TI Designs

Two-Phase Interleaved PFC Converter w/ Power Metering Test Results

TI Designs

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize and system. TI Designs help you accelerate your time to market.

Design Resources

ILPFC-Converter Design Files	Tool Folder Containing Design Files
F28xxx User's Guides	Product Folder

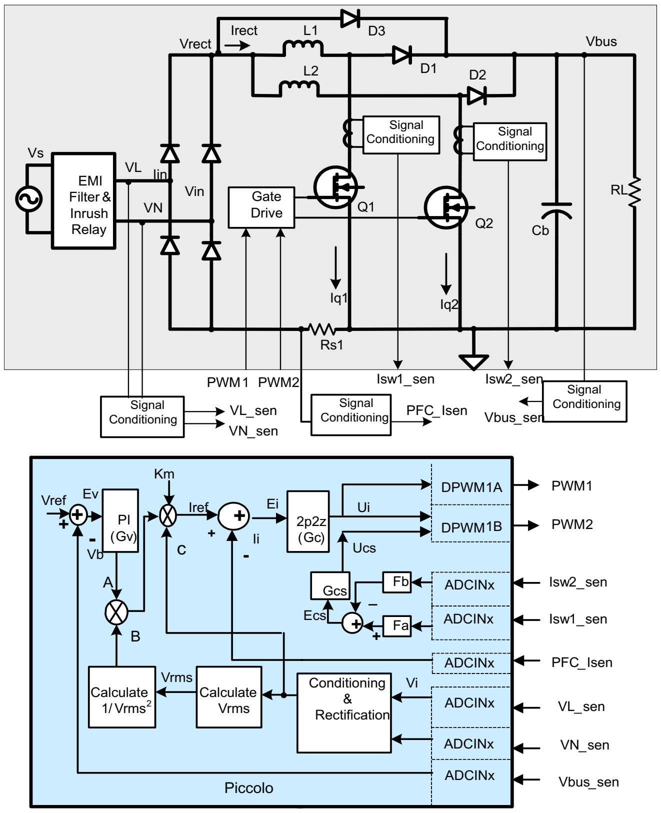


Figure 1. Interleaved PFC Converter Control Using the C2000 Microcontroller

Design Features

- 2 Phase Interleaved PFC Hardware and Software
  - 95~260 Vrms, 47~63Hz, 600W/750W
- Full Digital Control Loops
  - 200kHz Switching Frequency
- THD 1.5%, PF 0.99, HL Light Load - 0.95 (Min)
- Integrated Frequency Response Analyzer (FRA)

Featured Applications

- Fast Vin Feed Forward
- Improved THD with Adaptive Current Loop, Harmonic Compensation, and Oversampling
- Faster Voltage Loop with Non-Linear Control and Notch Filter
- Over-Voltage Protection
- Input RMS Current, Voltage, Power, and Frequency Measurement
- Zero Cross Detect, +ve/-ve Half-Cycle Detect

ASK Our E2E Experts
WebBench Calculator Tools

1 System Description

This document presents the implementation details of a digitally controlled 2-Phase Interleaved Power Factor Correction (ILPFC) converter. A C2000 Piccolo-B control card and a 700W ILPFC EVM are used to implement the complete system.

With various regulations limiting the input current harmonic content, especially with the IEC 61000-3-2 standard that defines the harmonic components that an electronic load may inject into the supply line, a power factor correction (PFC) stage has become an integral part of most rectifier designs. The PFC stage usually forms the front end of an isolated ac-dc rectifier system as shown in Figure 2.

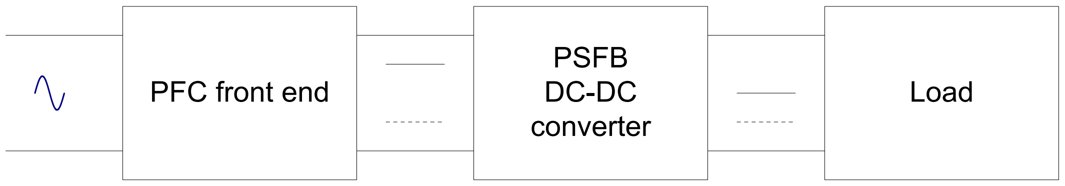


Figure 2. Isolated AC-DC Rectifier Block Diagram

The PFC converter provides power to non-linear loads from the AC mains while maintaining AC input current of the same wave-shape and phase of that of the AC mains voltage. At the same time, the PFC converter regulates its output DC voltage in order to provide a regulated high voltage bus to any downstream DC-DC converter connected to its output. The downstream DC-DC converter is usually a phase shifted full bridge (PSFB) converter which converts the high DC bus voltage from the PFC stage to a lower voltage such as, +12V, or, an intermediate distribution voltage, typically closer to 48V. The phase shifted full bridge (PSFB) stage provides the desired voltage translation and the high frequency isolation for this offline rectifier system. This document focuses on the implementation detail of the PFC stage. Specifically, it presents the hardware design and the corresponding software to control a 2-phase interleaved power factor correction (ILPFC) front end.

This PFC EVM comes with a Piccolo-B control card. However, the controller resources used for the PFC implementation shows that Piccolo EL can also be used to implement full control of the PFC stage.

2 Introduction

The function of a PFC stage is to convert the AC mains voltage to a regulated DC bus voltage while drawing a sine wave input current in phase with the AC input voltage. Typically, this is implemented using a bridge rectifier followed by a boost PFC stage. A C2000 piccolo microcontroller with its on-chip PWM and ADC modules is able to implement complete digital control of such interleaved PFC (ILPFC) system.

2.1 PFC Stage Implementation

Figure 3 illustrates a C2000 controller based interleaved PFC converter control system. The input AC voltage is applied to the PFC converter through the input EMI filter, followed by an inrush current limit and a bridge rectifier. The PFC stage consists of two interleaved boost converters each operating at 200kHz and phase shifted by 180 deg.

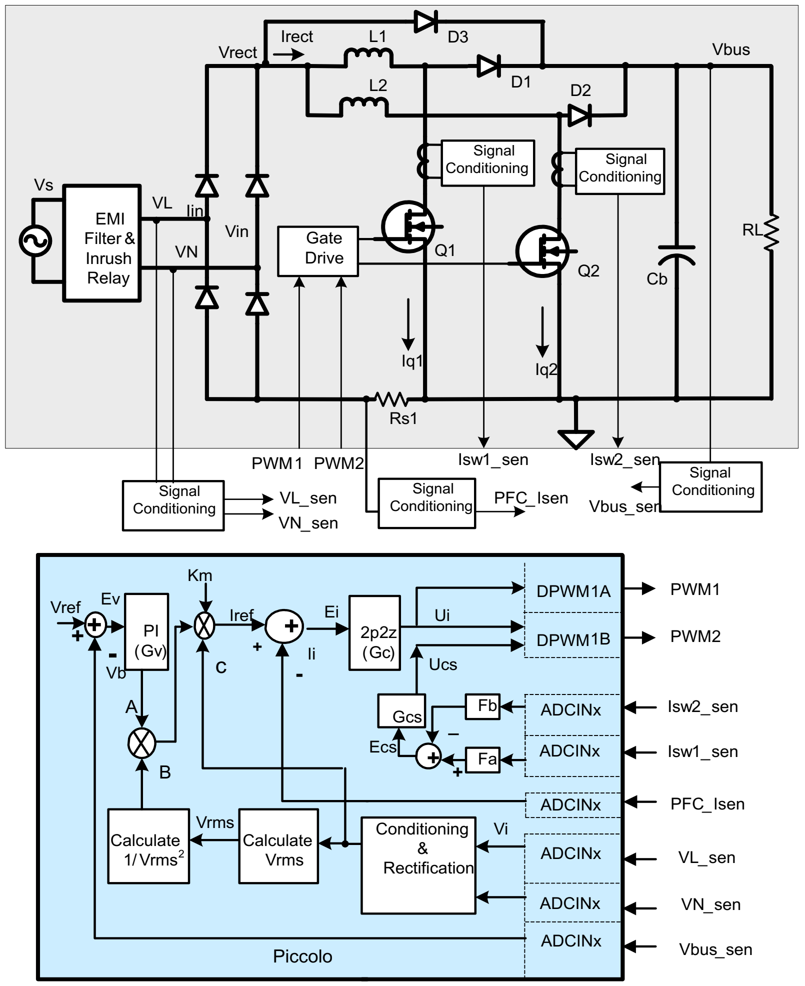


Figure 3. Interleaved PFC Converter Control Using the C2000 Microcontroller

Inductor L1, MOSFET switch Q1 and diode D1 together form one of the boost stages while, L2, Q2, and D2 form the other boost stage. A capacitor Cb at the boost converter output acts as an energy reservoir and this, in conjunction with closed loop PFC control, provides regulated dc voltage to the PFC load RL.

Figure 3 indicates all the interface signals needed for full control of this interleaved PFC converter using a C2000 micro-controller (MCU). The MCU controls the hardware using four feedback signals and two PWM outputs. The signals that are sensed and fed back to the MCU include, the line and neutral voltages (Vin_L & Vin_N), the PFC input current(Irect), and the DC bus output voltage (Vbus). These sensed signals are used to implement the voltage and current control loops for this IL PFC converter. For phase current balancing two PFC switch currents (Isw1, Isw2) can also be monitored. However, this feature is not implemented in this EVM

The dc bus voltage Vbus, sensed through one of the ADC channels, is compared against the reference bus voltage Vref. The error signal Ev is input to the voltage loop controller Gv which regulates the bus voltage at the reference level so as minimize Ev. The voltage controller Gv has the form of a two pole two zero (2P2Z) compensator. The output of Gv, denoted by the letter A in Figure 3, is proportional to the amount of power transfer by the PFC converter. This output A is then multiplied by three parameters, indicated by B, C and Km in Figure 3, in order to form the reference current command Iref for the PFC current control loop. The signal indicated by B is the inverse of the square of the RMS input voltage which also enables fast feed-forward control of the PFC system. The signal C is proportional to the rectified input voltage, which modulates the voltage controller output A such that the PFC input current has the same shape as that of the PFC input voltage. The parameter Km is the multiplier gain which is used to adjust the range of Iref corresponding to the full input voltage range of the PFC converter. The output of the multiplier block provides the reference signal Iref that is used for control of the total average inductor current, i.e., the PFC input current. This reference current command Iref for the PFC current control loop is compared against the sensed PFC input current Ii sensed through one ADC channel. The resulting current error signal Ei is then input to the current loop controller Gc which generates the PFC duty ratio command d such that the PFC input current tracks the reference current Iref.

In addition to implementing the voltage and current loop controllers, C2000 MCU also uses the sensed line and neutral voltage signals to determine the polarity of the input voltage (+ve & –ve half cycle) and calculates the rectified input voltage, the RMS input voltage, RMS input power and the input line frequency. All these time critical functions are implemented in a fast sampling loop enabled by the C2000 Micro-controller high speed CPU, interrupts, on chip 12-bit ADC module and high frequency PWM modules. A detailed description of the software algorithm is provided in the following chapters.

2.2 IL PFC Electrical Specifications

Following lists the key highlights of the C2000 IL PFC EVM:

- Input Voltage (AC Line): 100V (Min) to 260V (Max), 47~63Hz
- Rated Output Voltage 390Vdc
- Rated Output Power 700 Watts @220V input, 550W@110V input
- Full Load efficiency: 97% @220V input
- Power factor at 50% or greater load – 0.98 (Min)
- Input Power Monitoring
- PWM frequency 200kHz

3 Software Overview

3.1 Software Control Flow

The CCS project for C2000 ILPFC mostly makes use of the “C-background/ASM-ISR” framework. The main fast ISR (100kHz) runs in assembly environment. A slower ISR (10kHz) is also run from C environment. This slow ISR is made interruptible by the fast ISR.

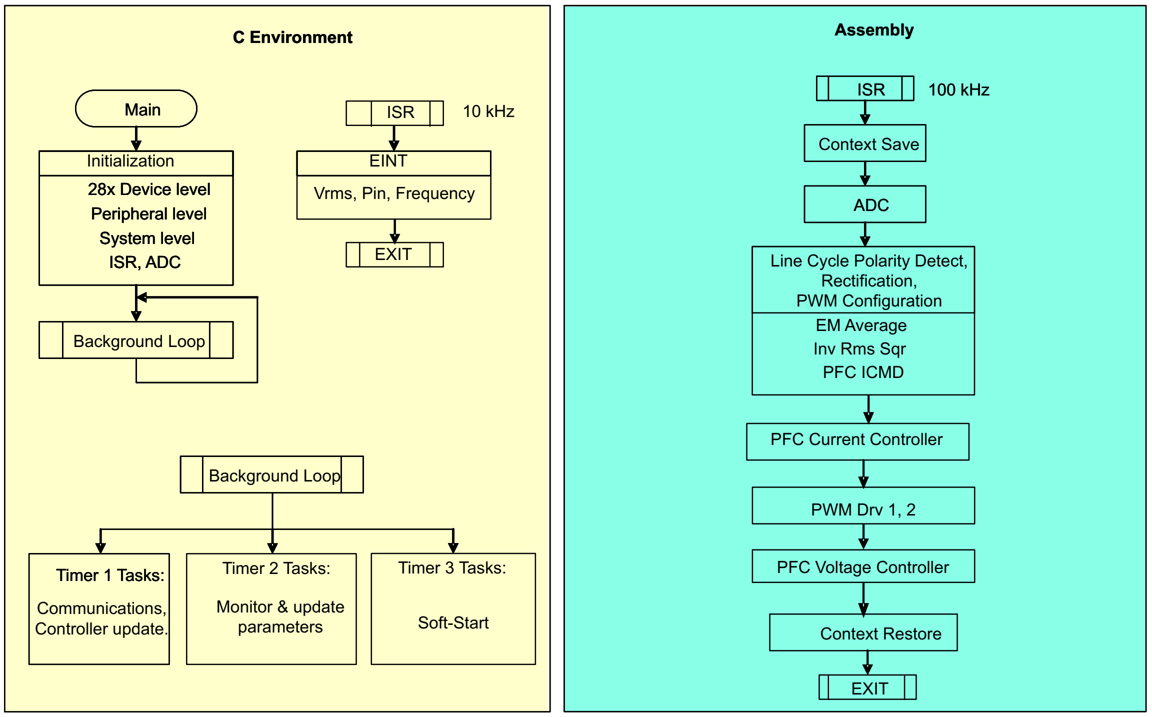


Figure 4. IL PFC Software Flow Diagram

The CCS project uses C-code as the main supporting program for the application, and is responsible for all system management tasks, monitoring, decision making, intelligence, and host interaction. The assembly code is strictly limited to the fast Interrupt Service Routine (ISR), which runs all the critical control code. Typically this includes reading ADC values, input line cycle polarity detect, sensed line volt rectification, control calculations, and PWM updates. The slower ISR in the C environment calculates the RMS input voltage, RMS input current, RMS input power and frequency of the input line voltage. Figure 4 depicts the general software flow for this project.

The key framework C files used in this project are:

InterleavedPFC-Main.c – this file is used to initialize, run, and manage the application. *InterleavedPFC-DevInit_F2803x.c* – this file is used for 2803x controller initialization. A 2803x control card is provided with the IL PFC EVM. This file is responsible for a one time initialization and configuration of the F280xx device, and includes functions such as setting up the clocks, PLL, GPIO, etc.

The fast ISR consists of a single file:

InterleavedPFC-DPL-ISR.asm – this file contains all time critical “control type” code. This file has an initialization section (one time execute) and a run-time section which executes at half the rate (100kHz) as the PWM time-base(200kHz) used to trigger it.

The slow ISR consists of a single file:

SineAnalyzer.h – this file contains code for calculating the RMS voltage, RMS input current, RMS input power and frequency of the input line voltage. This file has an initialization section (one time execute) and a run-time section which executes at 10kHz rate.

The Power Library functions (modules) are “called” from the fast ISR framework.

Library modules may have both a C and an assembly component. In this project, seven library modules are used. The C and corresponding assembly module names are:

Table 1. Library Modules

C configure function	ASM initialization macro	ASM run-time macro
PWM_2ch_UpDwnCnt_Cnf.c	PWMDRV_2ch_UpDwnCnt_INIT n	PWMDRV_2ch_UpDwnCnt n
ADC_SOC_Cnf.c	ADCDRV_1ch_INIT m,n,p,q	ADCDRV_1ch m,n,p,q
	PFC_InvRmsSqr_INIT n	PFC_ InvRmsSqr n
	MATH_EMAVG_INIT n	MATH_EMAVG n
	PFC_ICMD_INIT n	PFC_ICMD n
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

The modules can also be represented graphically as shown in Figure 5.

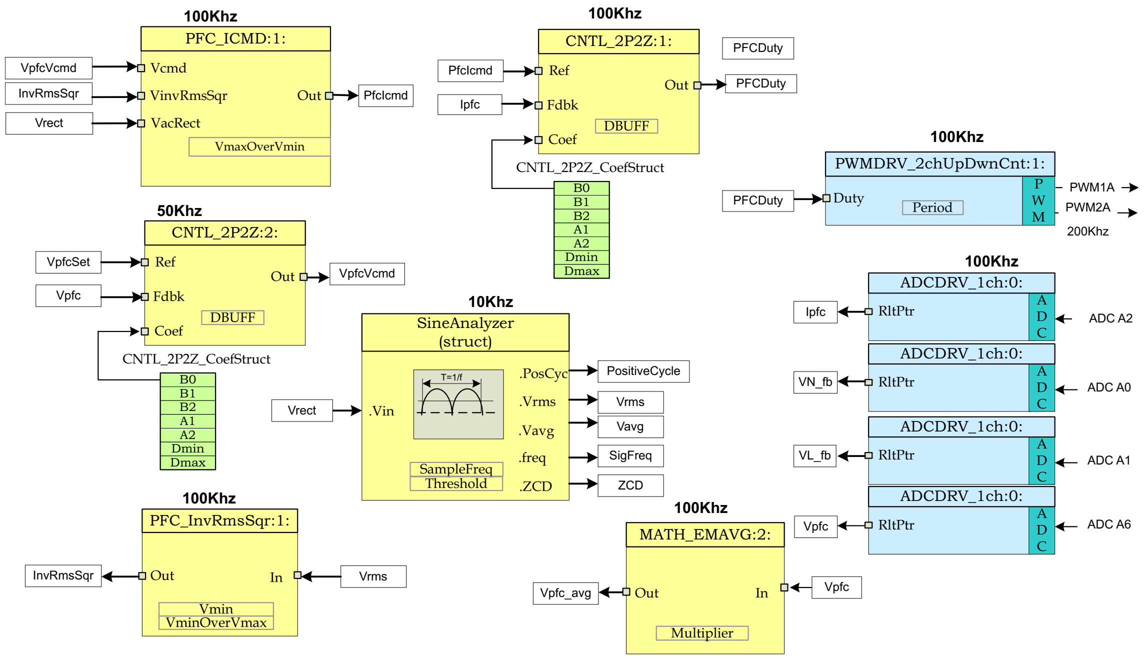


Figure 5. C2000 ILPFC Software Modules

NOTE: Note the color coding used for the modules in Figure 5. The blocks in 'dark blue' represent the on-chip hardware modules in C2000 controller. The blocks in 'blue' are the software drivers associated with these modules. The blocks in 'yellow' are part of the computation carried out on various signals. The controllers used for voltage and current loops have the form of a 2-pole 2-zero compensator. However these can be of other forms such as, PI, PID, 3-pole 3-zero or any other controller suitable for the application. The modular library structure makes it convenient to visualize and understand the complete system software flow as shown in Figure 5. It also allows for easy use and modifications of various functionalities. This fact is amply demonstrated in this project by implementing an incremental build approach. This is discussed in more detail in Section 3.2.

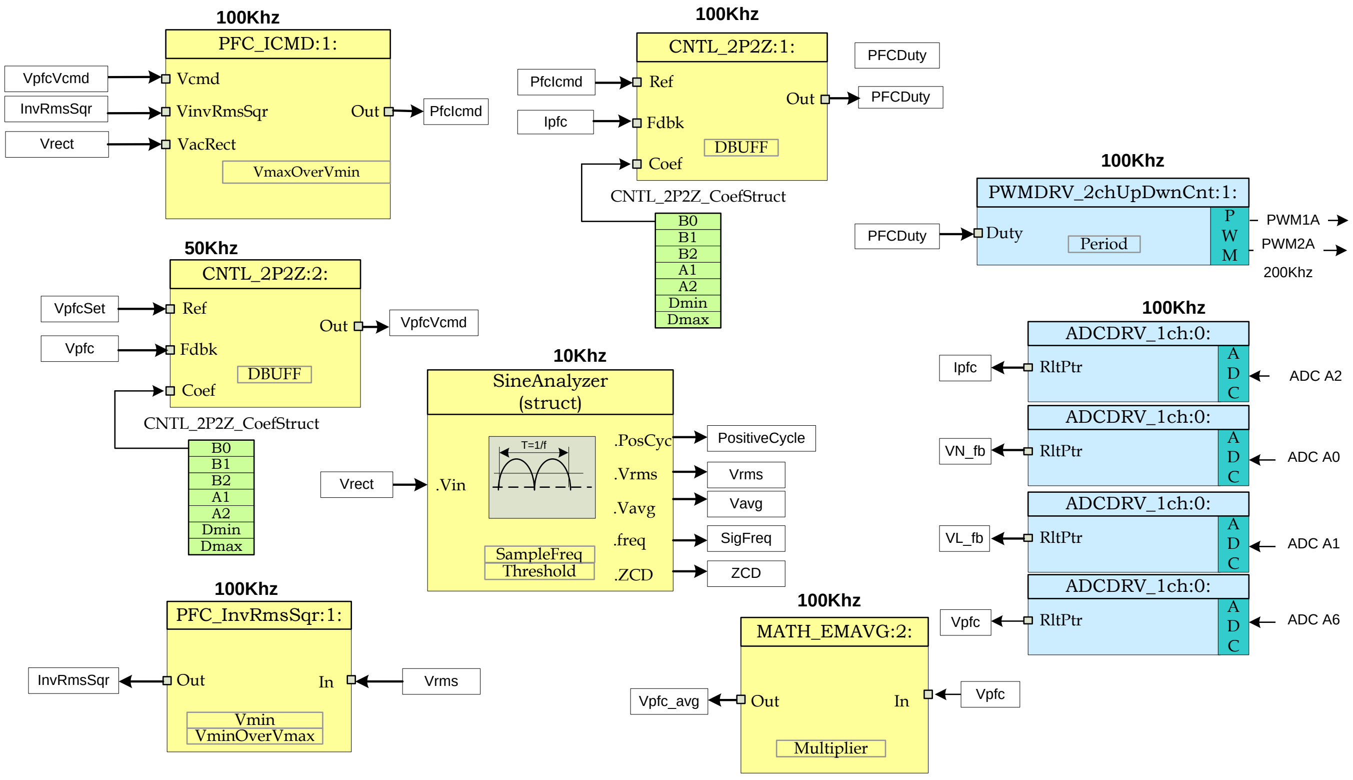


Figure 6. C2000 ILPFC Software Control Flow

As mentioned in Section 1, the IL PFC system is controlled by two feedback loops. The outer voltage loop regulates the DC bus voltage, while a faster inner current loop shapes the input current in order to maintain a high input power factor. Figure 6 also shows the rate at which the software modules are executed. For example, the current controller is executed at a rate of 100kHz (half of the PWM switching frequency) while the voltage controller is executed at 50kHz rate.

3.2 Incremental Builds

The complete CCS project for ILPFC is divided into three incremental builds. This approach provides the user with a step-by-step method to get familiar with the software and understand how it interacts with the IL PFC hardware. This approach also simplifies the task of debugging and testing the boards.

The build options are shown in Table 2 . To select a particular build option set the parameter INCR_BUILD to the corresponding build selection as shown. This parameter is found in *InterleavedPFC-Settings.h* file. Once the build option is selected, compile the complete project by selecting rebuild-all compiler option. Next chapter provides more details to run each of the build options.

Table 2. Incremental Build Options for PFC

INCR_BUILD = 1	Open loop test for boost PFC and ADC feedback (Check sensing circuitry)
INCR_BUILD = 2	Open voltage loop and closed current loop control of IL PFC
INCR_BUILD = 3	Closed voltage loop and closed current loop control of IL PFC

3.3 Procedure for running the incremental builds

All software files related to this C2x controlled IL PFC system i.e., the main source files, ISR assembly files and the project file for C framework, are located in the directory **…\controlSUITE\development_kits\ILPFC_v1.0\ILPFC**. The projects included with this software are targeted for CCSv4.

CAUTION

There are high voltages present on the board. It should only be handled by experienced power supply professionals in a lab environment. To safely evaluate this board an isolated AC source should be used to power up the unit. Before AC power is applied to the board a voltmeter and an appropriate resistive load (only) should be attached to the output. This will discharge the bus capacitor quickly when the AC power is turned off. The board has not been tested with electronic load and so it should not be used with such load. There is no output over-current protection implemented on the board and so the user should take appropriate measures for preventing any output short circuit condition. The ILPFC board should always be started with 110Vac (60Hz). Once the board is up and running the input voltage can be changed to any other voltage within the specification.

Follow the steps in the following "Build..." sections to build and run the example included in the PFC software.

3.3.1 Build 1: Open Loop Boost PFC with ADC

3.3.1.1 Objective

The objectives of this build are as follows:

1. Evaluate IL PFC PWM and ADC software driver modules
2. Verify MOSFET gate driver circuit, voltage, and current sensing circuit
3. Familiarize yourself with the operation of Code Composer Studio (CCS).

Under this build the system runs in open-loop mode and so the measured ADC values are used for circuit verification and instrumentation purposes only. Steps required for building and running a project is explained next.

3.3.1.2 Overview

The software in Build1 has been configured so that the user can quickly evaluate the PWM driver module (software driver) by viewing the related waveforms on a scope and observing the effect of duty cycle change on PFC output voltage. The user can adjust the PWM duty cycle from CCS watch window. The user can also evaluate the ADC driver module (software driver) by viewing the ADC sampled data in the CCS watch window. The PWM and ADC driver macro instantiations are executed inside the _DPL_ISR. Figure 7 shows the software blocks used in this build. The two PWM signals for the two PFC switches are obtained from ePWM module 1. ePWM1A drives one of the PFC switches while ePWM1B drives the other.

The signals that are sensed and input to the MCU include the following:

1. Line and neutral voltages (VL_fb, VN_fb)
2. PFC input current (Ipfc)
3. DC bus voltage (Vpfc).

These quantities are read using the ADC driver module and are indicated in Figure 7. The ADC driver module converts the 12-bit ADC result to a 32-bit Q24 value. A few lines of code in the ISR implements the detection of input AC line half cycle (positive & negative half cycles) and the rectification of the input voltage. This generates the input rectified signal Vrect.

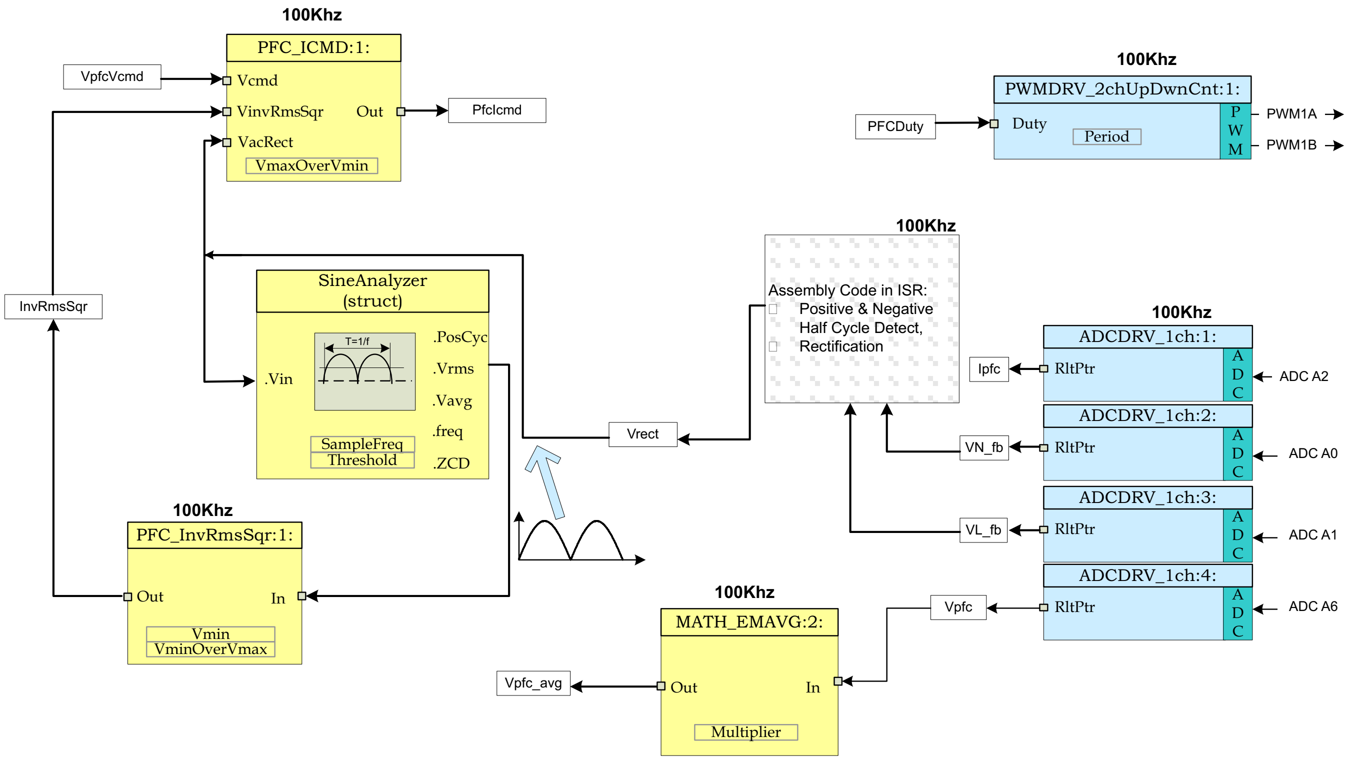


Figure 7. Build 1 Software Blocks

The PWM signals are generated at a frequency of 200 kHz i.e. a period of 5 us. With the controller operating at 60MHz, one count of the time base counter of ePWM1 corresponds to 16.6667ns. This implies a PWM period of 5us is equivalent to 300 counts of the time base counter (TBCNT1). The ePWM1 module is configured to operate in up-down count mode as shown in Figure 8. This means a time base period value of 150 (period register value) will give a total PWM period value of 300 counts (i.e. 5 us).

PFC total inductor current is sampled at the midpoint of the PWM ON pulse since the sampled value represents the average inductor current under CCM (continuous conduction mode) condition. Under DCM condition the sampled current value also represents an approximate average inductor current because of the oversampling action. PFC inductor current is also oversampled 8 times during each 10 us time period when both the PFC switches turn on once in their respective 5 us time slot (200kHz PWM). These 8 sampled values are then used to calculate the average PFC inductor current. This is illustrated in Figure 8.

The voltage signal conversions are also initiated when the PFC switch is on. This is indicated in Figure 8. The flexibility of ADC and PWM modules on C2000 devices allow for precise and flexible ADC start of conversions. In this case the time base counters (TBCNTx) of ePWM1~ePWM4 are used as 4 time bases to generate all the start of conversion (SOC) triggers.

Figure 8 shows 8 SOC triggers are generated when:

1. TBCNT1 reaches zero and period
2. TBCNT2 reaches the preset CMPA values during up and down count
3. TBCNT3 reaches zero and preset CMPB value during up count
4. TBCNT4 reaches preset CMPA and CMPB values during up count

Systems Applications Collateral A dummy ADC conversion is also performed at TBCNT3 zero point in order to ensure the integrity of the ADC results.

Figure 8 also shows the PWM outputs generated using ePWM1 module. PFC current is converted, averaged and then saved as Ipfc for PFC current loop control.

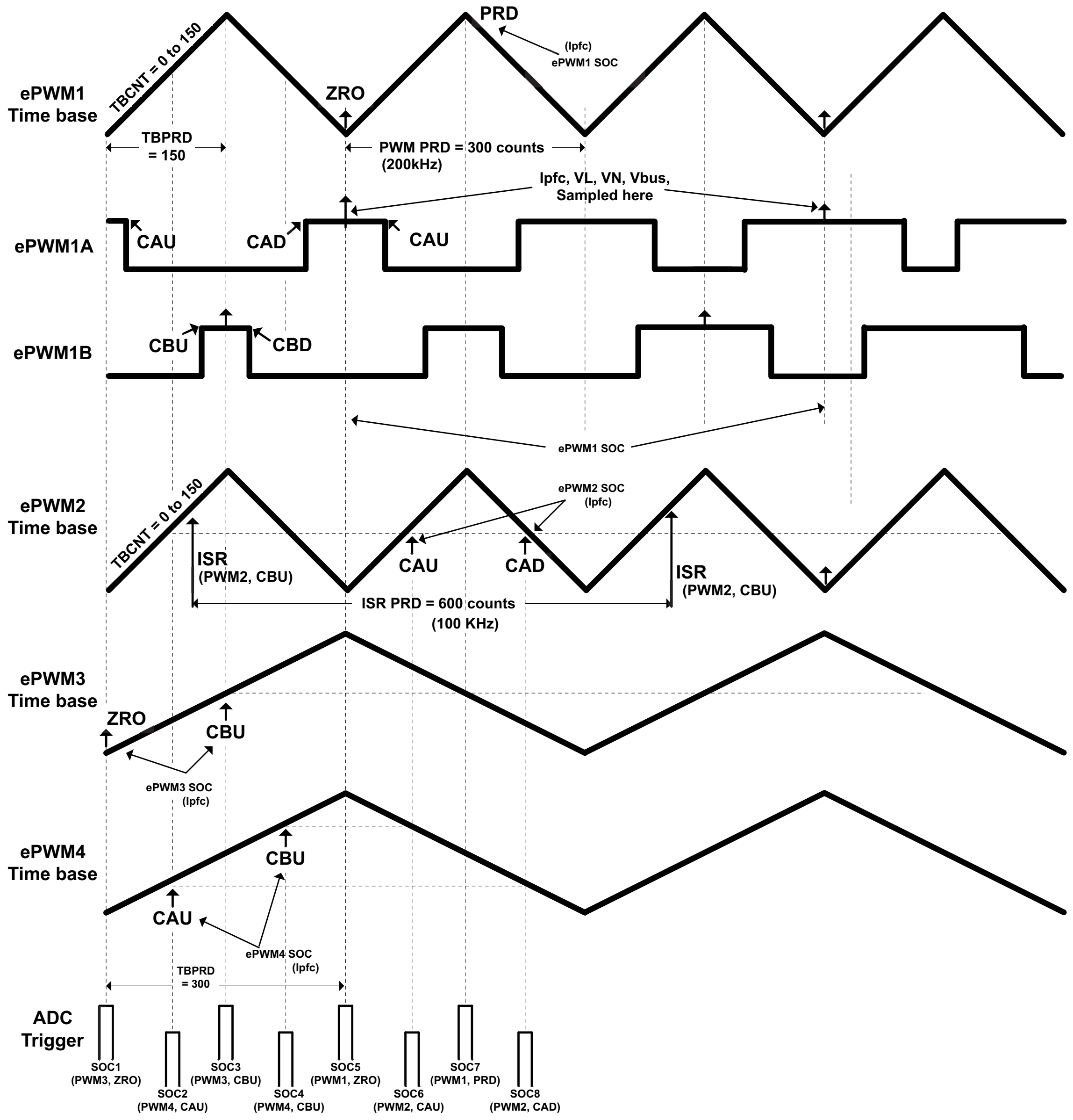


Figure 8. PWM generation and ADC Sampling

All ADC results are read in the ISR by executing the ADC driver module from the 100kHz ISR labeled as _DPL_ISR.

This ISR in assembly (_DPL_ISR) is triggered by EPWM2 on a CMPB match event on up count. CMPB is set to 80 so that the ISR is triggered only after the ADC conversions are complete. Inside the ISR PWMDRV_2ch_UpDwnCnt macros are executed and the PWM compare shadow registers are updated. These are loaded into the active register at the next TBCNT1 = ZERO event. Note that the ISR trigger frequency is half that of the PWM switching frequency as shown in Figure 8.

3.3.1.2.1 Protection

An overvoltage protection mechanism is implemented in software for this IL PFC EVM.

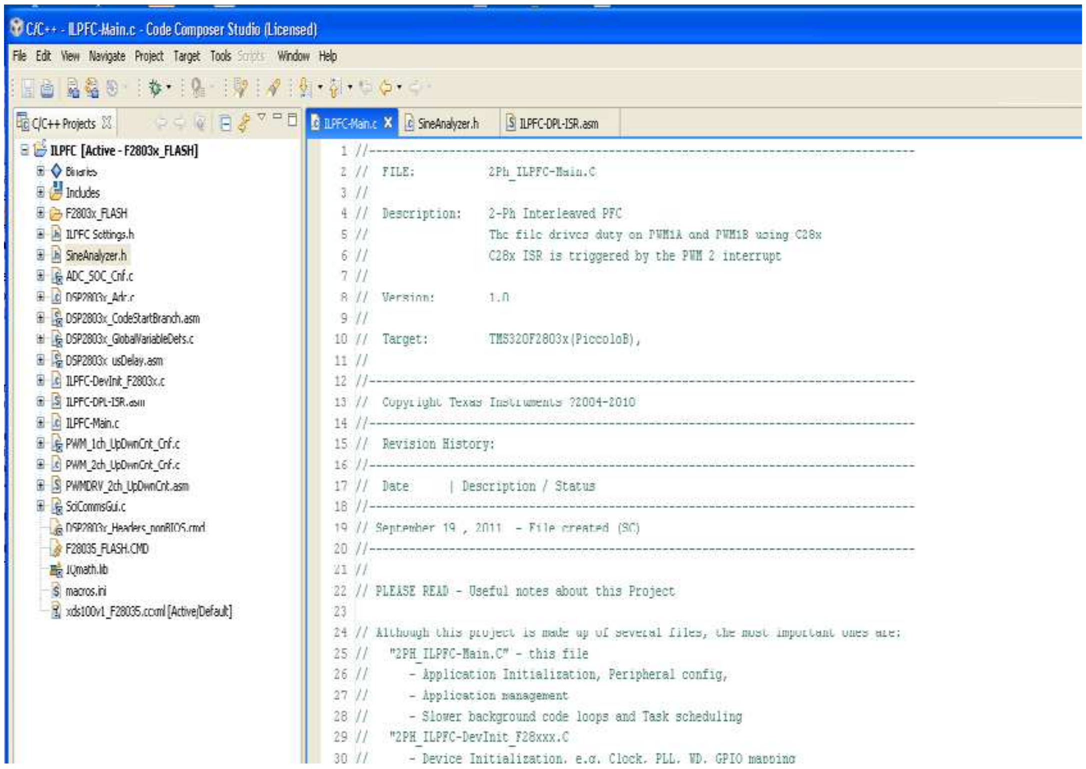
The sensed DC bus output voltage from the ADC input is compared against the overvoltage protection threshold set by the user. The OV threshold point for this ILPFC EVM has been set to 430V. This threshold parameter is labeled as VBUS_OVP_THRSHLD inside the file *InterleavedPFC-Settings.h*. In case of an OV condition the PWM outputs are shut off using the TZ (trip zone) registers. The flexibility of the trip mechanism on C2000 devices provides the possibilities for taking different actions on different trip events. In this project both PWM outputs will be driven low in case of a trip event. Both outputs are held in this state until a device reset is executed.

3.3.1.3 Procedure

3.3.1.3.1 Start CCS and Open a Project

Follow these steps to execute this build:

1. Connect USB connector to the Piccolo controller board for emulation. Connect the 12V bias supply output (external bias supply provided with the PFC EVM) to JP1 and apply this bias voltage to the board by setting the switch SW1 to position "Ext". By default, the Piccolo control card jumpers (see Piccolo control card documentation) are configured such that the device boot from FLASH. Change these jumper settings to allow code execution under CCS control.
2. Start Code Composer Studio (CCS). In CCS, a project contains all the files and build options needed to generate an executable output file (.out) which can run on the MCU hardware. On the menu bar click: Project -> Import Existing CCS/CCE Eclipse Project and under Select root directory navigate to and select *..\controlSUITE\development_kits\ILPFC_v1.0\ILPFC* directory. Make sure that under the Projects tab ILPFC is checked. Click Finish. This project will invoke all the necessary tools (compiler, assembler & linker) for building the project.
3. In the project window on the left, click the plus sign (+) to the left of Project. Your project window will look like the following in Figure 9:



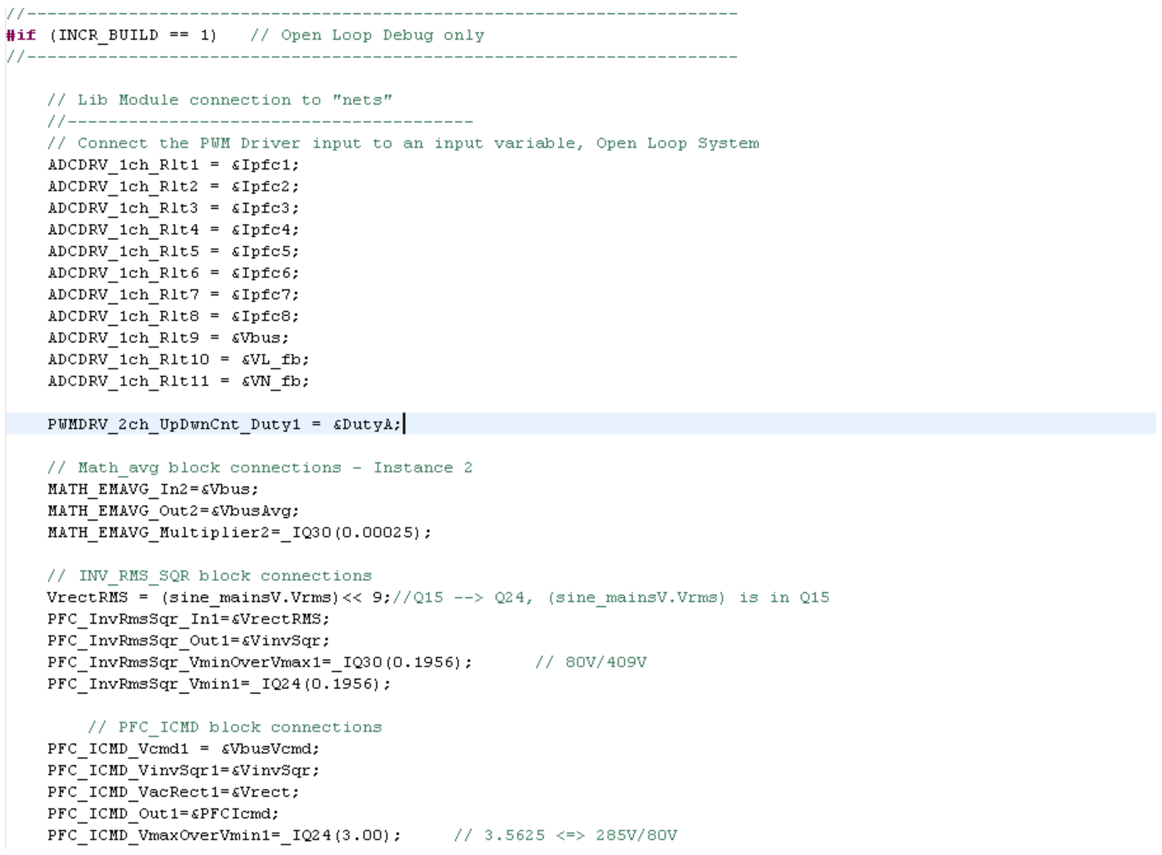
```
//------------------------------------------------------------------------------
// FILE:          2Ph_ILPFC-Main.C
//
// Description:   2-Ph Interleaved PFC
//                The file drives duty on PWM1A and PWM1B using C28x
//                C28x ISR is triggered by the PWM 2 interrupt
//
// Version:       1.0
//
// Target:        TMS320F2803x(PiccoloB),
//
//------------------------------------------------------------------------------
// Copyright Texas Instruments ?2004-2010
//------------------------------------------------------------------------------
// Revision History:
//------------------------------------------------------------------------------
// Date      | Description / Status
//------------------------------------------------------------------------------
// September 19 , 2011  - File created (SC)
//------------------------------------------------------------------------------
//
// PLEASE READ - Useful notes about this Project

// Although this project is made up of several files, the most important ones are:
//   "2PH_ILPFC-Main.C" - this file
//      - Application Initialization, Peripheral config,
//      - Application management
//      - Slower background code loops and Task scheduling
//   "2PH_ILPFC-DevInit_F28xxx.C
//      - Device Initialization, e.g. Clock, PLL, WD, GPIO mapping
```

Figure 9. CCS Project Window

3.3.1.3.2 Device Initialization, Main, and ISR Files

NOTE: Do **not** make any changes to the source files - **Only Inspect**

1. Open and inspect *InterleavedPFC-DevInit_F2803x.c* by double clicking on the filename in the project window. Note that system clock, peripheral clock prescale, and peripheral clock enables have been setup. Next, notice that the shared GPIO pins have been configured.
2. Open and inspect *InterleavedPFC-Main.c*. Notice the call made to DeviceInit() function and other variable initialization. Also notice code for different incremental build options, the ISR initialization and the background for(;;) loop.
3. Locate and inspect the following code in the main file under initialization code specific for build 1. This is where the PWMDRV_2ch_UpDwnCnt and ADCDRV_1CH blocks are connected in the control flow.

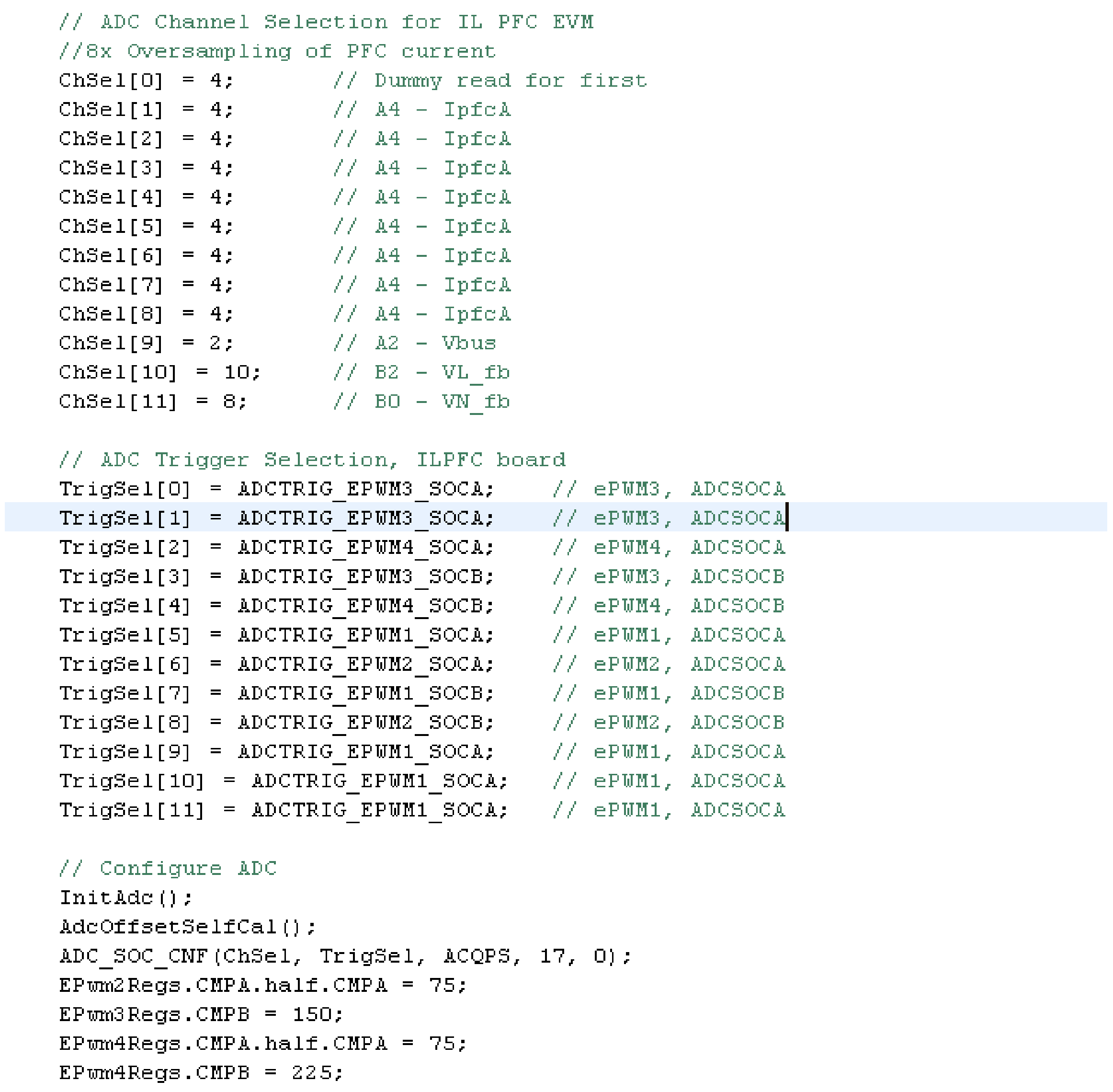


```
//----------------------------------------------------------------------
#if (INCR_BUILD == 1)   // Open Loop Debug only
//----------------------------------------------------------------------

    // Lib Module connection to "nets"
    //----------------------------------------
    // Connect the PWM Driver input to an input variable, Open Loop System
    ADCDRV_1ch_Rlt1 = &Ipfc1;
    ADCDRV_1ch_Rlt2 = &Ipfc2;
    ADCDRV_1ch_Rlt3 = &Ipfc3;
    ADCDRV_1ch_Rlt4 = &Ipfc4;
    ADCDRV_1ch_Rlt5 = &Ipfc5;
    ADCDRV_1ch_Rlt6 = &Ipfc6;
    ADCDRV_1ch_Rlt7 = &Ipfc7;
    ADCDRV_1ch_Rlt8 = &Ipfc8;
    ADCDRV_1ch_Rlt9 = &Vbus;
    ADCDRV_1ch_Rlt10 = &VL_fb;
    ADCDRV_1ch_Rlt11 = &VN_fb;

    PWMDRV_2ch_UpDwnCnt_Duty1 = &DutyA;

    // Math_avg block connections - Instance 2
    MATH_EMAVG_In2=&Vbus;
    MATH_EMAVG_Out2=&VbusAvg;
    MATH_EMAVG_Multiplier2=_IQ30(0.00025);

    // INV_RMS_SQR block connections
    VrectRMS = (sine_mainsV.Vrms)<< 9;//Q15 --> Q24, (sine_mainsV.Vrms) is in Q15
    PFC_InvRmsSqr_In1=&VrectRMS;
    PFC_InvRmsSqr_Out1=&VinvSqr;
    PFC_InvRmsSqr_VminOverVmax1=_IQ30(0.1956);      // 80V/409V
    PFC_InvRmsSqr_Vmin1=_IQ24(0.1956);

        // PFC_ICMD block connections
    PFC_ICMD_Vcmd1 = &VbusVcmd;
    PFC_ICMD_VinvSqr1=&VinvSqr;
    PFC_ICMD_VacRect1=&Vrect;
    PFC_ICMD_Out1=&PFCIcmd;
    PFC_ICMD_VmaxOverVmin1=_IQ24(3.00);     // 3.5625 <=> 285V/80V
```

4. Locate and inspect the following code in the main file under initialization code. This is where the PWMDRV_2ch_UpDwnCnt block is configured and initialized. This is common for all incremental builds. This PWM driver module inputs the total PWM period value of 300 and internally calculates the period register value of 150.

```
// Configure ePWMs to generate ADC SOC pulses for PFC current over sampling
EPwm1Regs.ETSEL.bit.SOCAEN = 1;              // Enable ePWM1 SOCA pulse
EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;   // SOCA from ePWM1 Zero event
EPwm1Regs.ETPS.bit.SOCAPRD = ET_1ST;         // Trigger ePWM1 SOCA on every event
EPwm1Regs.ETSEL.bit.SOCBEN = 1;              // Enable ePWM1 SOCB pulse
EPwm1Regs.ETSEL.bit.SOCBSEL = ET_CTR_PRD;    // SOCB from ePWM1 PRD event
EPwm1Regs.ETPS.bit.SOCBPRD = ET_1ST;         // Trigger ePWM1 SOCB on every event

EPwm2Regs.ETSEL.bit.SOCAEN = 1;              // Enable ePWM2 SOCA pulse
EPwm2Regs.ETSEL.bit.SOCASEL = ET_CTRU_CMPA;  // SOCA from ePWM2 CMPA up event
EPwm2Regs.ETPS.bit.SOCAPRD = ET_1ST;         // Trigger ePWM2 SOCA on every event
EPwm2Regs.ETSEL.bit.SOCBEN = 1;              // Enable ePWM2 SOCB pulse
EPwm2Regs.ETSEL.bit.SOCBSEL = ET_CTRD_CMPA;  // SOCB from ePWM2 CMPA down event
EPwm2Regs.ETPS.bit.SOCBPRD = ET_1ST;         // Trigger ePWM2 SOCB on every event

// Configure ePWMs to generate ADC SOC pulses
EPwm3Regs.ETSEL.bit.SOCAEN = 1;              // Enable ePWM3 SOCA pulse
EPwm3Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;   // SOCA from ePWM3 zero event
EPwm3Regs.ETPS.bit.SOCAPRD = ET_1ST;         // Trigger ePWM3 SOCA on every event
EPwm3Regs.ETSEL.bit.SOCBEN = 1;              // Enable ePWM3 SOCB pulse
EPwm3Regs.ETSEL.bit.SOCBSEL = ET_CTRU_CMPB;  // SOCB from ePWM3 CMPB up event
EPwm3Regs.ETPS.bit.SOCBPRD = ET_1ST;         // Trigger ePWM3 SOCB on every event

EPwm4Regs.ETSEL.bit.SOCAEN = 1;              // Enable ePWM4 SOCA pulse
EPwm4Regs.ETSEL.bit.SOCASEL = ET_CTRU_CMPA;  // SOCA from ePWM4 CMPA up event
EPwm4Regs.ETPS.bit.SOCAPRD = ET_1ST;         // Trigger ePWM4 SOCA on every event
EPwm4Regs.ETSEL.bit.SOCBEN = 1;              // Enable ePWM4 SOCB pulse
EPwm4Regs.ETSEL.bit.SOCBSEL = ET_CTRU_CMPB;  // SOCB from ePWM4 CMPB up event
EPwm4Regs.ETPS.bit.SOCBPRD = ET_1ST;         // Trigger ePWM4 SOCB on every event
```

5. Also locate and inspect the following code in the main file under initialization code. This is where the ADCDRV_1CH block is configured and initialized. This is also common for all incremental builds.



```
// ADC Channel Selection for IL PFC EVM
//8x Oversampling of PFC current
ChSel[0] = 4;       // Dummy read for first
ChSel[1] = 4;       // A4 - IpfcA
ChSel[2] = 4;       // A4 - IpfcA
ChSel[3] = 4;       // A4 - IpfcA
ChSel[4] = 4;       // A4 - IpfcA
ChSel[5] = 4;       // A4 - IpfcA
ChSel[6] = 4;       // A4 - IpfcA
ChSel[7] = 4;       // A4 - IpfcA
ChSel[8] = 4;       // A4 - IpfcA
ChSel[9] = 2;       // A2 - Vbus
ChSel[10] = 10;     // B2 - VL_fb
ChSel[11] = 8;      // B0 - VN_fb

// ADC Trigger Selection, ILPFC board
TrigSel[0] = ADCTRIG_EPWM3_SOCA;    // ePWM3, ADCSOCA
TrigSel[1] = ADCTRIG_EPWM3_SOCA;    // ePWM3, ADCSOCA
TrigSel[2] = ADCTRIG_EPWM4_SOCA;    // ePWM4, ADCSOCA
TrigSel[3] = ADCTRIG_EPWM3_SOCB;    // ePWM3, ADCSOCB
TrigSel[4] = ADCTRIG_EPWM4_SOCB;    // ePWM4, ADCSOCB
TrigSel[5] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA
TrigSel[6] = ADCTRIG_EPWM2_SOCA;    // ePWM2, ADCSOCA
TrigSel[7] = ADCTRIG_EPWM1_SOCB;    // ePWM1, ADCSOCB
TrigSel[8] = ADCTRIG_EPWM2_SOCB;    // ePWM2, ADCSOCB
TrigSel[9] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA
TrigSel[10] = ADCTRIG_EPWM1_SOCA;   // ePWM1, ADCSOCA
TrigSel[11] = ADCTRIG_EPWM1_SOCA;   // ePWM1, ADCSOCA

// Configure ADC
InitAdc();
AdcOffsetSelfCal();
ADC_SOC_CNF(ChSel, TrigSel, ACQPS, 17, 0);
EPwm2Regs.CMPA.half.CMPA = 75;
EPwm3Regs.CMPB = 150;
EPwm4Regs.CMPA.half.CMPA = 75;
EPwm4Regs.CMPB = 225;
```

```
// Configure ePWMs to generate ADC SOC pulses for PFC current over sampling
EPwm1Regs.ETSEL.bit.SOCAEN = 1;                 // Enable ePWM1 SOCA pulse
EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;      // SOCA from ePWM1 Zero event
EPwm1Regs.ETPS.bit.SOCAPRD = ET_1ST;            // Trigger ePWM1 SOCA on every event
EPwm1Regs.ETSEL.bit.SOCBEN = 1;                 // Enable ePWM1 SOCB pulse
EPwm1Regs.ETSEL.bit.SOCBSEL = ET_CTR_PRD;       // SOCB from ePWM1 PRD event
EPwm1Regs.ETPS.bit.SOCBPRD = ET_1ST;            // Trigger ePWM1 SOCB on every event

EPwm2Regs.ETSEL.bit.SOCAEN = 1;                 // Enable ePWM2 SOCA pulse
EPwm2Regs.ETSEL.bit.SOCASEL = ET_CTRU_CMPA;     // SOCA from ePWM2 CMPA up event
EPwm2Regs.ETPS.bit.SOCAPRD = ET_1ST;            // Trigger ePWM2 SOCA on every event
EPwm2Regs.ETSEL.bit.SOCBEN = 1;                 // Enable ePWM2 SOCB pulse
EPwm2Regs.ETSEL.bit.SOCBSEL = ET_CTRD_CMPA;     // SOCB from ePWM2 CMPA down event
EPwm2Regs.ETPS.bit.SOCBPRD = ET_1ST;            // Trigger ePWM2 SOCB on every event

// Configure ePWMs to generate ADC SOC pulses
EPwm3Regs.ETSEL.bit.SOCAEN = 1;                 // Enable ePWM3 SOCA pulse
EPwm3Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;      // SOCA from ePWM3 zero event
EPwm3Regs.ETPS.bit.SOCAPRD = ET_1ST;            // Trigger ePWM3 SOCA on every event
EPwm3Regs.ETSEL.bit.SOCBEN = 1;                 // Enable ePWM3 SOCB pulse
EPwm3Regs.ETSEL.bit.SOCBSEL = ET_CTRU_CMPB;     // SOCB from ePWM3 CMPB up event
EPwm3Regs.ETPS.bit.SOCBPRD = ET_1ST;            // Trigger ePWM3 SOCB on every event

EPwm4Regs.ETSEL.bit.SOCAEN = 1;                 // Enable ePWM4 SOCA pulse
EPwm4Regs.ETSEL.bit.SOCASEL = ET_CTRU_CMPA;     // SOCA from ePWM4 CMPA up event
EPwm4Regs.ETPS.bit.SOCAPRD = ET_1ST;            // Trigger ePWM4 SOCA on every event
EPwm4Regs.ETSEL.bit.SOCBEN = 1;                 // Enable ePWM4 SOCB pulse
EPwm4Regs.ETSEL.bit.SOCBSEL = ET_CTRU_CMPB;     // SOCB from ePWM4 CMPB up event
EPwm4Regs.ETPS.bit.SOCBPRD = ET_1ST;            // Trigger ePWM4 SOCB on every event
```

6. Open and inspect InterleavedPFC-DPL-ISR.asm. Notice the _DPL_Init and _DPL_ISR sections under build 1. This is where the PWM and ADC driver macro instantiation is done for initialization and runtime, respectively.

3.3.1.3.3 *Build and Load the Project*

1. Select the incremental build option as 1 in the *InterleavedPFC-Settings.h* file.

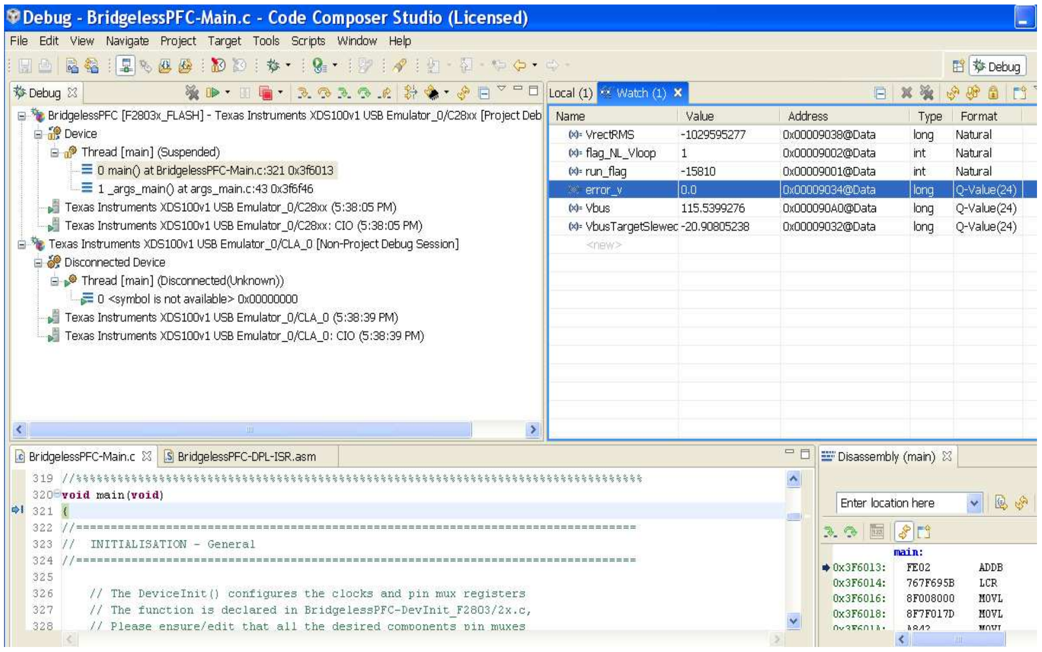
NOTE: Whenever you change the incremental build option in *InterleavedPFC-Settings.h*, always do a "Rebuild All".

2. Click Project -> “Rebuild All” button and watch the tools run in the build window.
3. Click Target ->”Debug Active Project”. CCS will ask you to open a new Target configuration file if one hasn’t already been selected. If a valid target configuration file has been created for this connection you may jump to Section 3.3.1.3.4. In the New target Configuration Window type in the name of the .ccxml file for the target you will be working with (Example: xds100-F28035.ccxml). Check “Use shared location” and click Finish.
4. In the .ccxml file that open up select Connection as “Texas Instruments XDS100v2 USB Emulator” and under the device, scroll down and select “TMS320F28035”. Click Save.
5. Click Targe -> -> ”Debug Active Project”. Select project configuration as F2803x_FLASH. The program will be loaded into the FLASH. You should now be at the start of Main().

3.3.1.3.4 Debug Environment Windows

It is standard debug practice to watch local and global variables while debugging code. There are various methods for doing this in Code Composer Studio, such as memory views and watch views. If a watch view did not open when the debug environment was launched, open a new watch view and add various parameters to it by following the procedure given below.

1. Click View -> Watch on the menu bar.
2. Click the “Watch (1)" tab at the top watch view. You may add any variables to the watch view. In the empty box in the "Name" column, type the symbol name of the variable you want to watch and press enter on keyboard. Be sure to modify the “Format” as needed. The watch view should look something like the following in Figure 10


Debug - BridgelessPFC-Main.c - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
Debug
Debug
Local (1)
Watch (1)
BridgelessPFC-Main.c
BridgelessPFC-DPL-ISR.asm
Disassembly (main)
Enter location here

```
BridgelessPFC [F2803x_FLASH] - Texas Instruments XDS100v1 USB Emulator_0/C28xx [Project Deb
  Device
    Thread [main] (Suspended)
      0 main() at BridgelessPFC-Main.c:321 0x3f6013
      1 _args_main() at args_main.c:43 0x3f6f46
  Texas Instruments XDS100v1 USB Emulator_0/C28xx (5:38:05 PM)
  Texas Instruments XDS100v1 USB Emulator_0/C28xx: CIO (5:38:05 PM)
Texas Instruments XDS100v1 USB Emulator_0/CLA_0 [Non-Project Debug Session]
  Disconnected Device
    Thread [main] (Disconnected(Unknown))
      0 <symbol is not available> 0x00000000
  Texas Instruments XDS100v1 USB Emulator_0/CLA_0 (5:38:39 PM)
  Texas Instruments XDS100v1 USB Emulator_0/CLA_0: CIO (5:38:39 PM)
```

Name	Value	Address	Type	Format
VrectRMS	-1029595277	0x00009038@Data	long	Natural
flag_NL_Vloop	1	0x00009002@Data	int	Natural
run_flag	-15810	0x00009001@Data	int	Natural
error_v	0.0	0x00009034@Data	long	Q-Value(24)
Vbus	115.5399276	0x000090A0@Data	long	Q-Value(24)
VbusTargetSlewec	-20.90805238	0x00009032@Data	long	Q-Value(24)
<new>				

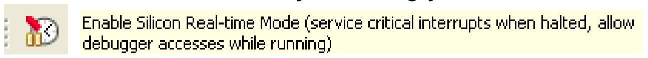
```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
void main(void)
{
//================================================================================
//  INITIALISATION - General
//================================================================================

    // The DeviceInit() configures the clocks and pin mux registers
    // The function is declared in BridgelessPFC-DevInit_F2803/2x.c,
    // Please ensure/edit that all the desired components pin muxes
```

```
                main:
0x3F6013:   FE02        ADDB
0x3F6014:   767F695B    LCR
0x3F6016:   8F008000    MOVL
0x3F6018:   8F7F017D    MOVL
0x3F601A:   A842        MOVL
```

Figure 10. CCS Watch View for Build 1

3.3.1.3.5 Using Real-Time Emulation

Real-time emulation is a special emulation feature that allows the windows within Code Composer Studio to be updated at a rate up to 10 Hz while the MCU is running. This not only allows graphs and watch views to update, but also allows the user to change values in watch or memory windows, and see the effect of these changes in the system. This is very useful when changing control law parameters on-the-fly, for example.

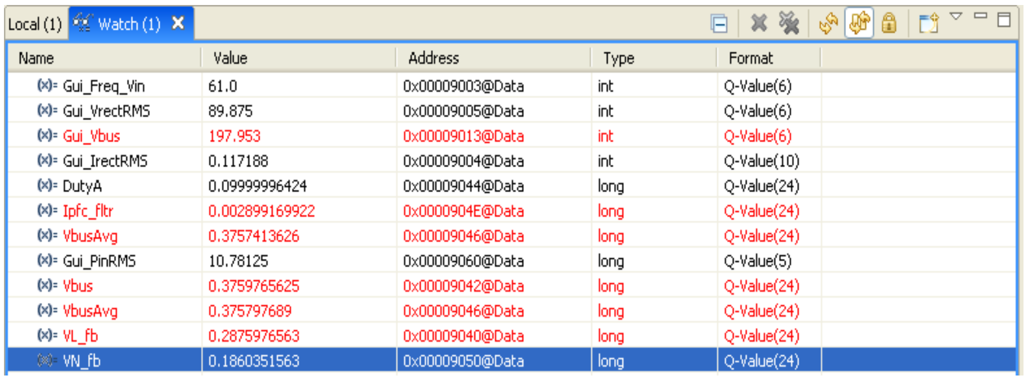
1. Enable real-time mode by hovering your mouse on the buttons on the horizontal toolbar and clicking  Enable Silicon Real-time Mode (service critical interrupts when halted, allow debugger accesses while running) button.
2. A message box may appear. If so, select YES to enable debug events. This will 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.
3. Click on Continuous Refresh buttons for the watch view.

3.3.1.3.6 Run the Code

1. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target -> Run on the menu bar.
2. In the watch view, add the variable DutyA and set it to 0.1 (=1677721 in Q24). This variable sets the duty cycle for the PFC converter.
3. Apply an appropriate resistive load to the PFC system at the DC output (8k/20W, or, 4K/40W or, 2k/80W).
4. Slowly apply AC Power to the board. Measure and verify the DC bus voltage corresponding to applied input voltage and the duty ratio.
5. Use DutyA to slowly change the duty from the watch window. The boost converter output voltage should change accordingly.

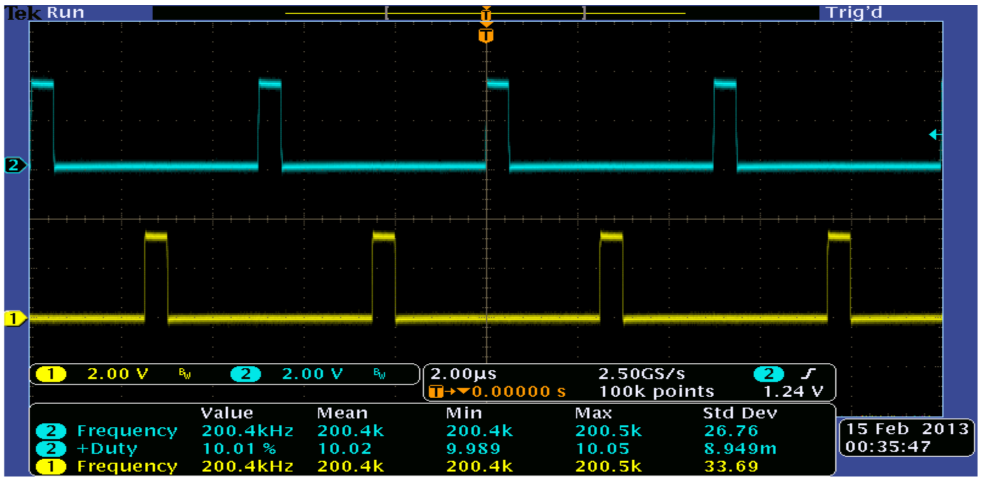
NOTE: Observe the output voltage carefully, the output voltage should **not** be allowed to exceed the maximum voltage rating of the board (400V).

6. Add the other variables such as, Vbus, VL_fb, VN_fb and verify the different ADC results in the watch view. For AC voltage input the sensed line and neutral voltage(VL_fb, VN_fb) will vary continuously in the watch window. Therefore, to verify the ADC readings and the line and neutral voltage sense circuits, the user may apply DC input voltage (20~200V) instead of AC (as stated in Step 4). In that case the PFC stage will temporarily operate in a pure dc-dc boost mode. CCS watch window below shows some of the variables under Build 1 when the input AC voltage is about 90Vrms, open loop duty is 10%, DC bus resistive load is about 4K ohm and DC bus voltage is 200V. Notice the Q-format used for each variable. Under this condition first four variables in the watch window show the AC line frequency, the RMS input voltage, the DC bus voltage and the RMS input current. The variable Gui_PinRMS shows the input power.



Name	Value	Address	Type	Format
Gui_Freq_Vin	61.0	0x00009003@Data	int	Q-Value(6)
Gui_VrectRMS	89.875	0x00009005@Data	int	Q-Value(6)
Gui_Vbus	197.953	0x00009013@Data	int	Q-Value(6)
Gui_IrectRMS	0.117188	0x00009004@Data	int	Q-Value(10)
DutyA	0.09999996424	0x00009044@Data	long	Q-Value(24)
Ipfc_fltr	0.002899169922	0x0000904E@Data	long	Q-Value(24)
VbusAvg	0.3757413626	0x00009046@Data	long	Q-Value(24)
Gui_PinRMS	10.78125	0x00009060@Data	long	Q-Value(5)
Vbus	0.3759765625	0x00009042@Data	long	Q-Value(24)
VbusAvg	0.375797689	0x00009046@Data	long	Q-Value(24)
VL_fb	0.2875976563	0x00009040@Data	long	Q-Value(24)
VN_fb	0.1860351563	0x00009050@Data	long	Q-Value(24)

7. The following oscilloscope capture shows two PWM outputs (Ch1 & Ch2) with duty ratio set to 10%. The PWM frequency is measured to be 200kHz.



	Value	Mean	Min	Max	Std Dev
2 Frequency	200.4kHz	200.4k	200.4k	200.5k	26.76
2 +Duty	10.01 %	10.02	9.989	10.05	8.949m
1 Frequency	200.4kHz	200.4k	200.4k	200.5k	33.69

8. Try different duty cycle values and observe the corresponding ADC results. Increase duty cycle value in small steps. Always observe the output voltage carefully, this should not be allowed to exceed the capabilities of the board. Different waveforms, like the PWM gate drive signals, input voltage and current and output voltage may also be probed and verified using an oscilloscope. Appropriate safety measures must be taken while probing these high voltage signals.
9. Fully halting the MCU when in real-time mode is a two-step process. With the AC input turned off wait until the DC bus capacitor is fully discharged. First, halt the processor by using the Halt button on the toolbar, or by using Target -> Halt. Then take the MCU out of real-time mode. Finally reset the MCU.
10. You may choose to leave Code Composer Studio running for the next exercise or optionally close CCS.
11. End of Exercise

3.3.1.4 Build 2: IL PFC with Closed-Current Loop

3.3.1.4.1 Objective

The objective of this build is to verify the operation of the IL PFC under closed current loop mode.

3.3.1.4.2 Overview

Figure 11 shows the software blocks used in this build. Notice that 1 additional software blocks are added to the Build 1 diagram (Figure 7) to implement this closed current loop system. The Sine Analyzer block calculates the RMS voltage and frequency of the input voltage. PFC InvRmsSqr block calculates the inverse of the square of the RMS input voltage. This calculated value together with the rectified voltage (Vrect) is used in the software block PFC_ICMD to generate the reference current command PfcIcmd for the PFC current control loop. PFC_ICMD block uses a 3rd input VpfcVcmd for controlling the magnitude of the reference current command. Since this software build implements only the PFC current loop (open voltage loop), this parameter VpfcVcmd needs to be varied from the CCS watch window in order to adjust the magnitude of the reference current and hence the PFC bus voltage. A two pole two zero (2p2z) controller is used to implement the current control loop. This is the 4th software block shown in Figure 11 as CNTL_2P2Z:1. Depending on the control loop requirements other control blocks such as a PI or a 3p3z controller can also be used.

As shown in Figure 11 the current loop control block is executed at a 100 KHz rate. CNTL_2P2Z is a 2nd order compensator realized from an IIR filter structure. This function is independent of any peripherals and therefore does not require a CNF function call.

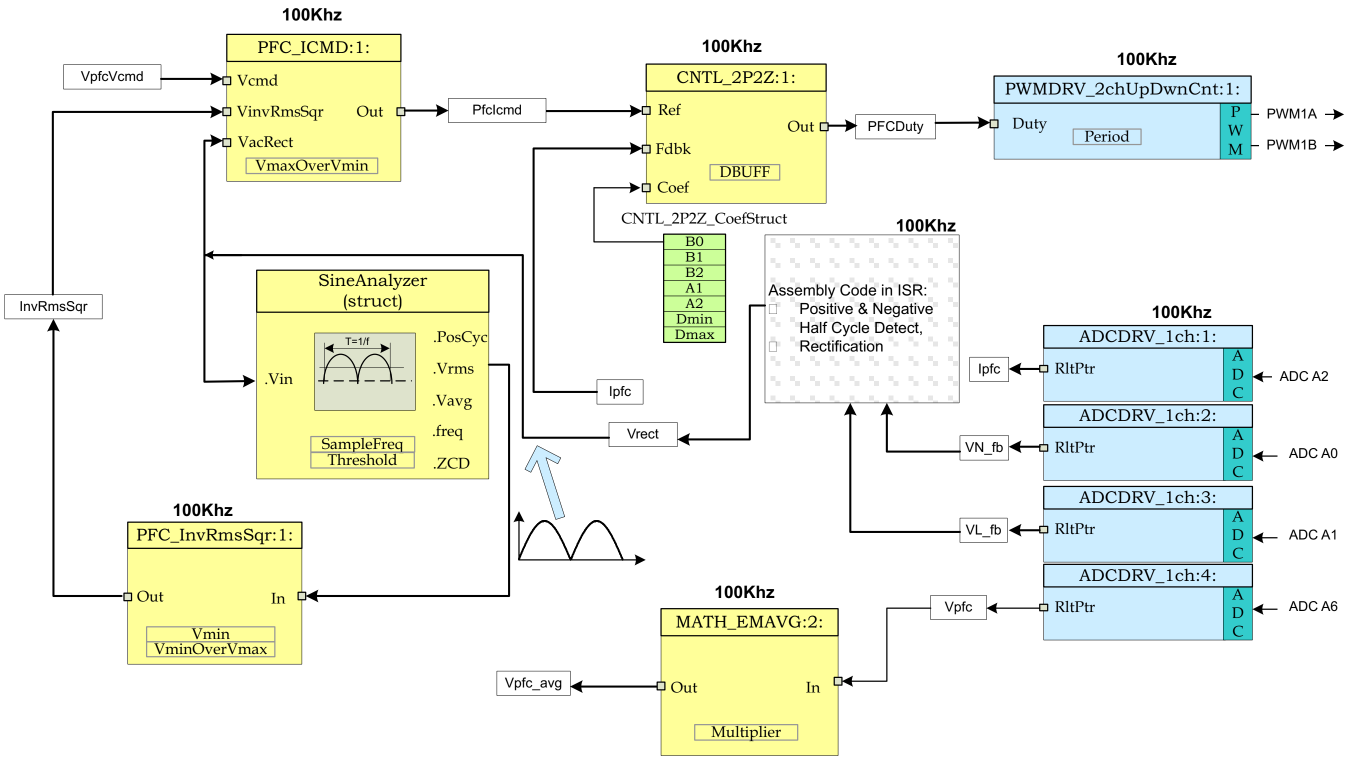


Figure 11. Build 2 Software Blocks

This 2p2z controller requires five control coefficients. These coefficients and the clamped output of the controller are stored as the elements of a structure named CNTL_2P2Z_CoefStruct1. The CNTL_2P2Z block can be instantiated multiple times if the system needs multiple loops. Each instance can have separate set of coefficients. The CNTL_2P2Z instance for the current loop uses the coefficients stored as the elements of structure CNTL_2P2Z_CoefStruct1. This way a second instantiation of CNTL_2P2Z with a different structure, CNTL_2P2Z_CoefStruct2, can be used for PFC voltage loop control, as we will see in Section 3.3.1.4.4 with Build 3.

The controller coefficients can be changed directly by modifying the values for B0, B1, B2, A1, and A2 inside the structure CNTL_2P2Z_CoefStruct1. Alternately, the 2p2z controller can be expressed in PID form and the coefficients can be changed by changing the PID coefficients. The equations relating the five controller coefficients to the three PID gains are given below. For the current loop these P, I and D coefficients are named as: Pgain_I, Igain_I and Dgain_I respectively. For the voltage loop, used in Build 3, these coefficients are named as: Pgain_V, Igain_V and Dgain_V respectively. These coefficients are used in Q26 format.

The compensator block (CNTL_2P2Z) has a reference input and a feedback input. The feedback input labeled as, Fdbk, comes from the ADC. The reference input labeled as, Ref, comes from PFC_ICMD block as mentioned before. The z-domain transfer function for CNTL_2P2Z is given by:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (1)$$

The recursive form of the PID controller is given by the difference equation:

$$u(k) = u(k-1) + b_0 e(k) + b_1 e(k-1) + b_2 e(k-2)$$

where

$$b_0 = K_p + K_i + K_d$$

$$b_1 = -K_p + K_i - 2K_d$$

- $b_2 = K_d$ (2)

And the z-domain transfer function of this PID is:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-1}} \tag{3}$$

Comparing this with the general form, we can see that PID is a special case of CNTL_2P2Z control where:

$$a_1 = -1 \quad \text{and} \quad a_2 = 0 \tag{4}$$

The MATH_EMAVG (Exponential Moving Average) block calculates the average of the output DC bus voltage. The output from this block is used to detect overvoltage condition followed by a PWM shutdown.

3.3.1.4.3 Procedure

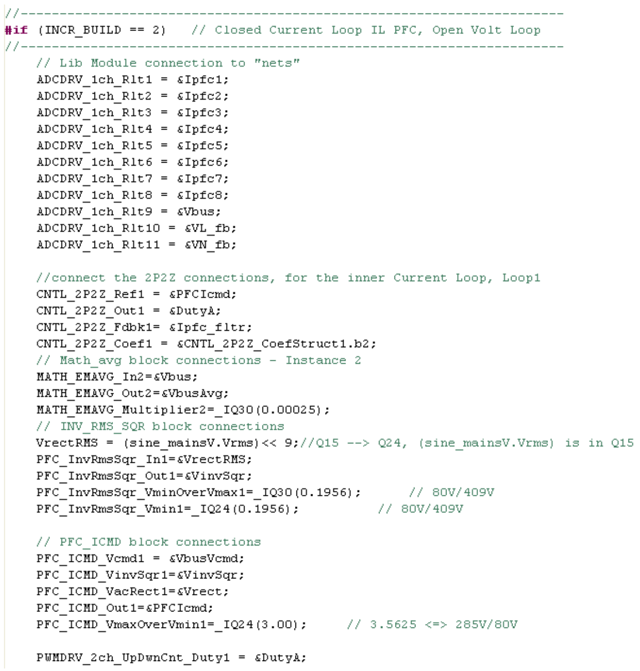
Follow the following steps to execute this build:

3.3.1.4.3.1 Build and Load Project

Follow the steps below to execute this build:

Follow the steps in Section 3.3.1.3.2 exactly as in build 1(Section 3.3.1) except that in Step 6, select build 2 option instead of build 1. Then complete Step 6 as below:

1. Locate and inspect the following code in the main file under initialization code specific for build 2. This is where all the software blocks related to build 2 are connected in the control flow.



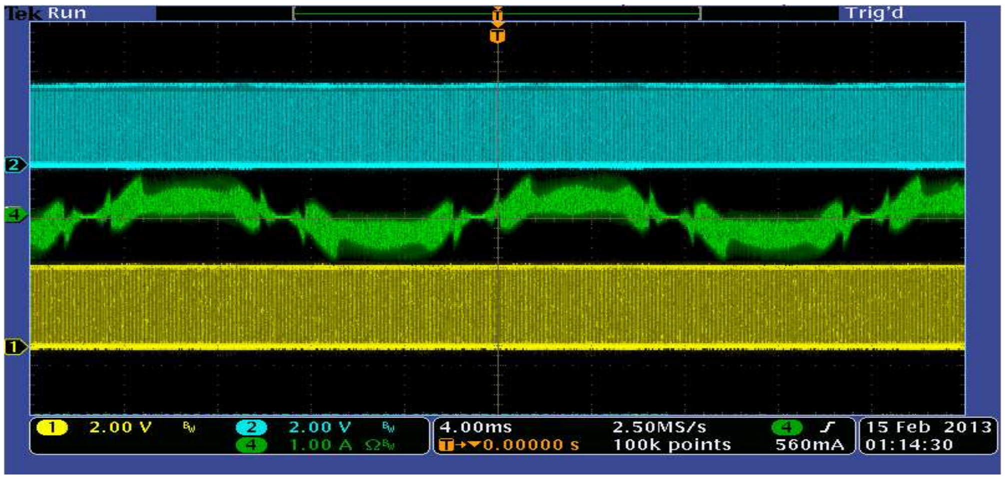
```
//-----------------------------------------------------------------------
#if (INCR_BUILD == 2)   // Closed Current Loop IL PFC, Open Volt Loop
//-----------------------------------------------------------------------
    // Lib Module connection to "nets"
    ADCDRV_1ch_Rlt1 = &Ipfc1;
    ADCDRV_1ch_Rlt2 = &Ipfc2;
    ADCDRV_1ch_Rlt3 = &Ipfc3;
    ADCDRV_1ch_Rlt4 = &Ipfc4;
    ADCDRV_1ch_Rlt5 = &Ipfc5;
    ADCDRV_1ch_Rlt6 = &Ipfc6;
    ADCDRV_1ch_Rlt7 = &Ipfc7;
    ADCDRV_1ch_Rlt8 = &Ipfc8;
    ADCDRV_1ch_Rlt9 = &Vbus;
    ADCDRV_1ch_Rlt10 = &VL_fb;
    ADCDRV_1ch_Rlt11 = &VN_fb;

    //connect the 2P2Z connections, for the inner Current Loop, Loop1
    CNTL_2P2Z_Ref1 = &PFCIcmd;
    CNTL_2P2Z_Out1 = &DutyA;
    CNTL_2P2Z_Fdbk1= &Ipfc_fltr;
    CNTL_2P2Z_Coef1 = &CNTL_2P2Z_CoefStruct1.b2;
    // Math_avg block connections - Instance 2
    MATH_EMAVG_In2=&Vbus;
    MATH_EMAVG_Out2=&VbusAvg;
    MATH_EMAVG_Multiplier2=_IQ30(0.00025);
    // INV_RMS_SQR block connections
    VrectRMS = (sine_mainsV.Vrms)<< 9;//Q15 --> Q24, (sine_mainsV.Vrms) is in Q15
    PFC_InvRmsSqr_In1=&VrectRMS;
    PFC_InvRmsSqr_Out1=&VinvSqr;
    PFC_InvRmsSqr_VminOverVmax1=_IQ30(0.1956);      // 80V/409V
    PFC_InvRmsSqr_Vmin1=_IQ24(0.1956);          // 80V/409V

    // PFC_ICMD block connections
    PFC_ICMD_Vcmd1 = &VbusVcmd;
    PFC_ICMD_VinvSqr1=&VinvSqr;
    PFC_ICMD_VacRect1=&Vrect;
    PFC_ICMD_Out1=&PFCIcmd;
    PFC_ICMD_VmaxOverVmin1=_IQ24(3.00);     // 3.5625 <=> 285V/80V

    PWMDRV_2ch_UpDwnCnt_Duty1 = &DutyA;
```

2. Open and inspect InterleavedPFC-DPL-ISR.asm. Notice the _DPL_Init and _DPL_ISR sections under build 2. This is where all the macro instantiations under build 2 are done for initialization and runtime, respectively.

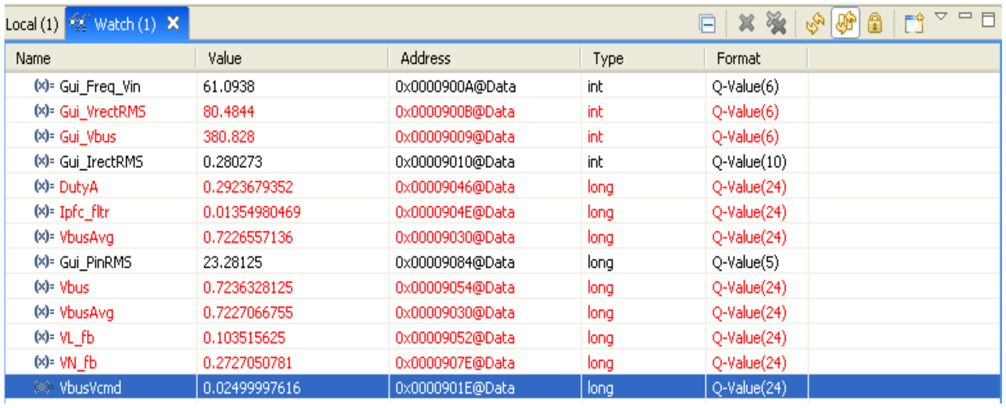
3. Select the Incremental build option as 2 in the *InterleavedPFC-Settings.h* file. Then follow the steps in Section 3.3.1.3.3 as in build 1 in order to run the code. When all these steps are completed you should now be at the start of Main().

NOTE: Whenever you change the incremental build option in *InterleavedPFCSettings.h* always do a "Rebuild All"

4. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target -> Run on the menu bar.
5. In the watch view, add the variable *VpfcVcmd* and set it to 0.025 (=419430 in Q24). This variable sets the magnitude of the reference current command for the current control loop.
6. Apply an appropriate resistive load to the PFC system at the DC output. For example, a 8.0Kohm resistor of 40W rating can be used. This will provide a load of 18W at 380V bus voltage.
7. Slowly apply AC Power to the board from an isolated AC source. Monitor the DC bus voltage as the input voltage is raised slowly to 82V rms. The bus voltage now should be around 380V. Use a current probe to observe the input current. With 82V rms input, 8.0kohm resistive load and bus voltage set to 380V you should see the following waveforms. Here Ch4 is the input current, Ch1/Ch2 are the PWM outputs. With the current loop closed the input current should have the same shape of the input voltage with good power factor.



CCS watch window below shows some of the variables under Build 2 when the input AC voltage is about 82Vrms, VbusVcmd is set to 0.025, DC bus resistive load is about 8K ohm and DC bus voltage is 380V.



Name	Value	Address	Type	Format
Gui_Freq_Vin	61.0938	0x0000900A@Data	int	Q-Value(6)
Gui_VrectRMS	80.4844	0x0000900B@Data	int	Q-Value(6)
Gui_Vbus	380.828	0x00009009@Data	int	Q-Value(6)
Gui_IrectRMS	0.280273	0x00009010@Data	int	Q-Value(10)
DutyA	0.2923679352	0x00009046@Data	long	Q-Value(24)
Ipfc_fltr	0.01354980469	0x0000904E@Data	long	Q-Value(24)
VbusAvg	0.7226557136	0x00009030@Data	long	Q-Value(24)
Gui_PinRMS	23.28125	0x00009084@Data	long	Q-Value(5)
Vbus	0.7236328125	0x00009054@Data	long	Q-Value(24)
VbusAvg	0.7227066755	0x00009030@Data	long	Q-Value(24)
VL_fb	0.103515625	0x00009052@Data	long	Q-Value(24)
VN_fb	0.2727050781	0x0000907E@Data	long	Q-Value(24)
VbusVcmd	0.02499997616	0x0000901E@Data	long	Q-Value(24)

8. Increase *VpfcVcmd* slightly (in steps of 0.002) and observe the bus voltage settles to a higher value. Increasing *VpfcVcmd* increases the magnitude of the current reference signal and, since the PFC voltage loop is open, the bus voltage will rise. Therefore, apply caution and set the overvoltage protection threshold to 400V. This threshold parameter is labeled as VBUS_OVP_THRSHLD inside the file *InterleavedPFC-Settings.h*. The default value is set to 430V. Now change the input voltage or the load resistance to see the PFC operation under current control loop.
9. Fully halting the MCU when in real-time mode is a two-step process. With the AC input turned off wait until the DC bus capacitor is fully discharged. First, halt the processor by using the Halt button on the toolbar, or by using Target -> Halt. Then take the MCU out of real-time mode. Finally reset the MCU.
10. You may choose to leave Code Composer Studio running for the next exercise or optionally close CCS.
11. End of Exercise

3.3.1.4.4 Build 3: IL PFC with Closed Voltage and Current Loop

3.3.1.4.4.1 Objective

The objective of this build is to verify the operation of the complete IL PFC project from the CCS environment.

3.3.1.4.4.2 Overview

Figure 12 shows the software blocks used in this build. Compared to build 2 in Figure 11 this build uses an additional 2p2z control block labeled as CNTL_2P2Z:2. This is the 2nd instantiation of the 2p2z control block in order to implement the IL PFC voltage loop control. This voltage loop controller is executed at 50kHz rate which is half the rate for current loop. The output from this control block drives the input node VpfcVcmd of the PFC_ICMD block. This is the main difference compared to build 2 where VpfcVcmd is updated by user from CCS watch window in an open voltage loop mode.

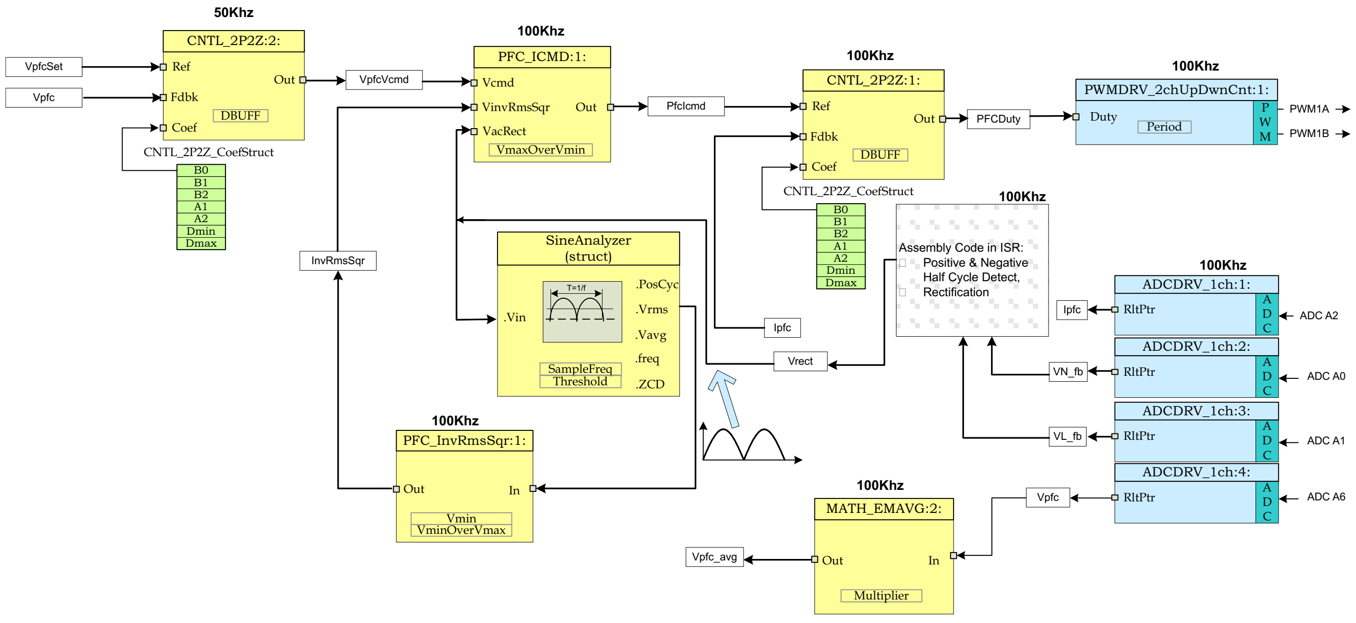


Figure 12. Build 3 Software Blocks

Similar to current loop controller, this voltage loop controller, CNTL_2P2Z:2, also requires five control coefficients. These coefficients and the clamped output of the controller are stored as the elements of a 2nd structure named CNTL_2P2Z_CoefStruct2. The coefficients for this controller can be changed directly by modifying the values for B0, B1, B2, A1, and A2 inside the structure CNTL_2P2Z_CoefStruct2, or by changing the equivalent PID gains as discussed in Section 3.3.1.4.

3.3.1.4.4.2.1 Start-Up, Inrush Current Control, and Slew-Limit

At start-up, the controller monitors the PFC DC bus voltage. When this voltage reaches a minimum level (this min level for this ILPFC EVM is set around 160Vdc) PFC action is enabled and the output DC bus slowly ramps up to the pre-set value of about 390Vdc. This output voltage level is set by the constant VBUS_RATED_VOLTS defined in the PFC header file *InterleavedPFC-Settings.h*. The ramp up speed is set by the parameter *VbusSlewRate* defined and implemented in the soft-start state machine task C2. This part of the software can be quickly modified to implement any other desired mode for PFC start-up.

3.3.1.4.4.2.2 Input Power Monitor

This ILPFC EVM has been equipped with input power monitoring feature. All the calculation for input power measurement is done inside a 10kHz ISR running the SineAnalyzer macro (Figure 12). This macro block takes the rectified input voltage and the rectified input current as its inputs. Then based on this information it calculates the RMS input power, RMS input voltage and RMS input current using the following equations:

$$V_{rms} = \sqrt{[\frac{1}{T}\int_{t}^{t+T} v^2(t)dt]} = \frac{1}{N}\sqrt{\sum_{n=1}^{N} V^2(n)}$$

$$N = T / T_s$$

$$I_{rms} = \sqrt{[\frac{1}{T}\int_{t}^{t+T} i^2(t)dt]} = \frac{1}{N}\sqrt{\sum_{n=1}^{N} I^2(n)}$$

$$P_{rms} = \frac{1}{T}\int_{t}^{t+T} [v(t)\cdot i(t)]dt = \frac{1}{N}\sum_{n=1}^{N} V(n)\cdot I(n) \tag{5}$$

The time period T of the rectified input voltage and current is calculated by setting a low threshold voltage level for the rectified voltage signal and then by measuring the time between the consecutive points when the signal crosses this threshold level. This is illustrated in Figure 13.

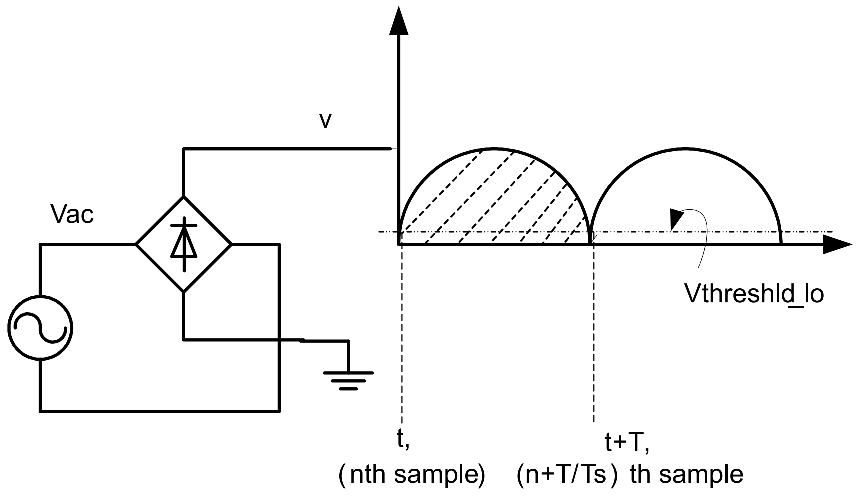


Figure 13. RMS Input Voltage and Current Calculation Scheme

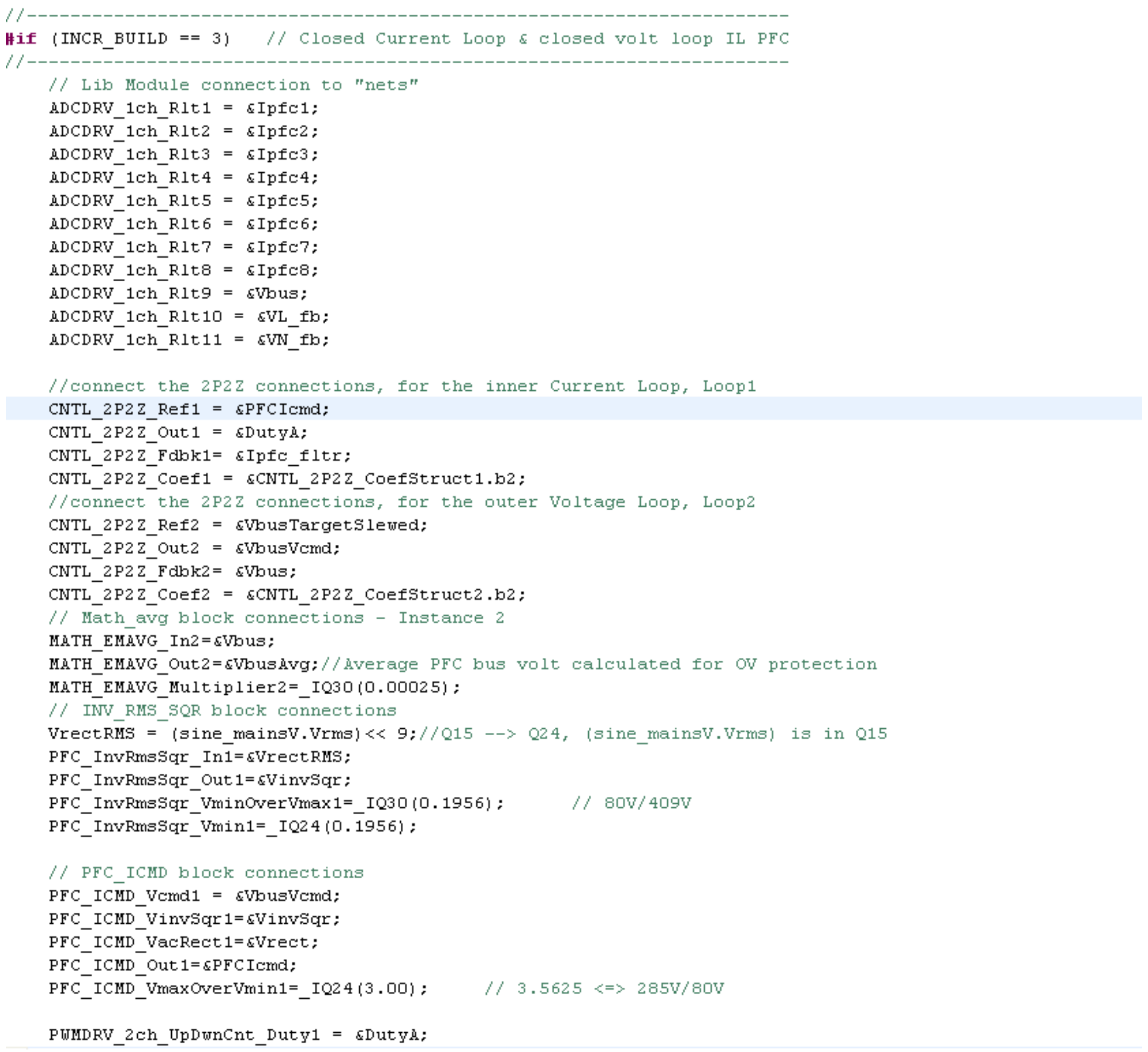
3.3.1.4.4.2.3 Procedure

3.3.1.4.4.2.4 Build and Load Project

Follow the steps below to execute this build.

Follow the steps in Section 3.3.1.3.1 exactly as in build 1(Section 3.3.1) except that in Step 6 select build 3 option instead of build 1. Then complete Step 6 as below:

1. Locate and inspect the following code in the main file under initialization code specific for build 3. This is where all the software blocks related to build 3 are connected in the control flow.



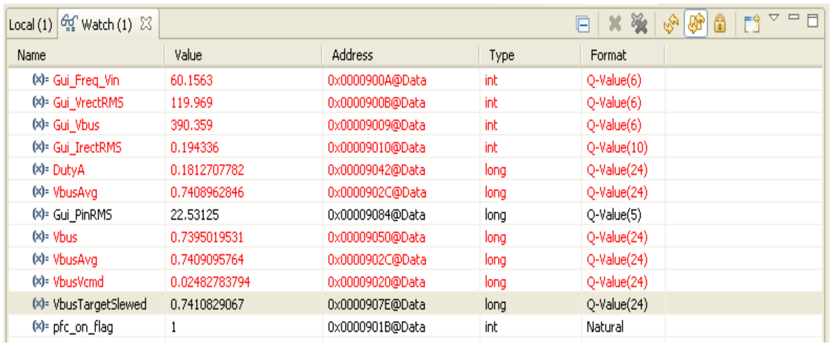
```
//----------------------------------------------------------------------
#if (INCR_BUILD == 3)   // Closed Current Loop & closed volt loop IL PFC
//----------------------------------------------------------------------
    // Lib Module connection to "nets"
    ADCDRV_1ch_Rlt1 = &Ipfc1;
    ADCDRV_1ch_Rlt2 = &Ipfc2;
    ADCDRV_1ch_Rlt3 = &Ipfc3;
    ADCDRV_1ch_Rlt4 = &Ipfc4;
    ADCDRV_1ch_Rlt5 = &Ipfc5;
    ADCDRV_1ch_Rlt6 = &Ipfc6;
    ADCDRV_1ch_Rlt7 = &Ipfc7;
    ADCDRV_1ch_Rlt8 = &Ipfc8;
    ADCDRV_1ch_Rlt9 = &Vbus;
    ADCDRV_1ch_Rlt10 = &VL_fb;
    ADCDRV_1ch_Rlt11 = &VN_fb;

    //connect the 2P2Z connections, for the inner Current Loop, Loop1
    CNTL_2P2Z_Ref1 = &PFCIcmd;
    CNTL_2P2Z_Out1 = &DutyA;
    CNTL_2P2Z_Fdbk1= &Ipfc_fltr;
    CNTL_2P2Z_Coef1 = &CNTL_2P2Z_CoefStruct1.b2;
    //connect the 2P2Z connections, for the outer Voltage Loop, Loop2
    CNTL_2P2Z_Ref2 = &VbusTargetSlewed;
    CNTL_2P2Z_Out2 = &VbusVcmd;
    CNTL_2P2Z_Fdbk2= &Vbus;
    CNTL_2P2Z_Coef2 = &CNTL_2P2Z_CoefStruct2.b2;
    // Math_avg block connections - Instance 2
    MATH_EMAVG_In2=&Vbus;
    MATH_EMAVG_Out2=&VbusAvg;//Average PFC bus volt calculated for OV protection
    MATH_EMAVG_Multiplier2=_IQ30(0.00025);
    // INV_RMS_SQR block connections
    VrectRMS = (sine_mainsV.Vrms)<< 9;//Q15 --> Q24, (sine_mainsV.Vrms) is in Q15
    PFC_InvRmsSqr_In1=&VrectRMS;
    PFC_InvRmsSqr_Out1=&VinvSqr;
    PFC_InvRmsSqr_VminOverVmax1=_IQ30(0.1956);      // 80V/409V
    PFC_InvRmsSqr_Vmin1=_IQ24(0.1956);

    // PFC_ICMD block connections
    PFC_ICMD_Vcmd1 = &VbusVcmd;
    PFC_ICMD_VinvSqr1=&VinvSqr;
    PFC_ICMD_VacRect1=&Vrect;
    PFC_ICMD_Out1=&PFCIcmd;
    PFC_ICMD_VmaxOverVmin1=_IQ24(3.00);     // 3.5625 <=> 285V/80V

    PWMDRV_2ch_UpDwnCnt_Duty1 = &DutyA;
```

2. Open and inspect *InterleavedPFC-DPL-ISR.asm*. Notice the _DPL_Init and _DPL_ISR sections under build 3. This is where all the macro instantiations under build 3 are done for initialization and runtime, respectively.
3. Select the Incremental build option as 3 in the *InterleavedPFC-Settings.h* file. Then follow steps in Section 3.3.1.3.3 and Section 3.3.1.3.5 as in build 1 in order to run the code. When all these steps are completed you should now be at the start of Main().

NOTE: Whenever you change the incremental build option in *InterleavedPFC-Settings.h* always do a "Rebuild All"

4. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target -> Run on the menu bar.

5. In the watch view, add the variables *VbusTargetSlewed*, *Vbus* and set the Q-format to Q24. These variables represent the p.u. reference bus voltage and the feedback bus voltage respectively (normalized or per unit values). These will slowly increase to the setpoint value as the PFC starts up when AC power is applied.
6. Apply an appropriate resistive load to PFC output. For example, a 8.0Kohm resistor of 40W rating can be used. This will provide a load of about 20W at 390V bus voltage. Configure the isolated AC source to output 120V, 60Hz, AC voltage output. Use a voltmeter to monitor the DC bus voltage. Turn on the AC source output for 120Vrms. When the DC bus voltage reaches 160V, PFC action will start and the bus voltage will slowly increase to about 390V. Notice that VbusTargetSlewed and Vbus variables on the watch window show a value of about 0.7414 (=390/526) when the Q format is set to Q24. The maximum bus voltage set by the Vbus sense resistors is about 526V that corresponds to maximum ADC input of 3.3V. Therefore, the normalized or per unit value will be about 0.7414 when the actual bus voltage is 390Vdc. CCS watch window below shows some of the variables under Build 3 when the input AC voltage is about 120Vrms, DC bus resistive load is about 8K ohm and DC bus voltage is about 390V. The variable “pfc_on_flag” is set to 1 when the soft-start is complete and PFC bus voltage reaches the desired set point. Use an oscilloscope with voltage and current probes to observe the input voltage and input current. With 110Vrms input, 277 ohm resistive load and bus voltage set to 390V you should see the waveforms shown below. Here Ch2 is the input voltage and Ch4 is the input current. Change the input voltage (100Vrms~260Vrms) or the load resistance (0~550W @110Vin, or, 0~700W @220Vin) to see the PFC operation under closed current and voltage control loop.



Name	Value	Address	Type	Format
Gui_Freq_Vin	60.1563	0x0000900A@Data	int	Q-Value(6)
Gui_VrectRMS	119.969	0x0000900B@Data	int	Q-Value(6)
Gui_Vbus	390.359	0x00009009@Data	int	Q-Value(6)
Gui_IrectRMS	0.194336	0x00009010@Data	int	Q-Value(10)
DutyA	0.1812707782	0x00009042@Data	long	Q-Value(24)
VbusAvg	0.7408962846	0x0000902C@Data	long	Q-Value(24)
Gui_PinRMS	22.53125	0x00009084@Data	long	Q-Value(5)
Vbus	0.7395019531	0x00009050@Data	long	Q-Value(24)
VbusAvg	0.7409095764	0x0000902C@Data	long	Q-Value(24)
VbusVcmd	0.02482783794	0x00009020@Data	long	Q-Value(24)
VbusTargetSlewed	0.7410829067	0x0000907E@Data	long	Q-Value(24)
pfc_on_flag	1	0x0000901B@Data	int	Natural

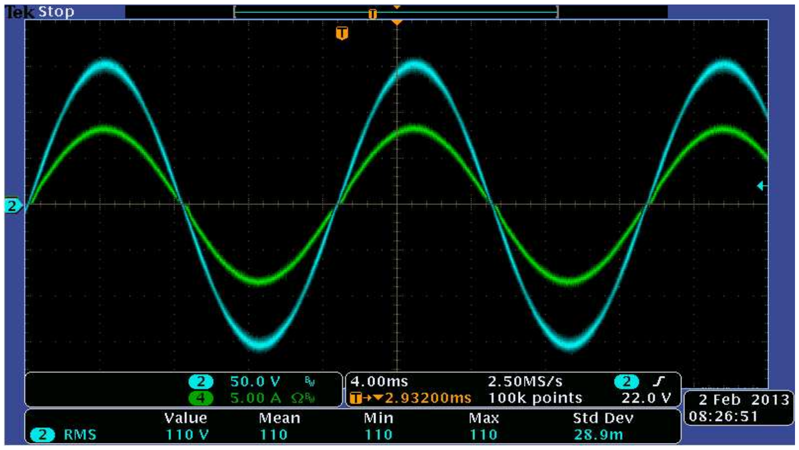
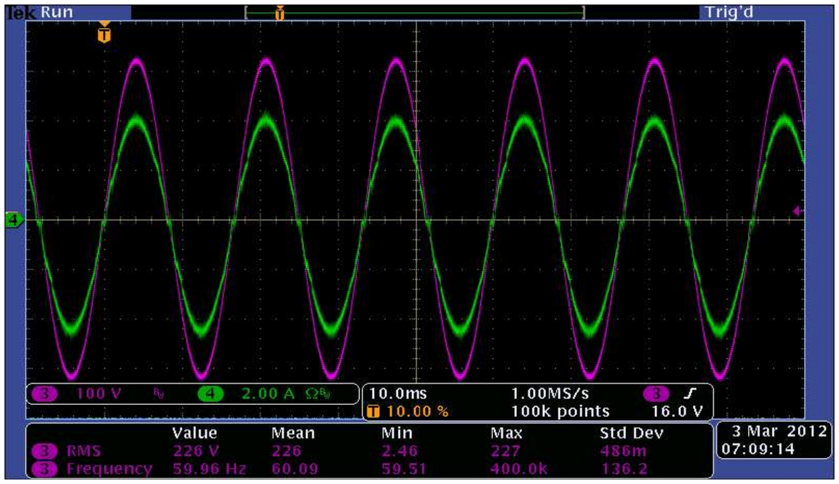


Figure 14. IL PFC Input Voltage and Current at 550W Load and 110Vrms Input

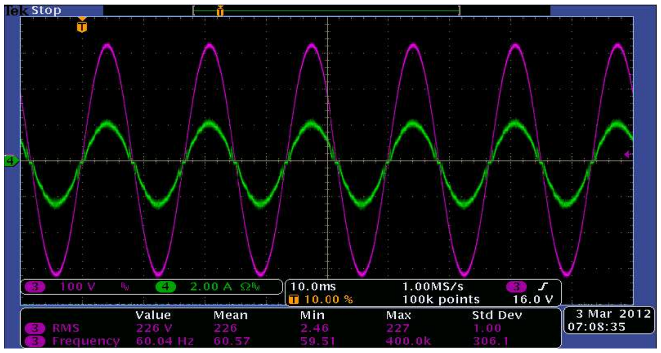
7. Fully halting the MCU when in real-time mode is a two-step process. With the AC input turned off wait until the DC bus capacitor is fully discharged. First, halt the processor by using the Halt button on the toolbar, or by using Target -> Halt. Then take the MCU out of real-time mode. Finally reset the MCU.
8. You may choose to leave Code Composer Studio running for the next exercise or optionally close CCS.
9. End of Exercise

3.3.1.4.4.3 Additional Test Results



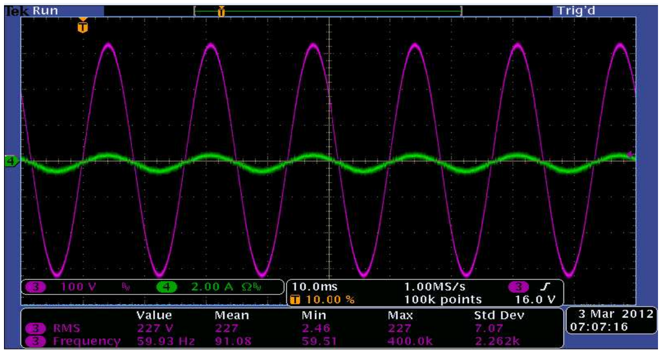
(1) Ch3 -Vin, Ch4-Iin, Vrms=230V, Vbus=390V, Pout=700W

Figure 15. IL PFC Input Voltage and Current Waveforms



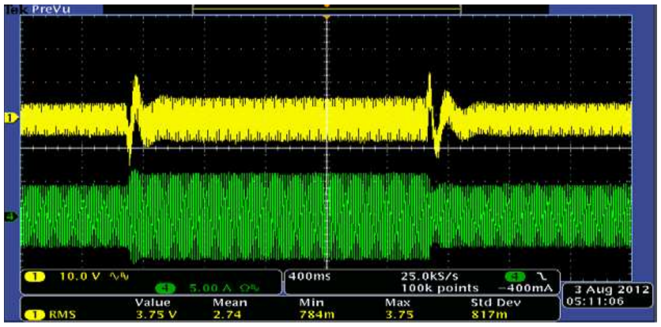
(1) Ch3 –Vin, Ch4-Iin, Vrms=230V, Vbus=390V, Pout=375W

Figure 16. IL PFC Input Voltage and Current Waveforms



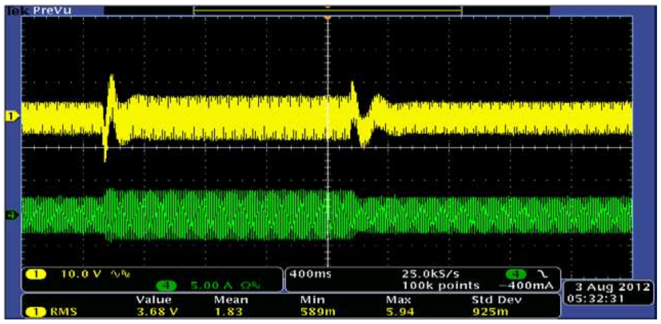
(1) Ch3-Vin, Ch4-Iin, Vrms = 230V, Vbus = 390V, Pout = 80W

Figure 18. IL PFC Input Voltage and Current Waveforms



(1) Ch1-Vbus, Ch4-Iin, Vrms = 120V, Vbus = 390V, Load Step 360W ~ 520W

Figure 17. IL PFC DC Bus Load Transient Response



(1) Ch1-Vbus, Ch4-Iin, Vrms = 220V, Vbus = 390 V, Load Step 370W ~ 530W

Figure 19. IL PFC DC Bus Load Transient Response

4 References

For more information, please refer to the following guides:

ILPFC-GUI_QSG— A quick-start guide for quick demo of the ILPFC EVM using a GUI interface.
..\controlSUITE\development_kits\ILPFC\~Docs\ILPFC-GUI_QSG.pdf

ILPFC_Rel-1.0-HWdevPkg— A folder containing various files related to the Piccolo-B controller card schematics and the IL PFC schematic.
..\controlSUITE\development_kits\ILPFC\ILPFC_HWDevPkg

F28xxx User's Guides— http://www.ti.com/f28xuserguides