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

Brett Larimore
C2000 Systems and Applications Team
Version 1.1 – October 2010

Abstract

This application note 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 to this, 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 note a DC/DC Sepic stage has been integrated in addition to the dimmable control of 8 individual LED strings.

This application note 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.

Table of Contents

LED Lighting Theory

The Benefit 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, on the order of 50,000 hours if designed correctly. Since, in applications such as street lighting bulb replacement can be quite expensive when labor and loss of service are considered, LEDs can be seen as having an advantage in these spaces. LEDs also have excellent vibration resilience, provide directional lighting, and allow for almost full dimmability. These features, and more, allow LEDs to be perceived as the future in lighting technology.

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 non-quantitative way. A better term is luminous flux which, measured in lumens, is the light power measured multiplied with the V-λ scaling function. This function is used to compensate for the human eye's sensitivity to different wavelengths.

Chromaticity, or color, is then an objective specification of the color regardless of the light's luminous flux. The chart below, the 1931 CIE Chromaticity Diagram, is a two-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 .

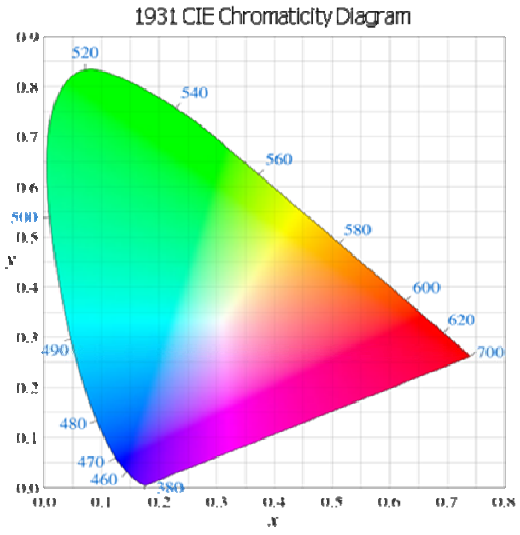


Figure 1: 1931 CIE Chromaticity Diagram

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 the image below, taken from an LED datasheet, it can be seen that current is nearly proportional to the luminous flux output from an LED. Because of this fact, in many systems the average current is edited in order to dim an LED or LED string. The only other

major variable that can affect luminous flux and chromaticity is temperature. Because of this, it is important to control temperature changes in an LED system.

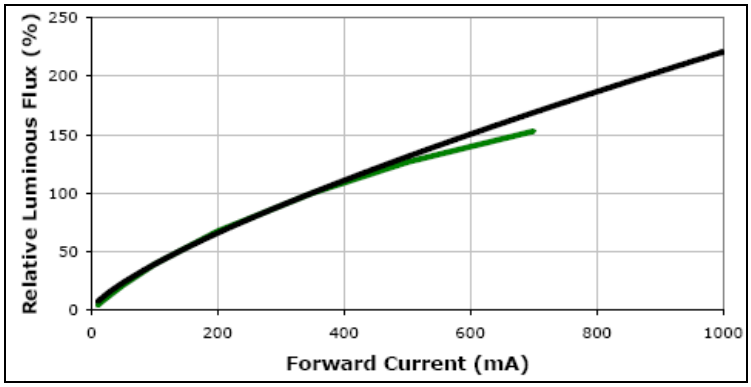


Figure 2: LED Current vs. Luminous Flux

The following image shows the relationship between forward voltage and current in an LED. Note that the LED behaves similarly to a diode in that it requires a certain threshold voltage before it will begin conducting. Once the forward voltage becomes greater than the threshold voltage the current will increase exponentially until it reaches the maximum current specification of the LED device. For a string of 6 LEDs it would be necessary to make the forward voltage larger than eight times the LED's threshold voltage in order to make the LEDs light up.

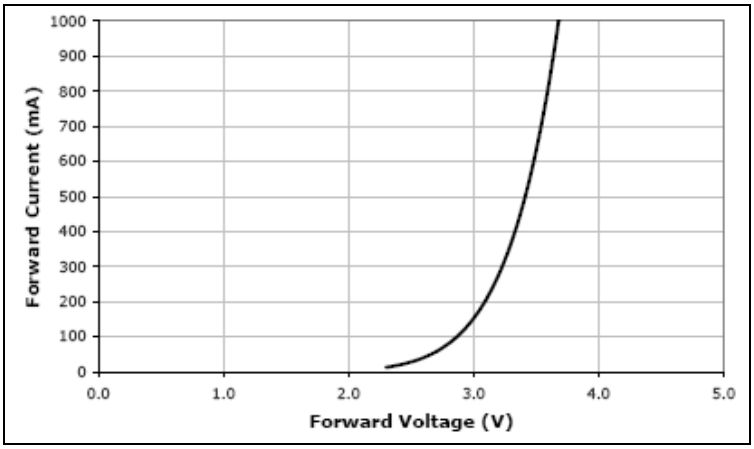


Figure 3: LED Forward Voltage vs. Current

Because of the LED's relationship between voltage and current it may seem reasonable that one could control the LED voltage in order to specify the LED current and therefore the luminous flux given by the LED string. The problem here is that as temperature even slightly increases the graph above will keep the same shape, but shift to the left. For instance, if one LED (with the specifications of the figure above) was controlled at 3.0V about 150mA of current would be drawn through the LED initially. However, as the LED warmed up, the graph will shift to the right and the current would increase slightly. This increased current would then increase the temperature. This cycle would continue until the LED device failed. 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.

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 cases the number of LEDs in a string may be large or the total luminous flux needed is expected to be large. In order to maintain efficiency a switched mode power supply will likely be needed. Below is one method of controlling one of these systems.

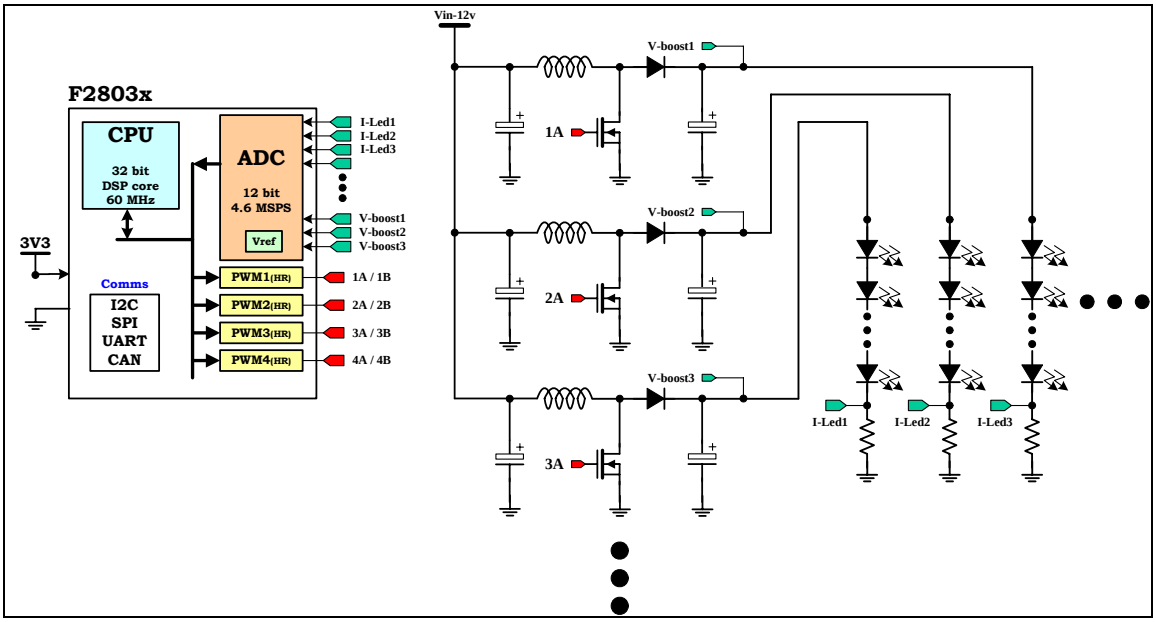


Figure 4: Individual Power Stage per String

In the solution above, 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 above 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 via peak or average current mode respectively.

On the following page, a different technique is shown. A single DC/DC stage is used to provide the voltage that will appear across the LED strings. This voltage will then correspond to a single current that will pass 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.

On the DC/DC LED Lighting Kit, this latter approach has been implemented. 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 be done to reduce the peak current and improve the overall efficiency. Piccolo's PWMs allow for synchronization between individual PWM modules and/or with an external synchronization pulse.

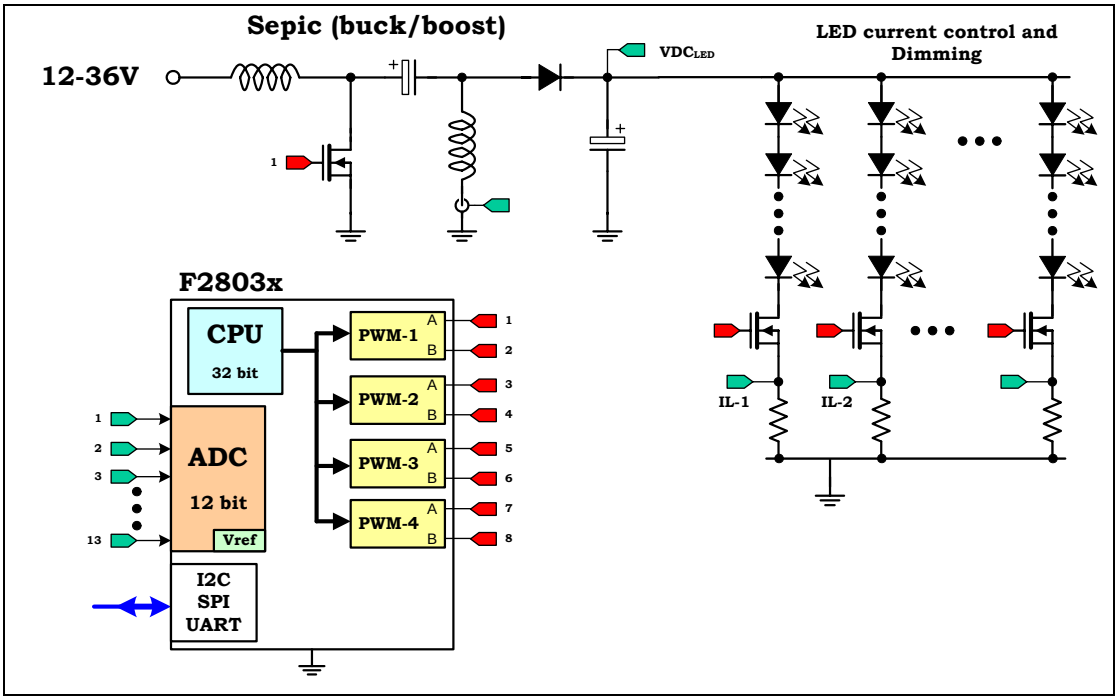


Figure 5: PWM Dimmed Strings with a Common Supply

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 the one below will have an AC/DC rectifier, followed by a PFC boost circuit and then one or more parallel DC/DC stages will be used to drive one or more LED strings. To create an efficient system each of these power stages must be designed to be efficient and the control of these power stages must be efficient.

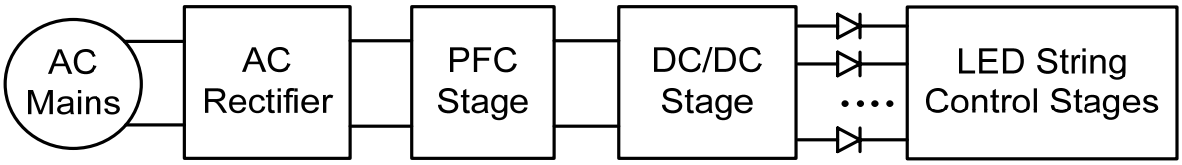


Figure 6: Lighting System

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 already exist such as DALI, KNX, and DMX512. These, 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's 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 only generate 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 pre-determined 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/or 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 will used less than 50% of the CPU bandwidth on the C2000.

Benefits of C2000 in LED Lighting

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 usage, aging, sensed external brightness, etc
- Easily synchronized to video clock via 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 I2C, 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 via software, creating products for multiple regions and multiple configurations often requires only minor changes in software.

System Overview

Hardware Overview

The DC/DC LED Lighting Developer's Kit takes in 12-36V DC. This voltage is then regulated to a different level by a SEPIC converter. SEPIC, as a step-up/step-down converter topology, is able to increase or decrease the input voltage. For our application, we will be setting the Sepic output voltage to approximately 20V independent of the board input. The SEPIC output is then connected to each of the LED strings. 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. Since the brightness of an LED is roughly proportional to the LED current we use the duty cycle of each string's PWM to control the average current drawn. The figure below illustrates the hardware present on the DC/DC LED Lighting Developer's Kit.

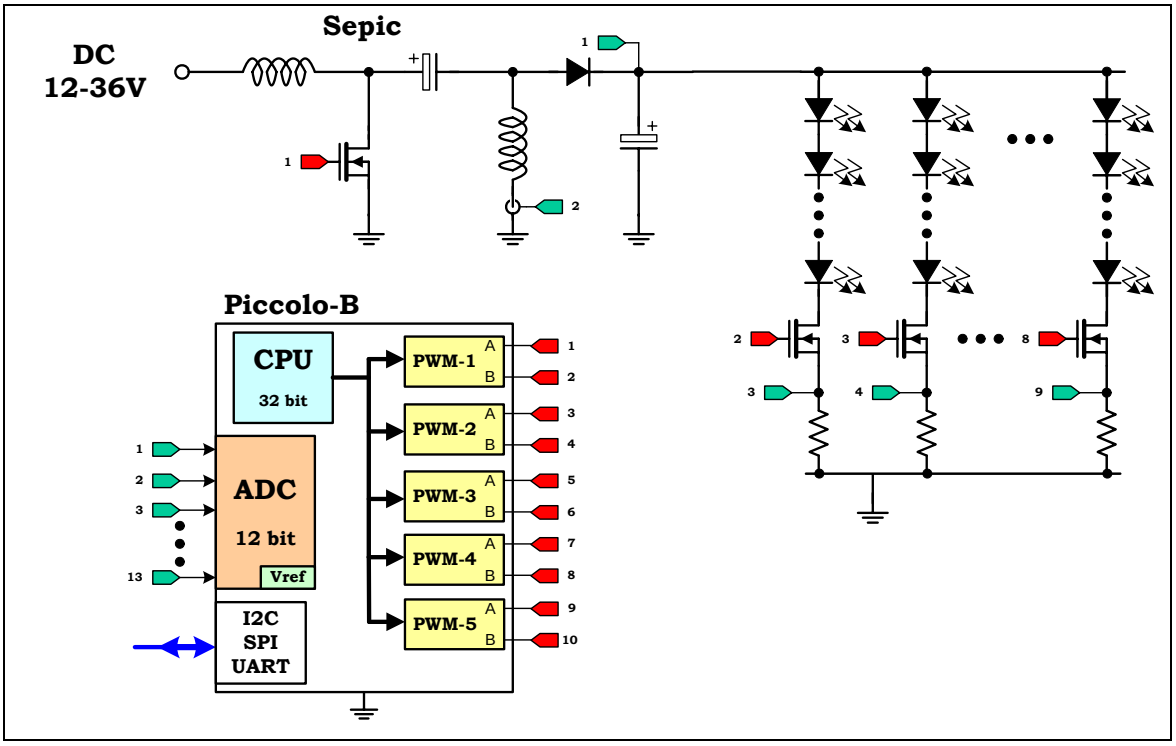


Figure 7: DC/DC LED Lighting Kit Hardware Block Diagram

The key signal connections between the F28035 microcontroller and the DC/DC LED Lighting board are listed in the table below:

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
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

Table 1: Key Peripherals Used

The Piccolo microcontroller, DC/DC LED Lighting board and CCS project can control 2 additional LED strings, however the LED panel ships with 6 strings and therefore this table contains only the hardware resources used for these 6.

Software Overview

Build Options

In order for the user to slowly build up and understand the project, the project is divided into various builds seperated 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. Below 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)

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 ~(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 via 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.

Table 2: Major Variables

Key Files Located in the Project

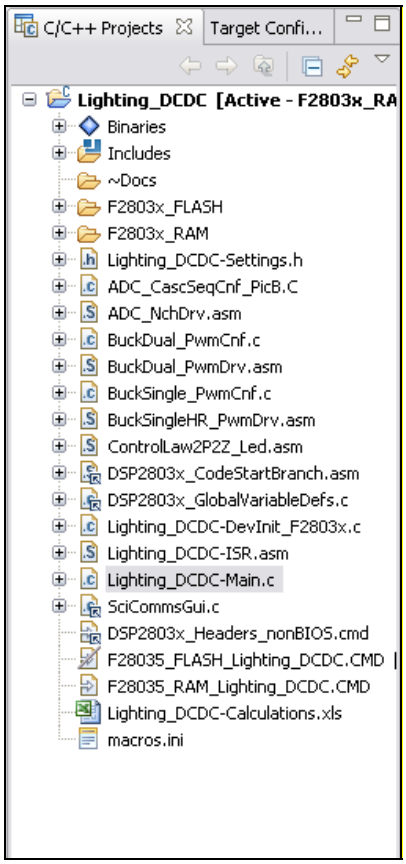


Figure 8: Lighting_DCDC Project Files

The key framework files used in this project are:

Lighting_DCDC-Main.c – this file is used to initialize, run, and manage the application. This is the “brains” behind 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, etc.

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 for the project (ie. build options). Note that it 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.

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. Following is the list of macros being used in this project.

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

Table 3: Macros Used

As the configuration of the peripherals is done in C, it can be easily modified. The ASM drive macro provide the necessary compact code to run in real time. Let's look at each of the macro to better understand it's role in the project.

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 please see the source file. Note 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 please see the source file. Note 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 2nd order compensator realized from an IIR filter structure. The 5 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 usage of ADC module, all that needs to 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.

The Lighting_DCDC project has the following properties:

Memory Usage of the Lighting_DCDC Project		
Build Level	**Program Memory Usage 2803x**	**Data Memory Usage[1] 2803x**
Build 3 (RAM)	4489 words	719 words

[1] Excluding the stack size

CPU Utilization of Build 3 of the Lighting_DCDC Project	
Name of Modules *	**Number of Cycles**
Context Save, etc	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 Utilization @ 60 Mhz	**31.0% ****
CPU Utilization @ 40 Mhz	**46.5% ****

* The modules are defined in the header files as “macros”
** At 100 kHz ISR freq.

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

Hardware Setup

In this guide each component is named first with their macro number follow by the reference name. For example, [M2]-J1 would refer to the jumper J1 located in the macro M2 and [Main]-J1 would refer to the J1 located on the board outside of the other defined macro blocks.Listed below are some of the major connectors and features of the DC/DC LED Lighting board.

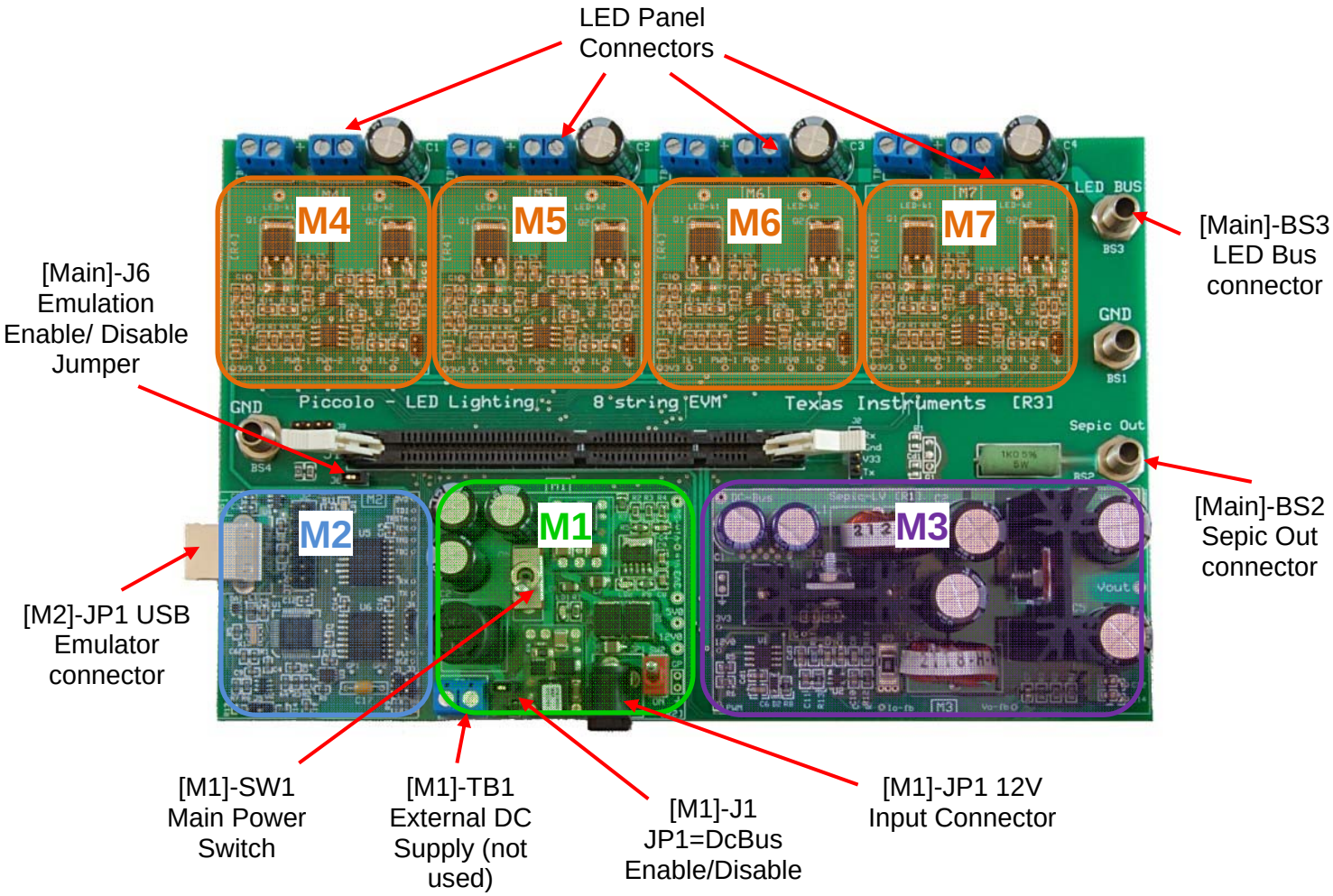
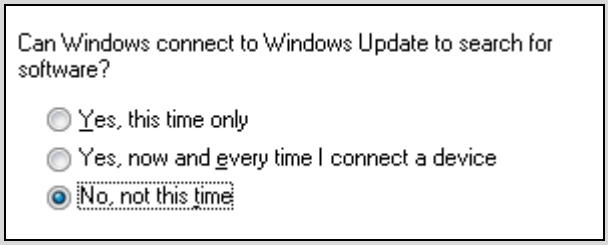


Figure 9: Hardware Features

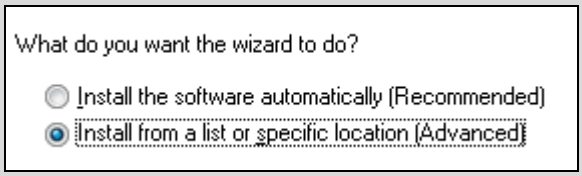
1) On the Piccolo F28035 controlCARD, check the following switches:
   - SW1, make sure this switch is in the “off” (down) position
   - SW2, make sure position 1 and 2 are both in the “on” (up) position.
2) Put a F28035 control card into the socket on the LED Lighting board and connect a cable from the USB connector on the board to the computer. [M2]-LD1, near the LED Lighting board’s USB connector, should turn on.

NOTE: If Code Composer Studio has never been installed, it may be necessary to install drivers to make the board work correctly. If a popup comes up when the USB cable is connected from the board to the computer, have the install wizard install drivers from the XDS100v1 directory of the USB drive included with this kit.

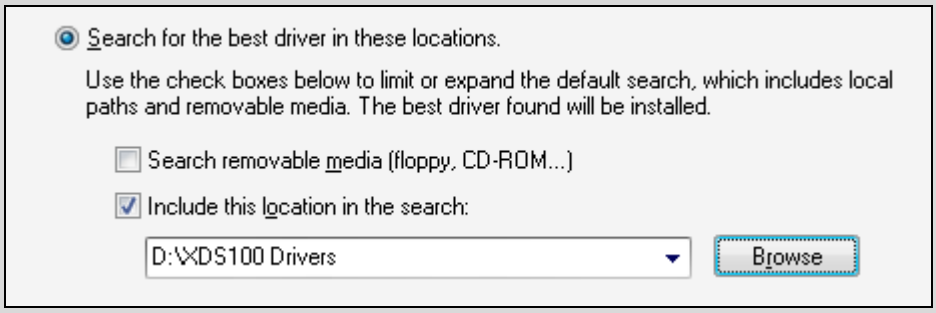
1) When Windows asks to search Windows Update, select “No, not at this time” and click Next



2) On the next screen select “Install from specific location” and click Next



3) Select “Search for Best Driver”, uncheck search removable media, and check include specific location and browse to [USB Drive]:\XDS100 Drivers



4) Click next and the drivers will be installed. The driver install screen will appear three times, repeat this procedure each time.

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

Software Setup

Installing Code Composer and controlSUITE

1) If not already installed, please install Code Composer v4.x from the DVD included with the kit.

2) Go to http://www.ti.com/controlsuite and run the controlSUITE installer. Select to install the "DC/DC LED Lighting" software and allow the installer to also download all automatically checked software.

Setup Code Composer Studio to Work with the DC/DC LED Lighting kit

3) Open "Code Composer Studio v4".

4) Once Code Composer Studio opens, the workspace launcher may appear that would ask to select a workspace location,: (please note workspace is a location on the hard drive where all the user settings for the IDE i.e. which projects are open, what configuration is selected etc. are saved, this 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 this dialog may not appear)
   - Click the "Browse…" button
   - Create the path below by making new folders as necessary.
   - "C:\Documents and Settings\My Documents\ CCSv4_workspaces\Lighting_DCDC"
   - Uncheck the box that says "Use this as the default and do not ask again".
   - Click "OK"

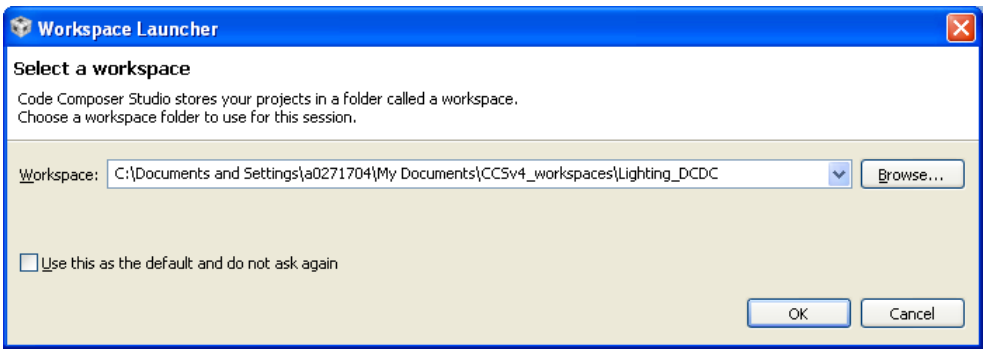


Figure 10: Workspace Launcher

5) Next we will configure Code Composer to know which MCU it will be connecting to. Click “Target -> New Target Configuration…”. Name the new configuration xds100-f28035.ccxml. Make sure that the “Use shared location” checkbox is checked and then click Finish.

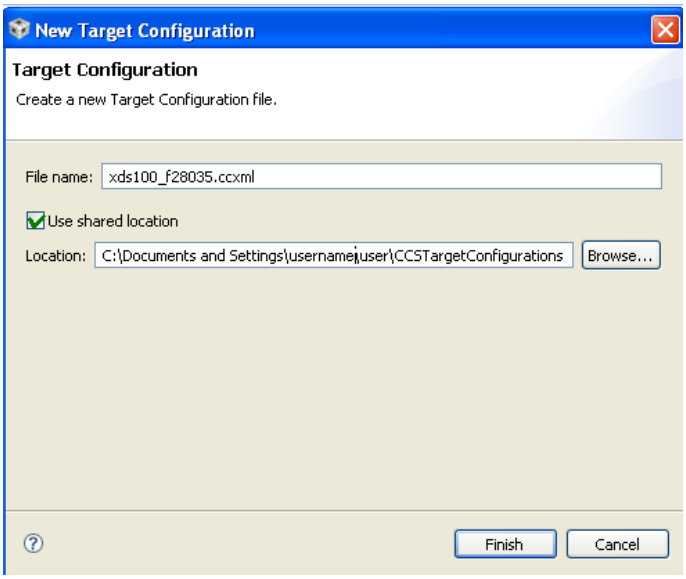


Figure 11: Creating a Target Configuration

6) This should open up a new tab as seen in Figure 2. Select and enter the options as shown:

- Connection – Texas Instruments XDS100v1 USB Emulator
- Device – TMS320F28035
- Click Save
- Close the xds100-f28035.ccxml tab

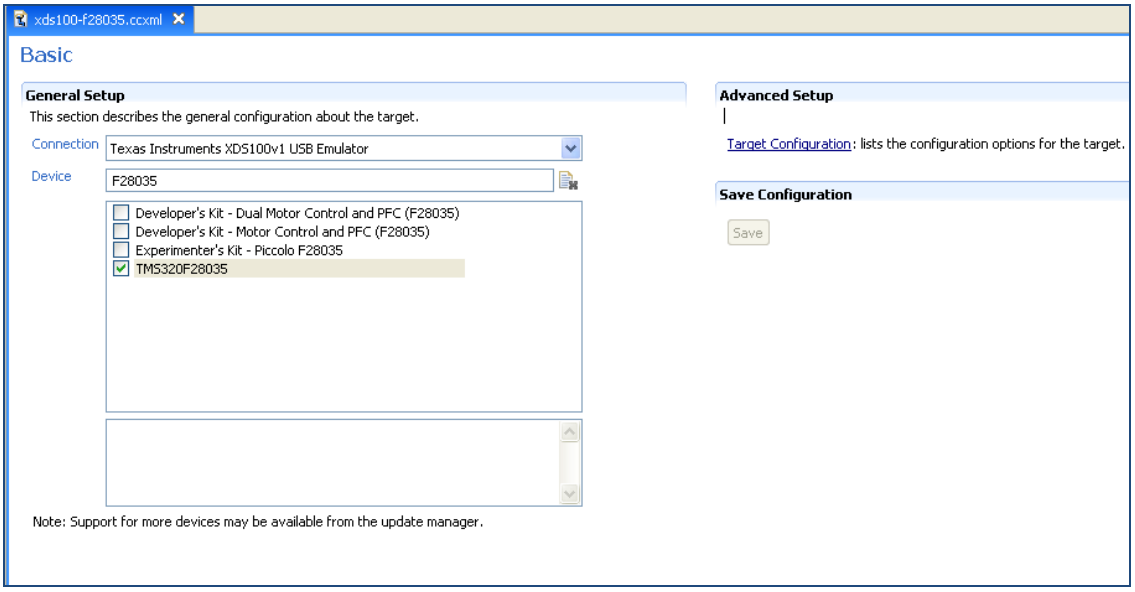


Figure 12: Configuring a New Target

7) Assuming this is your first time using Code Composer, the xds100-F28035 configuration is now set as the default target configuration for Code Composer. Please check this by going to "View->Target Configurations". In the "User Defined" section, right-click on the xds100-F28035.ccxml file and select "Set as Default". This tab also allows you to reuse existing target configurations and link them to specific projects.

8) Add the DC/DC LED Lighting project into your current workspace by clicking "Project->Import Existing CCS/CCE Eclipse Project".

   - Select the root directory of the DC/DC LED Lighting Kit. This will be: "C:\TI\controlSUITE\development_kits\Lighting_DCDC_vX.X\"

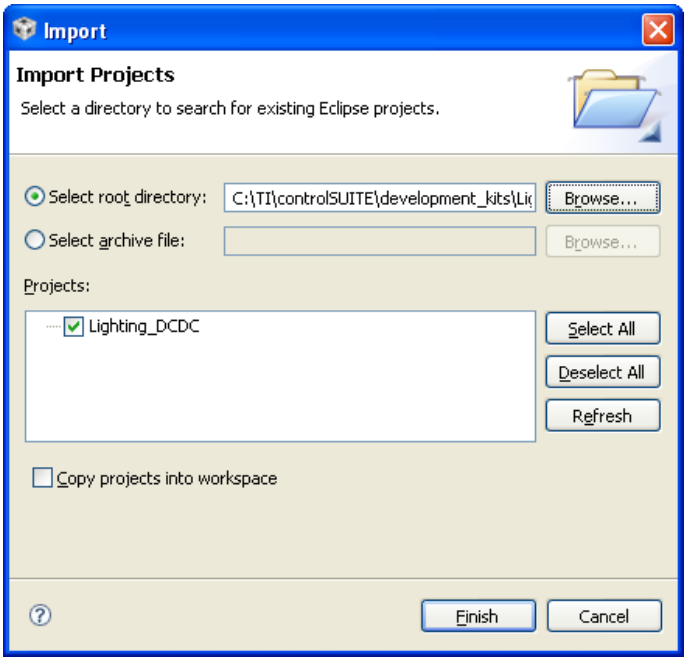


Figure 13: Adding the DC/DC LED Lighting Kit Project to your Workspace

   - Click Finish. This will copy the Lighting_DCDC project into the workspace.

Configuring a Project

9) The Lighting_DCDC project should be set as the active project. Right-click on the project name and click "Set as Active Project". Expand the file structure of the project.

10) 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. As shown in Figure 4, right-click on an individual project and select Active Build Configuration-> `F2803x_RAM` configuration.

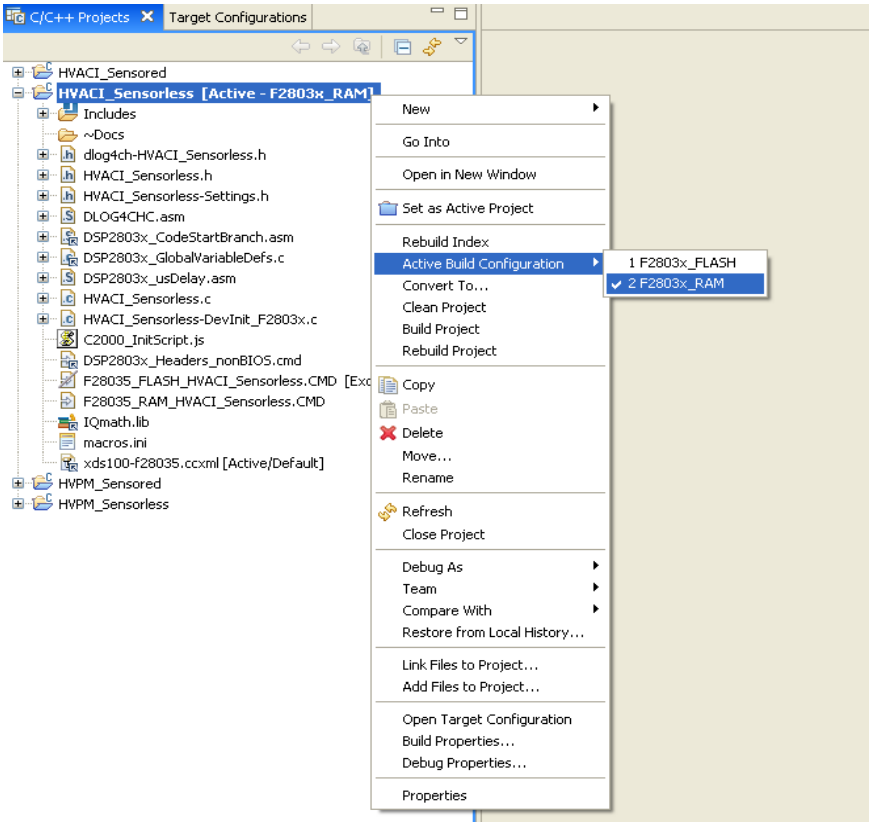


Figure 14: Selecting the F2803x_RAM configuration

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 is listed below.

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

Software Module	Phase 1	Phase 2	Phase 3
BuckSingleHR_DRV	√√	√	√
BuckDual_DRV	√√	√	√
ADC_NchDRV	√√	√	√
ControlLaw_2P2Z		√√	√√
Note: the symbol √ means this module is using and the symbol √√ means this module is testing in this phase.			

Table 4: Testing Modules in Each Incremental System Build

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

NOTE: This section assumes that the **Hardware Setup** and **Software Setup** sections have been completed. Please go through 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.

The components of the system as used in the software are described in the diagram below:

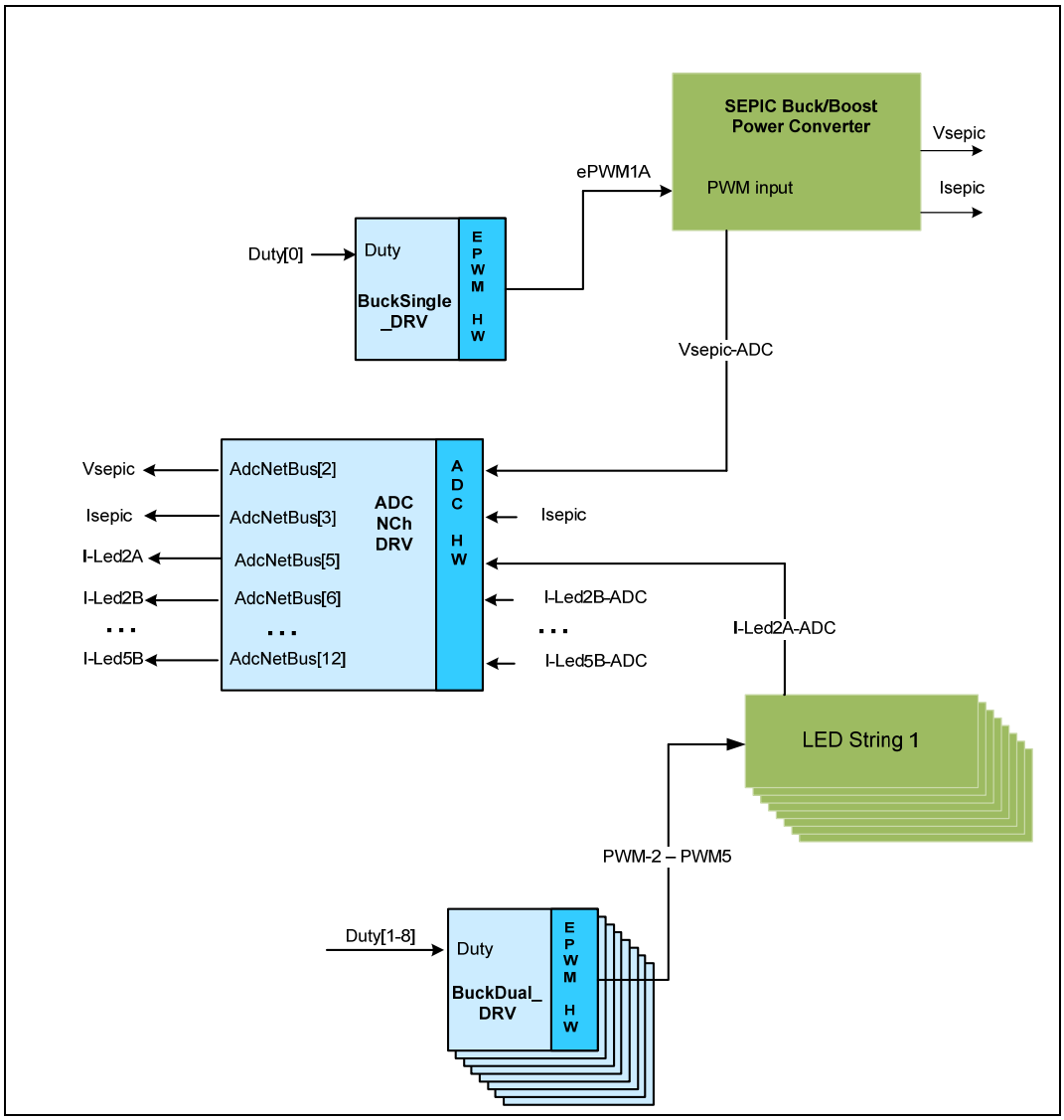


Figure 15: Build Level 1 Block Diagram

Inspect the Project

1. After initial boot processes are complete the software will begin from the main function. Open up Lighting_DCDC-Main.c and find the main() function in line 252.

2. The first thing that the software does in the main() function is call a function called DeviceInit() found in Lighting_DCDC-DevInit_F2803x.c. 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/disabled and the functional pinout, which configures which peripherals come out of which pins, is defined.

   - Confirm that the ADC and PWM1-8 peripheral clocks are enabled (lines 107-123). Also confirm that GPIO00 and GPIO02-GPIO09 are configured to be PWM outputs (lines 158-216).

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

   - From the C/C++ Project tab open the file Lighting_DCDC-Settings.h and make sure 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.

4. Go back to the Lighting_DCDC-Main.c file and notice the various incremental build 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. Note that 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 Lighting_DCDC-ISR.asm as well.

Build and Load the Project

5. 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.

6. On successful completion of the build click the  “Debug” button, located in the top-left side of the screen. The IDE will now automatically connect to the target, load the output file into the device and change to the Debug perspective.

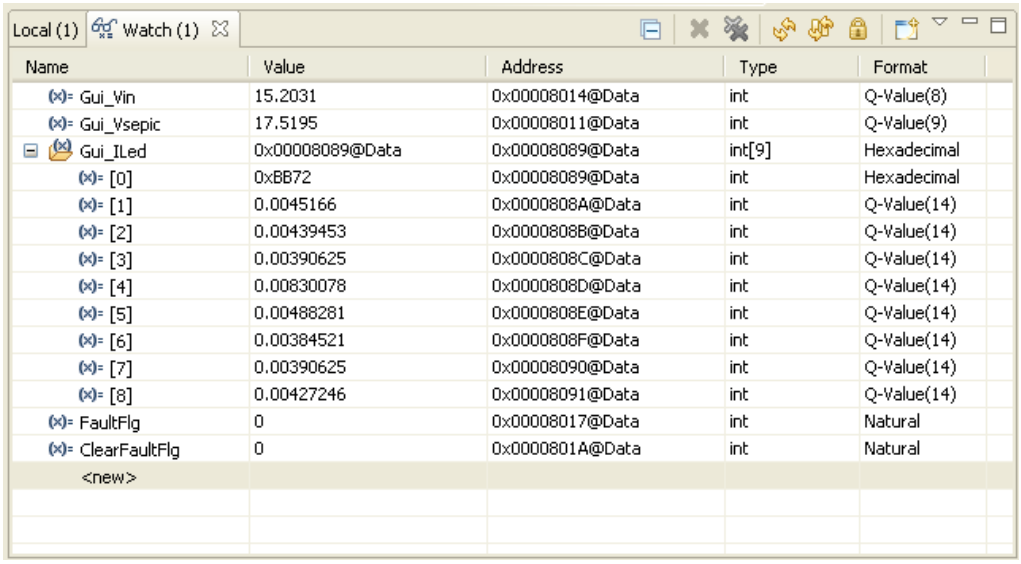
7. Now click the real-time mode  button that says “Enable silicon real-time mode”. This will allow the user to edit and view variables in real-time without halting the program.

8. A message box *may* appear. If so, 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.

Setup Watch Window & Graphs

9. Click: View → Watch on the menu bar to open a *watch window* to view the variables being used in the project. Add the variables found in Figure 17 to the watch window. The format of a variable can be changed by right-clicking on a particular variable then selecting a Q-Value. Change the format of each variable to match the figure. The variables in this watch window are largely mathematically altered ADC samples and will be used to monitor the status of the board.

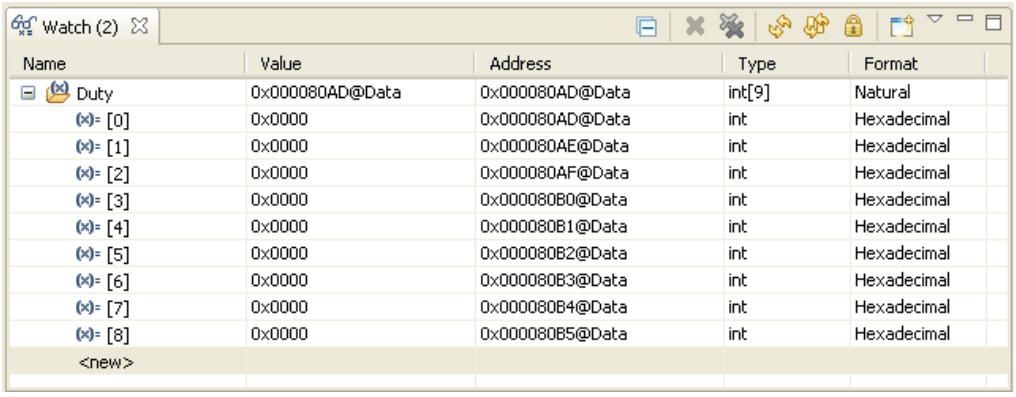
Hint: Shift-Click can be used select multiple variables or elements and then right-clicking and changing the format will affect all variables selected.



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

10. Add a second watch window by clicking on the button in the watch window. In the new watch window add the variable “Duty” and change its elements’ format to Hexadecimal as in Figure 18. This watch window holds variables that control the PWM duty cycle directly. 0x0000-0x7FFF corresponds 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

11. Click on the Continuous Refresh button in each watch window. This enables the window to run with real-time mode. By clicking the down arrow in any watch window, you may select “Customize Continuous Refresh Interval” and edit the refresh rate for the watch windows. Note that choosing too fast an interval may affect performance.

Run the Code

12. Run the code by pressing Run Button  in the Debug Tab.
13. The project should now run, and the values in the watch window should keep on updating. You may want to resize or reorganize the windows according to your preference. This can be done easily by dragging and docking the various windows.
14. In the second watch window set Duty[0] to 0x1600 and then 0x2600. This will alter the duty cycle of the SEPIC and will set the output voltage to around 20.0V. The SEPIC output voltage can be measured via a multimeter or via the variable Gui_Vsepic in the watch window.
15. Now set Duty[1] to 0x0800. This will set the duty cycle of the MOSFET controlling LED string 1 to be approximately 0x0800/0x3FFF = 6.25%. Note that once the LED string is on the SEPIC output voltage has decreased. This is because the SEPIC is currently operating in open loop and has more load than it did before.
16. LED string 2-8 can also be checked individual 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 LED string's threshold voltage.
17. Once complete, set Duty[1]-Duty[8] to 0x0000 and then Duty[0] to 0x0000.
18. Reset the processor  (Target->Reset->Reset CPU) and then terminate the debug session by clicking  (Target->Terminate All). This will halt the program and disconnect Code Composer from the MCU.

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/down

The components of the system as used in the software are described in the diagram below:

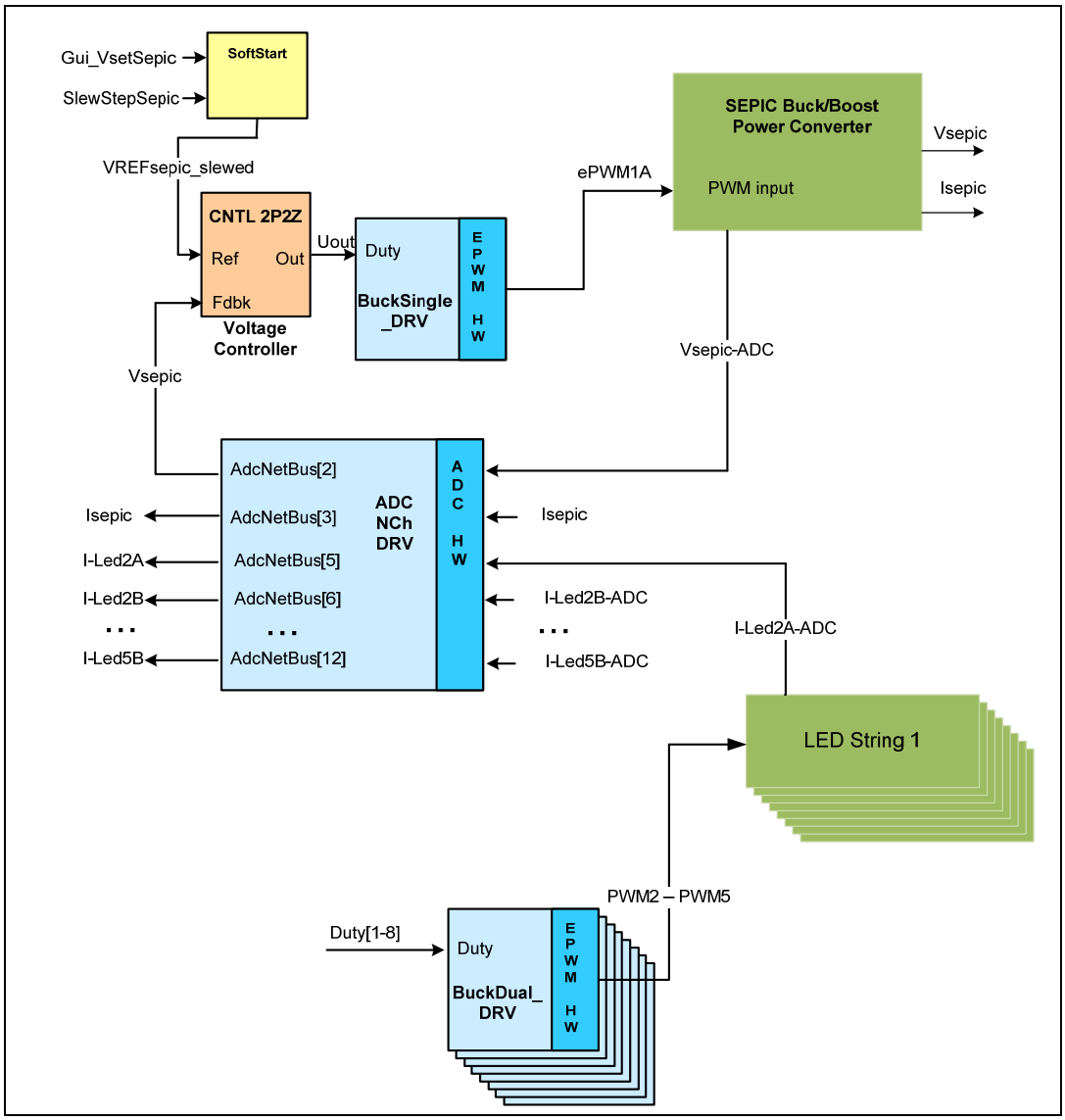


Figure 18: Build Level 2 Block Diagram

Run the Code

The instructions for this build will be more succinct than in the first build. Please refer to the instructions in Build 1 for further guidance. Please run build 1 prior to running build level 2.

1. Open Lighting_DCDC-Settings.h 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. On successful completion of the build click the “Debug” button, located in the top-left side of the screen. The IDE will now automatically connect to the target, load the output file into the device and change to the Debug perspective.
4. Now click the real-time mode button that says “Enable silicon real-time mode”. This will allow 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 20. This watch window will be used to control the SEPIC stage. Be sure to set the format of Gui_VsetSepic to Q9. Also, set this new watch window to continuously refresh.

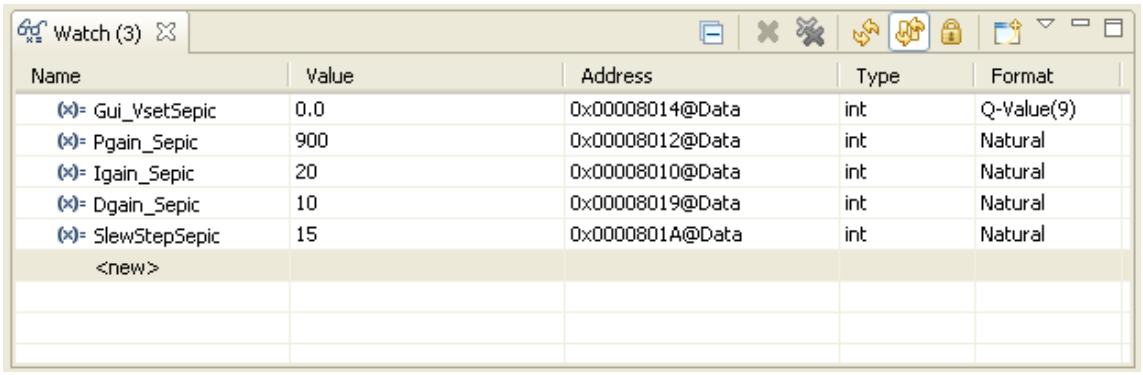


Figure 19: Configuring Watch Window 3

6. Run the code by pressing Run Button in the Debug Tab.
7. The project should now run, and the values in the watch window should keep on updating. You may want to resize or reorganize the windows according to your preference. This can be done easily by dragging and docking the various windows.
8. The watch windows used in this build will be Watch Windows 1, 2 and 3. Watch window 1 is used to monitor the system. Watch window 2 will be used to control the LED strings and Watch window 3 will be used to control the SEPIC stage.
9. In the Watch Window 3, set Gui_VsetSepic to 20 which corresponds to 20V. 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, near 20V.

10. In Watch Window 2, set the variables Duty[1-8] each to 0x0600. Each of the LEDs 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.

11. 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, will give better performance. This tuning could be assisted by a tool such as Mathlab. It is not recommended to edit these variables in this lab.

12. Once complete, set Duty[1]-Duty[8] to 0x0000 and then Gui_VsetSepic to 0.0V.

13. Reset the processor  (Target->Reset->Reset CPU) and then terminate the debug session by clicking  (Target->Terminate All). This will halt the program and disconnect Code Composer from the MCU.

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 Voltage Mode Control (VMC) with closed-loop feedback
- Regulate each LED string's current draw using average current mode control with closed-loop feedback.
- Use sequencing functions to ensure an "orderly" voltage or current ramp-up/down

The components of the system as used in the software are described in the diagram below:

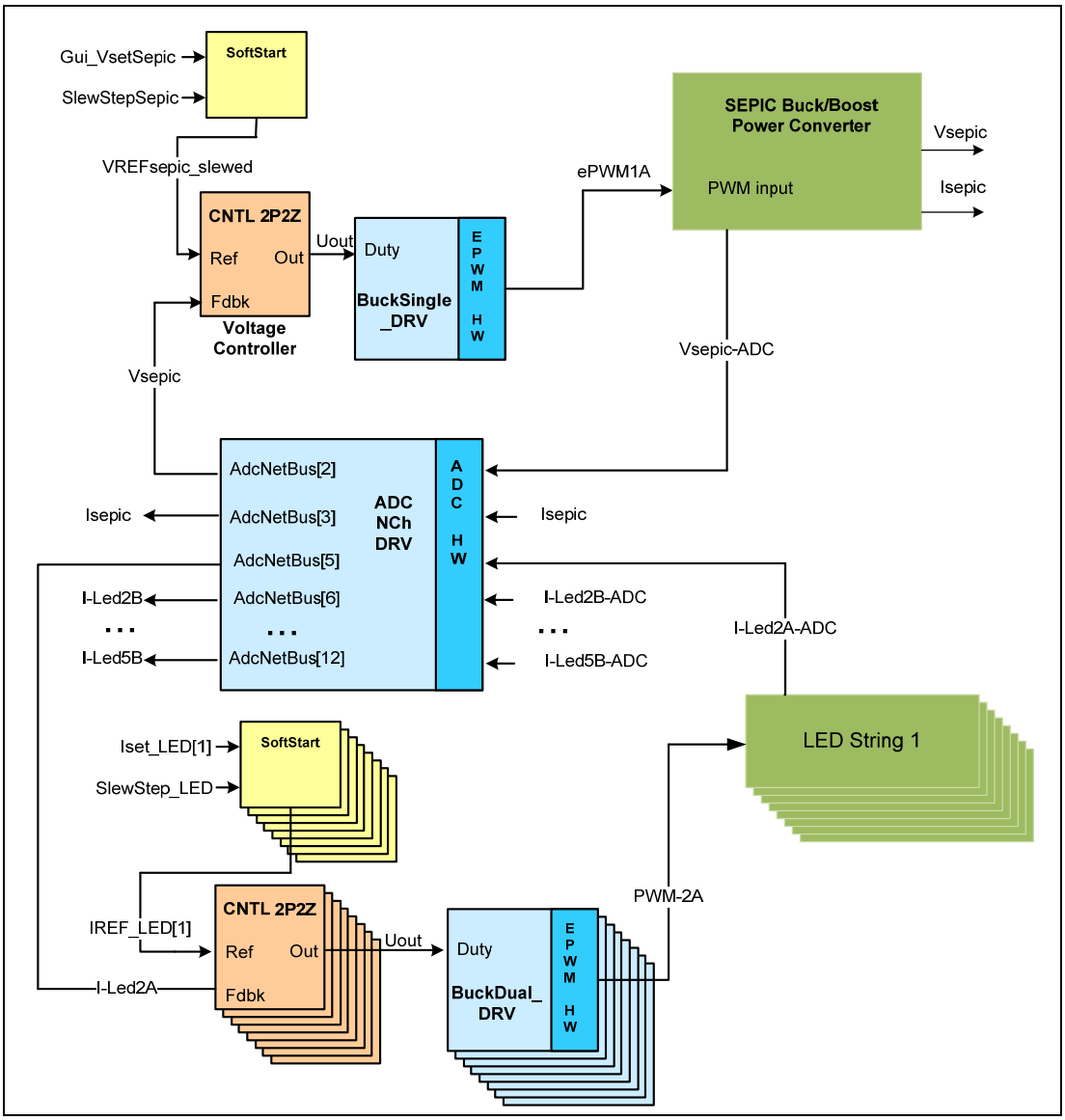
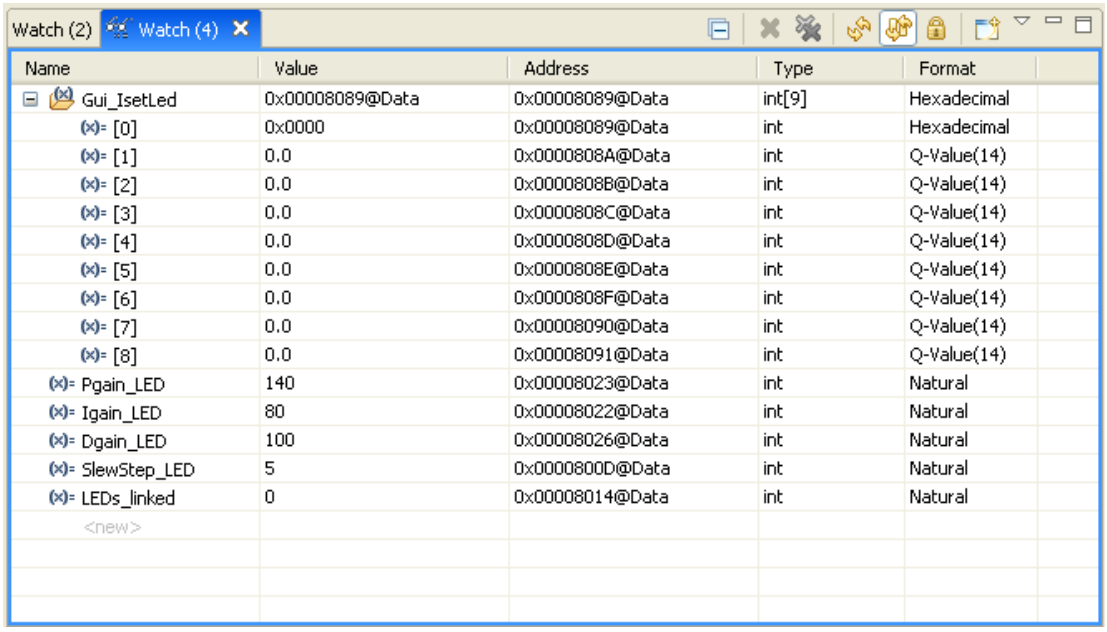


Figure 20: Build Level 3 Block Diagram

Run the Code

The instructions for this build will be more succinct than in the first build. Please refer to the instructions in Build 1 for further guidance. Please run build 1 prior to running build level 3.

1. Open Lighting_DCDC-Settings.h 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. On successful completion of the build click the  “Debug” button, located in the top-left side of the screen. The IDE will now automatically connect to the target, load the output file into the device and change to the Debug perspective.
4. Now click the real-time mode  button that says “Enable silicon real-time mode”. This will allow 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 and set it to use the correct format as shown in Figure 22. This watch window will be used to control the LED dimming stages. Also, 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

6. Run the code by pressing Run Button  in the Debug Tab.
7. The project should now run, and the values in the watch window should keep on updating. You may want to resize or reorganize the windows according to your preference. This can be done easily by dragging and docking the various windows.

8. The watch windows used in this build will be Watch Windows 1, 3 and 4. Watch window 1 will be used to monitor the system. Watch Window 3 will be used to control the SEPIC in closed loop and Watch Window 4 will be used to control the LED dimming stages in closed loop.

9. In Watch Window 3, set Gui_VsetSepic to 20 which corresponds to 20V. 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 20V.

10. In Watch Window 4, set the variable Gui_IsetLed[1] to 0.01, which corresponds to 0.01A. As the current reference is edited notice that Gui_ILed[1] (in Watch window 1) tries to match the reference.

11. Now change the value of Gui_IsetLed[1] to 0.03A. Notice that the LED string brightness has increased with the increased current.

12. Continue editing Gui_IsetLed[1] and also Gui_IsetLed[2-8] with various values from 0.0 to 0.15A.

    NOTE: Do not increase the total current used by all strings above 1.0A. This 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 will be some variation from LED to LED from 0.0 to approximately 0.02A. This is because the LEDs used are not guaranteed, from the manufacturer, to be identical until the LED current is greater than 0.05A.

13. Other variables that may be useful include:

    SlewStep_LED: 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/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 Piccolo MCU's on-chip comparators. 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.

14. Once complete, set Gui_IsetLed[1-8] to 0.0A and then Gui_VsetSepic to 0.0V.

15. Reset the processor  (Target->Reset->Reset CPU) and then terminate the debug session by clicking  (Target->Terminate All). This will halt the program and disconnect Code Composer from the MCU.

Ideas on Further Exploration

- **I^{2}C 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, be used to provide thermal protection for the LEDs, or (along with a lookup table) be 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 comes back to the MCU as feedback. Depending on the resistor and capacitor values chosen, the RC network will either slow down the response of the feedback signal or not be 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 variablility 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.

References

For more information please 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, etc).
  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