# AC LED Lighting & Communications Developer's Kit

## Controlling a Full Lighting System with one C2000 device
## CCS Guide

*Brett Larimore*
*C2000 Systems and Applications Team*
*Version 1.1 – December 2011*

## Abstract

This application note 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 design 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 it 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. On the board described in this application note the C2000 controls a LLC resonant converter stage, controls 6 individual LED strings and provides extra bandwidth to do communications.

See the following application notes:
TMDSIACLEDCOMKIT-DALIGuide.pdf
TMDSIACLEDCOMKIT-DMXGuide.pdf
TMDSIACLEDCOMKIT-PLCQSG.pdf
TMDSIACLEDCOMKIT-HWGuide.pdf

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 IsoACLighting-F28027 project.

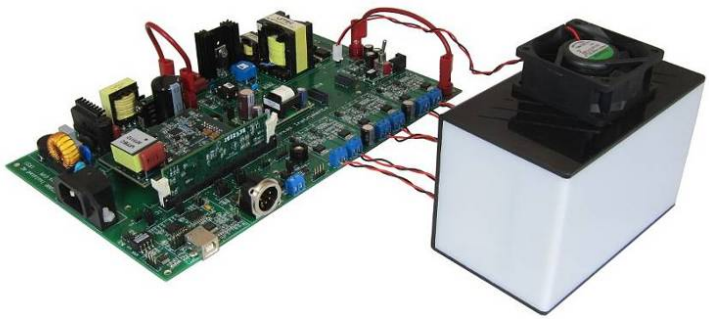


**Figure 1: TMDSIACLEDCOMKIT**

# 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 .

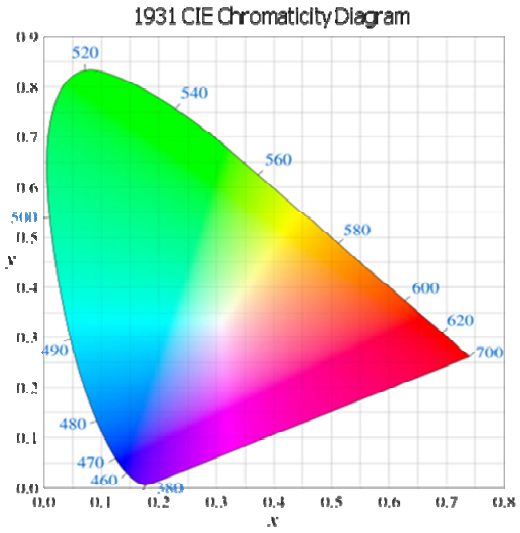



**Figure 2: 1931 CIE Chromaticity Diagram**

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 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.

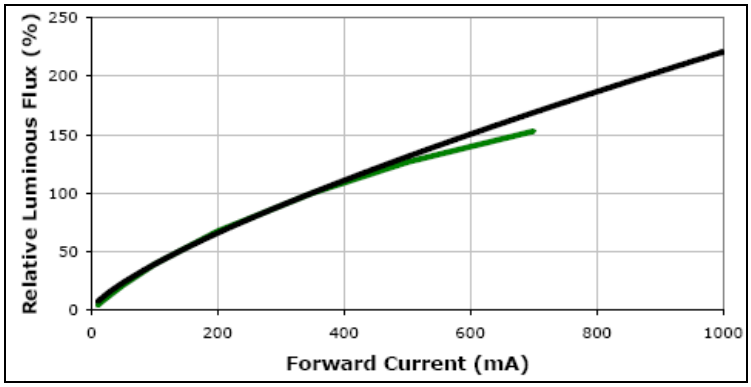



**Figure 3: LED Current vs. Luminous Flux**

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.

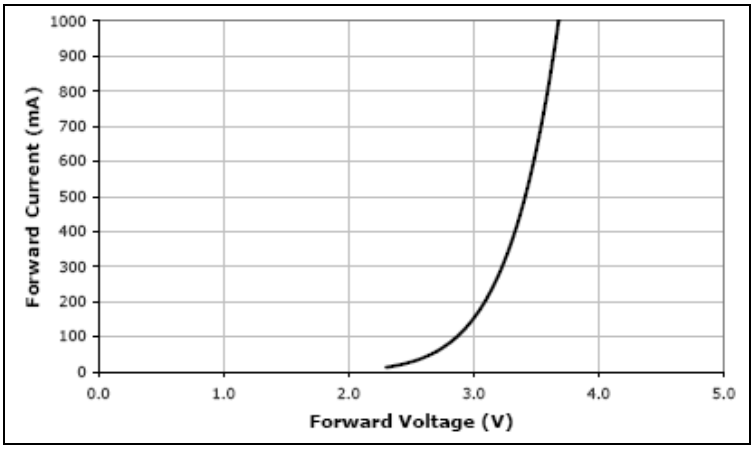



**Figure 4: LED Forward Voltage vs. Current**

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.

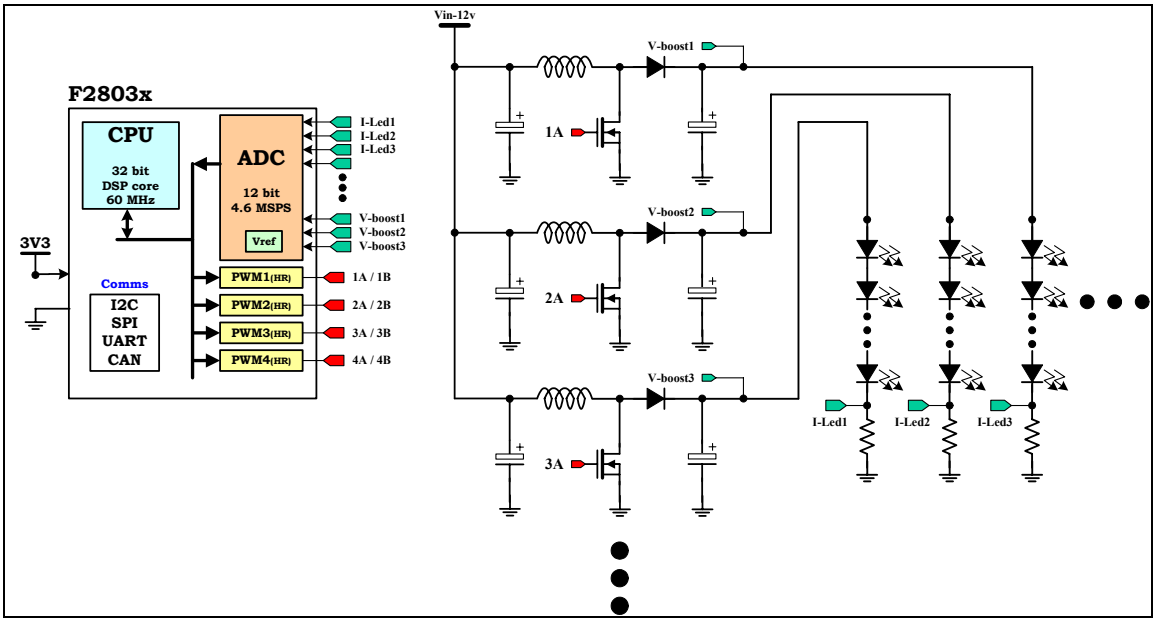



**Figure 5: Individual Power Stage per String**

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 AC LED Lighting and Communications board, this latter approach has been implemented. The single DC/DC stage is a LLC Resonant 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.

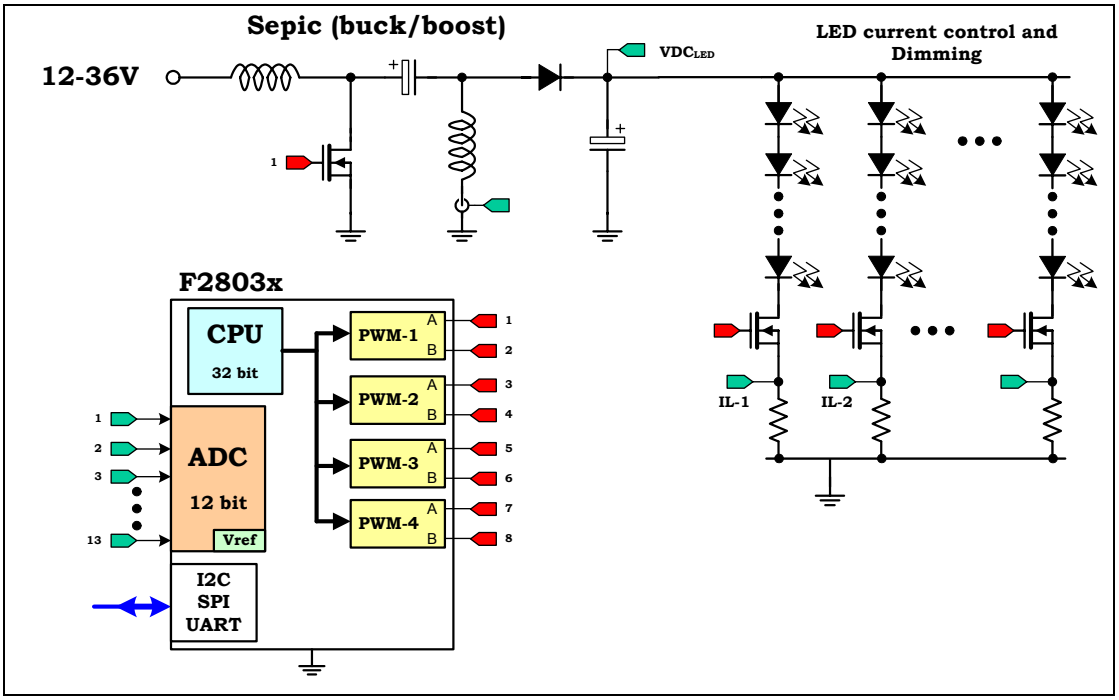



**Figure 6: PWM Dimmed Strings with a Common Supply**

## 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.

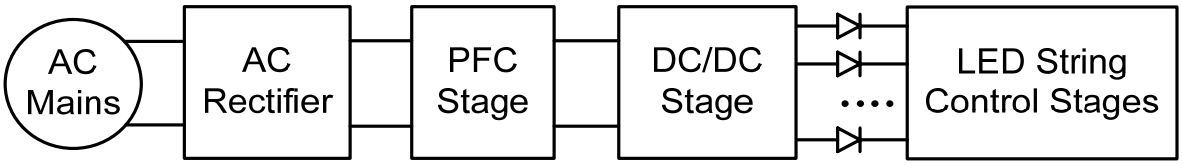



**Figure 7: Lighting System**

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 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 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, 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 AC LED Lighting and Communications Developer's Kit takes in universal AC (85-250Vac) input. This AC input then goes through a PFC stage in order to increase the power factor of the downstream power stages. A PFC stage, as shown here, 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 395V DC. To meet the LED string voltage that is required for each LED string to light, a 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 29-36V. The LLC resonant ouput 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 will then 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.

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. The figure below illustrates the hardware present on the AC LED Lighting & Communications Developer's Kit.

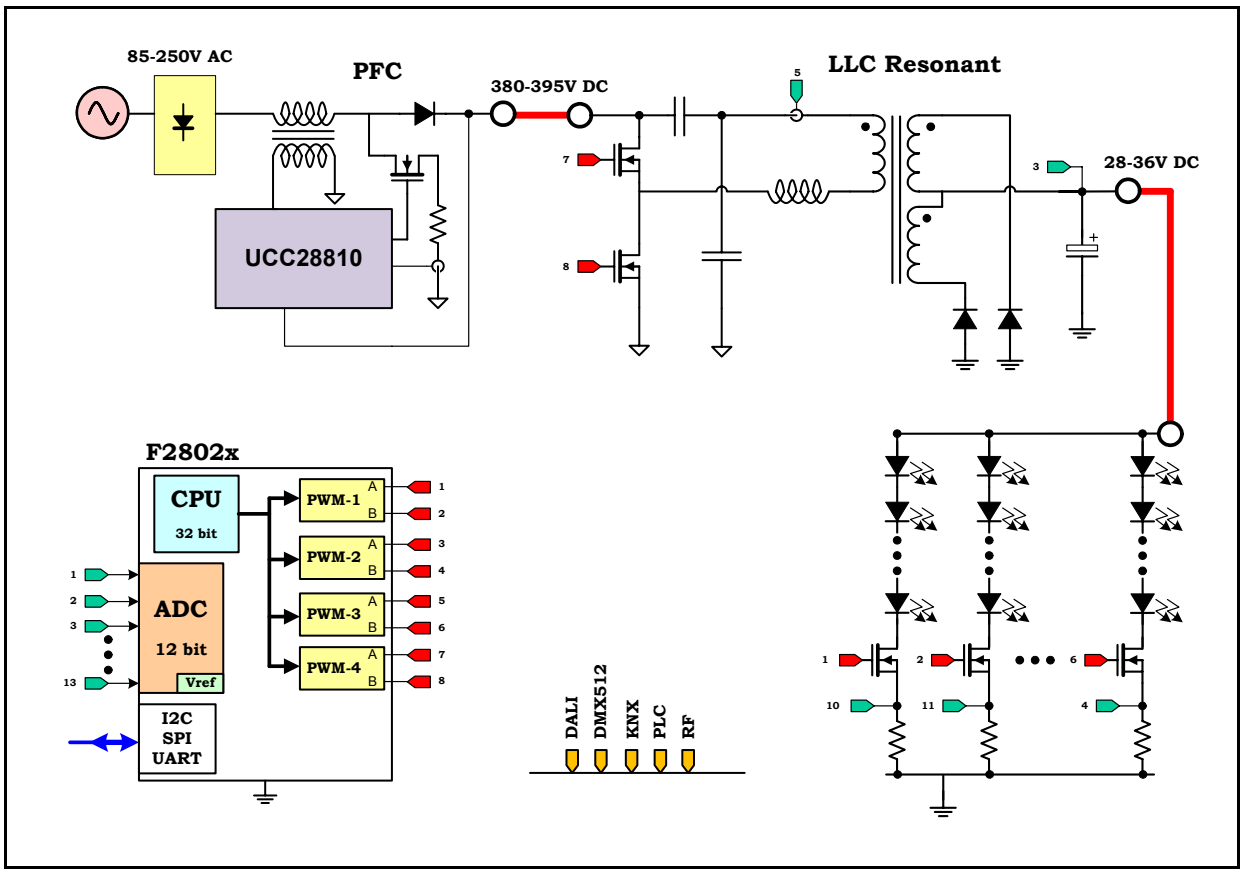



**Figure 8: AC LED Lighting and Communications board hardware block diagram**

The key signal connections between the F28027 microcontroller and the power stages located on the AC LED Lighting and Communications board are listed in the table below:

| Macro Name | | Signal Name | PWM Channel/ 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-Dimmming-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 |

**Table 1: Key Peripherals Used**

## 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 IsoACLighting-Main.c and IsoACLighting-ISR.asm files. Which build is used is set by the variable INCR_BUILD in IsoACLighting-Settings.h. Below 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

| | |
|---|---|
| Gui_Vout_LLC | The LLC Output Voltage (0-50V), for Instrumentation purpose only |
| Gui_Vset_LLC | The Voltage Reference for the LLC Output Voltage (0-50V). |
| 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 (0-0.20A), for Instrumentation purpose only |
| Gui_IsetLed[0-5] | This value determines the current reference for each LED string. (0-0.20A) |
| 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 1-6<br>• LEDs_linked = 1: Leds are all controlled via 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 therefore the on-time) of LED string 1-6. This duty cycle corresponds to a dimming control. |

**Table 2: Major Variables**

***Key Files Located in the Project***

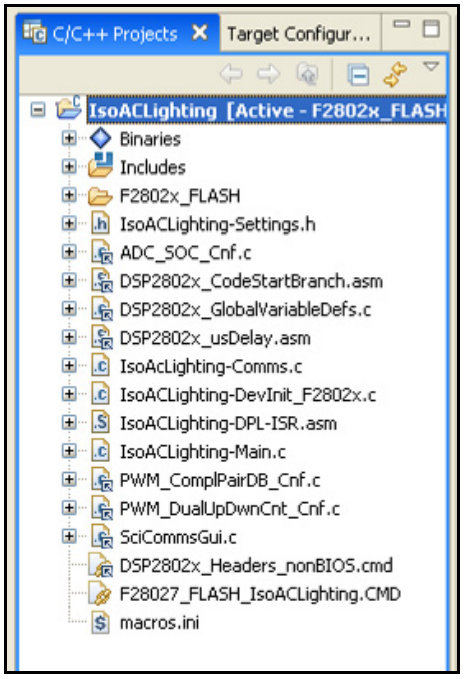



**Figure 9: IsoACLighting Project Files**

The key framework files used in this project are:

**IsoACLighting-Main.c** – this file is used to initialize, run, and manage the application. This is the "brains" behind the application.

**IsoACLighting-DevInit_F2802x.c** – this file is 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, etc.

**IsoACLighting-DPL-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 _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. It 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 which 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 – This file is used to set global definitions for the project (ie. build options). Note that it is linked to both IsoACLighting_Main.c and IsoACLighting-DPL-ISR.asm.

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

***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 |
|---|---|---|---|
| **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 |

**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 its role in the project.

**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 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. 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.

**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 & 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 2 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 degrees.

***The IsoACLighting project has the following properties:***

| 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)

| CPU Utilization of Build 2 of the IsoACLighting Project | |
|---|---|
| **Name of Modules *** | **Number of Cycles** |
| ISR Entry | 14-18 |
| Context Save, Restore, etc | 24 |
| Timeslicing Overhead | 34 |
| | |
| **LLC Control (updated every LED ISR)** | |
| ADCDRV_1ch | 5 |
| CNTL_2P2Z | 47 |
| | |
| **LED Control (2 strings updated every other LED ISR) **** | |
| ADCDRV_1ch * (2/2) | 5 |
| CNTL_2P2Z * (2/2) | 47 |
| PWMDRV_DualUpDwnCnt * (1/2) | 8 |
| | |
| **LLC Update ISR (runs at LLC frequency) ***** | |
| ISR Entry * 3 | 42-54 |
| ContextSave, Restore, etc * 3 | 72 |
| PWM_LLC_ComplPairDB * 3 | 48 |
| | |
| **Average Total Number of Cycles (loop rate = 62.5kHz)** | **362** |
| Average CPU Utilization @ 60 Mhz | **37.71%** |
| Average CPU Utilization @ 40 Mhz | **56.56%** |

* The modules are defined in the header files as “macros”

** Results for the LED Control subpart are shown as an average. In one LED ISR no cycles will be used for LED Control and PWM update, but in the next 120 will be used.

*** The LLC frequency is variable and therefore the worst case is shown (at no load an ISR happens three times in a 62.5kHz timeframe).

| System Features | |
|---|---|
| **Development /Emulation** | Code Composer Studio v4.1 (or above) with Real Time debugging |
| **Target Controller** | TMS320F2802x |
| **PWM Frequency** | 63.2-150kHz PWM (LLC); 31.25kHz PWM (LED sting) |
| **PWM Mode** | LLC – Asymmetric; LED – 6 Symmetric PWMs with phase shifts of 60° |
| **Interrupts** | • EPWM1 (LED PWM) counter equal to period or zero; drives controllers and LED PWM update<br>• EPWM4 (LLC PWM) counter equal to zero; drives LLC PWM update |

## 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 TMDSIACLEDCOMKIT Lighting board. See the TMDSIACLEDCOMKIT-HWGuide for a full description of connectors.

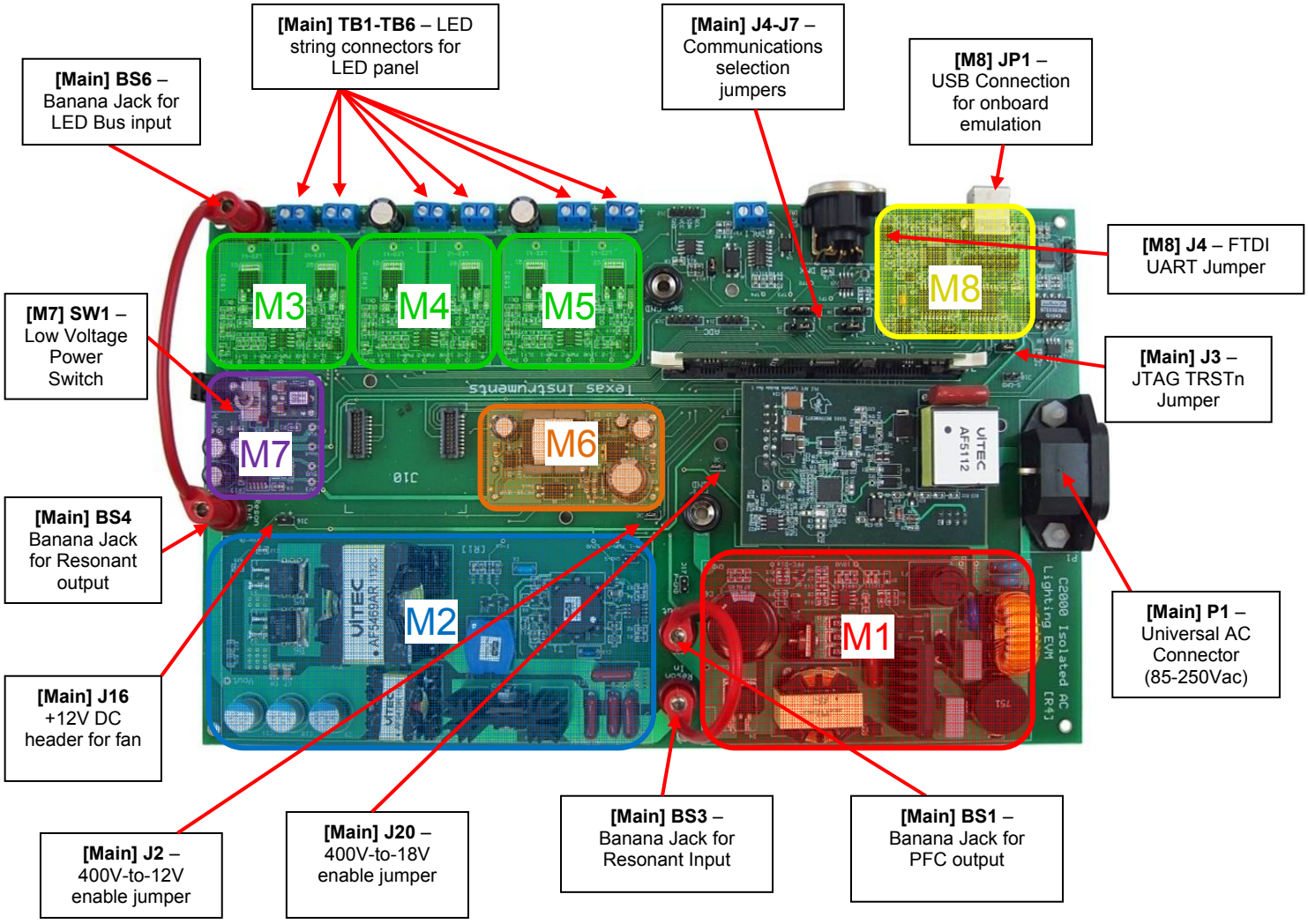
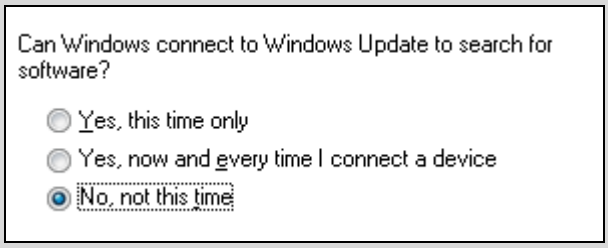



**Figure 10: Hardware Features**

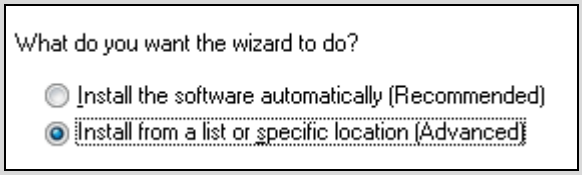
1) On the Piccolo F28027 controlCARD, check the following switches:
   - SW1, make sure position 1 and 2 are both in the “on” (up) position. This means the C2000 MCU is set to boot-to-flash.
   - SW2-SW4, all set such that all switches face downward (default setting)
   - There should be no jumper at 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 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.

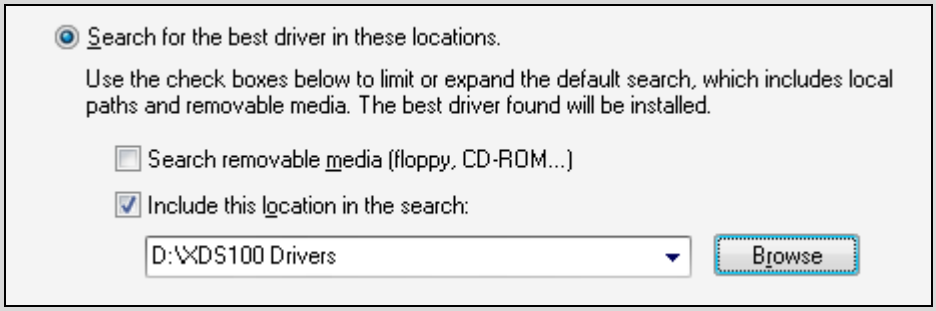
1) When Windows asks to search Windows Update, select “No, not at this time” and click Next




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 [Main]-TB1 to [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.
4) Connect or verify the following:
    - Switch [M7]-SW1 to the internal position (switched away from “Ext”)
    - Connect a jumper on [M8]-J4.
    - Connect a jumper on [Main]-J2.
    - Connect a jumper on [Main]-J20
    - Connect a jumper on [Main]-J3.
5) Connect the 12V power supply that was included in the kit to [M7]-JP1.
6) Connect the fan’s power cable to [Main]-J16. Connect the red wire toward “+”. (only needed if driving the total LED current above 1A (~35W))
7) Connet a red banana-to-banana cable between [Main]-BS4 and [Main]-BS6.

# Software Setup

### Installing Code Composer and controlSUITE

1) If not already installed, please install Code Composer 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.

### Setup Code Composer Studio to Work with the AC LED Lighting and Communications kit

3) Open “Code Composer Studio”.

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\ CCS_workspaces\IsoACLighting”
    - Uncheck the box that says “Use this as the default and do not ask again”.
    - Click “OK”

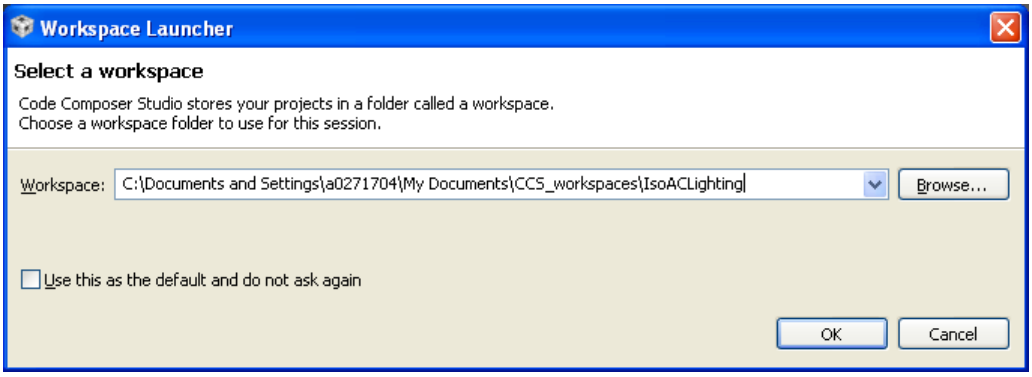



**Figure 11: Workspace Launcher**

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-f28027.ccxml. Make sure that the "Use shared location" checkbox is checked and then click Finish.

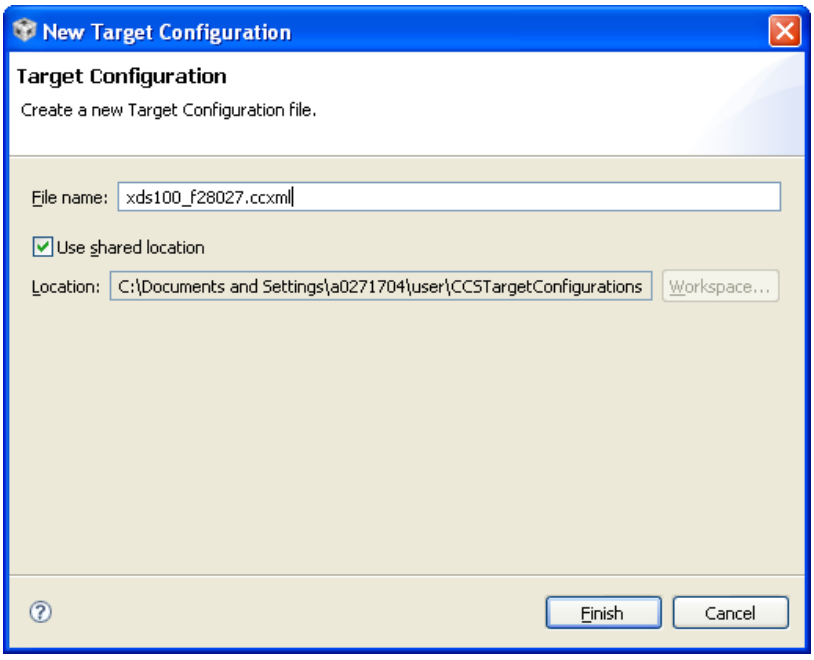



**Figure 12: Creating a Target Configuration**

6) This should open up a new tab as seen in Figure 12. Select and enter the options as shown:
   - Connection – Texas Instruments XDS100v1 USB Emulator
   - Device – TMS320F28027
   - Click Save
   - Close the xds100-f28027.ccxml tab

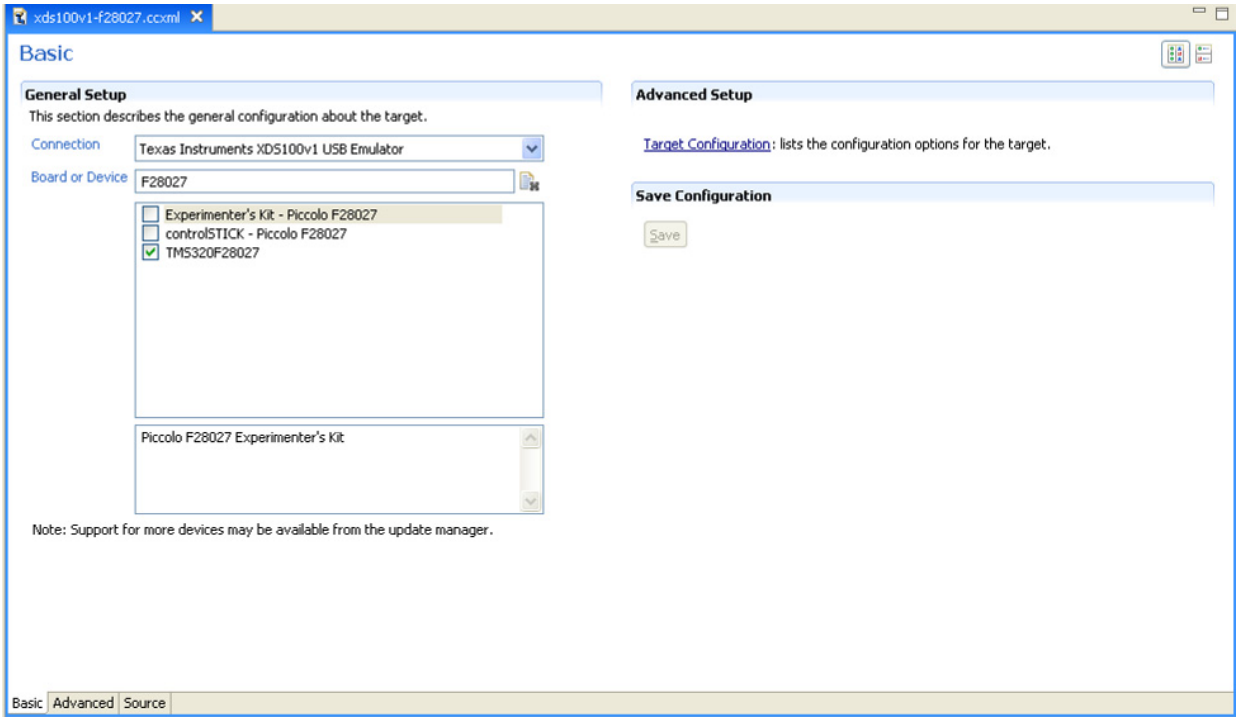



**Figure 13: Configuring a New Target**

7) Assuming this is your first time using Code Composer, the xds100-F28027 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-f28027.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 IsoACLighting project into your current workspace by clicking "Project->Import Existing CCS/CCE Eclipse Project". (If importing the project with CCSv5, check the box which allows CCS to "Automatically import referenced projects")

   - Select the root directory of the AC LED Lighting and Communications Developer's Kit. This will be:

     \TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X\IsoACLighting-F28027

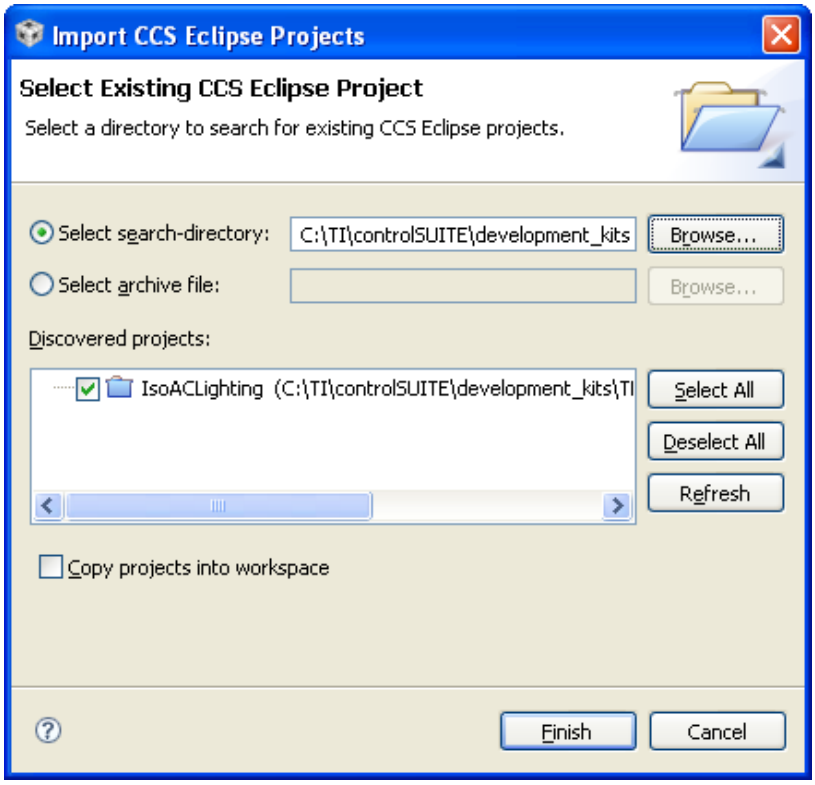



**Figure 14: Adding the IsoACLighting Project to your Workspace**

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

9) The IsoACLighting 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.

## 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 below.

**Build 1**: Open Loop Test, Check basic operation of the LLC Resonant and LED strings
**Build 2**: Closed Loop LLC Resonant and LED string control

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

The objective of this build is as follows:

- 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.

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

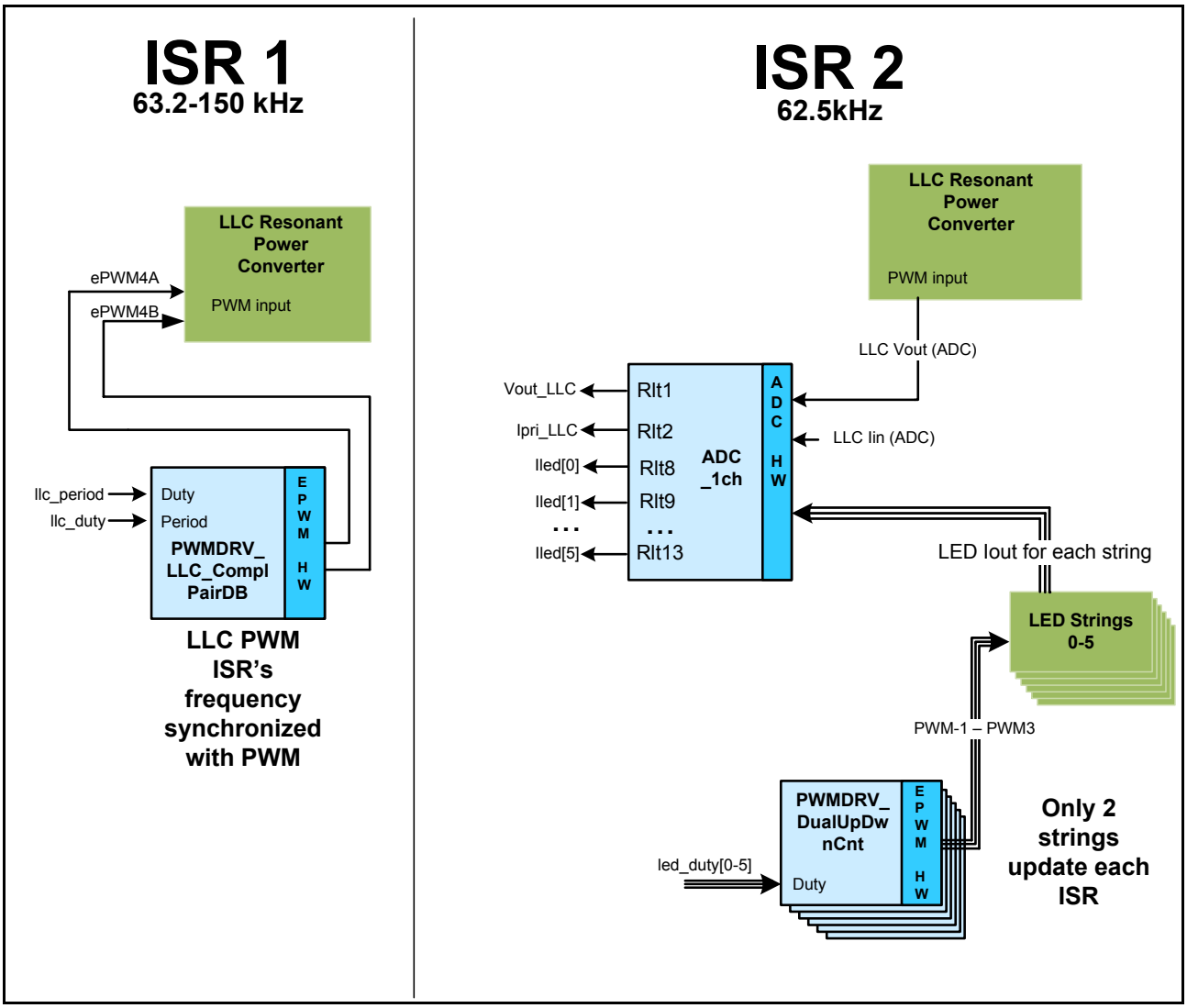



**Figure 15: Build Level 1 Block Diagram**

## Inspect the Project (Optional)

1. After initial boot processes are complete the software will begin 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 16. 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.

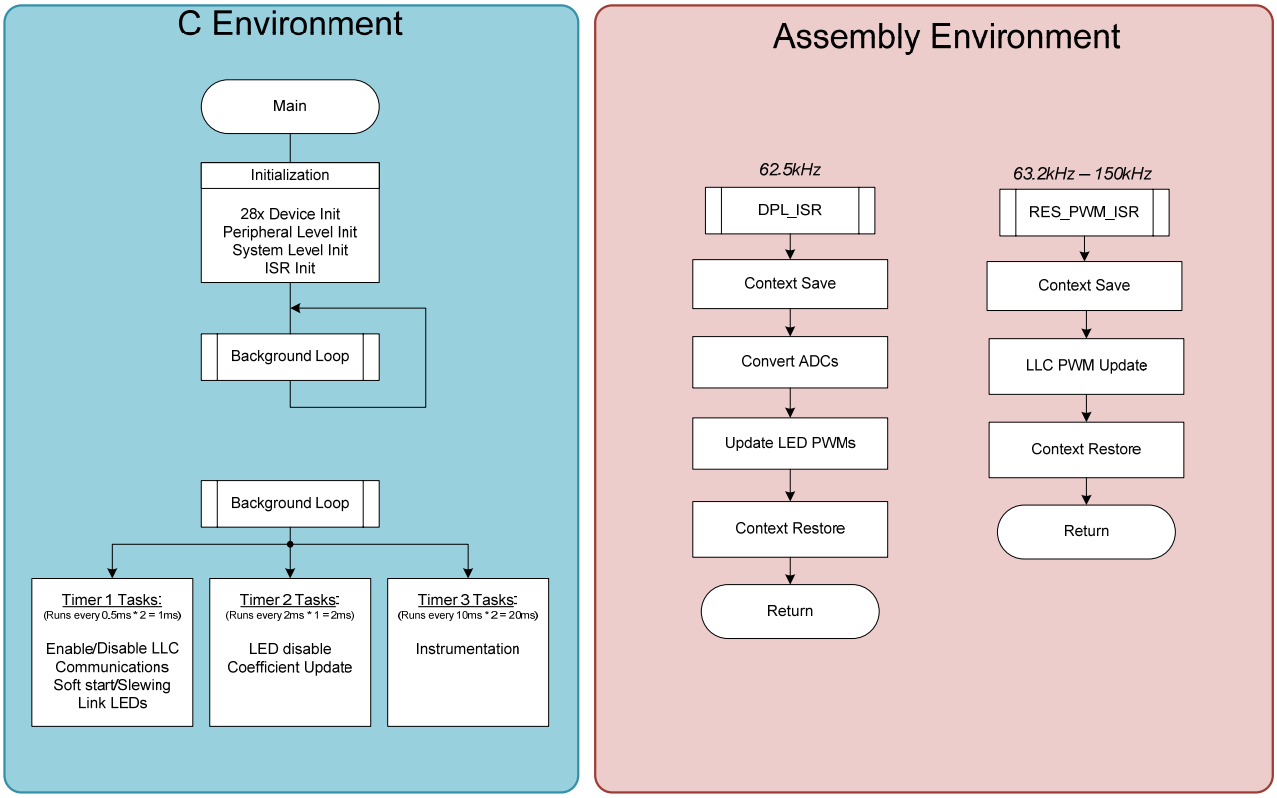



**Figure 16: Incremental Build 1 Software Flowchart**

2. The first thing that the software does in the main() function is call a function called DeviceInit() 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/disabled and the functional pinout, which configures which peripherals come out of which pins, is defined.
   - Confirm that the ADC and PWM1-4 peripheral clocks are enabled (lines 92-120). Also confirm that GPIO00-GPIO07 are configured to be PWM outputs (lines 122-184).
3. The IsoACLighting 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 IsoACLighting-Settings.h file 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. Note that the entire project must be recompiled if this variable is changed.

4. Go back to the IsoACLighting-Main.c file and futher inspect the software. In lines 322-399 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 degree shift between the 2 controlled strings. Each PWM module is then phase shifted by 60 degrees so that power is distributed more evenly within a cycle.

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

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

    - Notice 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 PWM1 SOCA trigger is defined to happen when the PWM1 timer reaches its period value (line 473-475). Other ADC channels are configured similarly.

6. In lines 554-651 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.

    - Notice that ADCDRV_1ch_Rlt8 is attached to the Iled[0] variable and that PWMDRV_DualUpDwnCnt_Duty1A is attached to the 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/write to. As defined, the ADCDRV_1ch macro will read from the ADC peripheral and write to Iled[0] and the PWMDRV_DualUpDwnCnt macro will read 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 the **Hardware Setup** and **Software Setup** sections from the previous pages have been completed. If not already done, please go through these sections before continuing!

### Build and Load the Project

1. From the C/C++ Project tab, open the file IsoACLighting-Settings.h file 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. Note that 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. 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.

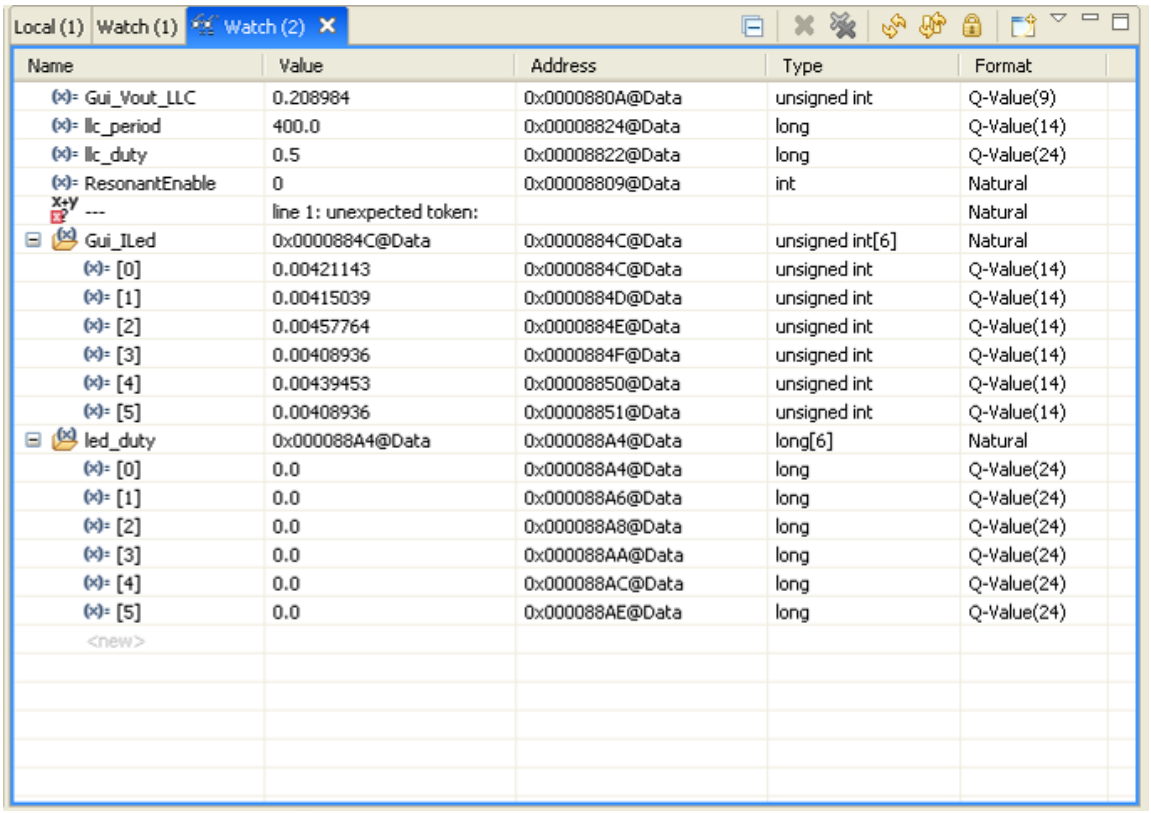
5. Now click the real-time mode  button that says “Enable silicon real-time mode”. This will allow the user to edit and view variables in real-time without halting the program.

6. 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

7. 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_Vout_LLC | 0.208984 | 0x0000880A@Data | unsigned int | Q-Value(9) |
| llc_period | 400.0 | 0x00008824@Data | long | Q-Value(14) |
| llc_duty | 0.5 | 0x00008822@Data | long | Q-Value(24) |
| ResonantEnable | 0 | 0x00008809@Data | int | Natural |
| --- | line 1: unexpected token: | | | Natural |
| Gui_ILed | 0x0000884C@Data | 0x0000884C@Data | unsigned int[6] | Natural |
| [0] | 0.00421143 | 0x0000884C@Data | unsigned int | Q-Value(14) |
| [1] | 0.00415039 | 0x0000884D@Data | unsigned int | Q-Value(14) |
| [2] | 0.00457764 | 0x0000884E@Data | unsigned int | Q-Value(14) |
| [3] | 0.00408936 | 0x0000884F@Data | unsigned int | Q-Value(14) |
| [4] | 0.00439453 | 0x00008850@Data | unsigned int | Q-Value(14) |
| [5] | 0.00408936 | 0x00008851@Data | unsigned int | Q-Value(14) |
| led_duty | 0x000088A4@Data | 0x000088A4@Data | long[6] | Natural |
| [0] | 0.0 | 0x000088A4@Data | long | Q-Value(24) |
| [1] | 0.0 | 0x000088A6@Data | long | Q-Value(24) |
| [2] | 0.0 | 0x000088A8@Data | long | Q-Value(24) |
| [3] | 0.0 | 0x000088AA@Data | long | Q-Value(24) |
| [4] | 0.0 | 0x000088AC@Data | long | Q-Value(24) |
| [5] | 0.0 | 0x000088AE@Data | long | Q-Value(24) |
| <new> | | | | |

**Figure 17: Configuring the Watch Window for Build 1**

8. Click on the Continuous Refresh button in each watch window. This enables the window to run with real-time mode which means that the variables will change with the software in real-time.

## Additional Hardware Setup Needed for this Build

***This set of instructions assumes that the Hardware Setup section has been done.***

9. 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.
10. Make sure that there is **NOT** a banana-to-banana cable connecting [Main]-BS1 and [Main]-BS3.
11. Connect a multimeter between the LLC ouput ([Main]-BS4) and secondary ground ([Main]-BS5)
12. Attach an appropriate DC load between the LLC output ([Main]-BS4) and the secondary ground ([Main]-BS5). About 25-50Ohm resistive load should work fine. (or 0.5A-1A load)
13. Connect an **unpowered** DC supply capable of delivering 390V between [Main]-BS1 (+) and [Main]-BS2 (-). Set the DC supply to have a current limit of 0.25A. This will allow only 100W to be delivered to the load.

**NOTE**: For safety reasons, it is recommended to use an isolated DC supply to supply 400V DC to the board.

## Run the Code

14. Run the code by pressing Run Button  in the Debug Tab.
15. 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.
16. Change `ResonantEnable` to 1. You should now see the below waveform from the LLC PWMs. Note that a `llc_period` of 550 means that the frequency is 60000000/(550) = ~109kHz . 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.

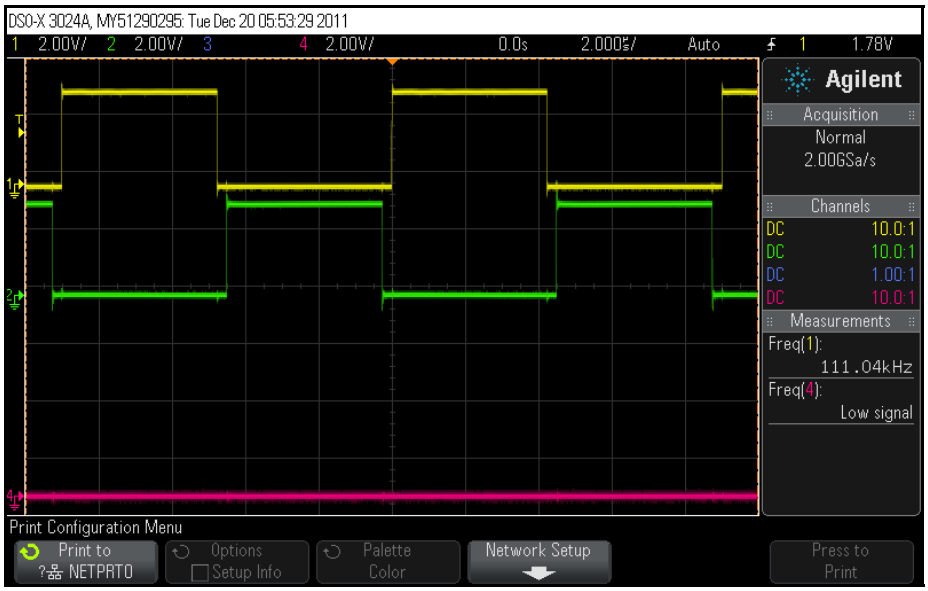



**Figure 18: LLC PWM Waveform at 110kHz**

17. Edit the `llc_period` to 700. The LLC PWM's frequency should have decreased to approximately 85.7kHz.

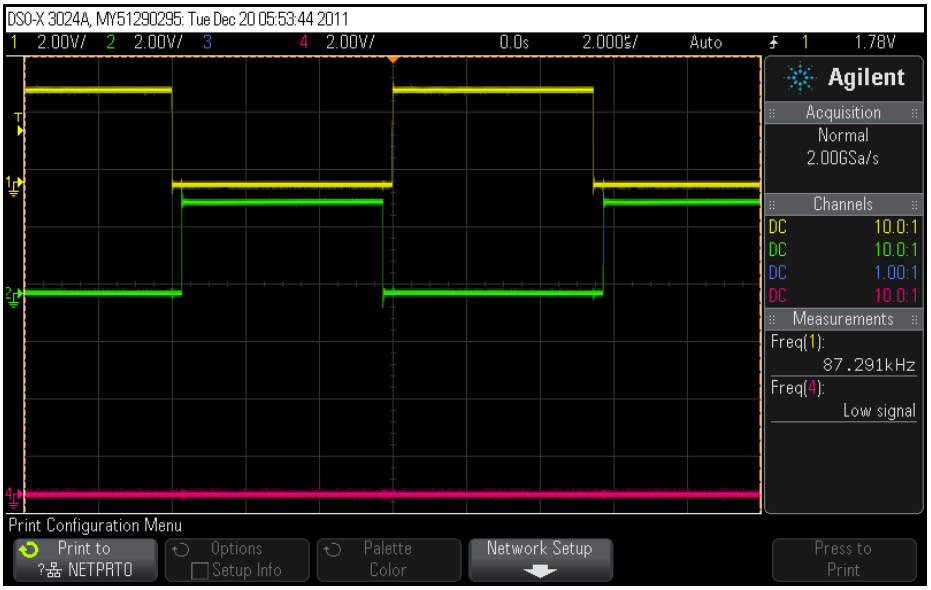



**Figure 19: LLC PWM Waveforms at 85.7kHz**

18. Now set `led_duty[0]` to 0.1. This will 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 31.25kHz signal that has 10% duty cycle.

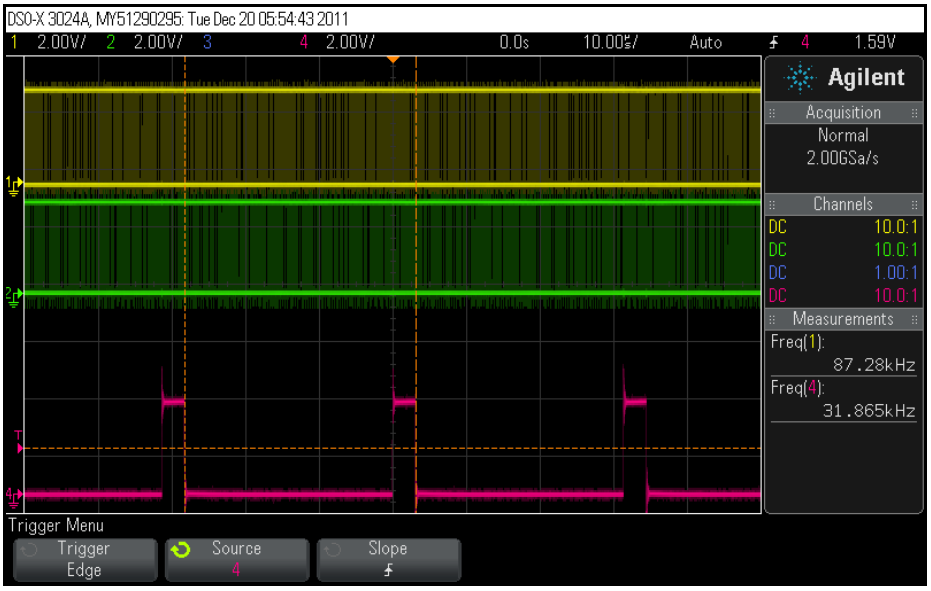



**Figure 20: LED0 PWM Waveform at 10% Duty Cycle**

19. Set `llc_period` to 400
20. Change `ResonantEnable` to 0
21. Set `led_duty` to 0.0

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

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

24. If the LLC output is less than 34VDC, increase `llc_period` in steps of 10-25 until the LLC Resonant output is about 33-34V. 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.

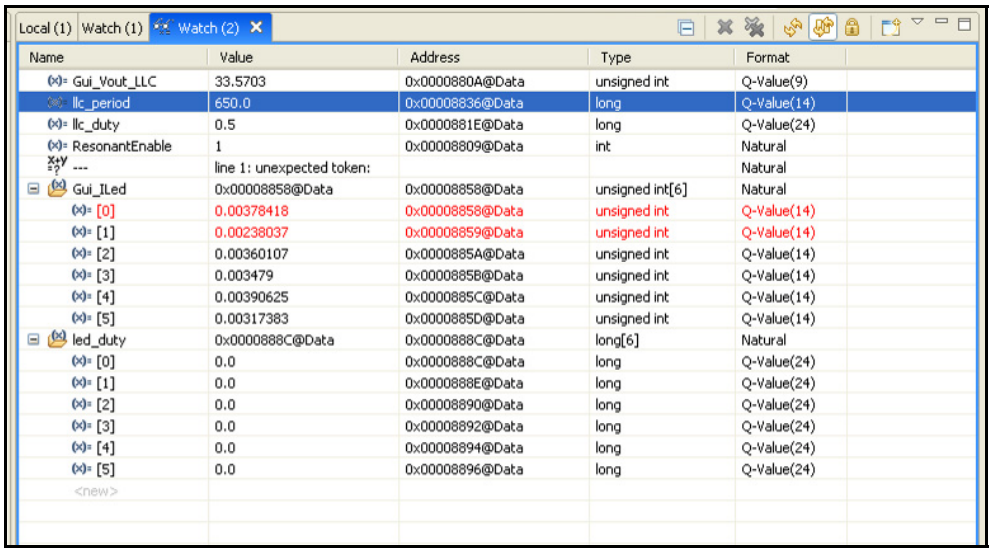



**Figure 21: Watch Window - Changing llc_period**

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

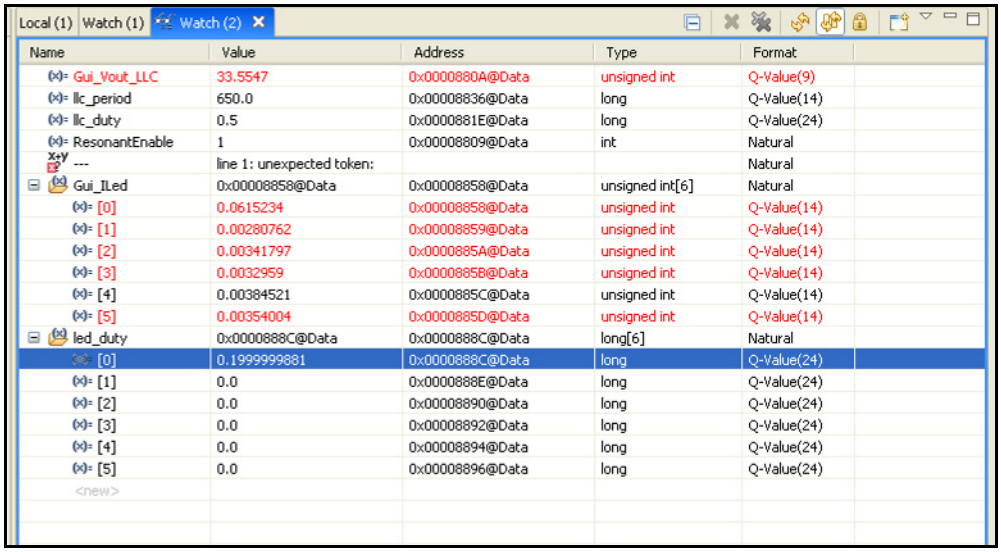



**Figure 22: Watch Window - Changing led_duty**

26. LED string 1-5 can also be checked individual 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 will decrease to the extent that it no longer surpasses the LED strings' threshold voltage.
27. Once complete, set `led_duty[0]`-`led_duty[5]` to 0.0 and then `llc_period` to 400.
28. Turn off the 400V DC Supply.
29. Set `ResonantEnable` to 0.
30. Halt the processor 
31. Stop Real-time mode by clicking 
32. 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 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 the diagram below:

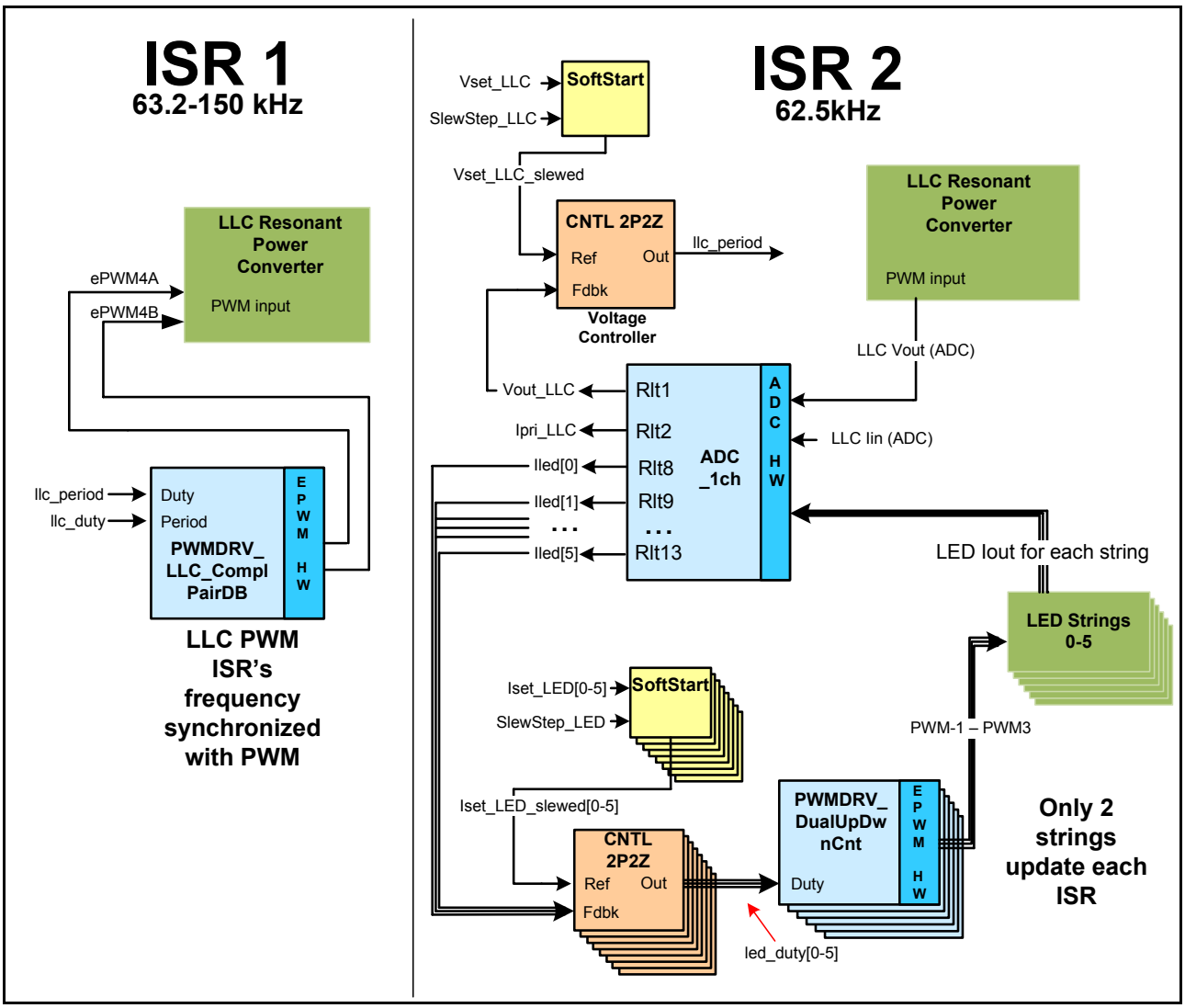



**Figure 23: Build Level 2 Block Diagram**

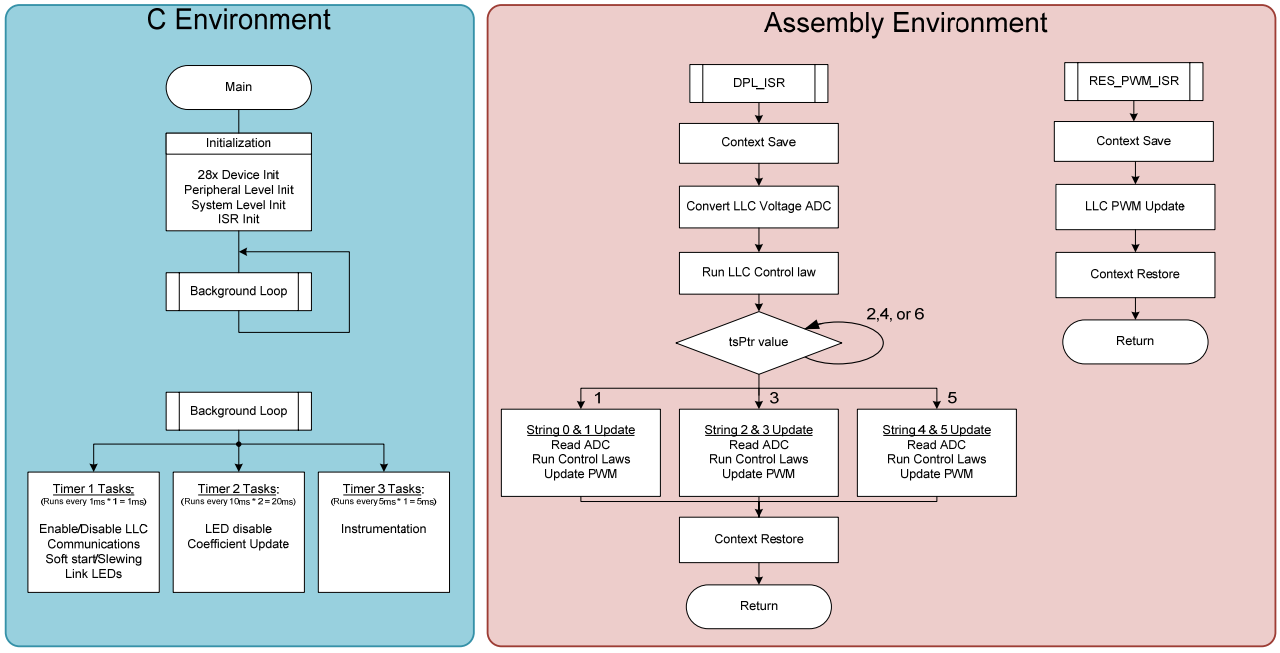



**Figure 24: Incremental Build 2 Software Flowchart**

## Additional Hardware Setup Needed for this Build

***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. It is recommended to run build level 1 prior to running build level 2.***

***This set of instructions assumes that the Hardware Setup section has been done.***

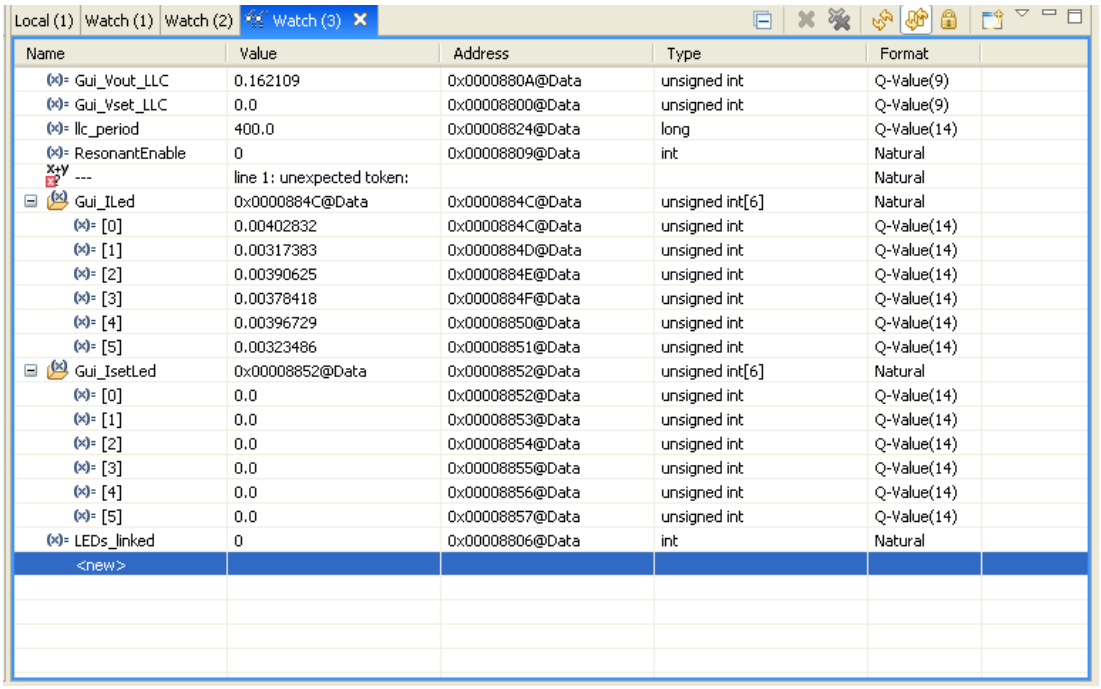
1. Connect a banana-to-banana cable between [Main]-BS1 and [Main]-BS3.
2. Connect a second banana-to-banana cable between [Main]-BS-BS4 and [Main]-BS6
3. Plug one side of the AC line cable to [Main]-P1. **DO NOT CONNECT THE OTHER END OF THE CABLE YET.**

## Run the Code

4. Flip [M7]-SW1 to the “Ext” position if not already done.
5. Open IsoACLighting-Settings.h and change the incremental build level to 2 (#define INCR_BUILD 2). Save the file.
6. Right-click on the project name and select “Rebuild Project”.
7. 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.

8. 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.

9. 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 LLC power stage and LED dimming stages. Also, set this new watch window to continuously refresh.




| Name | Value | Address | Type | Format |
|---|---|---|---|---|
| Gui_Vout_LLC | 0.162109 | 0x0000880A@Data | unsigned int | Q-Value(9) |
| Gui_Vset_LLC | 0.0 | 0x00008800@Data | unsigned int | Q-Value(9) |
| llc_period | 400.0 | 0x00008824@Data | long | Q-Value(14) |
| ResonantEnable | 0 | 0x00008809@Data | int | Natural |
| --- | line 1: unexpected token: | | | Natural |
| Gui_ILed | 0x0000884C@Data | 0x0000884C@Data | unsigned int[6] | Natural |
| [0] | 0.00402832 | 0x0000884C@Data | unsigned int | Q-Value(14) |
| [1] | 0.00317383 | 0x0000884D@Data | unsigned int | Q-Value(14) |
| [2] | 0.00390625 | 0x0000884E@Data | unsigned int | Q-Value(14) |
| [3] | 0.00378418 | 0x0000884F@Data | unsigned int | Q-Value(14) |
| [4] | 0.00396729 | 0x00008850@Data | unsigned int | Q-Value(14) |
| [5] | 0.00323486 | 0x00008851@Data | unsigned int | Q-Value(14) |
| Gui_IsetLed | 0x00008852@Data | 0x00008852@Data | unsigned int[6] | Natural |
| [0] | 0.0 | 0x00008852@Data | unsigned int | Q-Value(14) |
| [1] | 0.0 | 0x00008853@Data | unsigned int | Q-Value(14) |
| [2] | 0.0 | 0x00008854@Data | unsigned int | Q-Value(14) |
| [3] | 0.0 | 0x00008855@Data | unsigned int | Q-Value(14) |
| [4] | 0.0 | 0x00008856@Data | unsigned int | Q-Value(14) |
| [5] | 0.0 | 0x00008857@Data | unsigned int | Q-Value(14) |
| LEDs_linked | 0 | 0x00008806@Data | int | Natural |
| <new> | | | | |

**Figure 25: Configuring the Watch Window for Build 2**

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

11. 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.

12. Plug the other end of the AC cable into the wall. (or preferably, plug it into a power strip/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.**

13. In the Watch Window, set ResonantEnable to 1.

14. Set `Gui_Vset_LLC` to 34.5 which corresponds to setting the output reference to 34.5V. This variable is used as the reference to the LLC controller. The controller will edit the duty cycle as necessary to keep `Gui_Vout_LLC`, visible in Watch Window, at 34.5V.

**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.**

15. Set the variable `Gui_IsetLed[0]` to 0.1, which corresponds to 0.1A. As the current reference is edited notice that `Gui_ILed[0]` (in Watch window 1) tries to match the reference.
16. Now change the value of `Gui_IsetLed[0]` to 0.25A. Notice that the LED string brightness has increased with the increased current.
17. Continue editing `Gui_IsetLed[0]` and also `Gui_IsetLed[1-5]` with various values from 0.0 to 0.3A.

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

18. 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[0].

19. Once complete, set `Gui_IsetLed[0-5]` to 0.0A
20. Set `Gui_Vset_LLC` to 0.0V.
21. Set `ResonantEnable` to 0
22. Disconnect the AC cable from the wall. The board should be considered live for at least 20 seconds after the AC cable is disconnected.
23. Halt the processor 
24. Stop Real-time mode by clicking 
25. 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

- **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.

- **$I^2C$ 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.

- **$I^2C$ 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 TMDSIACLEDCOMKIT 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:

- **TMDSIACLEDCOMKIT-QSG –** a guide that allows a user to quickly demonstrate the capabilities of the IACLEDCOMKIT

  *C:\TI\controlSUITE\development_kits\TMDSIACLEDCOMKIT_vX.X\~Docs\TMDSIACLEDCOMKIT-QSG.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, etc)

  *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*