

# **Power Factor Correction on UCC28070EVM Using C2000 Piccolo-A**

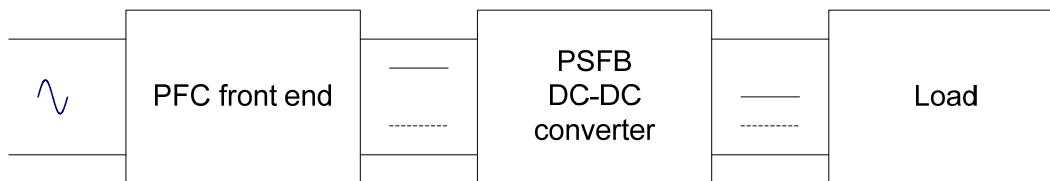
*CCS User Guide  
Version 1.1  
June 2010*

# Implementing Power Factor Correction with C2000 Micro-controllers

## Abstract

This document presents the implementation details of a digitally controlled Power Factor Correction (PFC) system. This forms the first stage of a rectifier system. The 300W UCC28070EVM analog reference design system is used to provide a means to learn and experiment digital control with C2000 controllers. Instead of using a UCC28070 control card to control this EVM, a C2000 Piccolo-A control card is used here. This provides for an easy way to control the same power stage using an analog or a digital solution from Texas Instruments.

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 stage has become an integral part of most rectifier designs. This stage forms the front end of a rectifier system as shown in Fig 1.



*Figure 1. System Block Diagram*

The phase shifted full bridge converter is used in DC-DC conversion, for example in telecom systems to convert a high voltage bus to an intermediate distribution voltage, typically closer to 48V. The phase shifted full bridge (PSFB) stage provides the voltage translation as well as the isolation from the line voltage, since the PSFB topology includes a transformer. This stage forms the second stage of the rectifier system as shown in Fig 1. This document focuses on the PFC stage.

The transition to digital power control means that functions that are previously implemented in hardware are now implemented in software. In addition to the intelligence and flexibility this adds to the system, this simplifies the system considerably.

This paper presents the hardware design and its corresponding software to control a two-phase interleaved power factor correction front end. It also implements a method to improve the input power factor (p.f.) and total harmonic distortion (T.H.D.) under low load and/or High Line conditions.

# 1 Introduction

The function of a PFC stage (sometimes called a pre-regulator) is to generate an intermediate DC bus from the AC source while drawing a sine wave input current. We begin to examine the functions of such a system in some detail.

## 1.1 Motivation

Of course, the question as to why a PFC stage is needed has not been answered yet. To understand this, a typical AC-Rectifier Input Circuit is shown in Fig 2a. For simplicity, consider the load being resistive. A sketch of the bridge rectifier input current (Fig 2b) shows that this is dominated by pulses that correspond to the time when the output DC bus capacitor is being charged. A Fourier analysis of the waveform shows the presence of harmonics in the line current. In a practical rectifier, the DC-DC converter stage acts as a load to the PFC stage. This represents a non-linear load and will draw harmonic currents from the line.

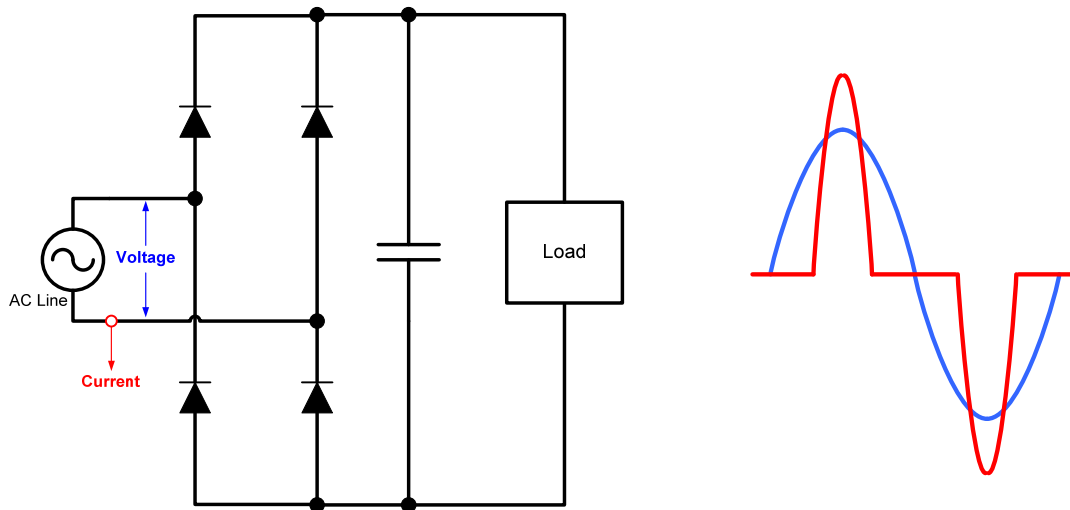


Figure 2. a) Typical AC Rectifier Input Circuit b) Input current waveform

These harmonics present a two-fold difficulty. Firstly they represent reactive power, which results in losses, and secondly, such currents can distort the line voltage. To avoid these undesirable effects, PFC attempts to draw a sinusoidal current from the line that is exactly in phase with the line voltage.

## 1.2 PFC Stage Implementation on the ACDC Board

At first, passive PFC solutions were used to achieve PFC, especially in higher power applications. However, these solutions required bulky 50/60 Hz Iron core inductors. These were replaced by Active PFC solutions that employed boost regulators and average current mode control techniques for wave shaping the input current. The boost converter switches at a much higher frequency (tens/hundreds of KHz) compared to the line frequency reducing the input filter and the boost inductor size drastically. Further improvements may be obtained by implementing an interleaved boost converter topology. This reduces the input current ripple, increases efficiency, reduces stress on power electronic components and also results in reduction in the overall inductor volume.

Fig 3 shows the simplified block diagram of the PFC circuit implemented here. The input AC line is passed through an EMI filter, and then through a bridge rectifier. The next stage is a two phase interleaved boost converter that boosts the AC line voltage, to the DC bus voltage. Inductor L1, MOSFET switch Q1 and diode D1, together, form phase A of the boost circuit, while L2, Q2 and D2 form the phase B. Finally a capacitor at the boost converter output acts as an energy reservoir, which reduces the output voltage ripple.

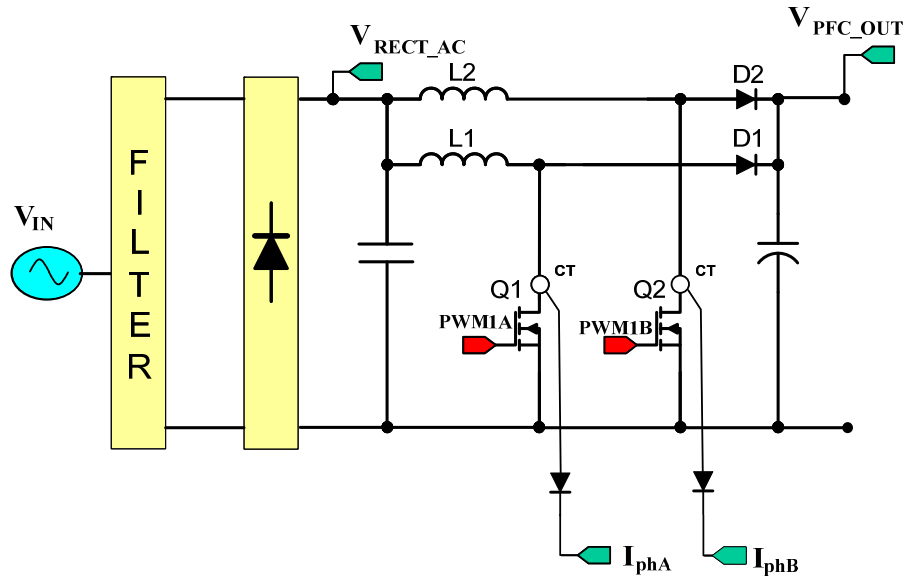


Figure 3. Two phase Interleaved PFC

The control algorithm is implemented on a C2000 micro-controller (MCU). The MCU interacts with the hardware by way of the feedback signals and the PWM outputs. The quantities that are sensed and fed back to the MCU include, the rectified input AC voltage ( $V_{RECT\_AC}$ ), the two PFC phase currents ( $I_{phA}$ ,  $I_{phB}$ ), and the boost stage output voltage ( $V_{PFC\_OUT}$ ), as shown in Fig 3. The duty cycle of the PWM outputs driving the two switches determines the amount of boost imparted by the boost converter. This is the controlled parameter.

MCU achieves PFC by controlling this duty cycle in a way such that the rectified input current, which is nothing but a numerical addition of the two phase currents, is made to follow the rectified input voltage while providing load and line regulation at the same time.

This is made possible by the high frequency PWM modules, an equally high control loop frequency and a 12-bit on chip ADC available on C2000 Micro-controllers. A detailed description of the software algorithm is provided in the following chapters.

### 1.3 PFC Electrical Specifications

Following lists the key highlights of the UCC28070 EVM.

- Input Voltage (AC Line): 85V (Min) to 265V (Max)
- 390Vdc Output
- 300 Watts Output Power
- Full Load efficiency greater than 90%
- Power factor at 50% or greater load – 0.9 (Min)
- 2-phase Interleaved Boost Circuit Topology
- 200kHz switch frequency, each phase
- Phase-shedding capable

## 2 Software overview

### 2.1 Software Control Flow

The PFC2PHIL project makes use of the “C-background/ASM-ISR” framework. It uses C-code as the main supporting program for the application, and is responsible for all system management tasks, decision making, intelligence, and host interaction. The assembly code is strictly limited to the ISR (Interrupt Service Routine), which runs all the critical control code and typically this includes ADC reading, control calculations, and PWM updates. Fig 4 depicts the general software flow for this project.

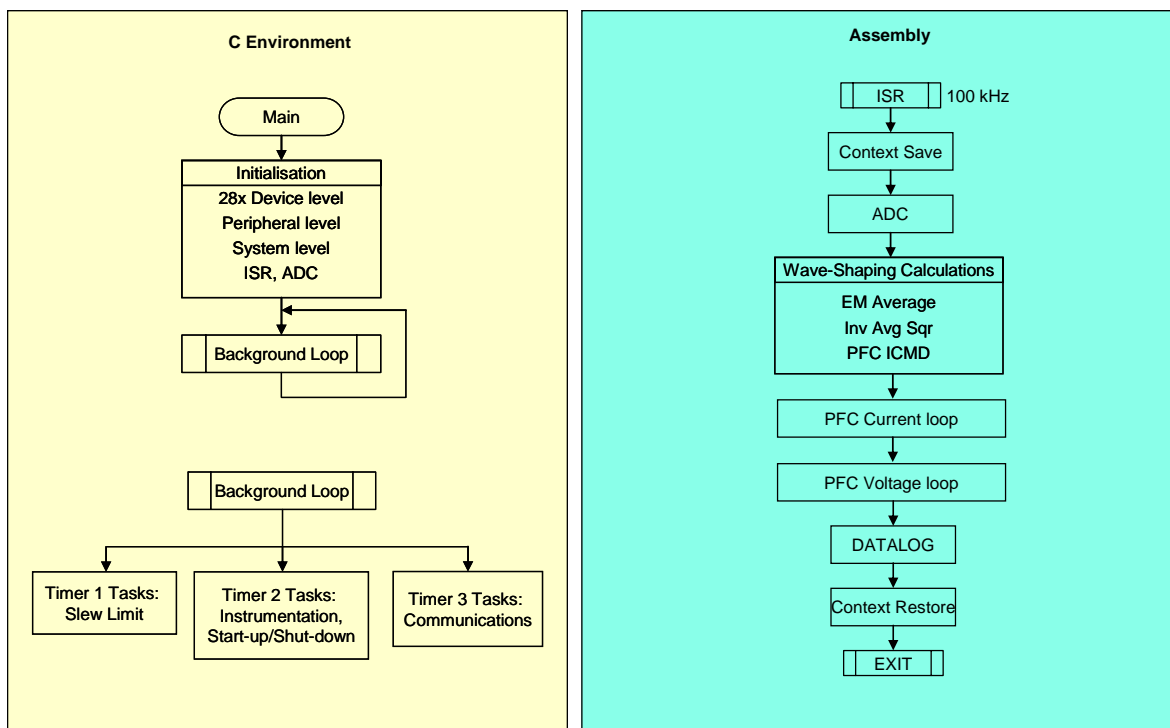


Figure 4. Software flow

The key framework C files used in this project are:

PFC2PHIL-Main.c – this file is used to initialize, run, and manage the application. This is the “brains” behind the application.

PFC2PHIL-DevInit\_F2802x.c – this file is responsible for a one time initialization and configuration of the F2802x device, and includes functions such as setting up the clocks, PLL, GPIO, etc.

The ISR consists of a single file:

PFC2PHIL-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 (typically) at the same rate as the PWM time-base used to trigger it.

The Power Library functions (modules) are “called” from this 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:

<b>C configure function</b>	<b>ASM initialization macro</b>	<b>ASM run-time macro</b>
PWM_PFC2PhiL_Cnf.c	PWMDRV_PFC2PhiL_INIT n	PWMDRV_PFC2PhiL n
ADC_SOC_Cnf.c	ADCDRV_4CH_INIT m,n,p,q	ADCDRV_4CH m,n,p,q
	PFC_INVSQR_INIT n	PFC_INVSQR n
	MATH_EMAVG_INIT n	MATH_EMAVG n
	PFC_ICMD_INIT n	PFC_ICMD n
	CNTL_2P2Z_INIT n	CNTL_2P2Z n
	DLOG_4CH_INIT n	DLOG_4CH n

*Table 1. Library Modules*

The control blocks can also be represented graphically as below. (Fig 5)

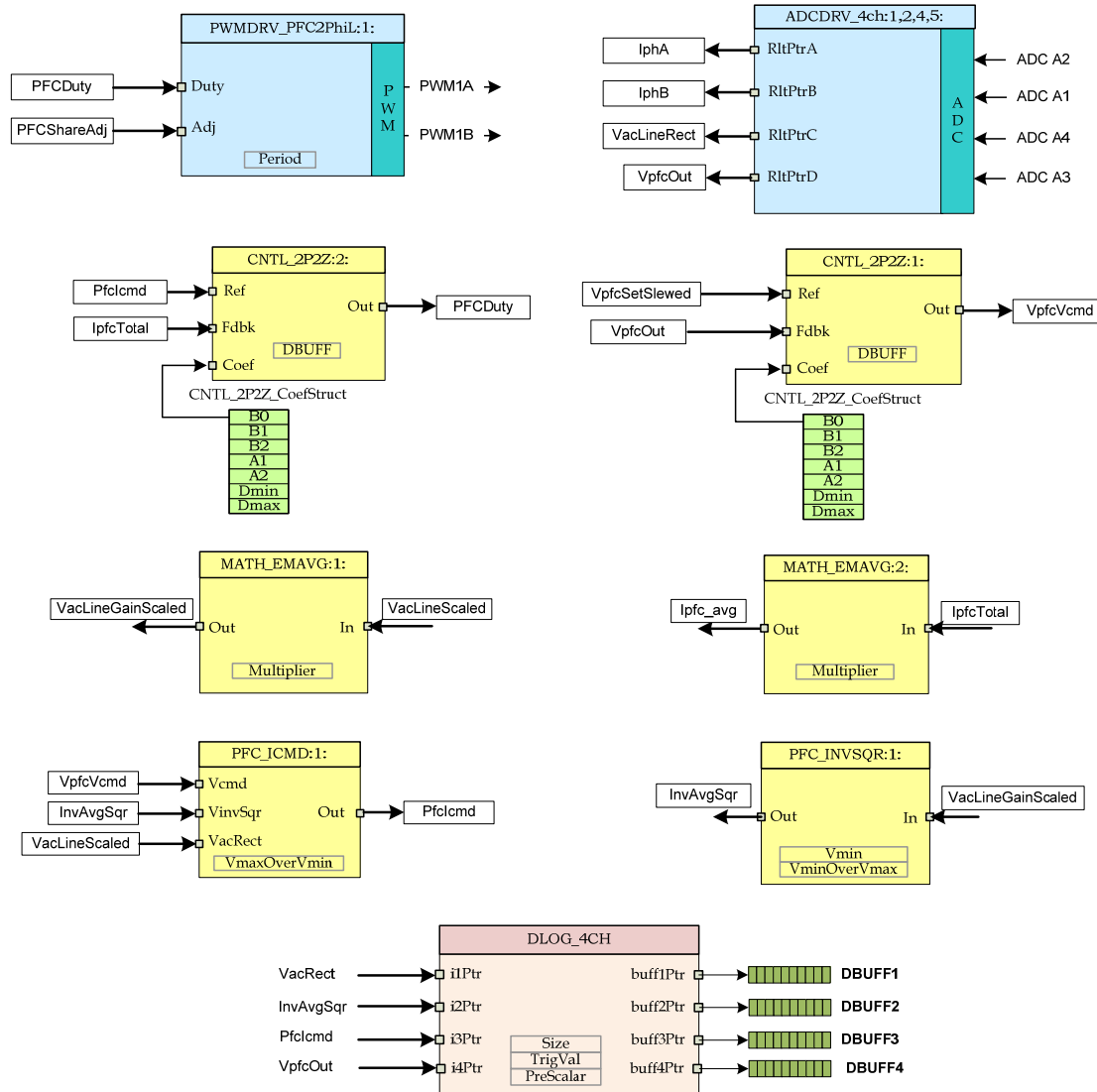
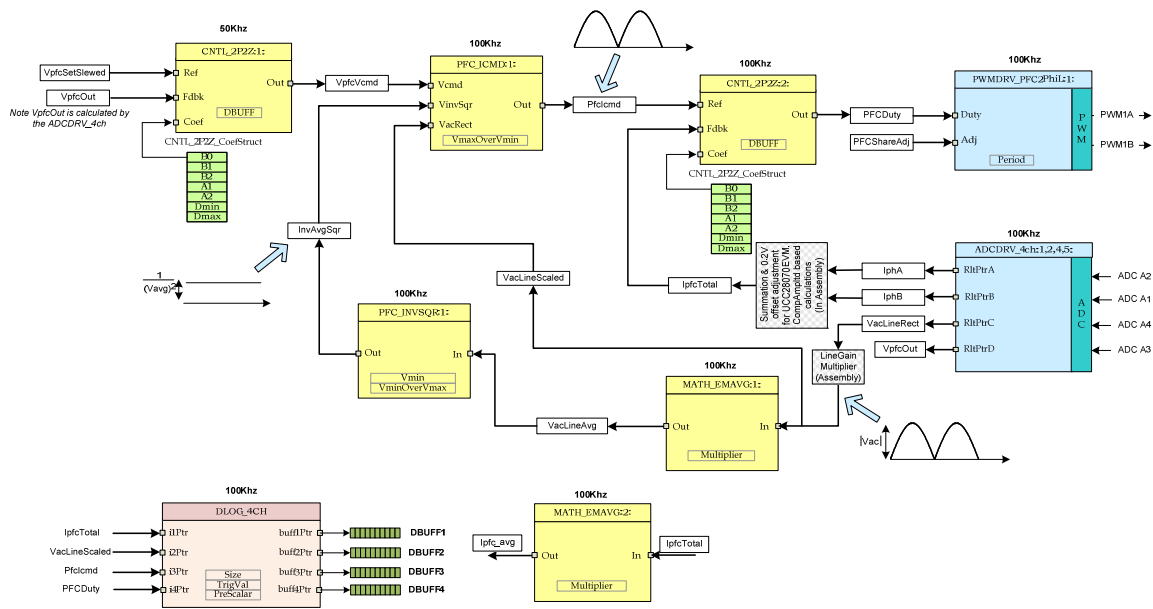


Figure 5. Software blocks

Notice the color coding for the modules in Fig 5. The blocks in 'dark blue' represent the hardware modules on the C2000 controller. The blocks in 'blue' are the software drivers for these modules. The blocks in 'yellow' are part of the mathematical processing carried out on various signals. Although a 2-pole 2-zero controller is used here, the controller could very well be a PI/PID, a 3-pole 3-zero or any other controller that can be suitably implemented for this application. Also notice the purple colored block used for logging data of up to four system parameters at a time. Such a modular library structure makes it convenient to visualize and understand the complete system software flow as shown in Fig 6. It also allows for easy use and additions/deletions of various functionalities. This fact is amply demonstrated in this project by implementing an incremental build approach. This is discussed in more detail in the next section.



The system is controlled by two feedback loops. The outer voltage loop regulates the output DC voltage, while a faster inner current loop wave shapes the input current so as to maintain a high power factor at the input. Fig 6 also gives the rate at which the control blocks are executed. For example, the current controller is executed at a rate of 100 kHz (half of the PWM switching frequency) while the voltage controller is executed 2x slower. The sensed signals, which were shown in Fig 3,  $V_{RECT\_AC}$ ,  $I_{phA}$ ,  $I_{phB}$  and  $V_{PFC\_OUT}$  are properly conditioned and fed back to the 4 ADC channels of the C2000 controller. Following explains the control implemented above.

Although it has not been implemented here, a phase management block can be added to this control flow. This provides a means to share the load between the two phases and facilitates ‘phase shedding’ to increase system efficiencies at low loads. In the implemented software, the control command is equally shared between the two phases by driving equal duty cycles for the two switches Q1 and Q2, thereby closely sharing the load between the two phases.



## 2.2 Incremental Builds

This project is divided into two incremental builds. This makes it easier to learn and get familiar with the board and the software. This approach is also good for debugging/testing boards.

The build options are shown below. To select a particular build option, set the macro INCR\_BUILD, found in the PFC2PHIL-Settings.h file, to the corresponding build selection as shown below. 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.

Incremental build options for PFC	
INCR_BUILD = 1	Open loop check for boost action and ADC feedback (Check sensing circuitry)
INCR_BUILD = 2	Full PFC

*Table 2. Incremental build options for PFC*

### 3 Procedure for running the incremental builds

The main source files, ISR assembly file and the project file for C framework to bring up the PFC system are located in the `..\controlSUITE\development_kits\HVPFC2PhiL\PFC2PhiL` directory. The projects included with this software are targeted for CCSv4.

#### Warning

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 isolation transformer should be connected between the AC source and the unit. Before AC power is applied to the board a voltmeter and an appropriate resistive or electronic load must be attached to the output. The unit should never be handled when the power is applied to it or when the output voltage is greater than 50V. There is no overvoltage protection implemented on the board and therefore zero load or very low load operation is highly discouraged.

Follow the steps below to build and run the example included in the PFC software.

#### 3.1 Build 1: Open loop boost with ADC measurements

##### ➤ Objective

The objective of this build is to evaluate the 2 phase interleaved PFC PWM and ADC driver modules, verify the PFC MOSFET driver circuit and sensing circuit on the board and become familiar with the operation of Code Composer Studio (CCS). Since this system is running open-loop, the ADC measured values are only used for instrumentation purposes in this build. Steps required to build and run a project will be explored.

##### ➤ Overview

The software in Build1 has been configured so that the user can quickly evaluate the 2 phase interleaved PFC PWM driver module by viewing the various waveforms on a scope and observing the effect of change in duty cycle on the output voltage by interactively adjusting the duty cycle on CCS. Additionally, the user can evaluate the 2 phase interleaved PFC ADC driver module by viewing the ADC sampled data in the watch view and also by using the CCS graphical capabilities.

The PWM and ADC driver macro instantiations are executed inside the `_DPL_ISR` as mentioned in the previous chapter. Fig 7 shows the software blocks used in this build. The two PWM signals for the two phases are obtained from ePWM module 1. ePWM1A drives phase1 of the PFC while ePWM1B drives phase 2. The quantities that are sensed and fed back to the MCU include, the rectified input AC voltage ( $V_{RECT\_AC}$ ), the two PFC phase currents ( $I_{phA}$ ,  $I_{phB}$ ), and the boost stage output voltage ( $V_{PFC\_OUT}$ ), as was shown in Fig 3. These quantities are read using the ADC driver module and are shown as `VacLineRect`, `IphA`, `IphB` and `Vpfcout` in Fig 7. The ADC driver module converts the 12-bit ADC result to a Q24 value. A few lines of code in the ISR adds the two individual PFC phase currents and also accounts for the 0.2V offset on the UCC28070EVM to give the total PFC input current.

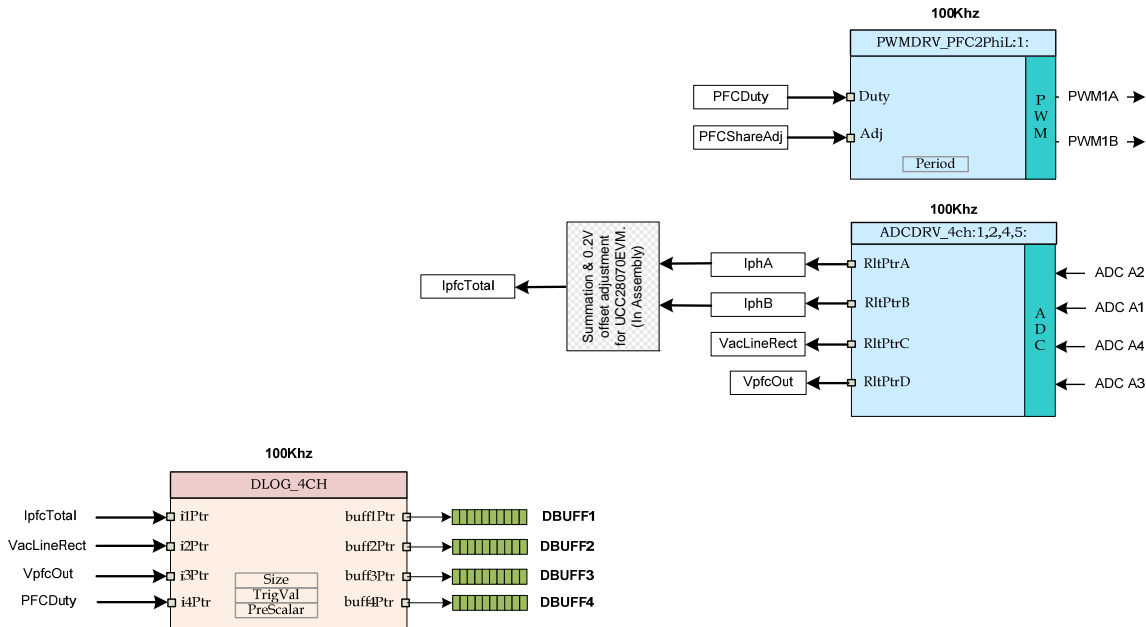


Figure 7. Build 1 software blocks

The PWM signals need to be generated at a frequency of 200 kHz i.e. a period of 5 us. With the controller operating at 60 MHz, 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 an up-down count mode as shown in Fig 8, which means a time base period value of 150 will give a total PWM period value of 300 counts (i.e. 5 us) as shown.

For a two phase interleaved boost converter it is advantageous to operate the two phases 180 deg out of phase. This attenuates the input ripple current to the best possible extent and also reduces the output capacitor RMS current. Here this is achieved by configuring ePWM1 as shown in Fig 8.

There is an important consideration regarding where the ADC input is sampled. The integrity of the ADC input signals is of high importance since this is where signals from the analog and digital domains interface. Turning a switch ON/OFF in the power stage may result in some noise or disturbance on the signals that are to be sensed around this point in time. Even with all the filtering provided on these signals to avoid this noise from showing up at the ADC inputs, it is prudent to sample the ADC inputs at a time such that this disturbance is avoided. To achieve this, the ADC input signals are sampled at an appropriate time so as to get as noise free a sample as possible. For the PFC this is achieved by sampling at the midpoint of the PWM signal i.e. as far away from the MOSFET switching as possible. This avoids any switching noise to be reflected on the ADC result. This is shown in Fig 8. The flexibility of ADC and PWM modules on C2000 devices allow such precise and flexible triggering of ADC conversions. In this case ePWM2 is used as a time base to generate a start of conversion (SOC) trigger just before the aforementioned mid-point. A dummy ADC conversion is performed here to ensure the integrity of the ADC results.

Fig 8 also shows (at a compressed scale) how the PWM duty cycle will correspond to one quarter of the input AC voltage. Corresponding current is also depicted for one of the PFC phases.

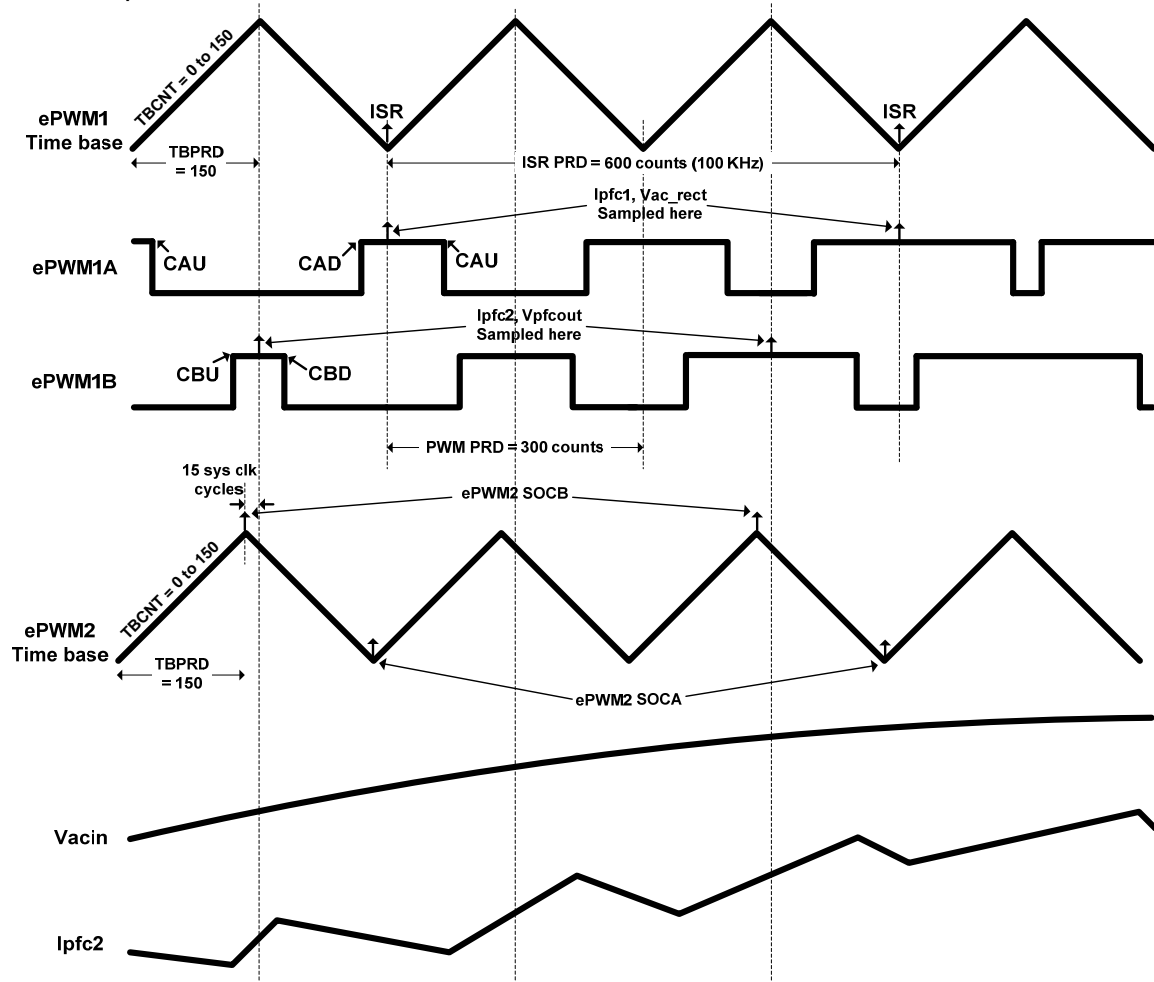


Figure 8. PWM generation and ADC sampling

On a CAU event (TBCNT1 = CMPA and counting up), ePWM1A output is Reset, while on a CAD event (TBCNT1 = CMPA and counting down), ePWM1A output is Set. For phase 2, on a CBU event (TBCNT1 = CMPB and counting up), ePWM1B output is Set, while on a CBD event (TBCNT1 = CMPB and counting down), ePWM1B output is Reset.

The CMPA value is derived from the input "PFCDuty" (Q24 variable) command. The CMPB value is in turn derived from the CMPA value as follows:

$$CMPB = TBPRD - CMPA \quad \dots\dots\dots (1)$$

Table 3 gives example CMPA, CMPB values calculated for a TBPRD value of 149.

<b>PFCDuty (Q24)</b>	<b>CMPA = (PFCDuty*TBPRD/2^15)</b>	<b>CMPB = TBPRD – CMPA</b>	<b>% Duty</b>
2097152d (0x00200000)	18	131	12.4
8388608d (0x00800000)	75	74	50
16776704d (0x00FFFE00)	149	0	100

*Table 3. Duty values for reference*

This ensures a 180 deg phase shift between the two PWM signals as shown in Fig 8.

The input “Phadj” can also be used in equation 1 to provide added flexibility to implement better phase management between the two PFC phases. In this project, this parameter is always set to ‘0’, such that the commanded duty cycle is always equal for both phases as shown in Fig 8.

The ADC module is configured to use both SOCA and SOCB of ePWM2 such that, SOCA is triggered at TBCNT2 = ZERO event while SOCB is triggered at TBCNT2 = PRD event. At an SOCA event *Ipfc1* and *Vac\_rect* are sampled and at an SOCB event *Ipfc2* and *Vpfcout* are sampled as shown in Fig 8. These 4 ADC results are read in the ISR by executing the ADC driver module in *\_DPL\_ISR*.

The assembly ISR *\_DPL\_ISR* routine is triggered by EPWM1. This is where the *PWMDRV\_PFC2PHIL* macros are executed and the PWM compare shadow registers updated. These are loaded in to the active register at the next TBCNT = ZERO event. Note that the ISR trigger frequency is half that of the PWM switching frequency as shown in Fig 8.

Each PFC phase current is sensed by sensing the current through its MOSFET leg. Thus, only the inductor charging current is sensed. Here, current transformers, with a turns-ratio of 1:50, and sense resistors (33.2 ohm) sense the current through the two switches. On this board, there is also an offset of 0.2V added to the current sense signal (please refer to the UCC28070EVM user guide for a detailed description). This offset shows up on the ADC input and is accounted for in the software. Please refer to *PFC2PHIL-Calculations.xls* file for details on these calculations.

### **VacLineRect and Vpfcout**

Simple voltage dividers are used to sense these two quantities by bringing them within the 0 – 3.3 V ADC input range.

### **Protection**

At this stage, it is appropriate to introduce the trip mechanism used on this board for the PFC stage. Here an **overcurrent protection** is implemented for the two PFC phase

currents. Please note that although no overvoltage protection is implemented here, it is possible to do so in the software or using external comparator and TZ pins.

The sensed current signals are fed to the ADC inputs that also serve as the non-inverting input of the two on-chip analog comparators on Piccolo-A. The reference trip voltage level is set using the internal 10-bit DACs and fed to the inverting terminal of the comparators. The comparator outputs are configured to generate a one-shot trip action whenever the sensed current is greater than the set limit. 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 the trip flag is cleared in the software or a device reset is executed. Please refer to PFC2PHIL-Calculations.xls file for details on these calculations.

The key signal connections between the C2000 Controller and the PFC stage are summarized in the table below.

Signal Name	Description	Connection to C2000 Controller
EPWM-1A	PWM Duty control signal for PFC phase 1	GPIO-00
EPWM-1B	PWM Duty control signal for PFC phase 2	GPIO-01
Ipfc1	PFC Phase1 current	ADC-A2/COMP1A
Vpfcout	PFC Output Voltage	ADC-A3
Ipfc2	PFC Phase 2 current	ADC-A4/COMP2A
Vac_rect	Input rectified AC voltage	ADC-A1

Table 4. PFC Signal Interface reference

## ➤ Procedure

### Start CCS and Open a Project

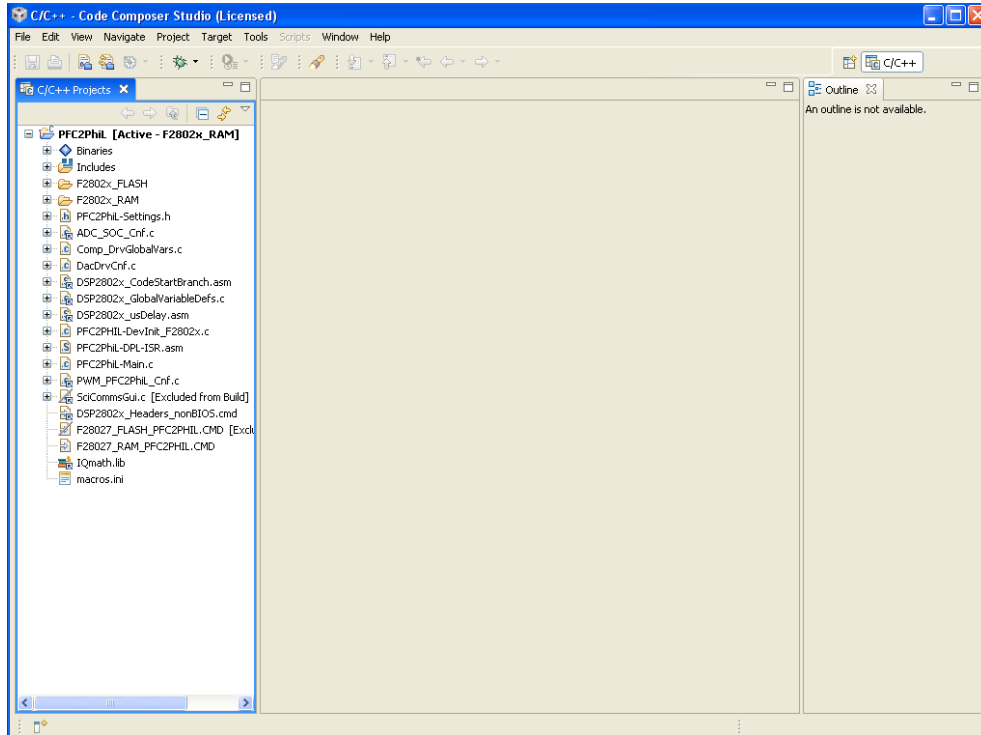
To quickly execute this build, follow the following steps:

1. Connect USB connector to the Piccolo controller board for emulation. Power up the 14 V bias supply at J2. By default, resistors R6 and R7 on the Piccolo Macro of the control card are removed to enable – boot from FLASH. Re-populate these resistors to run and program RAM or program FLASH.
2. Double click on the Code Composer Studio icon on the desktop. Maximize Code Composer Studio to fill your screen. Close the Welcome screen if it opens up.
3. A *project* contains all the files and build options needed to develop an executable output file (.out) which can be 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\HVPFC2Phil\PFC2Phil` directory. Make sure that under the Projects tab PFC2PHIL is checked. Click Finish.

This project will invoke all the necessary tools (compiler, assembler, linker) to build the project.

4. In the project window on the left, click the plus sign (+) to the left of Project. Your project window will look like the following:



## Device Initialization, Main, and ISR Files

---

**Note:** *DO NOT* make any changes to the source files – *ONLY INSPECT*

---

5. Open and inspect `PFC2PHIL-DevInit_F2802x.c` by double clicking on the filename in the project window. Notice that `system clock`, peripheral clock prescale, and `peripheral clock` enables have been setup. Next, notice that the shared `GPIO pins` have been configured.
6. Open and inspect `PFC2PHIL-Main.c`. Notice the call made to `DeviceInit()` function and other variable initialization. Also notice code for different incremental build options (specifically the build you are going to compile now), the ISR initialization and the background for(;;) loop.
7. Locate and inspect the following code in the main file under initialization code specific for build 1. This is where the `PWMDRV_PFC2Phil` block is connected and initialized in the control flow.

```
// Lib Module connection to "nets"
//-----
// Connect the PWM Driver input to an input variable, Open Loop System
PWMDRV_PFC2PHIL_Duty1 = &PFCDuty;
PWMDRV_PFC2PHIL_Adj1 = &PFCShareAdj;
```

8. Locate and inspect the following code in the main file under initialization code specific for build 1. This is where the ADCDRV\_4CH block is configured, initialized and connected in the control flow.

```
#define      Ipfc1R      AdcResult.ADCRESULT1      //
#define      Vac_rectR  AdcResult.ADCRESULT2      //
#define      Ipfc2R      AdcResult.ADCRESULT4      //
#define      VpfcOutR   AdcResult.ADCRESULT5      //

// ADC Channel Selection for Configuring the ADC
//Map channel to ADC Pin
ChSel[0] = 2;      // A2 - PFC Phase1 current - Dummy read
ChSel[1] = 2;      // A2 - PFC Phase1 current - IpfcR1
ChSel[2] = 1;      // A1 - VacLine Rect - Vac_rectR
ChSel[3] = 4;      // A4 - PFC Phase2 current- Dummy read
ChSel[4] = 4;      // A4 - PFC Phase2 current - IpfcR2
ChSel[5] = 3;      // A3 - PFC O/P Voltage - Vpfc_outR

// Select Trigger Event
TrigSel[0] = ADCTRIG_EPWM2_SOCa;      // PFC Phase 1 current sampling triggered by EPWM2 SOCA
TrigSel[1] = ADCTRIG_EPWM2_SOCa;      // PFC Phase 1 current sampling triggered by EPWM2 SOCA
TrigSel[2] = ADCTRIG_EPWM2_SOCa;      // VacLine Rect sampling triggered by EPWM2 SOCA
TrigSel[3] = ADCTRIG_EPWM2_SOCb;      // PFC Phase 2 current sampling triggered by EPWM2 SOCB
TrigSel[4] = ADCTRIG_EPWM2_SOCb;      // PFC Phase 2 current sampling triggered by EPWM2 SOCB
TrigSel[5] = ADCTRIG_EPWM2_SOCb;      // PFC O/P voltage sampling triggered by EPWM2 SOCB

// Configure PWM1 with TBPRD of 300 clock cycles, in master mode
PWM_PFC2PHIL_CNF(1, 300); // for 200 Khz swithcing frequency
PWM_PFC2PHIL_CNF(2, 300); // ePWM2 used for ADC Trigger

// Sync PWM2 with PWM1 with a phase shift = time required for one ADC dummy conversion
EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;
EPwm2Regs.TBCTL.bit.PHSDIR = TB_UP;
EPwm2Regs.TBPHS.half.TBPHS = 15;

ADC_SOC_CNF(ChSel,TrigSel,ACQPS,15,0);

// Configure the Start of Conversion for the ADC. ePWM2 is the trigger.
// At TBCNT1 = 0 convert Ipfc1 and VacLine and TBCNT1 = PRD convert Ipfc2 and Vpfcout
// At TBCNT1 = 0 SOCA is triggered and at TBCNT1 = PRD SOCB is triggered
EPwm2Regs.ETSEL.bit.SOCAEN = 1;
EPwm2Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO; // Use ZRO event as trigger
EPwm2Regs.ETPS.bit.SOCAPRD = ET_2ND;      // Generate pulse on 2nd event

EPwm2Regs.ETSEL.bit.SOCBEN = 1;
EPwm2Regs.ETSEL.bit.SOCBSEL = ET_CTR_PRD; // Use PRD event as trigger as well
EPwm2Regs.ETPS.bit.SOCBPRD = ET_2ND;      // Generate pulse on 2nd event

//ADCDRV_4CH block output connections
ADCDRV_4ch_RltPtrA = &IphA;
ADCDRV_4ch_RltPtrB = &VacLineRect;
ADCDRV_4ch_RltPtrC = &IphB;
ADCDRV_4ch_RltPtrD = &VpfcOut;
```

9. Open and inspect PFC2PHIL-DPL-ISR.asm. Notice the \_DPL\_Init and \_DPL\_ISR sections. This is where the PWM and ADC driver macro instantiation is done for initialization and runtime, respectively. Optionally, you can close the inspected files.



## Build and Load the Project

10. Select the Incremental build option as 1 in the **PFC2PHIL-Settings.h** file.  
**Note:** Whenever you change the incremental build option in **PFC2PHIL-Settings.h** always do a “Rebuild All”.
11. Click Project→“Rebuild All” button and watch the tools run in the build window.
12. 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 Step 14. In the New target Configuration Window type in the name of the .ccxml file for the target you will be working with (Example: xds100-F28027.ccxml). Check “Use shared location” and click Finish.
13. In the .ccxml file that open up select Connection as “Texas Instruments XDS100v2 USB Emulator” and under the device, scroll down and select “TMS320F28027”. Click Save.
14. Click Target→“Debug Active Project”. The program will be loaded into the FLASH or RAM memory depending on the project configuration selected (Default configuration is F2802x\_FLASH, which will be used for this exercise. Please refer to the note below if you want to change the project configuration). You should now be at the start of Main().

**NOTE:** When using a certain project configuration (F2802x\_RAM/ F2802x\_FLASH), please make sure you change the following lines of code in PFC2PHIL-Main.c file appropriately:

```
#define DLOG_SIZE 256    // Uncomment for FLASH configuration only
// #define DLOG_SIZE 32  // Uncomment for RAM configuration only
```

**We will be using the FLASH configuration option for this exercise.**

## Debug Environment Windows

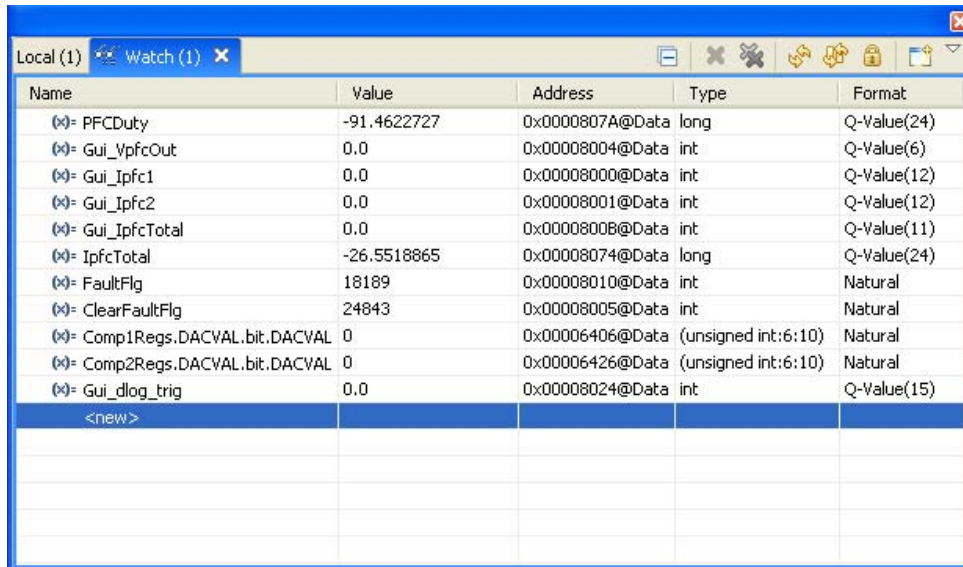
It is standard debug practice to watch local and global variables while debugging code. There are various methods for doing this in Code Composer Studio, such as memory views and watch views. Additionally, Code Composer Studio has the ability to make time (and frequency) domain plots. This allows us to view waveforms using graph windows.

15. 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.

Click: View → Watch on the menu bar.

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. Please note that some of the variables have not been initialized at this point in the main code and may contain some garbage values.

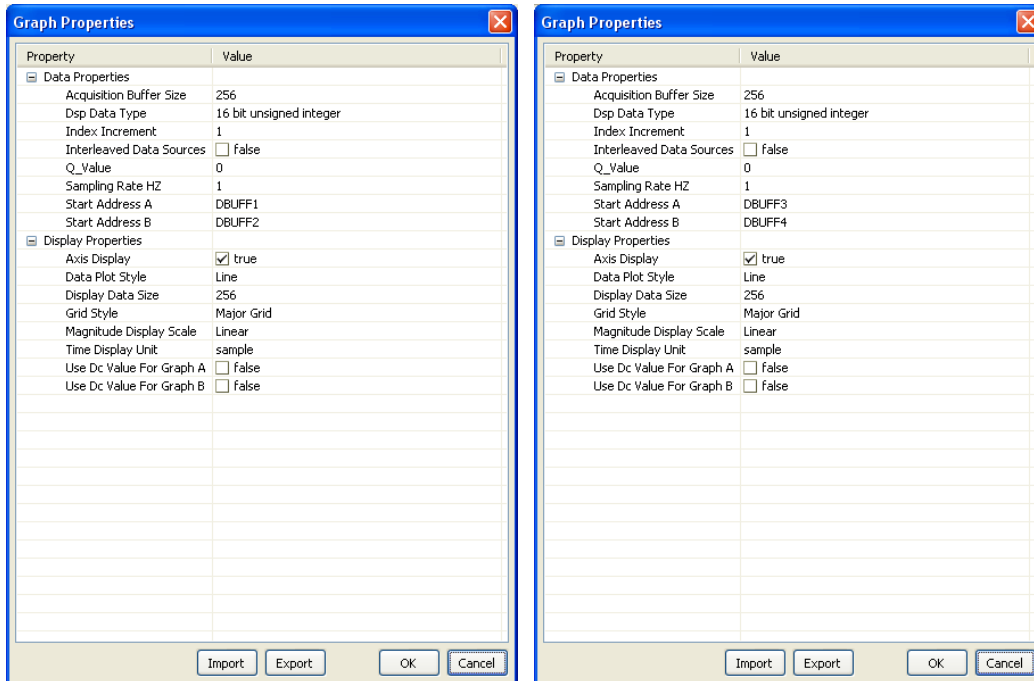


Name	Value	Address	Type	Format
(*)= PFCduty	-91.4622727	0x0000807A@Data	long	Q-Value(24)
(*)= Gui_VpfcOut	0.0	0x00008004@Data	int	Q-Value(6)
(*)= Gui_Ipfc1	0.0	0x00008000@Data	int	Q-Value(12)
(*)= Gui_Ipfc2	0.0	0x00008001@Data	int	Q-Value(12)
(*)= Gui_IpfcTotal	0.0	0x00008008@Data	int	Q-Value(11)
(*)= IpfcTotal	-26.5518865	0x00008074@Data	long	Q-Value(24)
(*)= FaultFlg	18189	0x00008010@Data	int	Natural
(*)= ClearFaultFlg	24843	0x00008005@Data	int	Natural
(*)= Comp1Regs.DACVAL.bit.DACVAL	0	0x00006406@Data	(unsigned int:6:10)	Natural
(*)= Comp2Regs.DACVAL.bit.DACVAL	0	0x00006426@Data	(unsigned int:6:10)	Natural
(*)= Gui_dlog_trig	0.0	0x00008024@Data	int	Q-Value(15)
<new>				

Figure 9. CCS watch view for Build 1


FaultFlg, if set, indicates a PFC over current condition (discussed above). The trip mechanism immediately shuts down the two PWM outputs in such a case. The PWM outputs are held in this state until a device reset (follow the proper procedure in Step 31) or until ClearFaultFlg is set in the watch view. The *CompXRegs.DACVAL.bit.DACVAL* directly set the internal 10-bit DAC reference level for the on-chip comparators. Please note that this is a 10-bit (Q10) number.

- Open two graph windows by clicking Tools → Graph → Dual Time. These are dual time graph windows to plot the four data log buffers DBUFF1, DBUFF2, DBUFF3 and DBUFF4. Set the graph properties for the two graphs as shown below. Please note that when using the F2802x\_RAM project configuration the Acquisition Buffer Size and Display Data Size should be changed to 32. For the F2802x\_FLASH configuration a value of 256 is used for these fields as shown below.





## Using Real-time Emulation

Real-time emulation is a special emulation feature that allows the windows within Code Composer Studio to be updated at up to a 10 Hz rate *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 have those changes affect the MCU behavior. This is very useful when tuning control law parameters on-the-fly, for example.

17. 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.

18. 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.

19. Now click  **Enable Polite Real-time Mode** button on the same horizontal toolbar.

20. When a large number of windows are open, as bandwidth over the emulation link is limited, updating too many windows and variables in continuous refresh can cause the refresh frequency to bog down. Right click on the  button in the watch view and select "Customize Continuous Refresh Interval..". You can slow down the refresh rate for the watch view variables by changing the *Continuous refresh interval (milliseconds)* value. A rate of 4000 ms is usually enough for these exercises.

21. Click on Continuous Refresh buttons  for the watch view and the graphs.

## Run the Code

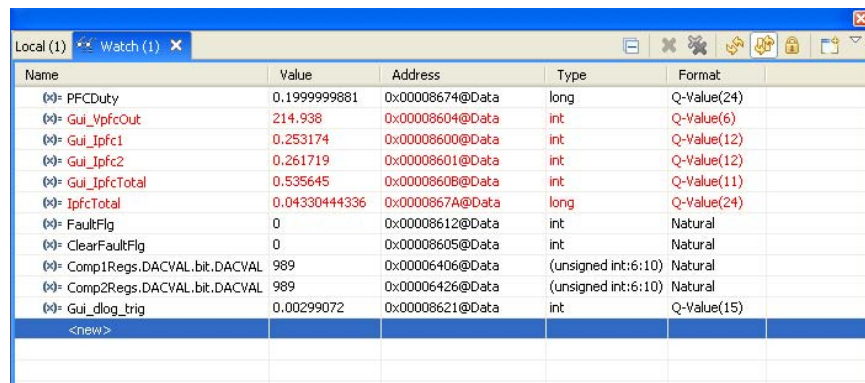
22. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.
23. In the watch view, the variable *PFCDuty* should be set to 0.0. This variable sets the duty cycle for the two phases of PFC boost converter as shown in Table 3.
24. Apply an appropriate resistive load to the PFC system at the DC output.

**NOTE: For safety reasons it is recommended to use an appropriate isolated AC source.**

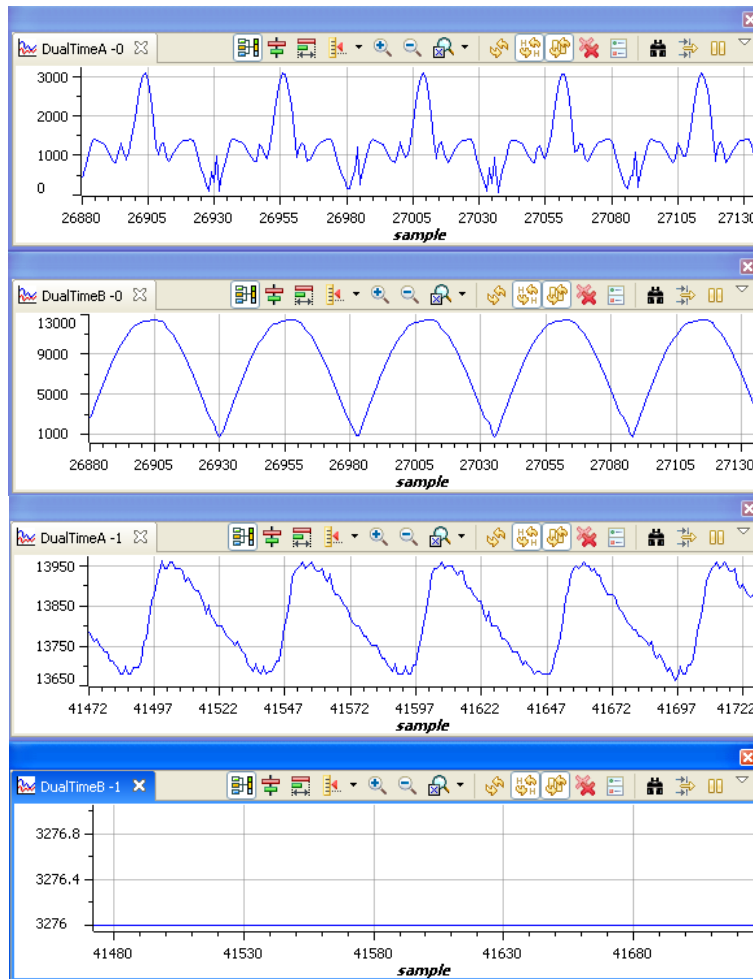
25. Power the AC input with an AC input voltage within the abilities of the board.
26. Increase the duty cycle by setting *PFCDuty* to a higher value 0.04(say) in the watch view. (Refer to Table 3 for correspondence of different duty values with *PFCDuty*). The boost converter should now start boosting the input voltage.

**Observe the output voltage carefully, this should not be allowed to exceed the capabilities of the board.**

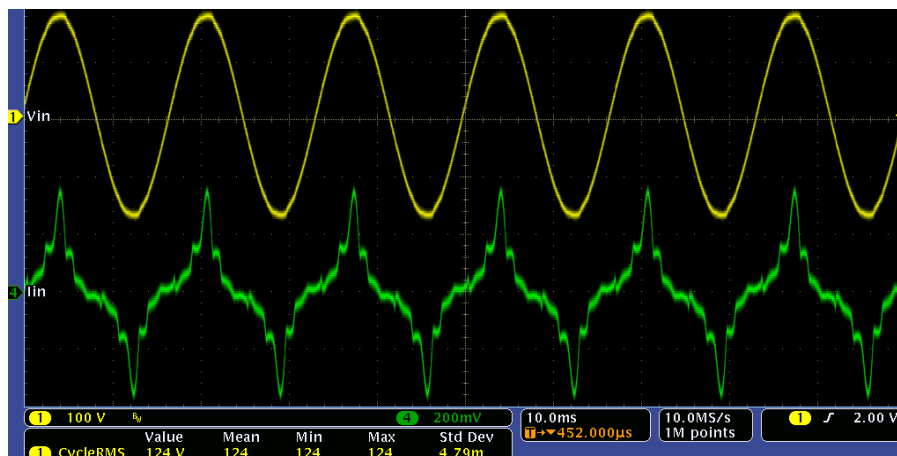
27. Observe the different ADC results in the watch view and the graph windows for different duty cycle values. You may have to adjust the “Gui\_dlog\_trig\_value” in the watch view for the graphical display. Data logging is triggered with respect to DBUFF1 (IpfcTotal). A good approximation for “dlog\_trig\_value” (in Q15) is to make it close to the IpfcTotal value (in Q24).
28. Here are the watch view and graph windows that correspond to the operation of the system with a *PFCDuty* command of 0.2 per unit (Q24) with the input voltage of around 125Vac and a resistive load of around 1200 ohm. DualtimeA-0 corresponds to IpfcTotal(Q15), DualtimeB-0 corresponds to VacLineRect(Q15), DualtimeA-1 corresponds to VpfcOut(Q15) and DualtimeB-1 corresponds to *PFCDuty*(Q15).




Name	Value	Address	Type	Format
PFCDuty	0.1999999881	0x00008674@Data	long	Q-Value(24)
Gui_VpfcOut	214.938	0x00008604@Data	int	Q-Value(6)
Gui_Ipfc1	0.253174	0x00008600@Data	int	Q-Value(12)
Gui_Ipfc2	0.261719	0x00008601@Data	int	Q-Value(12)
Gui_IpfcTotal	0.535645	0x00008608@Data	int	Q-Value(11)
IpfcTotal	0.04330444336	0x0000867A@Data	long	Q-Value(24)
FaultFlg	0	0x00008612@Data	int	Natural
ClearFaultFlg	0	0x00008605@Data	int	Natural
Comp1Regs.DACVAL.bit.DACVAL	989	0x00006406@Data	(unsigned int:6:10)	Natural
Comp2Regs.DACVAL.bit.DACVAL	989	0x00006426@Data	(unsigned int:6:10)	Natural
Gui_dlog_trig	0.00299072	0x00008621@Data	int	Q-Value(15)
<new>				



29. The following oscilloscope captures show the input line voltage and current seen under the conditions described above.



30. 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 using an oscilloscope. Appropriate safety precautions must be taken while probing these high voltage, high power signals.
31. Fully halting the MCU when in real-time mode is a two-step process. With the AC input turned off wait a few seconds. First, halt the processor by using the Halt button on the toolbar, or by using Target → Halt. Then click clicking  button again to take the MCU out of real-time mode and then reset the MCU.
  32. You may choose to leave Code Composer Studio running for the next exercise or optionally close CCS.

### **End of Exercise**



These mapping equations are given below. For the current loop these P, I and D coefficients are: **Pgain\_I, Igain\_I and Dgain\_I** respectively. For the voltage loop these P, I and D coefficients are: **Pgain\_V, Igain\_V and Dgain\_V** respectively. These P, I and D coefficients are used in **Q26** format. To simplify tuning from the GUI environment (or from CCS watch views) these three coefficients are further scaled to values from **0 to 999** and the corresponding P, I and D GUI gain variable is used. In the lab exercise, you will inspect the C code in which these coefficients are initialized.

The compensator block (CNTL\_2P2Z) has 2 poles and 2 zeros and is based on the general IIR filter structure. It has a reference input and a feedback input coming from the ADC. For the current loop the feedback is the total PFC input current *I<sub>pfcTotal</sub>*. For the voltage loop the feedback is the output voltage *V<sub>pfcOut</sub>*. The transfer function is given by:

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

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:

$$\begin{aligned} b_0 &= K_p' + K_i' + K_d' \\ b_1 &= -K_p' + K_i' - 2K_d' \\ b_2 &= K_d' \end{aligned}$$

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

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

Comparing this with the general form, we can see that PID is nothing but a special case of CNTL\_2P2Z control where:

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

The MATH\_EMAVG (Exponential Moving Average) block calculates the **average of the input rectified AC voltage**. The **Inverse average square** (PFC\_INVSQR) block normalizes this average value with its own (AC input) maximum. The PFC\_INVSQR block also provides a feed-forward compensation so as to maintain a constant output power for a given load by appropriately responding to changes in the input AC voltage. For example for a constant load if the input voltage decreases INV SQR block output increases proportionately. This block drives the PFC current command block (PFC\_ICMD). Therefore, an increase in the PFC\_INVSQR block output in turn increases the current command, which leads to an increase in the input current so as to maintain the same power level.

Implemented in the .asm file (ISR code) is a gain on the *VacLineRect* variable such that the maximum input voltage of **285V** on this board is scaled to **0 to 32767d in Q15 format** (or **0 to 16776704d in Q24**) for the variable *VacLineScaled*. This is done in order to scale the possible input voltage range to the full Q24 format range before applying it to the



other mathematical blocks. After applying this gain the maximum possible input voltage on this board will correspond to a value of **0x7fff (32767d)** for the variable **VacLineScaled** in Q15 format.

The PFC current command block is responsible for calculating the current command required for the power factor correction. Its output is half-sinusoidal with an amplitude proportional to the commanded voltage. The voltage controller output sets the voltage command for the PFC\_ICMD block. The output of the PFC\_ICMD block drives the current controller.

### Improving Input p.f. and T.H.D. under High-line and Low-load Conditions

In this implementation the ADC inputs are sampled once in **each switching cycle**. As mentioned in the previous chapter these inputs are sampled at the mid-point of the ON time so as to get as noise free a signal as possible. The two PFC phase currents are also measured in the same way. An advantage of this way of sensing is that when the PFC stage is operating in continuous conduction mode, the current measured this way gives an average input current measurement (assuming a linear charging-discharging inductor current).

Under high-line and low-load conditions the input current stays **discontinuous** for a longer portion in each sine quarter of the input line voltage. The lower the load or higher the input rms voltage, the longer the PFC stage operates in discontinuous conduction mode (DCM) in each sine quarter of the input line voltage. The amount of time the system operates in DCM mode is dependent on the inductor and the power stage design and the line-load conditions. During this time of operation (DCM) the sensed current is not equal to the average (averaged over one switching cycle) current. **In fact this measured current is greater than the average.** This difference in the *measured* average and *true* average current is greater for duty cycle and line-load conditions when operating in deep DCM mode. Therefore the current loop feedback input is higher than the set reference input. **Consequently the controller drives a smaller control action/duty cycle where a higher duty cycle is required.** This results in a distorted input current waveform during the period of DCM operation in each quarter cycle and possibly a jump in the input current during the input voltage sine quarter when the input current crosses DCM-CCM mode boundary and vice-versa. Apart from reducing the input power factor (p.f.) and increasing the total harmonic distortion (T.H.D.), a side effect of this behavior is a possible reduction in the efficiency of the system under these high-line and low-load conditions.

In this implementation, to compensate for the difference between the *measured* and *true* average current values under different line-load conditions a compensating factor is subtracted from the measured current. This factor is estimated to vary in an inverse-sine fashion with the input line voltage. This is because around the zero crossing of the input line voltage the discrepancy between the measured and true average current seems to be the largest, while it is minimum at the peak of the line voltage. The amplitude of this inverse-sine (*CompAmpltd*) is dependent on the different line-load conditions (input rms voltage and power drawn). The higher the line or lower the load, the higher is the *CompAmpltd* value. Thus the largest *CompAmpltd* values correspond to the high-line and low-load operation. This way of compensation also helps improve the system efficiency under different operating conditions.

In this implementation following equation was used to calculate the *CompAmpltd*. Please note that this equation was arrived at by way of experimentation.

$$CompAmpltd = (Input\_Power(Max) - Input\_Power) * Scale\_Factor * (Vac_{(rms)} - 85) / (285 - 85)$$

where,

*CompAmpltd* = the amplitude of the inverse-sine compensation

*Input\_Power(Max)* = Maximum input power rating (~312VA)

*Input\_Power*<sup>1</sup> = Input Power calculated from r.m.s. input current and r.m.s input voltage measurement.

*Scale\_Factor* = this factor is a function of the line voltage. Here different values are used for different input line voltage ranges. These ranges may be increased decreased for increased granularity.

$(Vac_{(rms)} - 85) / (285 - 85)$  = this factor is a normalizing factor for the input line voltage within its range of operation from 85V – 285V AC.

These calculations are performed in the slower state machine task B3. Please note that this is only one way of implementation and an approximate equation that seems to work well for the power stage used here. Other better functions and equations based on the inductor and power stage designs and Mathcad and Matlab simulation tools are certainly possible to provide better results and granularity.

### Adding a Shoulder

When operating under high-line low load conditions, it is likely that the PWM switching around the input voltage zero crossing may contribute more to switching losses than be useful for power factor correction action. It may also be the case that efficiency is a major concern under certain operating conditions than achieving the very small power factor increases that may be obtained by driving duty cycle around the zero voltage crossing points. Therefore, it might be advantageous to not drive the two switches around the zero crossing. A factor *shoulder* is used to make the duty cycle zero for input voltages below a certain level defined by this value. Higher the line voltage and lower the load, higher the shoulder value. Please note that a large *shoulder* value may result in a huge jump in the input current when the input voltage goes above this value. Also, the calculated r.m.s. input voltage will be smaller than actual.

---

<sup>1</sup> The input power calculated here is based on r.m.s values calculated from average values obtained using moving average filters for the input current and input voltage. Better results may be obtained by implementing true r.m.s calculations for input voltage and current to obtain better granularity and possibly better results across the whole line-load range.

## Start-up, Shut-down and Slew-limit

At start-up if the input line voltage is above ~ 115V r.m.s. for greater than ~ 1.5 secs the PFC action is enabled and the output will ramp up to the desired value of ~ 390Vdc. This ramp up speed is defined by the slew limit block implemented in the slower state machine task C3. Further, the current controller output is clamped and this clamping limit is also ramped up slowly at start-up to ensure smooth start-up operation. Whenever the input voltage falls below ~ 90V the PFC action is immediately disabled and not enabled till the input voltage falls in the acceptable range and stays good for more than 1.5 secs. As such please note that this mechanism prevents a one cycle input voltage drop-out operation. This part of the software, among other things, is required to be changed if such an operation is desired.

## ➤ Procedure

### Build and Load Project

To quickly execute this build using the pre-configured work environment, follow the following steps:

- 1-2. Follow Steps 1 to 2 exactly as in build 1. If you were working on build 1 the last time CCS was used, the same workspace should open up with the project. If this is not the case, you can open the workspace used for build1 by clicking File → Switch Workspace and then navigating to the correct workspace. If a workspace was not saved or got deleted, please follow steps 3 and 4 exactly as in build1.
3. Locate and inspect the initialization code specific to build 2 in the main file. This is where all the control blocks are configured, initialized and connected in the control flow.
4. Select the Incremental build option as 2 in the **PFC2PHIL-Settings.h**.  
**Note:** Whenever you change the incremental build option in **PFC2PHIL-Settings.h** always do a “Rebuild All”.
5. Click Project→“Rebuild All” button and watch the tools run in the build window.
6. Click Target→“Debug Active Project”. The program will be loaded into the FLASH or RAM memory depending on the project configuration selected (Default configuration is F2802x\_FLASH, which will be used for this exercise). You should now be at the start of Main().

**NOTE: We will be using the F2802x\_FLASH configuration option for this exercise. It is important to make the appropriate changes to the code for the configuration being used, as mentioned in Step 14 of Build 1.**

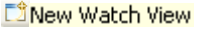
## Debug Environment Windows

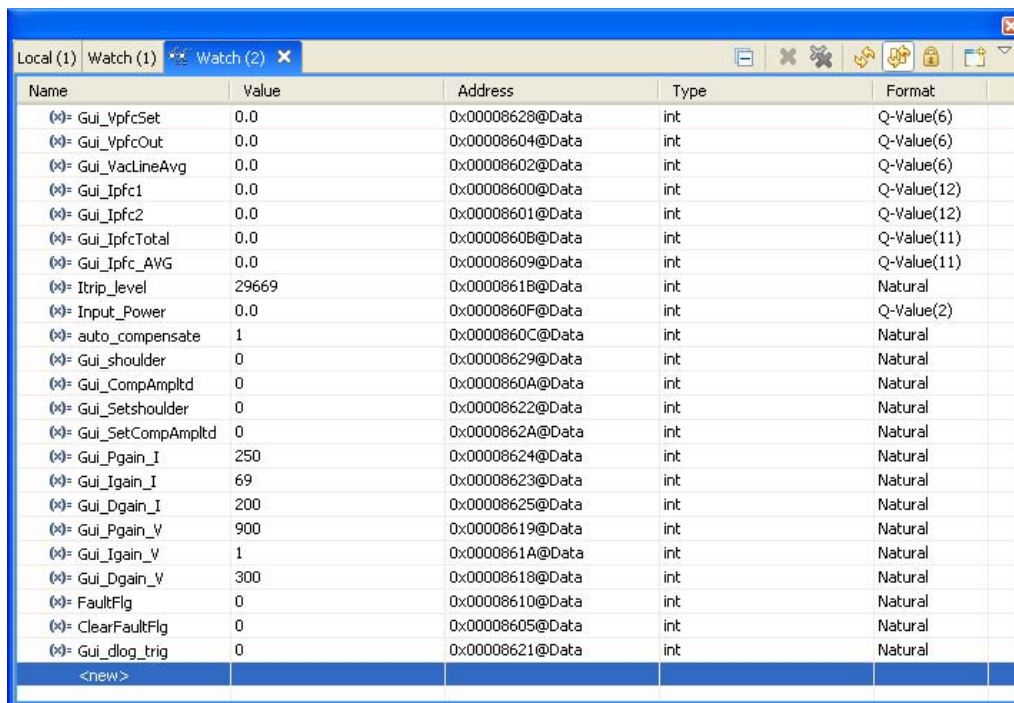
It is standard debug practice to watch local and global variables while debugging code. There are various methods for doing this in Code Composer Studio, such as memory views and watch views. Additionally, Code Composer Studio has the ability to make

time (and frequency) domain plots. This allows us to view waveforms using graph windows.

7. 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.

Click: View → Watch on the menu bar.

Click the “Watch (1)” tab at the top watch view. If a watch view is already open from the debug environment saved for build 1, click on  **New Watch View** button in the watch view. A “Watch (2)” tab will open and you may then drag it to be viewed in a window of your choice. You may add any variables to this watch view tab. 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. Please note that some of the variables have not been initialized at this point in the main code and may contain some garbage values.



Name	Value	Address	Type	Format
(*)= Gui_VpfcSet	0.0	0x00008628@Data	int	Q-Value(6)
(*)= Gui_VpfcOut	0.0	0x00008604@Data	int	Q-Value(6)
(*)= Gui_VacLineAvg	0.0	0x00008602@Data	int	Q-Value(6)
(*)= Gui_Ipfc1	0.0	0x00008600@Data	int	Q-Value(12)
(*)= Gui_Ipfc2	0.0	0x00008601@Data	int	Q-Value(12)
(*)= Gui_IpfcTotal	0.0	0x00008608@Data	int	Q-Value(11)
(*)= Gui_Ipfc_AVG	0.0	0x00008609@Data	int	Q-Value(11)
(*)= Itrip_level	29669	0x00008618@Data	int	Natural
(*)= Input_Power	0.0	0x0000860F@Data	int	Q-Value(2)
(*)= auto_compensate	1	0x0000860C@Data	int	Natural
(*)= Gui_shoulder	0	0x00008629@Data	int	Natural
(*)= Gui_CompAmpltd	0	0x0000860A@Data	int	Natural
(*)= Gui_Setshoulder	0	0x00008622@Data	int	Natural
(*)= Gui_SetCompAmpltd	0	0x0000862A@Data	int	Natural
(*)= Gui_Pgain_I	250	0x00008624@Data	int	Natural
(*)= Gui_Igain_I	69	0x00008623@Data	int	Natural
(*)= Gui_Dgain_I	200	0x00008625@Data	int	Natural
(*)= Gui_Pgain_V	900	0x00008619@Data	int	Natural
(*)= Gui_Igain_V	1	0x0000861A@Data	int	Natural
(*)= Gui_Dgain_V	300	0x00008618@Data	int	Natural
(*)= FaultFlg	0	0x00008610@Data	int	Natural
(*)= ClearFaultFlg	0	0x00008605@Data	int	Natural
(*)= Gui_dlog_trig	0	0x00008621@Data	int	Natural
<new>				

Figure 11. CCS watch view for Build 2

Notice the additional variables in the watch view. *Itrip\_level* is a Q15 number and sets the DAC reference level for the two comparators. The default value corresponds to a current trip level of 4.5A. *Input\_Power* is the estimated input power based on average input current and average input voltage calculated using exponential moving average blocks. The GUI variables for the P, I and D gains of the current and voltage loops are in a scaled format and have a range of 0 to 999. It is extremely important that these values be never changed to values outside this range.

8. Notice the variable *auto\_compensate* in the watch view. When this parameter is set to 1 (default) the algorithm calculates the *CompAmpltd* (Q30) and *shoulder* (Q24) values on its own depending on the line-load conditions. A value of 0 for *auto\_compensate* lets the user set the *CompAmpltd* and *shoulder* values using the variables *Gui\_SetCompAmpltd* (Q15) and *Gui\_Setshoulder* (Q15) respectively. This provides for an easy means to experiment with/without this compensation mechanism. *Gui\_CompAmpltd* and *Gui\_shoulder* variables allow the *CompAmpltd* and *shoulder* values to be viewed in Q15 format.
9. If the graph windows used in build 1 don't open correctly, then create new graph windows exactly as in step 16 of build 1.
10. We'll use *Gui\_VpfcSet* and *Gui\_Vpfcout* to set the voltage command and read the output voltage instead of the Q24 variables *VpfcSet* and *VpfcOut*.

## Run the Code

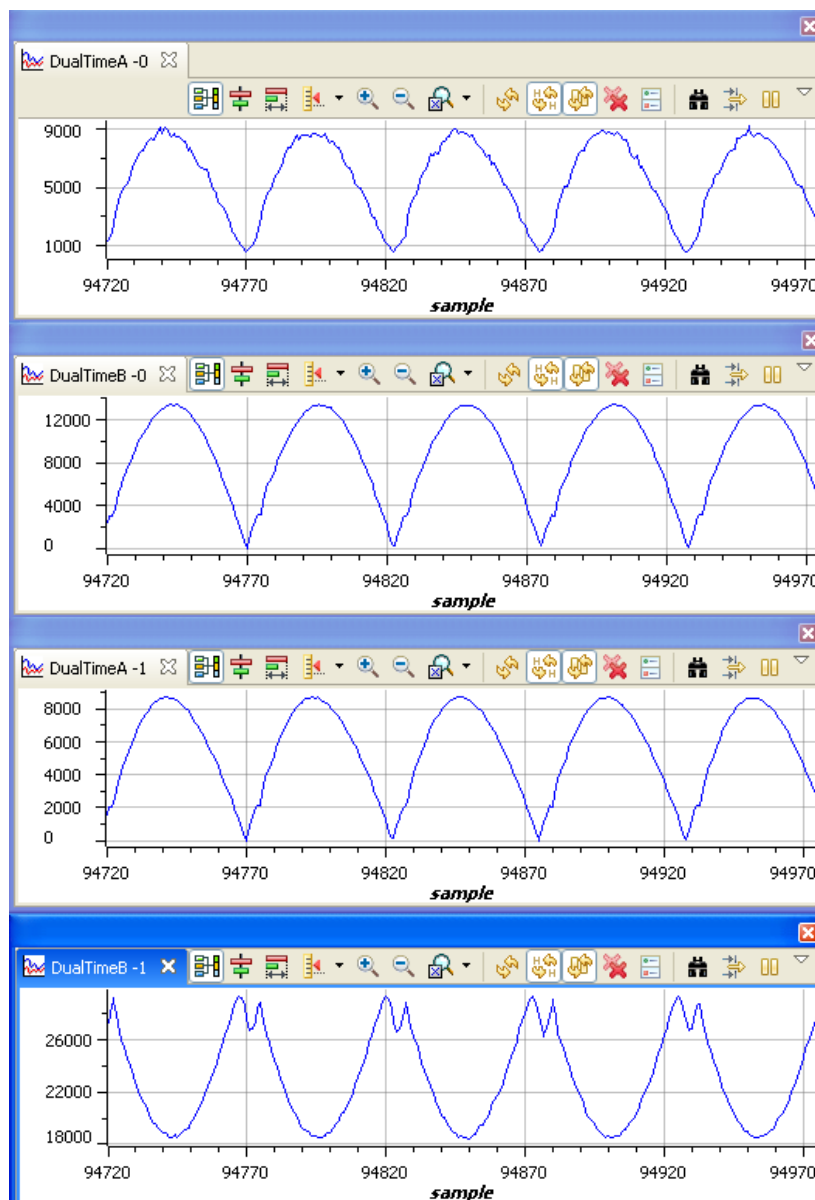
11. Follow steps 17 to 21 of build 1 procedure to enable real time mode and continuous refresh for the watch views and graph windows and also for changing the continuous refresh interval for the watch view.
12. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.
13. Apply an appropriate resistive load to the PFC system at the DC output (Resistive loads above 80W and below 300W are recommended).
14. Power the AC input with an AC input voltage within the abilities of the board (110Vac to 230Vac is recommended).

**NOTE: For safety reasons it is recommended to use an appropriate isolated AC source.**

Once the system is running in realtime and is powered with the AC input, observe the measured output voltage (*Gui\_Vpfcout*). The PFC output should ramp according to the slew rate to the desired set value (~385Vdc). The power factor correction should be maintained during this process.

15. Here are the watch view and graph windows that correspond to the operation of the system with the input voltage of around 120Vac and a resistive load of around 700 ohm. You may have to adjust the "Gui\_dlog\_trig\_value" in the watch view for the graphical display. *DualtimeA-0* corresponds to *IpfcTotal*(Q15), *DualtimeB-0* corresponds to *VacLineScaled*(Q15), *DualtimeA-1* corresponds to *PFCcmd*(Q15) and *DualtimeB-1* corresponds to *PFCDuty*(Q15).

Local (1) Watch (1) Watch (2) X					
Name	Value	Address	Type	Format	
(X) Gui_VpfcSet	383.0	0x00008628@Data	int	Q-Value(6)	
(X) Gui_VpfcOut	382.641	0x00008604@Data	int	Q-Value(6)	
(X) Gui_VacLineAvg	117.906	0x00008602@Data	int	Q-Value(6)	
(X) Gui_Ipfc1	0.883057	0x00008600@Data	int	Q-Value(12)	
(X) Gui_Ipfc2	0.900391	0x00008601@Data	int	Q-Value(12)	
(X) Gui_IpfcTotal	1.87109	0x0000860B@Data	int	Q-Value(11)	
(X) Gui_Ipfc_AVG	1.7876	0x00008609@Data	int	Q-Value(11)	
(X) Itrip_Level	29669	0x0000861B@Data	int	Natural	
(X) Input_Power	212.25	0x0000860F@Data	int	Q-Value(2)	
(X) auto_compensate	1	0x0000860C@Data	int	Natural	
(X) Gui_shoulder	200	0x00008629@Data	int	Natural	
(X) Gui_CompAmpltd	60	0x0000860A@Data	int	Natural	
(X) Gui_Setshoulder	0	0x00008622@Data	int	Natural	
(X) Gui_SetCompAmpltd	0	0x0000862A@Data	int	Natural	
(X) Gui_Pgain_I	250	0x00008624@Data	int	Natural	
(X) Gui_Igain_I	69	0x00008623@Data	int	Natural	
(X) Gui_Dgain_I	200	0x00008625@Data	int	Natural	
(X) Gui_Pgain_V	900	0x00008619@Data	int	Natural	
(X) Gui_Igain_V	1	0x0000861A@Data	int	Natural	
(X) Gui_Dgain_V	300	0x00008618@Data	int	Natural	
(X) FaultFlg	0	0x00008610@Data	int	Natural	
(X) ClearFaultFlg	0	0x00008605@Data	int	Natural	
(X) Gui_dlog_trig	1000	0x00008621@Data	int	Natural	
<new>					



16. The following oscilloscope captures show the input line voltage and current seen under the conditions described above is shown in Fig 12. This clearly illustrates the PFC action.



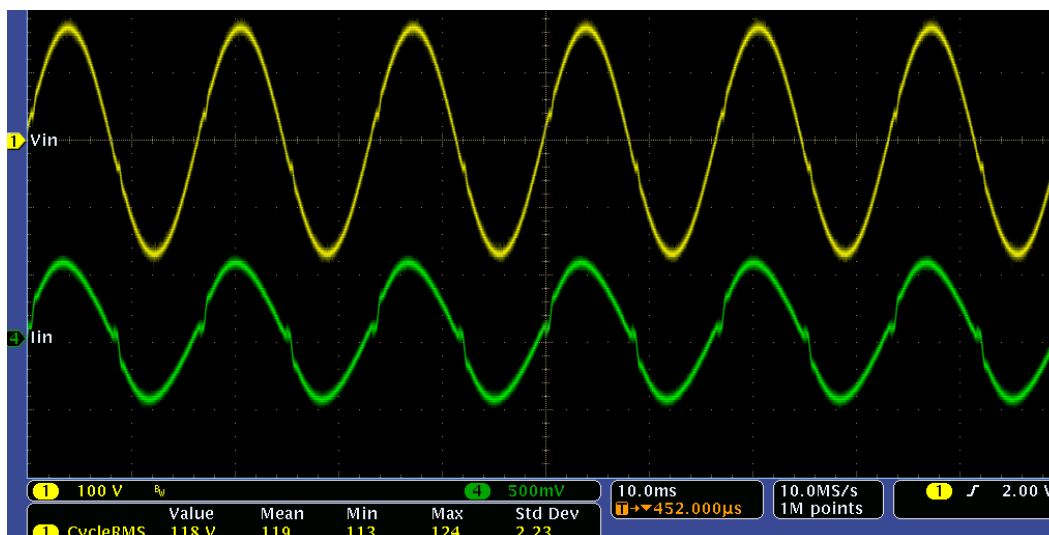



Figure 12. Scope capture of Input current and voltage waveforms for Build 2  
(Output Power ~ 211W – Input Voltage ~ 120Vac)

17. Observe the effect of varying the input voltage on the output voltage and input current. There should be virtually no effect on the output voltage. Also the power factor correction should be maintained.
18. Similarly observe the effect of varying the load. Again there should be virtually no effect on the output voltage and the power factor correction should be maintained.
19. Fully halting the MCU when in real-time mode is a two-step process. With the AC input turned off wait a few seconds. First, halt the processor by using the Halt button on the toolbar, or by using Target → Halt. Then click clicking  button again to take the MCU out of real-time mode and then reset the MCU.
20. Close CCS.

**End of Exercise**



# References

For more information please refer to the following guides:

- **HVPFC2PHIL-GUI-QSG** – gives an overview on how to quickly demo the HVPFC project using an intuitive GUI interface.  
`..\controlSUITE\development_kits\HVPFC2PhiL\~Docs\HVPFC2PHIL-GUI-QSG.pdf`
- **HVPFC2PHIL** – provides detailed information on the PFC2PHIL project within an easy to use lab-style format.  
`..\controlSUITE\development_kits\HVPFC2PhiL\~Docs\HVPFC2PHIL.pdf`
- **HVPFC2PHIL-Calculations** – a spreadsheet showing a few of the key calculations made within the PFC2PHIL project.  
`..\controlSUITE\development_kits\HVPFC2PhiL\HVPFC2PHIL-Calculations.xls`
- **HVPFC2PHIL\_Rel-1.0-HWdevPkg** – a folder containing various files related to the Piccolo-A controller card schematics.  
`..\controlSUITE\development_kits\HVPFC2PhiL\HVPFC2PhiL_HWDevPkg`
- **UCC28070EVM power board** related useful files found at <http://focus.ti.com/docs/toolsw/folders/print/ucc28070evm.html> –
  - HPA225A - Gerber files for the EVM mother board
  - User Guide - UCC28070, 300-W Interleaved PFC Pre-Regulator (Rev. B)
- **F28xxx User's Guides**  
<http://www.ti.com/f28xuserguides>