AC LED Lighting and Communications Developer's Kit

User's Guide

![Texas Instruments]

Contents

List of Figures

List of Figures   3

List of Tables

AC LED Lighting and Communications Developer's Kit

This document presents a solution to control an LED lighting system using TMS320F2802x microcontrollers. The Piccolo™ series of devices are part of the family of C2000™ microcontrollers, which enable cost-effective designs of LED lighting systems. With these devices, it is possible to control multiple strings of LEDs in an efficient and accurate way. In addition to this, the speed of the C2000 microcontroller allows the board to integrate many supplemental tasks that would, in a normal system, increase chip count and complexity. These tasks could include the control of a DC/DC conversion stage, system management, and various communication protocols such as DALI, DMX512 or even power line communication (PLC). On the board described in this application note, the C2000 controls a LLC resonant converter stage, controls six individual LED strings and provides extra bandwidth to do communications.

1   Overview

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 IsoACLighting-F28027 project
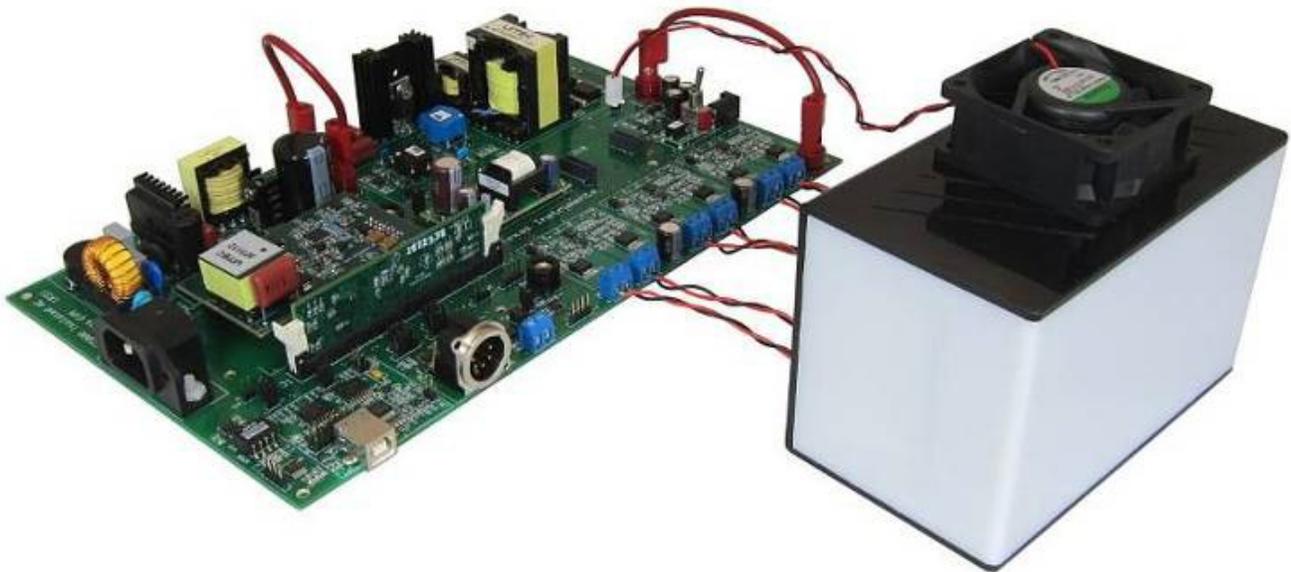
Figure 1. TMDSIACLEDCOMKIT

2 LED Lighting Theory

2.1 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. LEDs have an advantage in applications that can be expensive when considering labor and loss of service (such as replacing lighting bulbs). 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.

2.2 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 non-quantitative way. A better term is luminous flux, measured in lumens, which 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 in Figure 2, 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.
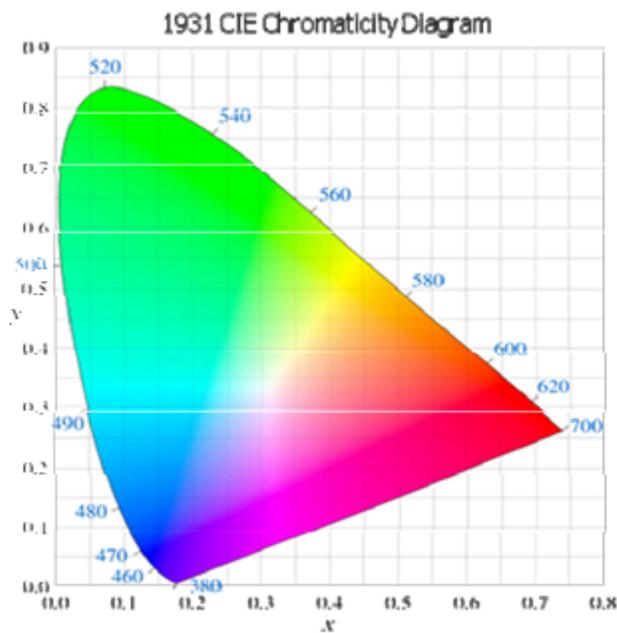
Figure 2. 1931 CIE Chromaticity Diagram

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 3, taken from an LED datasheet, 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. Because of this, it is important to control temperature changes in an LED system.
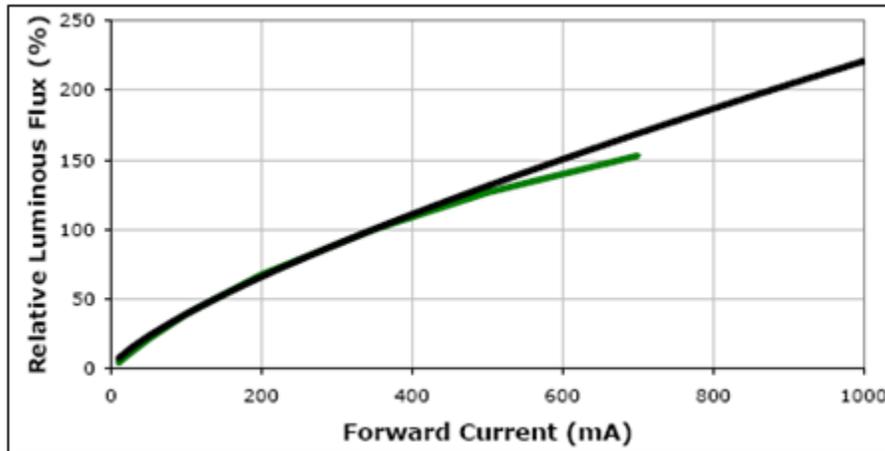
Figure 3. LED Current versus Luminous Flux

Figure 4 shows the relationship between forward voltage and current in an LED. Note that the LED behaves similarly to a diode in that the LED requires a certain threshold voltage before it begins to conduct. 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, make the forward voltage larger than eight times the LED's threshold voltage to make the LEDs light up.
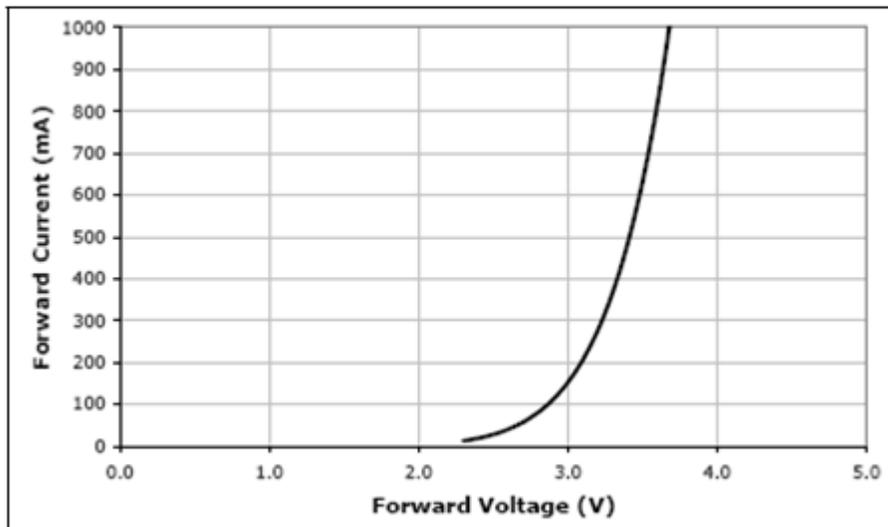
Figure 4. LED Forward Voltage versus Current

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 keeps the same shape but shifts to the left. For instance, if one LED (with the specifications of Figure 4) was controlled at 3.0 V, about 150 mA of current would be drawn through the LED initially. However, as the LED warms up, the graph shifts to the right and the current increases slightly. This increased current then increases the temperature. This cycle continues until the LED device failed. Because of this cycle, current control of LED strings is highly preferred. It also helps to show the importance of thermal management in an LED system.

2.3 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 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, the board likely needs a switched-mode power supply. Figure 5 shows one method of controlling one of these systems.
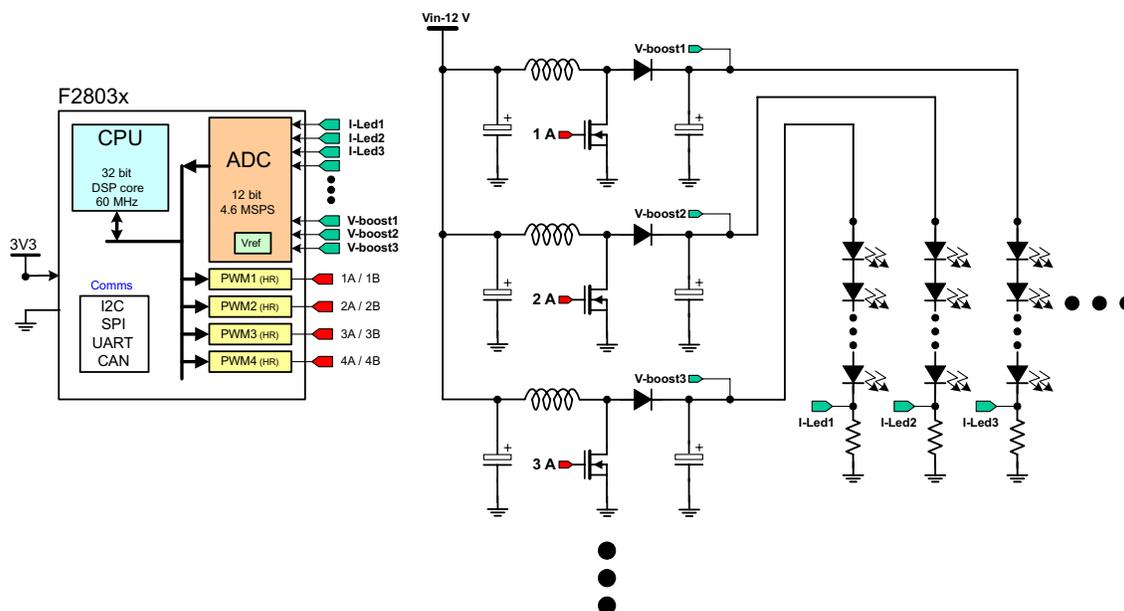
Figure 5. Individual Power Stage per String

In Figure 5, 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 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 through the peak or average current mode, respectively.

Figure 6 shows a different technique is shown. A single DC/DC stage is used to provide the voltage that appears across the LED strings. This voltage then 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.

On the AC LED lighting and communications board, this latter approach has been implemented. The single DC/DC stage is an LLC resonant converter and the Piccolo MCU also controls up to eight LED strings.

In either of these strategies, interleave strings to reduce the peak current and improve the overall efficiency. The Piccolo's PWMs allow for synchronization between individual PWM modules and with an external synchronization pulse.
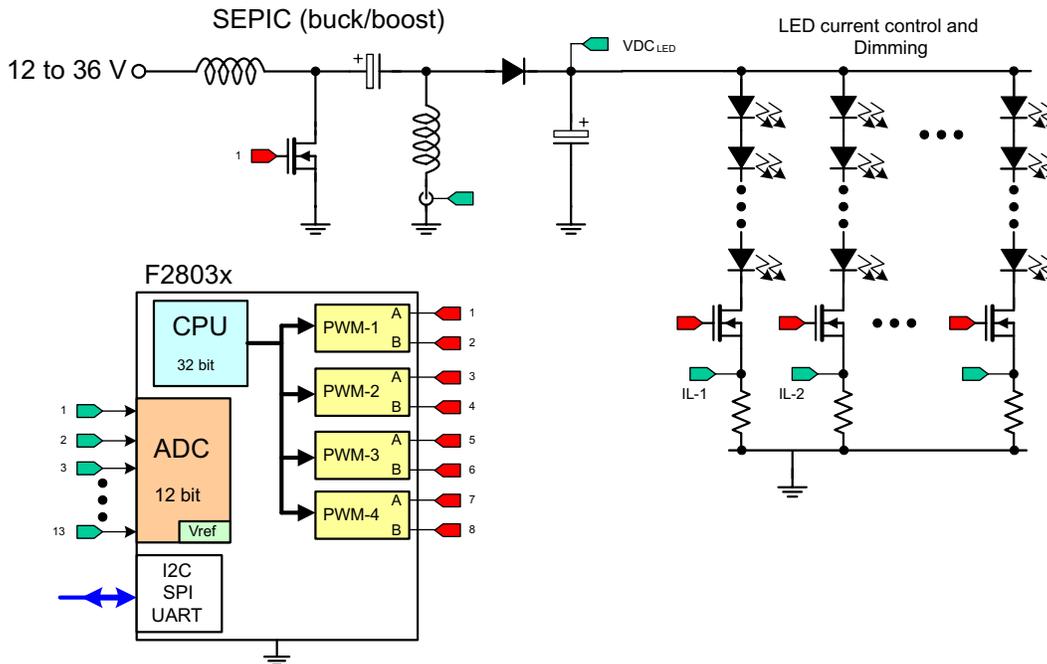

Figure 6. PWM Dimmed Strings with a Common Supply

2.4   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 7 has an AC/DC rectifier, followed by a PFC boost circuit, and then one or more parallel DC/DC stages 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.
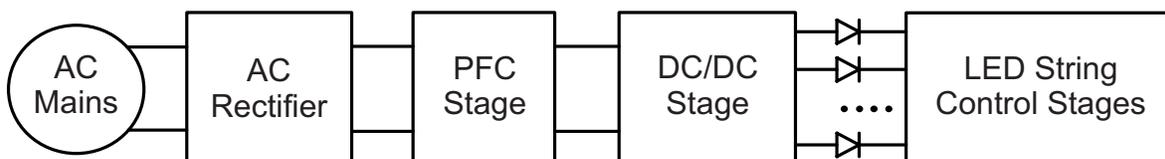
Figure 7. Lighting System

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 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, where 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 intelligent lighting control can all be done on a single chip. Obviously, processor bandwidth limits 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.

3   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 that meet safety requirements like on-chip comparators and trip zones for the pulse width modulators (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 controller device. If a greater number of LEDs is needed in a particular product, only the power stage would need to be checked and any software changes should be minor
- Adaptive dimming based on LED usage, aging, sensed external brightness, and so forth
- Easily synchronized to video clock through capture (CAP) and quadrature encoder pulse (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 inter-integrated circuit (I2C), serial peripheral interface (SPI) and universal asynchronous receiver/transmitter (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

4   System Overview

4.1   Hardware Overview

The AC LED lighting and communications developer's kit takes in universal AC (85 to 250-V AC) input. This AC input then goes through a PFC stage to increase the power factor of the downstream power stages. A PFC stage, as shown in Figure 8, helps the board to meet IEC61000-3-2 and other on-line regulations. At the output of the PFC stage, the voltage will be roughly 395-V DC. To meet the LED string voltage that is required for each LED string to light, an LLC resonant DC/DC stage is used. The LLC provides isolation between the mains and the LED output, and its turns ratio is chosen such that it can output approximately 29 to 36 V. The LLC resonant output is then connected to each of the LED strings. In order to perform independent LED string dimming, a MOSFET is placed in series with each string. The on-time of this string's MOSFET then controls the average current through an LED string. Since the brightness of an LED is roughly proportional to the LED current, use the duty cycle of each string's PWM to control the average current drawn.

In this board, a UCC28810 transition-mode PFC controller is used to manage the PFC stage and the C2000 controls the LLC resonant and lumen output of each LED string. In addition, spare bandwidth on the MCU allows communications and system supervisory tasks to also be done by the C2000 device. Figure 8 illustrates the hardware present on the AC LED lighting and communications developer's kit.
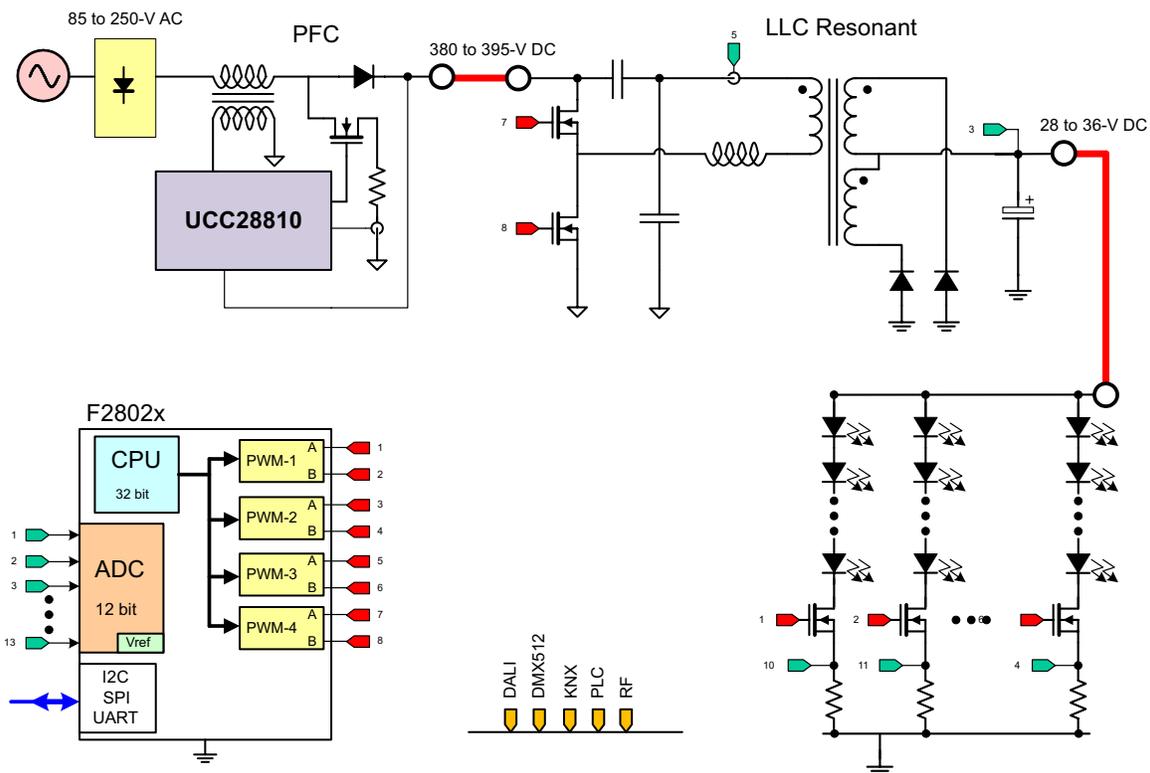

Figure 8. AC LED Lighting and Communications Board Hardware Block Diagram

The key signal connections between the F28027 microcontroller and the power stages located on the AC LED lighting and communications board are listed in Table 1.

Table 1. Key Peripherals Used

MACRO NAME		SIGNAL NAME	PWM OR ADC CHANNEL NO MAPPING FOR F28035	FUNCTION
Resonant LLC Stage		PWM	PWM-4A	LLC converter PWM1
		PWM	PWM-4B	LLC converter PWM2
		Vout-Fb	ADC-A2	LLC output voltage feedback
		Iout-Fb	ADC-A4	LLC current sense
LED-Dimming—Dual Stages	M3	PWM-1	PWM-1A	String 1 PWM
		PWM-2	PWM-1B	String 2 PWM
		IL-1	ADC-B1	String 1 current sense
		IL-2	ADC-B2	String 2 current sense
	M4	PWM-1	PWM-2A	String 3 PWM
		PWM-2	PWM-2B	String 4 PWM
		IL-1	ADC-B3	String 3 current sense
		IL-2	ADC-B4	String 4 current sense
	M5	PWM-1	PWM-3A	String 5 PWM
		PWM-2	PWM-3B	String 6 PWM
		IL-1	ADC-B6	String 5 current sense
		IL-2	ADC-A3	String 6 current sense
Auxiliary Power Module (M6)		Vpfcout-meas	ADC-A1	PFC output voltage monitor
PLC Systems Module (J9)		PLC-ADCRx	ADC-A0	Conditioned PLC receive signal
Main		ADC-A5	ADC-A5	Spare
Main		ADC-A6	ADC-A6	Spare
Main		ADC-B5	ADC-B5	Spare

4.2   Software Overview

4.2.1   Build Options

For the user to slowly build up and understand the project, the project is divided into various builds separated by *#if* options in the IsoACLighting-Main.c and IsoACLighting-ISR.asm files. The build used is set by the variable INCR_BUILD in IsoACLighting-Settings.h. Table 2 is a short description of the different builds available in the IsoACLighting project.

- Build 1: Open loop test. Check basic operation of the LLC resonant and LED strings
- Build 2: Closed loop LLC output voltage and closed loop LED dimming control for each individual string

Table 2. Major Variables

VARIABLE	DESCRIPTION
Gui_Vout_LLC	The LLC Output Voltage (0 to 50 V) for instrumentation purpose only
Gui_Vset_LLC	The voltage reference for the LLC output voltage (0 to 50 V)
PgainLLC	Proportional gain for the SEPIC; value adjustment : 0 – 1 (in Q26)
IgainLLC	Integral gain for the SEPIC; value adjustment : 0 – 1 (in Q26)
DgainLLC	Derivative gain for the SEPIC; value adjustment : 0 – 1 (in Q26)
SlewStep_LLC	This value determines how fast the soft start mechanism would let the duty cycle reach the desired output of the LLC resonant stage. Higher number means slower soft start
Gui_ILed[0-5]	Each LED strings' current (0 to 0.20 A) for instrumentation purpose only
Gui_IsetLed[0-5]	This value determines the current reference for each LED string. (0 to 0.20 A)
Pgain_LED	Proportional gain for each LED string; value adjustment: 0 – 1 (in Q26)
Igain_LED	Integral gain for each LED string; value adjustment: 0 – 1 (in Q26)
Dgain_LED	Derivative gain for each LED string; value adjustment: 0 – 1 (in Q26)
Dmax_LED	Maximum duty cycle allowed for each LED string
SlewStep_LED	This value determines how fast the soft start mechanism would let the current reach the desired LED string current. Higher number means slower soft start.
LEDs_linked	- LEDs_linked = 0: LEDs are independently controlled. Gui_IsetLed[0-5] control strings 1 through 6 - LEDs_linked = 1: LEDs are all controlled through the Gui_IsetLed[0] variable
llc_period	Sets the LLC period in Incremental Build 1
llc_duty	Fixed to 50% in all builds (0.5 in Q24)
led_duty[0-5]	Sets the duty cycle (and the on-time) of LED string 1 through 6. This duty cycle corresponds to a dimming control.

4.2.2   Key Files Located in the Project
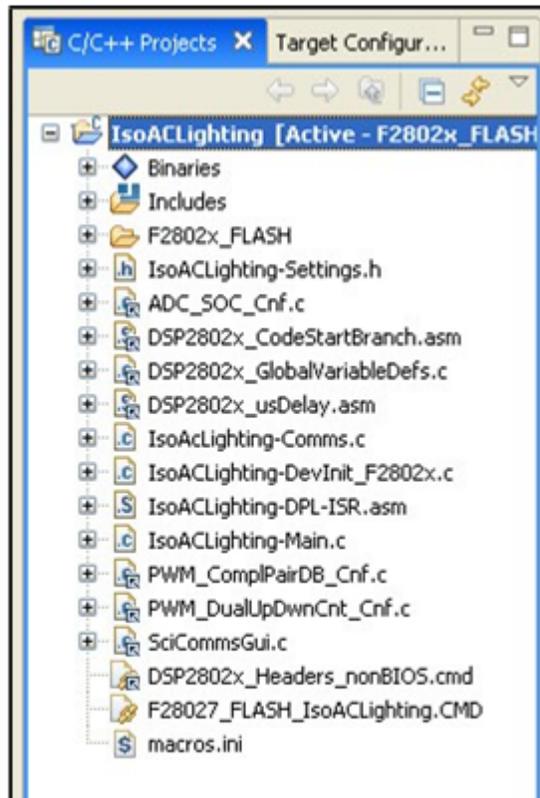
Figure 9. IsoACLighting Project Files

The key framework files used in this project are:

- IsoACLighting-Main.c: Used to initialize, run, and manage the application. This is the "brains" behind the application.

- IsoACLighting-DevInit_F2802x.c: Responsible for a one-time initialization and configuration of the device (in this case, the F28027), and includes functions such as setting up the clocks, PLL, GPIO, and so on.

- IsoACLighting-DPL-ISR.asm: Contains all time critical "control type" code. This file has an initialization section that is executed one time by the C-callable assembly subroutine _DPL_Init. The file also contains the _DPL_Run routine, which executes all the control loop code and runs at the same rate as the LED string 1 PWM, which is used to trigger it. The file also contains the _RESPWM_ISR routine, which runs at the resonant LLC PWM frequency and is responsible for updating the PWM registers based on what the control loop calculates.

- IsoACLighting-Comms.c: Serves as a file that bridges between the main code and the communications code. For this project, the communications code is SCI-UART and handled in SciCommsGui.c.

- IsoACLighting-Settings.h: Used to set global definitions for the project (for example, build options). Note that this file is linked to both IsoACLighting_Main.c and IsoACLighting-DPL-ISR.asm.

- IsoACLighting-Calculations.xls: 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. This document is found in *\TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X* .

4.2.3   Macros Used

To reduce the effort for developing code each time, the board uses macro blocks. These macro blocks are written in C-callable assembly and can have a C and assembly component. Table 3 lists the macros being used in this project.

Table 3. Macros Used

PART	C CONFIGURATION FUNCTION	ASM INITIALIZATION MACRO	ASM RUN-TIME MACRO
ADC	ADC_SOC_CNF	ADCDRV_1ch_INIT n	ADCDRV_1ch n
2P2Z controllers	None	CNTL_2P2Z_INIT n	CNTL_2P2Z n
LLC PWM	PWM_ComplPairDB_CNF	PWMDRV_LLC_ComplPairDB_INIT n	PWMDRV_LLC_ComplPairDB n
LED PWMs	PWM_DualUpDwnCnt_CNF	PWM_DualUpDwnCnt_INIT n	PWMDRV_DualUpDwnCnt n

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.

ADC_SOC_CNF(), ADCDRV_1ch n

Defined in ADC_SOC_Cnf.c and ADCDRV_1ch.asm, this macro abstracts the usage of ADC module. The configuration function should be called with the correct array of channel numbers and trigger sources. In the ISR, the driver macro copies the result from the hardware ADC registers and puts it in the area of memory that ADCDRV_1ch_Rltn is attached to. This variable can then be connected to other library macros.

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. The assembly macros are defined in CNTL_2P2Z.asm.

PWM_ComplPairDB_CNF ()

Defined in PWM_ComplPairDB_Cnf.c, this function configures and initializes the PWM module that will be used to drive the high and low FETs of the LLC resonant converter.

4.2.3.1   PWMDRV_LLC_ComplPairDB n

Defined in PWMDRV_LLC_ComplPair.asm, this macro is used to update the period and duty cycles of PWM[n]A and PWM[n]B in the ISR.

PWM_DualUpDwnCnt_CNF ()

Defined in BuckDual_PwmCnf.c, this function configures and initializes a PWM module that will be used to drive two separate stages. In this case, each PWM module used will drive two LED strings.

PWMDRV_DualUpDwnCnt n

Defined in PWMDRV_DualUpDwnCnt.asm, this macro is used to update the duty cycle of PWM[n]A and PWM[n]B in the ISR. PWM[n]A and PWM[n]B are defined to be phase shifted by 180°.

The IsoACLighting project has the properties shown in Table 4 through Table 6.

Table 4. Memory Usage of the IsoACLighting Project

BUILD LEVEL	FLASH MEMORY USAGE F2802x	PROGRAM RAM USAGE F2802x	DATA RAM USAGE[1] F2802x
Build 2 (FLASH)	4532 words	561 words	705 words

[1] Excluding the stack size (in the project, the stack is defined to be the default 0x380 words)

Table 5. CPU Utilization of Build 2 of the IsoACLighting Project

NAME OF MODULES[1]	NUMBER OF CYCLES
ISR Entry	14 to 18
Context Save, Restore, and so on	24
Timeslicing Overhead	34
LLC Control (updated every LED ISR)	
ADCDRV_1ch	5
CNTL_2P2Z	47
LED Control (two strings updated every other LED ISR) [2]	
ADCDRV_1ch * (2/2)	5
CNTL_2P2Z * (2/2)	47
PWMDRV_DualUpDwnCnt * (1/2)	8
LLC Update ISR (runs at LLC frequency) [3]	
ISR Entry * 3	42 to 54
ContextSave, Restore, etc * 3	72
PWM_LLC_ComplPairDB * 3	48
Average Total Number of Cycles (loop rate = 62.5 kHz)	**362**
Average CPU Utilization at 60 Mhz	37.71%
Average CPU Utilization at 40 Mhz	56.56%

[1] The modules are defined in the header files as "macros".
[2] Results for the LED Control subpart are shown as an average. In one LED ISR, no cycles are used for LED Control and PWM update, but in the next cycle, 120 will be used.
[3] The LLC frequency is variable and therefore the worst case is shown (at no load an ISR happens three times in a 62.5-kHz timeframe).

Table 6. System Features

HEADER 1	HEADER 2
Development and Emulation	Code Composer Studio™ v4.1 (or above) with real-time debugging
Target Controller	TMS320F2802x
PWM Frequency	63.2 to 150 kHz PWM (LLC); 31.25 kHz PWM (LED sting)
PWM Mode	LLC: Asymmetric; LED: Six symmetric PWMs with phase shifts of 60°
Interrupts	• EPWM1 (LED PWM) counter equal to period or zero; drives controllers and LED PWM update • EPWM4 (LLC PWM) counter equal to zero; drives LLC PWM update

5   Hardware Setup

Each component in this document is named first with their macro number followed by the reference name. For example, [M2]-J1 refers to the jumper J1 located in the macro M2, and [Main]-J1 refers to the J1 located on the board outside of the other defined macro blocks. Figure 10 lists some of the major connectors and features of the TMDSIACLEDCOMKIT lighting board. For a full description of connectors, see *AC LED Lighting and Communications Hardware Developer's Kit* (SPRABS2).
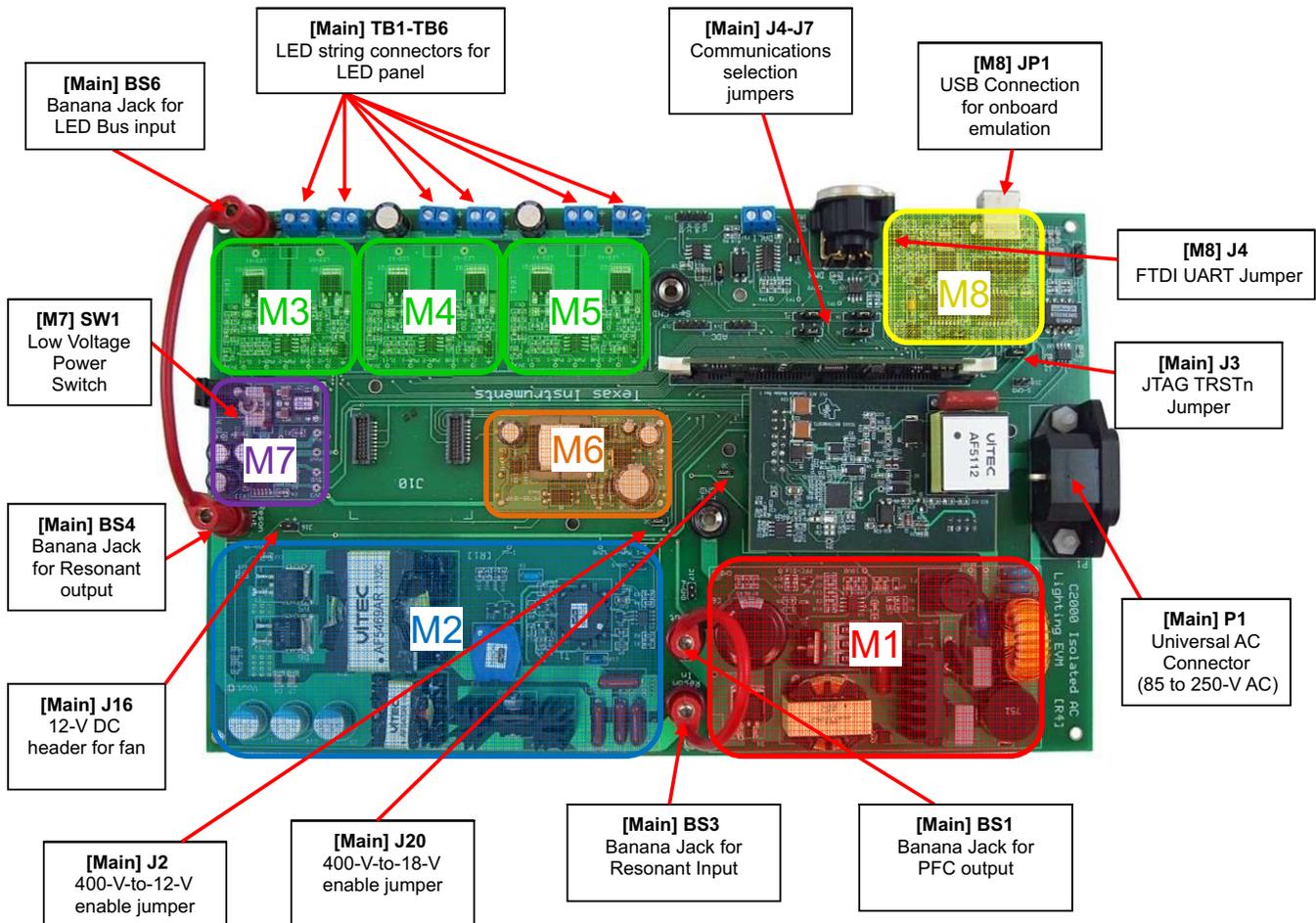
Figure 10. Hardware Features

1. On the Piccolo F28027 controlCARD, check the following switches:

   (a) SW1: make sure position 1 and 2 are both in the on (up) position so the C2000 MCU is set to boot-to-flash.

   (b) SW2 to SW4: set all switches to face downward (default setting).

   (c) No jumper at R10 (if applicable).

2. Put a F28027 control card into the socket on the AC LED lighting and communications board, and connect a cable from the USB connector ([M8]-JP1) on the board to the computer. [M8]-LD1, near the LED lighting board's USB connector, should turn on once everything is connected correctly.

NOTE: If CCS has never been installed, it may be necessary to install drivers to make the board work correctly. If a window 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.

---

3. When Windows® asks to search for Windows Update, select *No, not this time* and click *Next* (see Figure 11).
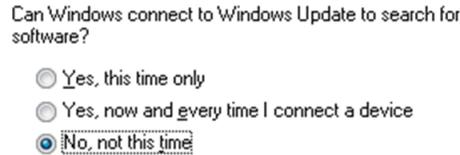
Figure 11. Windows Update Options

4. On the next screen, select *Install from a list or specific location* and click *Next* (see Figure 12).
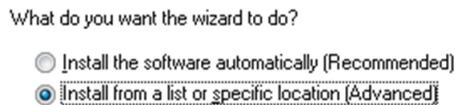
Figure 12. Wizard Options

5. Select *Search for the Best Driver in these locations*, uncheck *Search removable media*, check *Include this location in the search*, and browse to [USB Drive]:\XDS100 Drivers (see Figure 13).
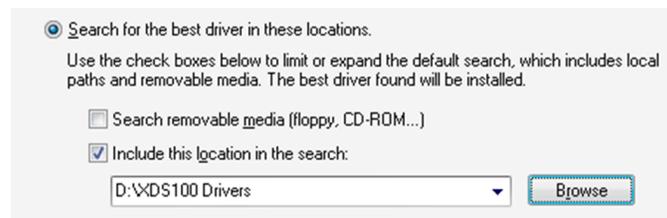
Figure 13. Driver Options

6. Click *Next* to install the drivers. The driver install screen will appear three times. Repeat this procedure each time.

7. Connect the LED panel to [Main]-TB1 through [Main]-TB6 on the AC LED Lighting and Communication board. For each twisted cable from the LED panel, make sure to connect the red wire to the positive, "**+**", terminal and the black wire to the negative, "**–**", terminal.

8. Connect or verify the following:

    (a) Switch [M7]-SW1 to the internal position (switched away from "Ext").

    (b) Connect a jumper on [M8]-J4.

    (c) Connect a jumper on [Main]-J2.

    (d) Connect a jumper on [Main]-J20.

    (e) Connect a jumper on [Main]-J3.

9. Connect the 12-V power supply that was included in the kit to [M7]-JP1.

10. Connect the fan's power cable to [Main]-J16. Connect the red wire toward "+" [only needed if driving the total LED current above 1 A (approximately 35 W)].

11. Connect a red banana-to-banana cable between [Main]-BS4 and [Main]-BS6.

6   Software Setup

6.1   Installing CCS and controlSUITE™

1. If not already installed, install CCS from the DVD included with the kit.
2. Go to http://www.ti.com/controlsuite and run the controlSUITE installer. Select to install the AC LED lighting and communications kit software. If desired, allow the installer to also automatically check and download software updates.

6.2   Setting Up CCS to Work With the AC LED Lighting and Communications Kit

1. Open Code Composer Studio. Once Code Composer Studio opens, the workspace launcher may ask to select a workspace location.
2. Click the *Browse…* button.
3. Create this path by making new folders as necessary:

    (a) *C:\Documents and Settings\My Documents\ CCS_workspaces\IsoACLighting*

4. Uncheck the box that says *Use this as the default and do not ask again* (see Figure 14).
5. Click *OK.*.
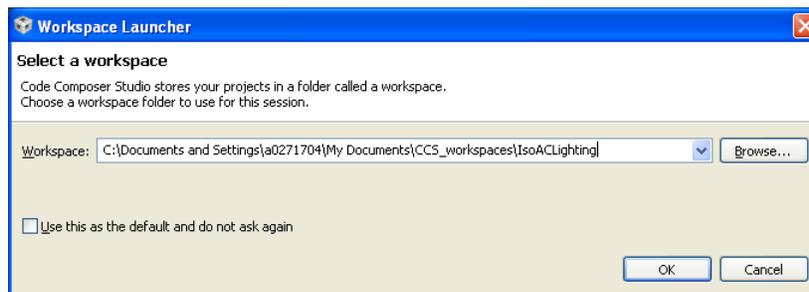
Figure 14. Workspace Launcher

NOTE: A workspace is a location on the hard drive where all the user settings for the IDE (which projects are open, what configuration is selected, and so on) are saved. This location can be anywhere on the disk; the location mentioned above is just for reference. Also note that if this is not the first time CCS has run, this dialog may not appear.

---

6. Next, configure CCS to know which MCU to connect to. Click *Target→New Target Configuration...* . Name the new configuration "xds100-f28027.ccxml". Make sure that the *Use shared location* checkbox is checked and then click *Finish* (see Figure 15).
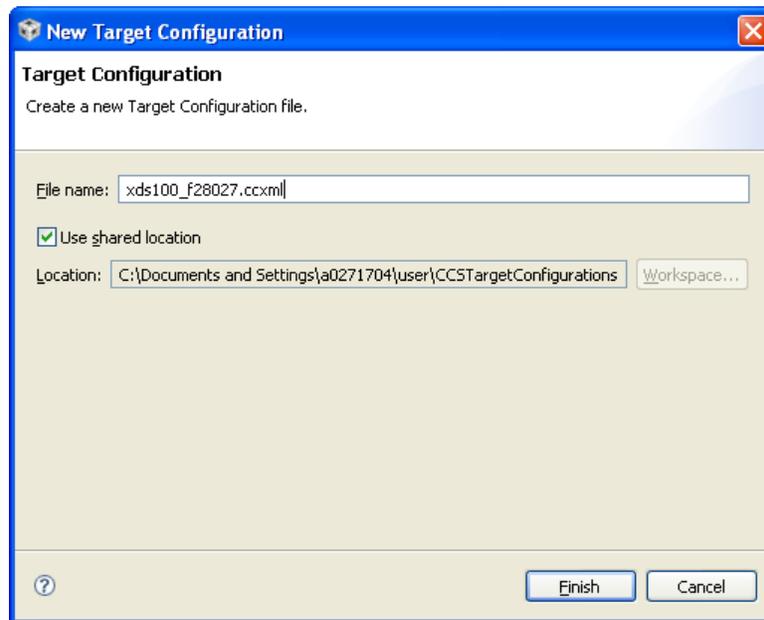
Figure 15. Creating a Target Configuration

7. This should open up a new tab as seen in Figure 16. Select and enter the options as shown:
   (a) In the *Connection* field, write "Texas Instruments XDS100v1 USB Emulator".
   (b) Under *Board or Device*, check the box marked *TMS320F28027*.
   (c) Click *Save*.
   (d) Close the *xds100-f28027.ccxml* tab.
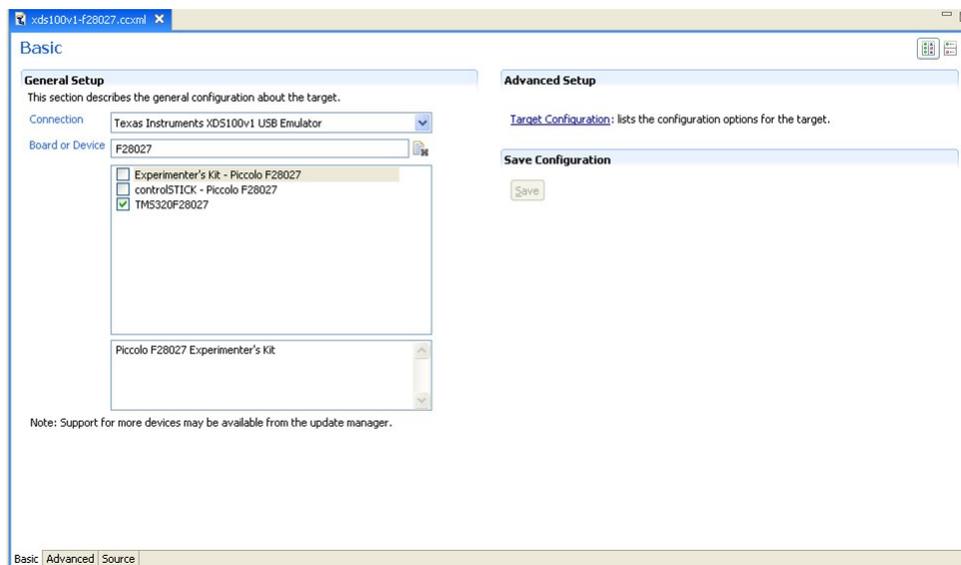
Figure 16. Configuring a New Target

8. Assuming this is the first time CCS has been used, the xds100-F28027 configuration is now set as the default target configuration. Confirm this by going to *View →Target Configurations*. In the *User Defined* section, right click on the xds100-f28027.ccxml file and select *Set as Default*. This tab also allows the user to reuse existing target configurations and link them to specific projects.

9. Add the IsoACLighting project into your current workspace by clicking *Project→Import Existing CCS/CCE Eclipse Project*. If importing the project with CCSv5, check the box that allows CCS to *Automatically import referenced projects*.

   (a) Select the root directory of the AC LED lighting and communications developer's kit:

\TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X\IsoACLighting-F28027
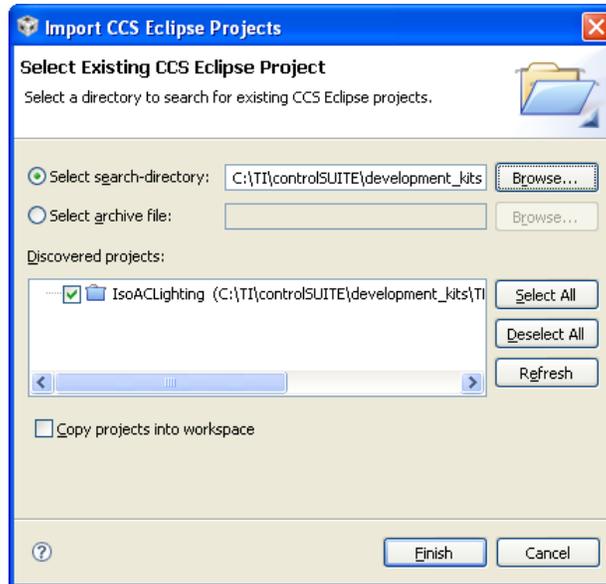
Figure 17. Adding the IsoACLighting Project to Workspace

   (b) Click *Finish* to copy the IsoACLighting project into the workspace.

10. To set the IsoACLighting project as the active project, right-click on the project name and click *Set as Active Project*. Expand the file structure of the project.

6.3 Incremental System Build for IsoACLighting 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 here:

- Build 1: Open loop test, and check basic operation of the LLC resonant and LED strings
- Build 2: Closed loop LLC resonant and LED string control

7 Build 1: Open Loop LLC Resonant Control and Open Loop LED String Control

These are the objectives of this build:

- Verify that the LLC and LED power stages on the board are working correctly.
- Control the frequency of the LLC stage and the duty cycles of the LED strings.
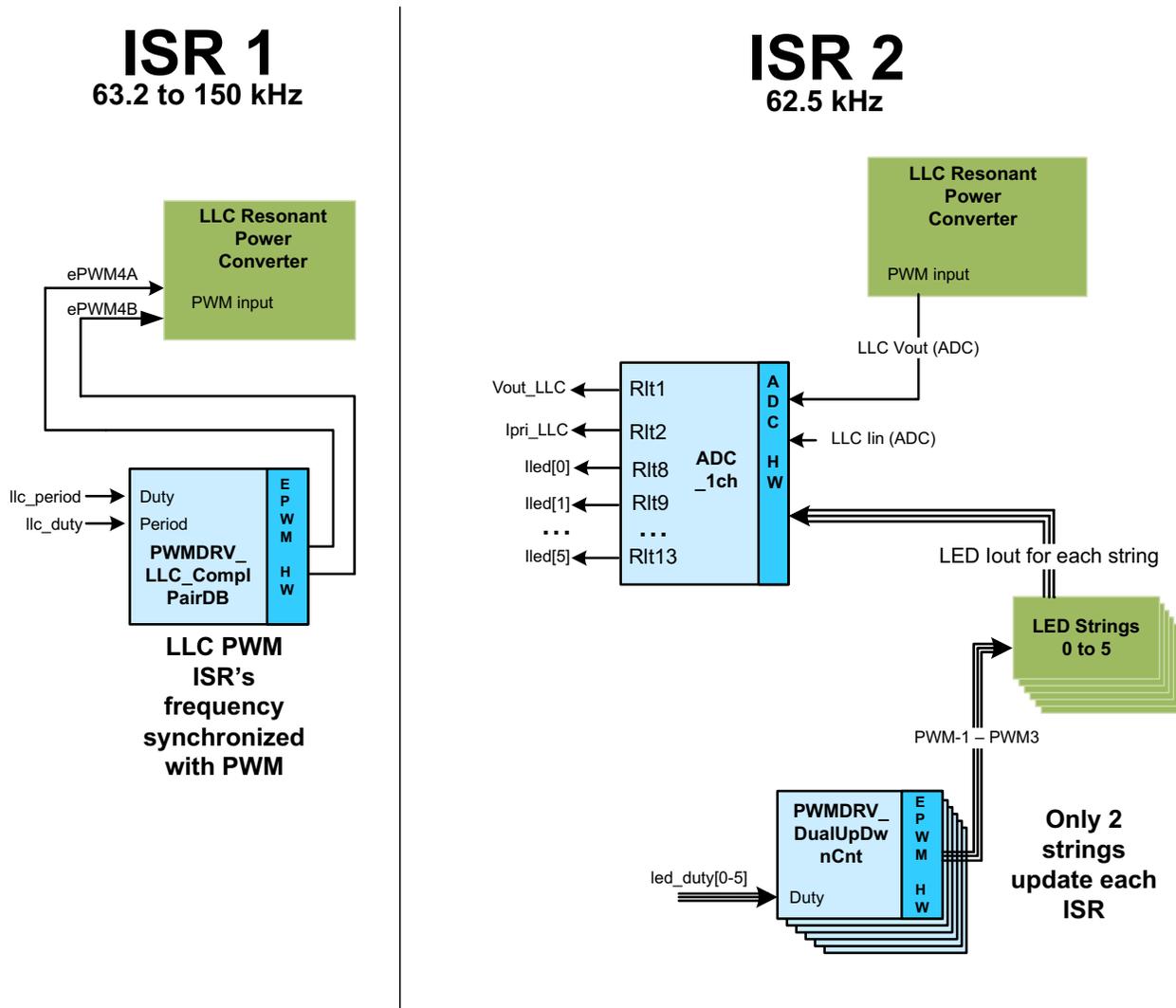
Figure 18. Build Level 1 Block Diagram

7.1 *Inspect the Project (Optional)*

1. After initial boot processes finish, the software begins from the main function. Open up IsoACLighting-Main.c and find the *main()* function in line 290. After the device begins at main, the software flow looks as shown in Figure 19. Most of the software that affects the power stages is done in assembly with the help of C2000's digital power library modules. The DPL_ISR is started by the PWM that drives LED0 and the RES_PWM_ISR is triggered by the varying frequency PWM that controls the LLC resonant power stage.
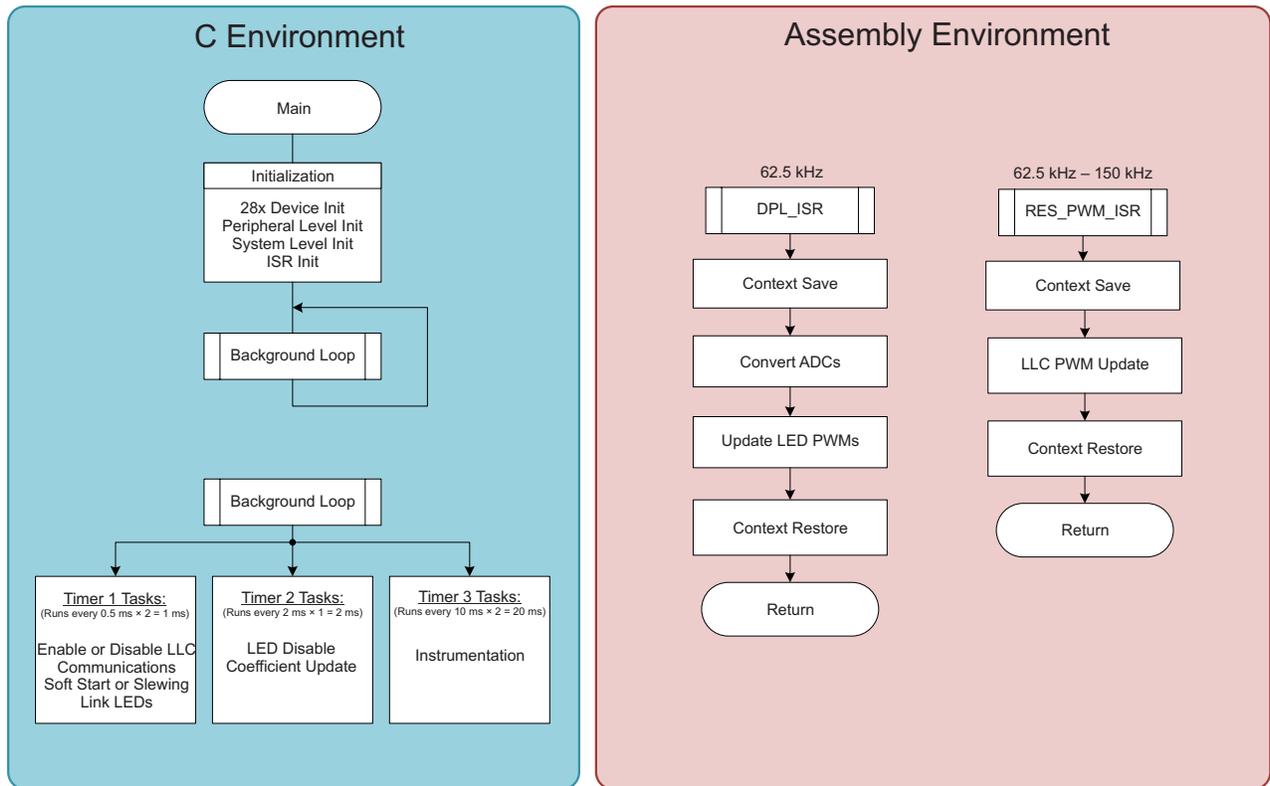

Figure 19. Incremental Build 1 Software Flowchart

2. The first thing that the software does in the *main()* function is to call the *DeviceInit()* function found in IsoACLighting-DevInit_F2802x.c. Open and inspect IsoACLighting-DevInit_F2802x.c by double clicking on the filename in the project window. In this file, the various peripheral clocks are enabled or disabled and the functional pinout that configures which peripherals comes out of which pins is defined.

   (a) Confirm that the ADC and PWM1 to 4 peripheral clocks are enabled (lines 92 to 120). Also confirm that GPIO00-GPIO07 are configured to be PWM outputs (lines 122 to 184).

3. The IsoACLighting 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. This process helps in the step-by-step debug and understanding of the system.

   (a) From the *C/C++ Project* tab, open the file IsoACLighting-Settings.h file and make sure that INCR_BUILD is set to 1. Save the file. After testing Build 1, this variable needs to be redefined to move on to Build 2, and so on until all builds are complete.

   NOTE: The entire project must be recompiled if this variable is changed.

4. Go back to the IsoACLighting-Main.c file and further inspect the software. In lines 322 to 399, many of the variables needed by the project are initialized to their default values. In lines 400-547, the ADC and PWM peripherals are initialized. Note that two LED strings are controlled per PWM module and there is a 180° shift between the 2 controlled strings. Each PWM module is then phase shifted by 60° so that power is distributed more evenly within a cycle.

    (a) Confirm that the LLC PWM is configured to begin at 120 KHz and that the LED PWMs is configured to run at 31.25 KHz (lines 427 to 434).

5. The ADC is configured in lines 439 to 467.

    (a) Note that ADC-B1 (LED string 1) is configured to be ADC result channel 8 (line 444) and that this result is configured to start conversion on a PWM1 SOCA trigger (line 457). Note that the PWM1 SOCA trigger is defined to happen when the PWM1 timer reaches its period value (line 473 to 475). Other ADC channels are configured similarly.

6. In lines 554 to 651, notice the various incremental build configuration code. The build specific configuration code configures how the power library blocks are connected. In INCR_BUILD==1, the system is set to be open loop. In INCR_BUILD==2, the power library pointers are connected together by system variables such that a closed loop system is created.

    (a) Note that ADCDRV_1ch_Rlt8 is attached to the Iled[0] variable and that PWMDRV_DualUpDwnCnt_Duty1A is attached to led_duty[0] variable. The digital power library macros rely on the Main.c file to define addresses that the digital power library macros will read and write to. As defined, the ADCDRV_1ch macro reads from the ADC peripheral and write to Iled[0], and the PWMDRV_DualUpDwnCnt macro reads from the led_duty[0] variable and write to the PWM peripheral. These macros run based on the timing of the ISR in the DPL-ISR.asm file.

    NOTE: This section assumes that Section 5 and Section 6 have been completed. If not already done, go through these sections before continuing.

7.2 Build and Load the Project

1. Open the IsoACLighting-Settings.h file from the *C/C++ Project* and make sure that INCR_BUILD is set to 1. Save the file. After testing Build 1, this variable needs to be redefined to move on to Build 2, and so on until all builds are complete.

    NOTE: The entire project must be recompiled if this variable is changed.

2. Turn on bias supply power by switching [M7]-SW1 to the *Ext* position. A green LED on the controlCARD should light up.

3. 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.

4. After successfully completing the build, click the *Debug* button 🐞 located in the top-left side of the screen. The IDE is now automatically connected to the target, load the output file into the device, and change to the *Debug* perspective.

5. Click the *Real-Time Mode* button 🔲 *Enable silicon real-time mode* to edit and view variables in real time without halting the program.

6. A message box *may* appear. If so, select *YES* to enable debug events and 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.

7.3 Setup 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. Add the variables found in Figure 20 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 are used to monitor the status of the board.

   Hint: *Shift+Click* can be used select multiple adjacent variables or elements, and then right-clicking and changing the format affects all variables selected.

Figure 20. Configuring the Watch Window for Build 1

2. Click on the *Continuous Refresh* button in each watch window to enable the window to run with real-time mode, which means that the variables change with the software in real time.

7.4 Additional Hardware Setup Needed for this Build

This set of instructions assumes that Section 5 has been completed:

1. Connect an oscilloscope probe to the vias *PWM-1* and *PWM-2* in the [M2] section (LLC) of the board. Connect a third probe up to *PWM-1* in the [M3] section (LED). The pointed tips of most oscilloscope probes work well for this.

2. Make sure that there is not a banana-to-banana cable connecting [Main]-BS1 and [Main]-BS3.

3. Connect a multi-meter between the LLC output ([Main]-BS4) and secondary ground ([Main]-BS5).

4. Attach an appropriate DC load between the LLC output ([Main]-BS4) and the secondary ground ([Main]-BS5). A 25 to 50-Ω resistive load should work fine (or a 0.5 to 1-A load).

5. Connect an unpowered DC supply capable of delivering 390 V between [Main]-BS1 (+) and [Main]-BS2 (–). Set the DC supply to have a current limit of 0.25 A to allow only 100 W to be delivered to the load.

> **NOTE:** For safety reasons, use an isolated DC supply to supply 400-V DC to the board.

7.5 Run the Code

1. Run the code by pressing the *Run* button [▷] in the *Debug* tab.

2. The project should now run, and the values in the watch window should keep on updating. To resize or reorganize the windows, drag and dock the various windows.

3. Change `ResonantEnable` to '1'. Figure 21 should be visible. Note that a `llc_period` of 550 means that the frequency is 60000000/(550) = approximately 109 kHz. Most resonant LLC stages keep the LLC PWM's duty cycle at 50%. Therefore, always keep `llc_duty` at 0.5. Notice the deadband between the two LLC PWM waveforms.
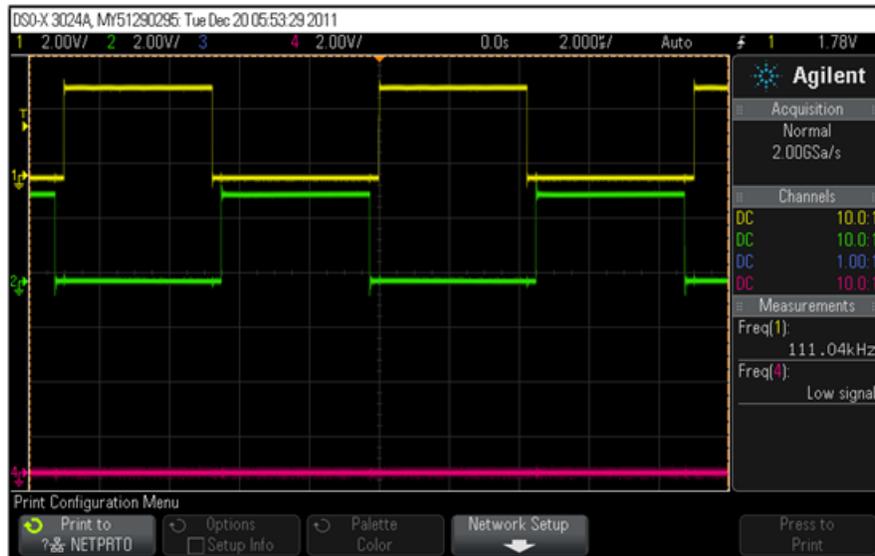
Figure 21. LLC PWM Waveform at 110 kHz

4. Edit the `llc_period` to 700. The LLC PWM's frequency should have decreased to approximately 85.7 kHz.
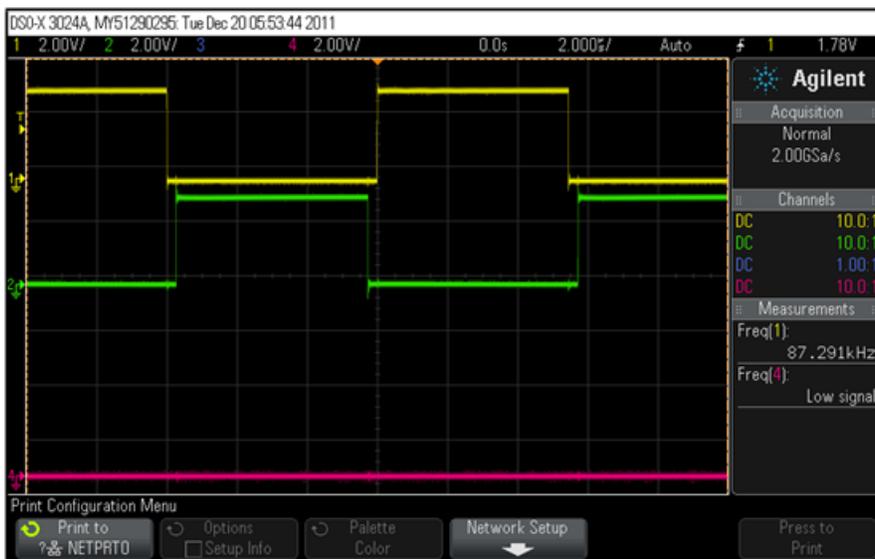
Figure 22. LLC PWM Waveforms at 85.7 kHz

5. Set `led_duty[0]` to 0.1 to set the duty cycle of the MOSFET controlling LED string 0 to be approximately 10% duty cycle. The oscilloscope waveform for the LED PWM should now show a 31.25-kHz signal that has 10% duty cycle.

Figure 23. LED0 PWM Waveform at 10% Duty Cycle

6. Set `llc_period` to 400.

7. Change `ResonantEnable` to 0.

8. Set `led_duty` to 0.0.

9. Enable the DC supply to give 390-V DC to the board. The LLC output voltage should stay around 0.0 V.

10. Change `ResonantEnable` to 1. The output voltage should ramp quickly to a voltage between 26 to 36-V DC.

11. If the LLC output is less than 34-V DC, increase `llc_period` in steps of 10 to 25 until the LLC Resonant output is about 33 to 34 V. Because the system is being run in open loop, do not make any large step changes to `llc_period`. Note that the ADC is converting in real time and its converted reading can be found in Gui_Vout_LLC.

Figure 24. Watch Window—Changing llc_period

---

12. Change `led_duty[0]` to 0.2. Note that once any LED string is on, the LLC output voltage decreases because the LLC is currently operating in an open loop and has more load than it did before.

Figure 25. Watch Window—Changing led_duty

13. LED strings 1 to 5 can also be checked individually by setting them to 0.1 while the rest of the strings are at 0.0. Note that if the LLC stage is loaded too strongly, its output voltage decreases to the extent that it no longer surpasses the LED strings' threshold voltage.

14. Once complete, set `led_duty[0]`-led_duty[5] to 0.0 and then `llc_period` to 400.

15. Turn off the 400-V DC supply.

16. Set `ResonantEnable` to 0.

17. Halt the processor ▯▯ .

18. Stop real-time mode by clicking 🔳 .

19. Reset the processor 🔲 (*Target→Reset→Reset CPU*) and then terminate the debug session by clicking 🔲 (*Target→Terminate All*) to halt the program and disconnect CCS from the MCU.

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

The objective of this build is as follows:

- Regulate LLC resonant 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 Figure 26.
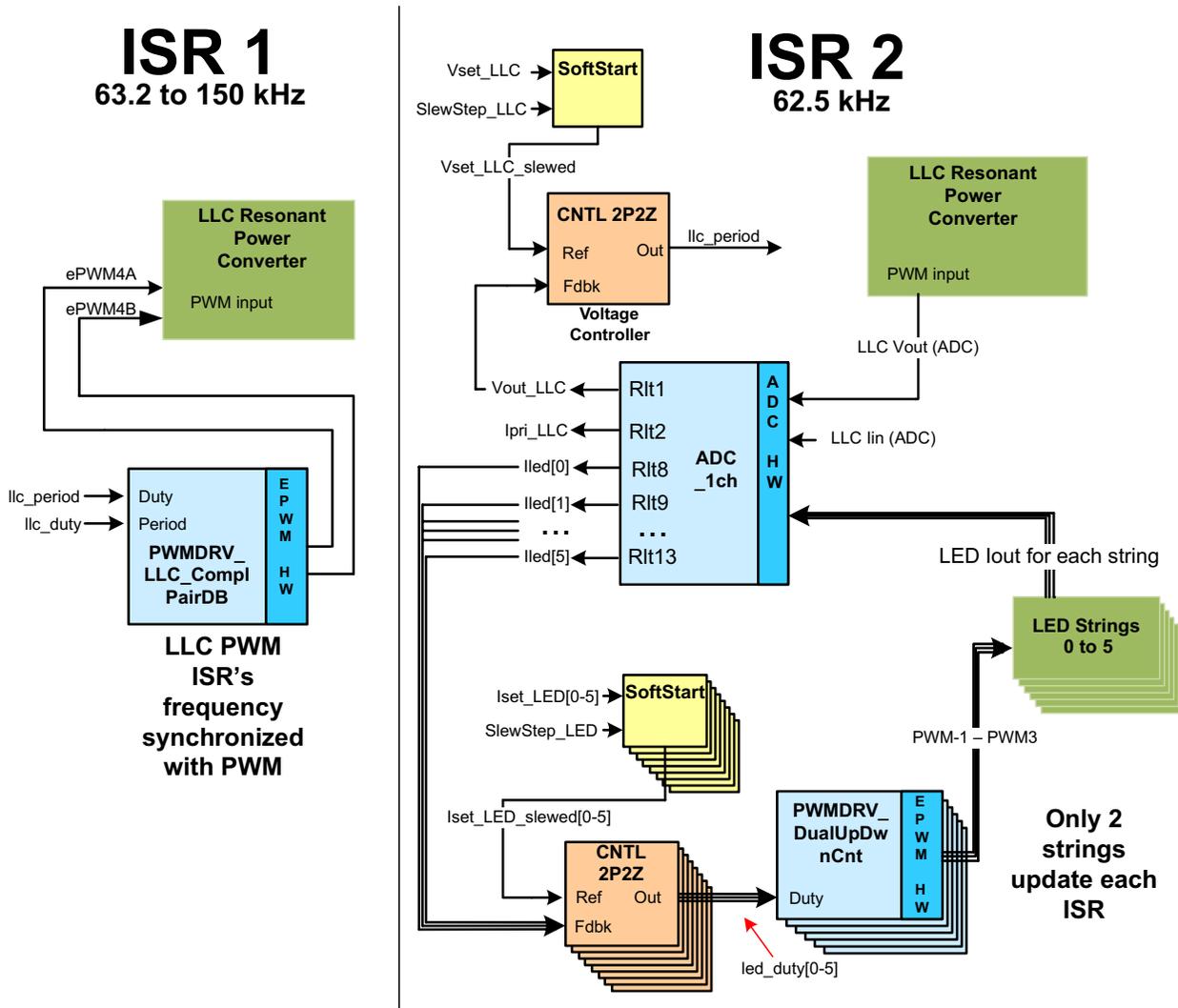
**Figure 26. Build Level 2 Block Diagram**
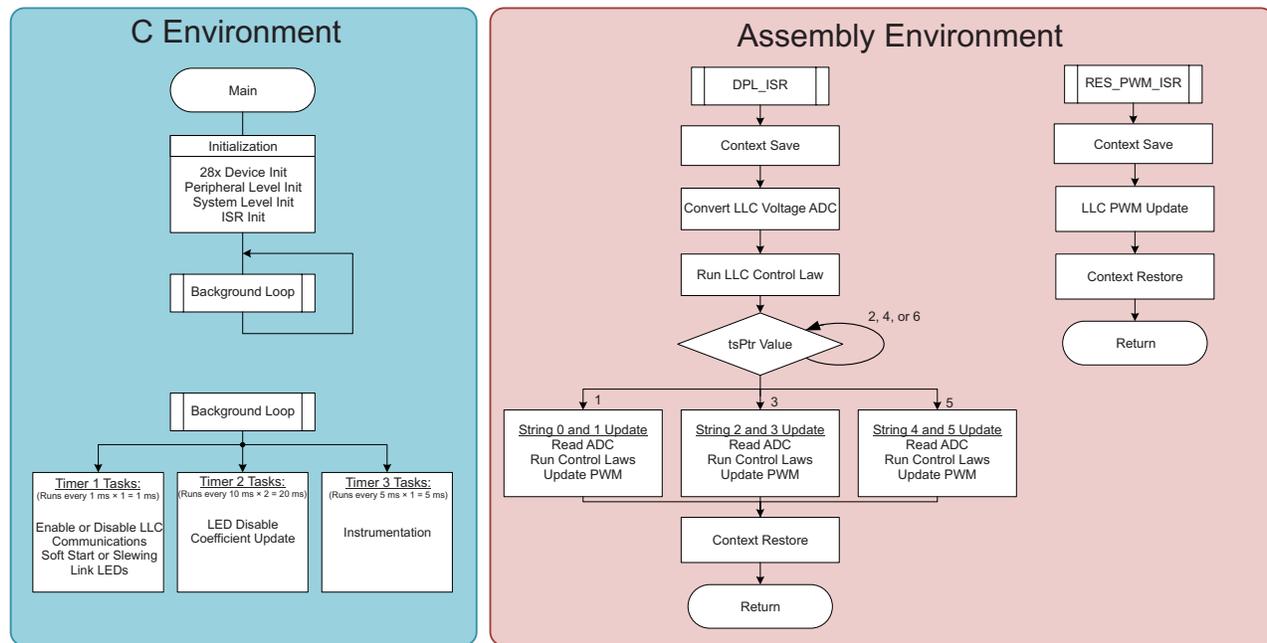
Figure 27. Incremental Build 2 Software Flowchart

8.1 Additional Hardware Setup Needed for this Build

The instructions for this build are more succinct than in the first build. For further guidance, see the instructions in Section 7. Run build level 1 prior to running build level 2.

This set of instructions assumes that Section 5 has been done.

1. Connect a banana-to-banana cable between [Main]-BS1 and [Main]-BS3.
2. Connect a second banana-to-banana cable between [Main]-BS-BS4 and [Main]-BS6.
3. Plug one side of the AC line cable to [Main]-P1. **Do not connect the other end of the cable yet.**

8.2 Run the Code

1. Flip [M7]-SW1 to the *Ext* position if not already done.
2. Open IsoACLighting-Settings.h and change the incremental build level to 2 (#define INCR_BUILD 2). Save the file.
3. Right-click on the project name and select *Rebuild Project*.
4. 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.
5. Click the *Real-Time Mode* button that says *Enable silicon real-time mode* to allow the user to edit and view variables in real time without halting the program.

6. Add a new watch window to the workspace and add the variables and set it to use the correct format as shown in Figure 28. This watch window is used to control the LLC power stage and LED dimming stages. Also, set this new watch window to continuously refresh.

Figure 28. Configuring the Watch Window for Build 2

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 keep on updating.

8. To resize or reorganize the windows, drag and dock the various windows.

9. Plug the other end of the AC cable into the wall (or preferably, plug it into a power strip or surge protector, then flip its switch to enable output).

CAUTION

After the AC cable is plugged in, the board is considered live and has the potential for hazardous shock. Take all necessary precautions before completing this step.

---

10. In the watch window, set ResonantEnable to 1.

11. Set `Gui_Vset_LLC` to 34.5, which corresponds to setting the output reference to 34.5 V. This variable is used as the reference to the LLC controller. The controller edits the duty cycle as necessary to keep `Gui_Vout_LLC`, visible in the watch window, at 34.5 V.

NOTE: With no load, the resonant DC/DC stage may not be able to regulate the output to exactly the reference given. Once loaded, the output will stay at the given reference voltage.

WARNING

The LED panel is capable of driving the LEDs at a very high intensity. It is recommended to face the LED panel away from people and eye protection is recommended.

---

12. Set the variable `Gui_IsetLed[0]` to 0.1, which corresponds to 0.1 A. As the current reference is edited notice that Gui_ILed[0] (in watch window 1) tries to match the reference.

13. Change the value of `Gui_IsetLed[0]` to 0.25 A. Notice that the LED string brightness has increased with the increased current.

14. Continue editing `Gui_IsetLed[0]` and also `Gui_IsetLed[1-5]` with various values from 0.0 to 0.3 A.

> **NOTE:** `Gui_IsetLed` values of up to 0.95 A are possible, but make sure that a fan (such as the one that comes with this kit) is used.

15. Other variables that may be useful include:

    (a) `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.

    (b) `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[0].

16. Once complete, set `Gui_IsetLed[0-5]` to 0.0 A.

17. Set Gui_Vset_LLC to 0.0 V.

18. Set `ResonantEnable` to 0.

19. Disconnect the AC cable from the wall. The board should be considered live for at least 20 seconds after the AC cable is disconnected.

20. Halt the processor ⏸ .

21. Stop real-time mode by clicking 🔲 .

22. Reset the processor 🔳 (*Target→Reset→Reset CPU*) and then terminate the debug session by clicking 🔳 (*Target→Terminate All*) to halt the program and disconnect CCS from the MCU.

9   Ideas on Further Exploration

- **Communications**

  Because the F28027 MCU is using only about 40% of the CPU bandwidth, there is extra bandwidth to do communications, host control or other system tasks.

- **Exponential dimming**

  The eye does not see lumen output increases linearly. Instead, the eye is exponentially less sensitive to small changes as the lumen output increases. A new variable could be set by the user and then squared (or multiplied by some factor) to give the current reference.

- **I2C EEPROM for storing settings**

  The on-board EEPROM could be used to preserve the light's settings once it is powered off. The `i2c_eeprom` CCS project within the device_support section of controlSUITE can be used as a template to begin experimenting with this.

- **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, 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 gives 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 TMDSIACLEDCOMKIT to jump-start the 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

- *DALI Communication on the AC LED Lighting and Communications Developer's Kit* (SPRABR8)

- *DMX512 Communication on the AC LED Lighting and Communications Developer's Kit* (SPRABR9)

- *PLC With the AC LED Lighting and Communications Developer's Kit* (SPRABS0)

- *AC LED Lighting and Communications Hardware Developer's Kit* (SPRABS2)

- TMDSIACLEDCOMKIT_CCS: Provides detailed information on the IsoACLighting project within Code Composer Studio. The document goes through the project in an easy to use lab-style format.
    - *C:\TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X\~Docs\TMDSIACLEDCOMKIT_CCS.pdf*

- TMDSIACLEDCOMKIT-HWdevPkg: A folder containing various files related to the hardware on the AC LED Lighting and Communications Developer's Kit board (schematics, bill of materials, Gerber files, PCB layout, and so on)
    - *C:\TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X \ ~TMDSIACLEDCOMKIT-HwdevPkg[R4]*

- TMDSIACLEDCOMKIT-HWGuide: Presents full documentation on the hardware found on the AC LED Lighting and Communications Developer's board.
    - *C:\TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X\~Docs\ TMDSIACLEDCOMKIT-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
Copyright © 2014, Texas Instruments Incorporated