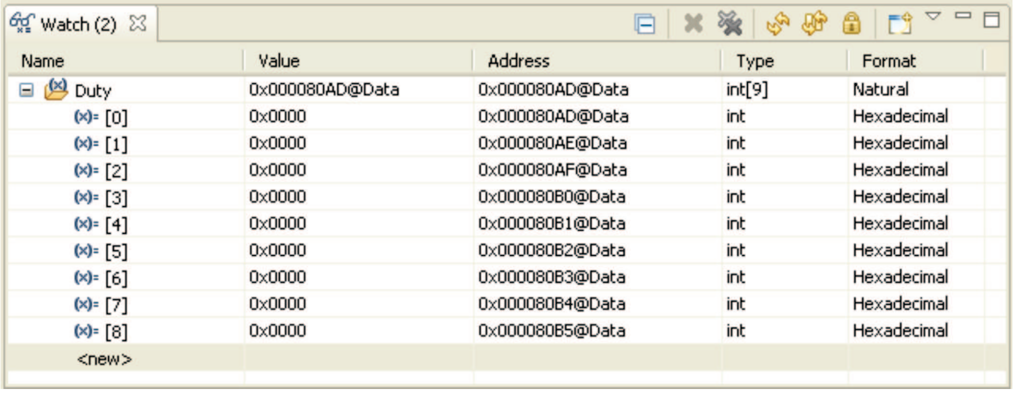


Name	Value	Address	Type	Format
Gui_Vin	15.2031	0x00008014@Data	int	Q-Value(8)
Gui_Vsepic	17.5195	0x00008011@Data	int	Q-Value(9)
Gui_ILed	0x00008089@Data	0x00008089@Data	int[9]	Hexadecimal
[0]	0xBB72	0x00008089@Data	int	Hexadecimal
[1]	0.0045166	0x0000808A@Data	int	Q-Value(14)
[2]	0.00439453	0x0000808B@Data	int	Q-Value(14)
[3]	0.00390625	0x0000808C@Data	int	Q-Value(14)
[4]	0.00830078	0x0000808D@Data	int	Q-Value(14)
[5]	0.00488281	0x0000808E@Data	int	Q-Value(14)
[6]	0.00384521	0x0000808F@Data	int	Q-Value(14)
[7]	0.00390625	0x00008090@Data	int	Q-Value(14)
[8]	0.00427246	0x00008091@Data	int	Q-Value(14)
FaultFlg	0	0x00008017@Data	int	Natural
ClearFaultFlg	0	0x0000801A@Data	int	Natural
<new>				

Figure 16. Configuring Watch Window 1

4. Add a second watch window by clicking on the button in the watch window.
5. In the new watch window, add the variable “Duty” and change its format to hexadecimal, as Figure 17 shows.

   This watch window holds variables that control the PWM duty cycle directly. 0x0000–0x7FFF correspond to 0–100% duty cycle.



Name	Value	Address	Type	Format
Duty	0x000080AD@Data	0x000080AD@Data	int[9]	Natural
[0]	0x0000	0x000080AD@Data	int	Hexadecimal
[1]	0x0000	0x000080AE@Data	int	Hexadecimal
[2]	0x0000	0x000080AF@Data	int	Hexadecimal
[3]	0x0000	0x000080B0@Data	int	Hexadecimal
[4]	0x0000	0x000080B1@Data	int	Hexadecimal
[5]	0x0000	0x000080B2@Data	int	Hexadecimal
[6]	0x0000	0x000080B3@Data	int	Hexadecimal
[7]	0x0000	0x000080B4@Data	int	Hexadecimal
[8]	0x0000	0x000080B5@Data	int	Hexadecimal
<new>				

Figure 17. Configuring Watch Window 2

6. Click on the Continuous Refresh button in each watch window to enable the windows to run with real-time mode.

   By clicking the down arrow in any watch window, you can select “Customize Continuous Refresh Interval” and edit the refresh rate for the watch windows. Note: Choosing an interval that is too fast may affect performance.

6.4 Run the Code

1. Run the code by pressing the Run Button in the Debug Tab.

   The project should now run, and the values in the watch window should continue to update. Resize or reorganize the windows according to your preference by dragging and docking the various windows.

2. In the second watch window, set Duty[0] to 0x1600 and then 0x2600. This value alters the duty cycle of the SEPIC and sets the output voltage to approximately 20.0 V. The SEPIC output voltage can be measured through a multimeter or through the variable Gui_Vsepic in the watch window.

3. Set Duty[1] to 0x0800. This value sets the duty cycle of the MOSFET controlling LED string 1 to be approximately 6.25% (0x0800/0x3FFF). Note: Once the LED string is on, the SEPIC, the output voltage decreases because the SEPIC is currently operating in an open loop and has a greater load than it did before.

   You can also check LED strings 2–8 individually by setting them to 0x0800 while the rest of the strings are at 0x0000. If the SEPIC is loaded too strongly, its output voltage will decrease to the extent that it no longer surpasses the threshold voltage of the LED string.

4. Once complete, set Duty[1]–Duty[8] to 0x0000 and Duty[0] to 0x0000.

5. Reset the processor (Target → Reset → Reset CPU).

6. Terminate the debug session by clicking (Target → Terminate All). This halts the program and disconnects Code Composer from the MCU.

7 Build 2: Closed-Loop SEPIC (Voltage) with Closed-Loop LED Strings (Current)

The objective of this build is as follows:

- Regulate SEPIC output voltage using voltage mode control (VMC) with closed-loop feedback.
- Use the duty cycle to dim the LED strings in open loop.
- Use sequencing functions to ensure an “orderly” voltage ramp-up and ramp-down.

Figure 18 describes the components of the system as used in the software.

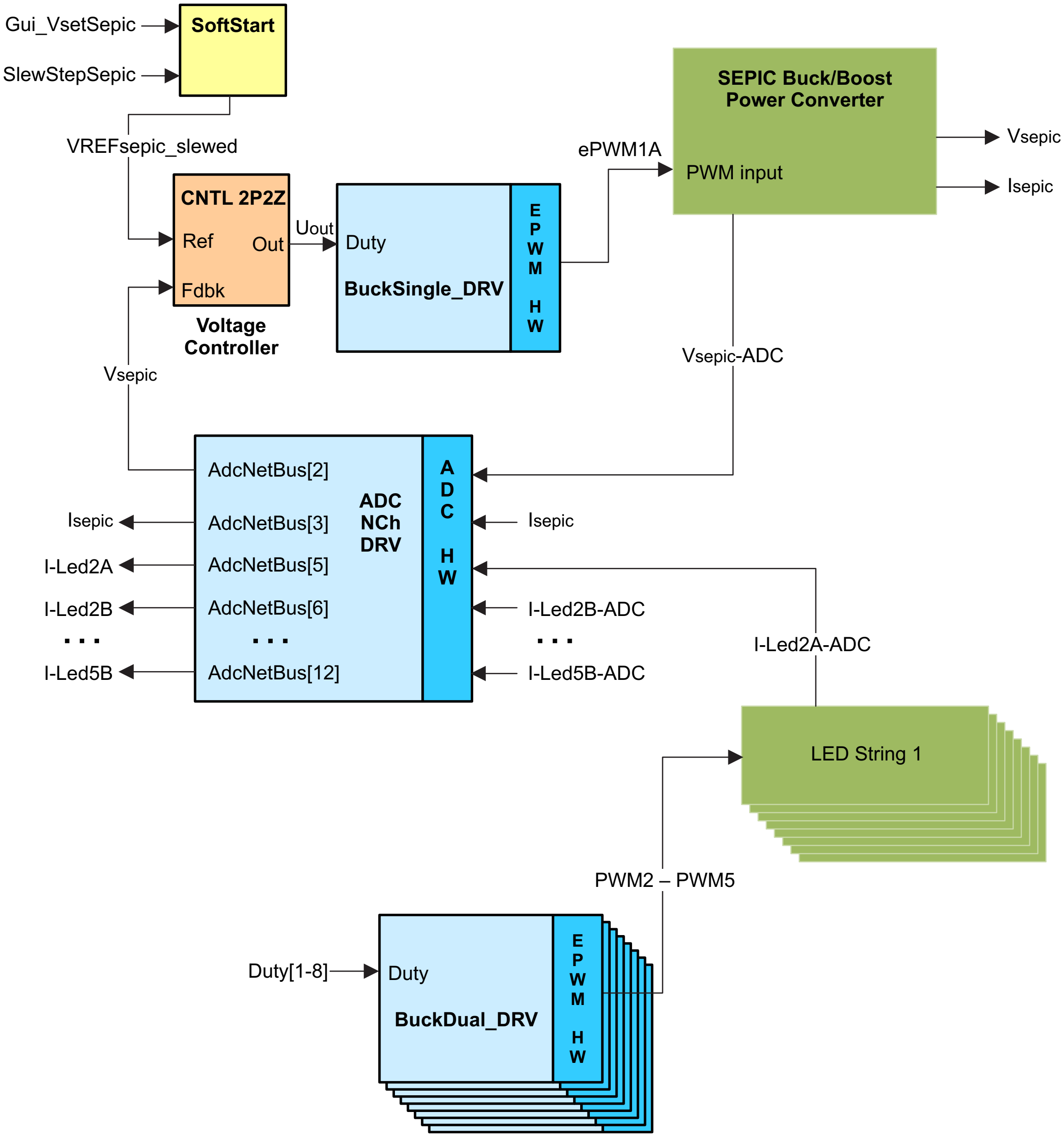
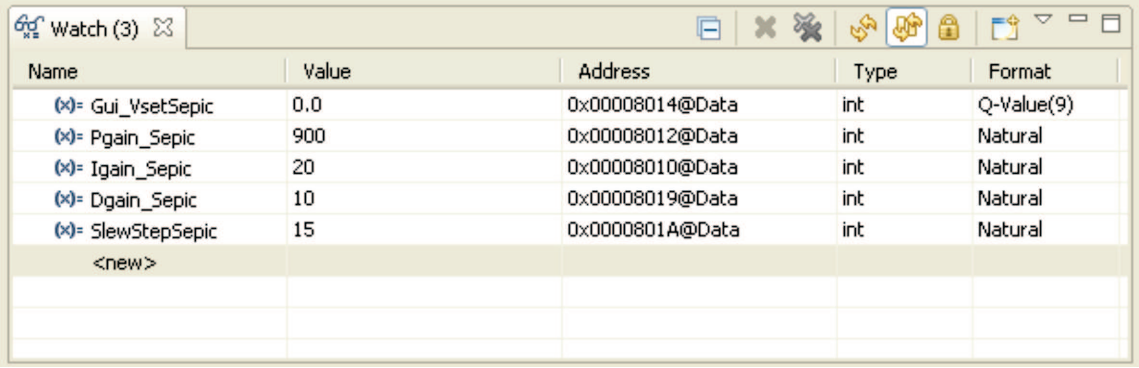


Figure 18. Build Level 2 Block Diagram

7.1 Run the Code

NOTE: The instructions for Build 2 are shorter than Build 1. Run Build 1 before running Build 2. For more details, see Section 6, *Build 1*.

1. Open the Lighting_DCDC-Settings.h file and change the incremental build level to 2 (#define INCR_BUILD 2). Save the file.
2. Right-click on the project name and select “Rebuild Project”.
3. After successfully completing the build, click the “Debug” button , located in the top-left side of the screen. The IDE automatically connects to the target, loads the output file into the device, and changes to the debug perspective.
4. Click the real-time mode button labeled “Enable silicon real-time mode”. This allows the user to edit and view variables in real time without halting the program.
5. Add a new watch window to the workspace and add the variables shown in Figure 19. This watch window will be used to control the SEPIC stage.
6. Set the format of Gui_VsetSepic to Q9. Also, set this new watch window to continuously refresh.



Name	Value	Address	Type	Format
(x)= Gui_VsetSepic	0.0	0x00008014@Data	int	Q-Value(9)
(x)= Pgain_Sepic	900	0x00008012@Data	int	Natural
(x)= Igain_Sepic	20	0x00008010@Data	int	Natural
(x)= Dgain_Sepic	10	0x00008019@Data	int	Natural
(x)= SlewStepSepic	15	0x0000801A@Data	int	Natural
<new>				

Figure 19. Configuring Watch Window 3

7. Run the code by pressing the Run Button in the Debug Tab.

   The project should now run, and the values in the watch window should continue to update. Resize or reorganize the windows according to your preference by dragging and docking the various windows.

   Watch windows 1, 2, and 3 are used in this build.. Watch window 1 monitors the system. Watch window 2 controls the LED strings. Watch window 3 controls the SEPIC stage.
8. In watch window 3, set Gui_VsetSepic to 20, which corresponds to 20 V. This variable is used as the reference to the controller. The controller edits the duty cycle as necessary to keep Gui_Vsepic, visible in watch window 1, near 20 V.
9. In watch window 2, set the variables Duty[1-8] each to 0x0600. Each LED should turn on to approximately 4.68% Duty Cycle. Unlike in build 1, the output voltage of the SEPIC is regulated and does not drop as more load is delivered.

   The variables Pgain_Sepic, Igain_Sepic, and Dgain_Sepic correspond to the PID coefficients of the voltage controller. Tuning these parameters, or the underlying 2P2Z coefficients, returns better performance. Tuning is assisted by a tool such as MATHLAB. Do not edit these variables in this lab.
10. Once complete, set Duty[1]–Duty[8] to 0x0000 and then Gui_VsetSepic to 0.0 V.
11. Reset the processor (Target → Reset → Reset CPU).
12. Terminate the debug session by clicking (Target → Terminate All). This halts the program and disconnects Code Composer from the MCU.

8 Build 3: Closed-Loop SEPIC (Voltage) With Closed-Loop LED Strings (Current)

The objective of this build is as follows:

- Regulate SEPIC output voltage using VMC with closed-loop feedback.
- Regulate the current draw of each LED using average current mode control with closed-loop feedback.
- Use sequencing functions to ensure an “orderly” voltage or current ramp-up and ramp-down.

Figure 20 describes the components of the system as used in the software.

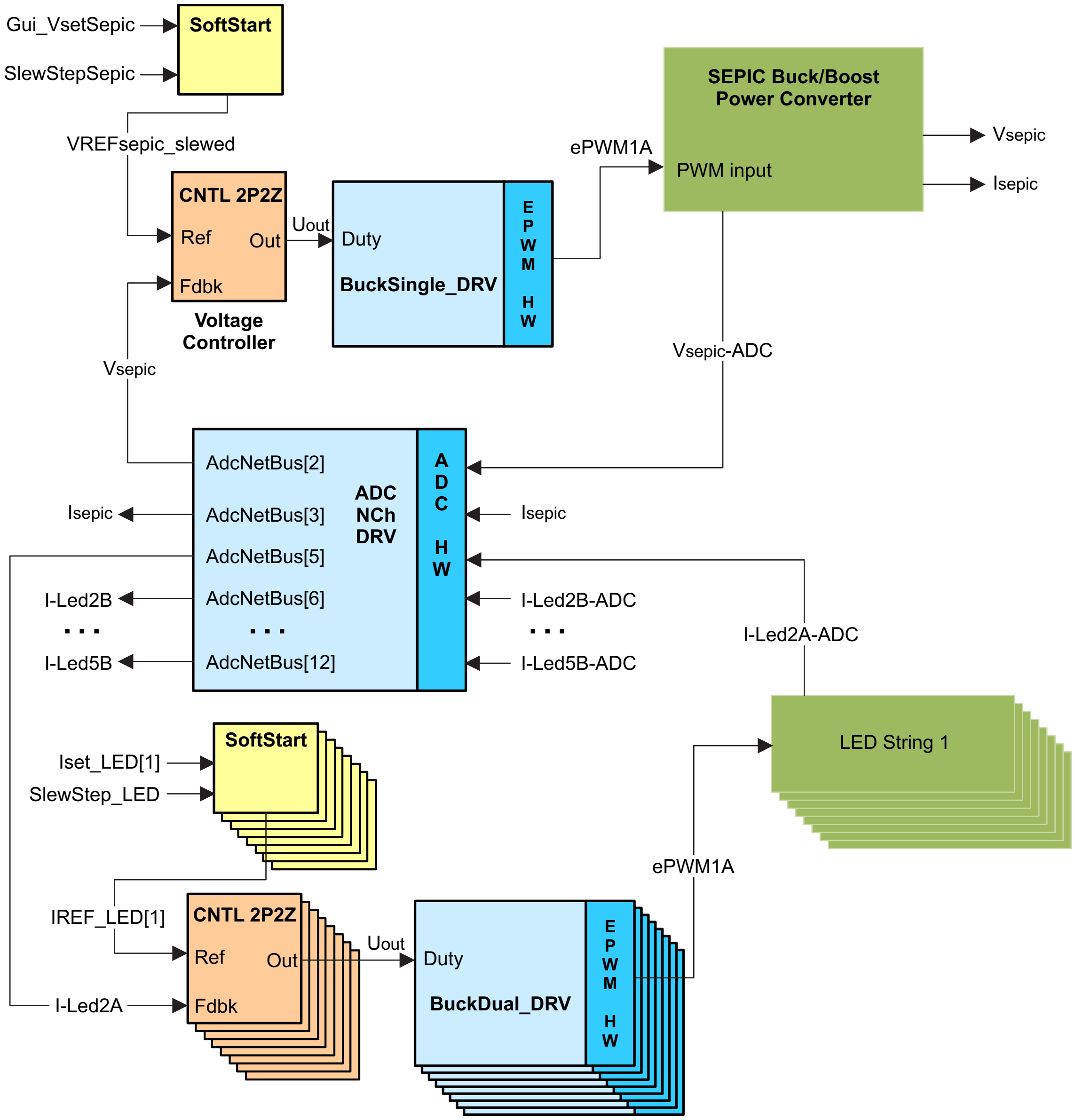
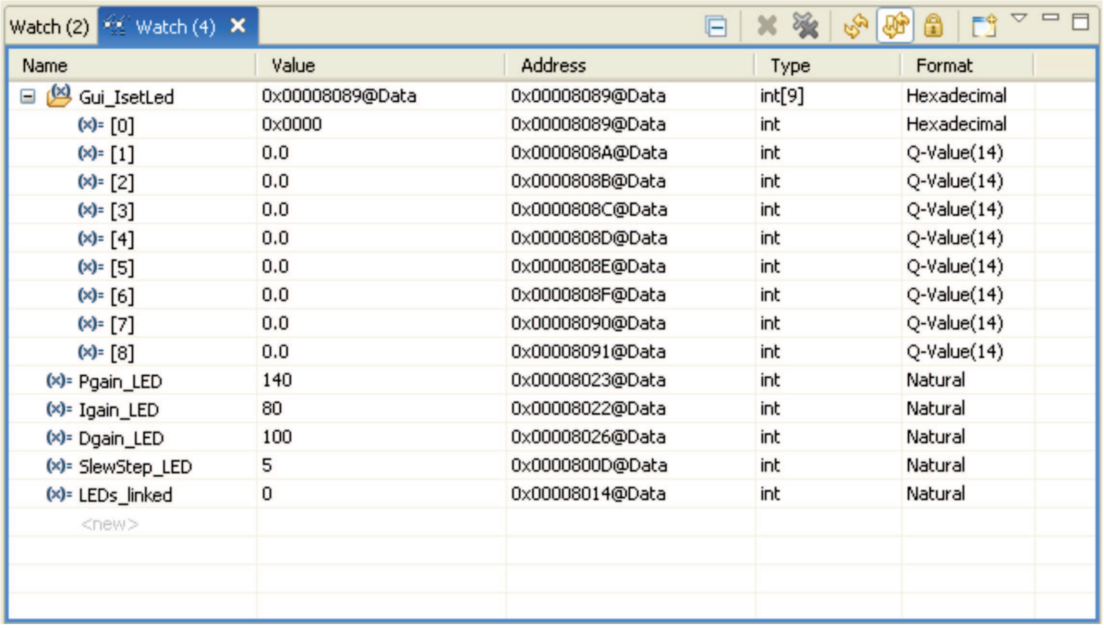


Figure 20. Build Level 3 Block Diagram

8.1 Run the Code

NOTE: The instructions for Build 3 are shorter than Build 1. Run Build 1 before running Build 3. For more details, see Section 6, *Build 1*.

1. Open the Lighting_DCDC-Settings.h file and change the incremental build level to 3 (#define INCR_BUILD 3). Save the file.
2. Right-click on the project name and select “Rebuild Project”.
3. After successfully completing the build, click the “Debug” button , located in the top-left side of the screen. The IDE automatically connects to the target, loads the output file into the device, and changes to the Debug perspective.
4. Click the real-time mode button labeled “Enable silicon real-time mode”. This allows the user to edit and view variables in real time without halting the program.
5. Add a new watch window to the workspace, add the variables, and set them to the correct format, as Figure 21 shows. This watch window will be used to control the LED dimming stages.
6. Set this new watch window to continuously refresh.



Name	Value	Address	Type	Format
Gui_IsetLed	0x00008089@Data	0x00008089@Data	int[9]	Hexadecimal
[0]	0x0000	0x00008089@Data	int	Hexadecimal
[1]	0.0	0x0000808A@Data	int	Q-Value(14)
[2]	0.0	0x0000808B@Data	int	Q-Value(14)
[3]	0.0	0x0000808C@Data	int	Q-Value(14)
[4]	0.0	0x0000808D@Data	int	Q-Value(14)
[5]	0.0	0x0000808E@Data	int	Q-Value(14)
[6]	0.0	0x0000808F@Data	int	Q-Value(14)
[7]	0.0	0x00008090@Data	int	Q-Value(14)
[8]	0.0	0x00008091@Data	int	Q-Value(14)
Pgain_LED	140	0x00008023@Data	int	Natural
Igain_LED	80	0x00008022@Data	int	Natural
Dgain_LED	100	0x00008026@Data	int	Natural
SlewStep_LED	5	0x0000800D@Data	int	Natural
LEDs_linked	0	0x00008014@Data	int	Natural
<new>				

Figure 21. Configuring Watch Window 4

7. Run the code by pressing the Run Button in the Debug Tab.

   The project should now run, and the values in the watch window should continue to update. Resize or reorganize the windows according to your preference by dragging and docking the various windows.

   The watch windows used in this build will be watch windows 1, 3 and 4. Watch window 1 monitors the system. Watch window 3 controls the SEPIC in closed loop. Watch window 4 controls the LED dimming stages in closed loop.
8. In watch window 3, set Gui_VsetSepic to 20 which corresponds to 20 V. This variable is used as the reference to the controller. The controller will edit the duty cycle as necessary to keep Gui_Vsepic, visible in watch window 1, at 20 V.
9. In watch window 4, set the variable Gui_IsetLed[1] to 0.01, which corresponds to 0.01 A. As the current reference is edited notice that Gui_ILed[1] (in watch window 1) tries to match the reference.

10. Change the value of Gui_IsetLed[1] to 0.03 A. Notice that the LED string brightness has increased with the increased current.
11. Continue editing Gui_IsetLed[1] and also Gui_IsetLed[2–8] with various values from 0.0 to 0.15 A.

    Note: Do not increase the total current used by all strings above 1.0 A. The reason for this restriction is because the power supply is not rated to output this much power. If a larger amount of current is required, an external power supply could be used. See the document Lighting_DCDC-HWGuide.pdf for further information.

    Note: There is a variation of 0.0 to approximately 0.02 A among the LEDs. This variation exists because the manufacturer does not ensure that the LEDs are identical until the LED current is greater than 0.05 A.

    Other variables that may be useful include:

    - SlewStep_LED: This variable changes the rate at which the LEDs change from one reference point to another. Smaller values mean more delay while larger values equate to a smaller delay.
    - LEDs_linked: This variable enables and disables individual control of each LED string and has the controller try and output the same current for each string. The combined reference is set by a Gui_IsetLed[1].
    - FaultFlg: The SEPIC stage is being over-current and over-voltage protected by the on-chip comparators of the Picollo MCU. Under large variation in references or under fault conditions, the MCU will force all the PWM outputs to low in an attempt to protect the system. This variable is a flag to show if this has occurred.
    - ClearFaultFlg: Clears any trip that may have occurred and, assuming no damage was done, allows the system to again work as specified.
12. Once complete, set Gui_IsetLed[1–8] to 0.0A and then Gui_VsetSepic to 0.0 V.
13. Reset the processor (Target → Reset → Reset CPU)
14. Terminate the debug session by clicking (Target → Terminate All). This halts the program and disconnects Code Composer from the MCU.

9 Ideas for Further Exploration

- I2C Temperature Sensing

  A temperature sensor could be placed on the LED panel in order to give the system information on the temperature of the LEDs on the panel. This temperature information could then be sent to an external host, used to provide thermal protection for the LEDs, or (along with a lookup table) used to provide thermal compensation.
- Using Duty Cycle to Compute Average LED Current

  On the DC-DC LED Lighting Kit a resistor-capacitor network has been used to average out the current flowing through the LED string before it returns to the MCU as feedback. Depending on the resistor and capacitor values chosen, the RC network either slows down the response of the feedback signal or is not fully DC.

  Instead of doing this, the RC-filter could be removed, and then the string current feedback would follow the PWM waveform directly. The configurability of the ADC triggering on Piccolo could then be used to sample each LED string’s current feedback signal while it is on. The MCU could then take this current signal and multiply it by the duty cycle for that string, which would give the average current. This would give a more accurate and cheaper method of controlling the LED strings.
- LED Current Offset Compensation

  Resistors, op-amps, and other discrete components have a specified error inherent in their design. These errors often manifest themselves as ADC voltage offset errors in a system. To reduce this variability and error, the ADC voltage could be measure a voltage at a known state and derive the error. The digital controller could then subtract this error within the interrupt.
- Use the DC-DC LED Lighting Kit to Jump-Start Your Prototype

  All software and hardware that is bundled with this kit is free to use with no licenses. Feel free to use any of the hardware and software as resources for your own design.

10 References

For more information, see the following guides:

- QSG-Lighting_DCDC

  Provides detailed information on the CCS4 Lighting_DCDC project within an easy-to-use lab-style format.

  C:\TI\controlSUITE\development_kits\Lighting_DCDC_vX.X\Lighting_DCDC\~GUI\QSG-Lighting_DCDC.pdf

- Lighting_DCDC-HWdevPkg

  A folder containing various files related to the hardware on the DC-DC LED Lighting board (schematics, bill of materials, Gerber files, PCB layout, and more).

  C:\TI\controlSUITE\development_kits\Lighting_DCDC_vX.X\~Lighting_DCDC-HwdevPkg\

- Lighting_DCDC-HWGuide

  Presents full documentation on the hardware found on the Lighting_DCDC board.

  C:\TI\controlSUITE\development_kits\Lighting_DCDC_vX.X\~Docs\Lighting_DCDC-HWGuide.pdf

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265