Digitally Controlled Bridgeless Power Factor Correction (BL PFC) Converter Using C2000 Piccolo-A Microcontroller

CCS User Guide

Version 1.1

April 2012

Abstract

This document presents the implementation details of a digitally controlled Bridgeless Power Factor Correction (BL PFC) converter. A C2000 Piccolo-B control card and a 300W BL PFC 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. This PFC stage forms the front end of an isolated ac-dc rectifier system as shown in Fig 1.

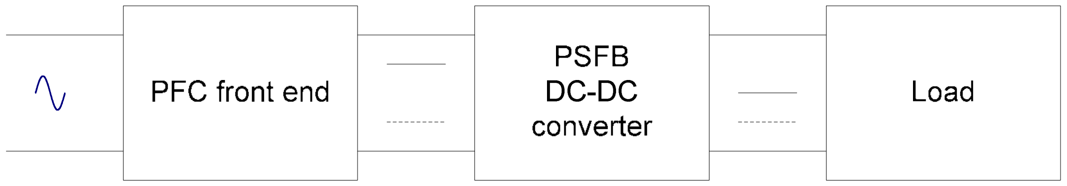


Figure 1. Isolated AC-DC Rectifier System Block Diagram

The PFC converter draws sinusoidal input current from the AC mains and at the same time regulates its output voltage in order to provide a regulated high DC bus voltage to the downstream DC-DC converter. The DC-DC stage 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 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 bridgeless power factor correction (BL PFC) front end.

This PFC EVM uses a Piccolo-B control card and not the Piccolo-A card just because of the absence of a RC filter in the Piccolo-A card. This RC filter is available on both the ADC channels on Piccolo-B card that sense two BL PFC switch currents. However, on Piccolo-A card only one of the two ADC channels has this RC filter. A revised version of this Picocolo-A control card can be easily used to implement full control of this BL PFC EVM.

1 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. Typically, this is implemented using a bridge rectifier followed by a boost PFC stage. However, to reduce the power losses in the diode bridge and to improve the system efficiency an alternative approach is to use a bridgeless system and implement both the rectification and power factor control using two boost stages. In this approach the two boost stages are operated alternately in the positive and negative half cycles of the ac mains voltage. A C2000 piccolo microcontroller with its on-chip PWM and ADC modules is able to implement complete digital control of such bridgeless PFC (BL PFC) system.

1.1. PFC Stage Implementation

Fig 1.1 illustrates a C2000 based bridgeless PFC converter control system. The input AC voltage is applied to the PFC converter through the input EMI filter followed by an inrush control relay. The PFC stage consists of two boost converters each operating alternately in boost mode during half of the input AC line cycle.

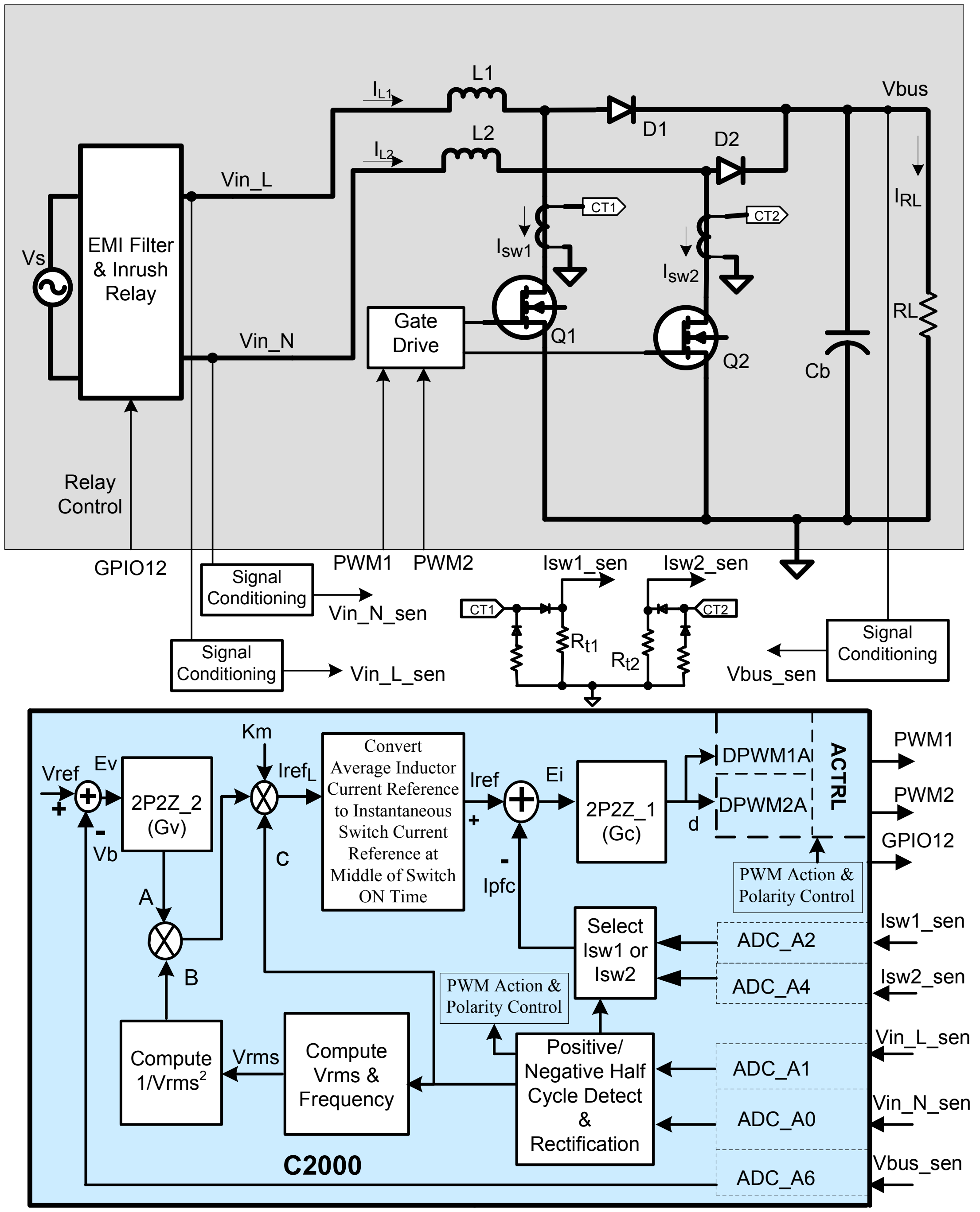


Figure 1.1 Bridgeless PFC Converter Control using C2000 Micro-controller

During one half of the line cycle one converter converts the AC line voltage to the DC bus voltage, while the other converter performs the same action during the other half of the line cycle. Inductor L1, MOSFET switch Q1, current sense transformer CT1 and diode D1 together form one of the boost stages while, L2, Q2, CT2 and D2 form the other boost stage. A capacitor Cb at the boost converter output acts as an energy reservoir and provides regulated dc voltage to the PFC load denoted by RL.

Figure 1.1 indicates all the interface signals needed for full control of this bridgeless PFC converter using a C2000 micro-controller (MCU). The MCU controls the hardware using five feedback signals, two PWM outputs and one GPIO output. The signals that are sensed and fed back to the MCU include, the line and neutral voltages (Vin_L & Vin_N), the two PFC switch currents (I_{sw1}, I_{sw2}), and the boost output voltage (Vbus). These sensed signals are used to implement the voltage and current control loops for this BL PFC converter.

The dc bus voltage Vbus, sensed through one of the ADC channels, is compared against the reference bus voltage Vref. The resulting error signal Ev is then input the voltage loop controller Gv which regulates the bus voltage at the reference level. 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 1.1, 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 1.1, 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 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 of the PFC input voltage. The parameter Km is called 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 provides the reference signal for control of average inductor current $Iref_L$. However, for BL PFC system the current feedback is from the PFC switches (Q1 & Q2), and not from the boost inductors (L1 & L2). This means the reference signal $Iref_L$ for average inductor current control has to be converted before it is used for PFC switch current control. In Figure 1.1 this is indicated by the conversion block between $Iref_L$ and Iref. This reference current command Iref for the PFC current control loop is then compared against the PFC switch current Ipfc sensed through two ADC channels. The resulting current error signal Ei is then input the current loop controller Gc which generates the PFC duty ratio command d such that the PFC switch 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 to calculate the rectified input voltage, the RMS input voltage and the input line frequency. Based on the polarity of the input voltage the C2000 controller selects the appropriate PFC switch current (Isw1 or Isw2) to be used as PFC current feedback Ipfc. The polarity information is also used to set the appropriate PFC switch (Q1 & Q2) to either in PWM mode (boost PFC) or in forced ON mode. 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.

1.2. BL PFC Electrical Specifications

Following lists the key highlights of the C2000 BL PFC EVM.

- Input Voltage (AC Line): 95V (Min) to 240V (Max), 47~63Hz
- 400Vdc Output
- 300 Watts Output Power
- Full Load efficiency greater than 93%
- Power factor at 50% or greater load – 0.98 (Min)
- PWM frequency 200kHz

2 Software overview

2.1 Software Control Flow

The C2000 BLPFC project mostly makes use of the "C-background/ASM-ISR" framework. The main fast ISR (100kHz) runs in assembly environment. However, a slower ISR (10kHz) is also run from C environment. This slow ISR is made interruptible by the fast ISR.

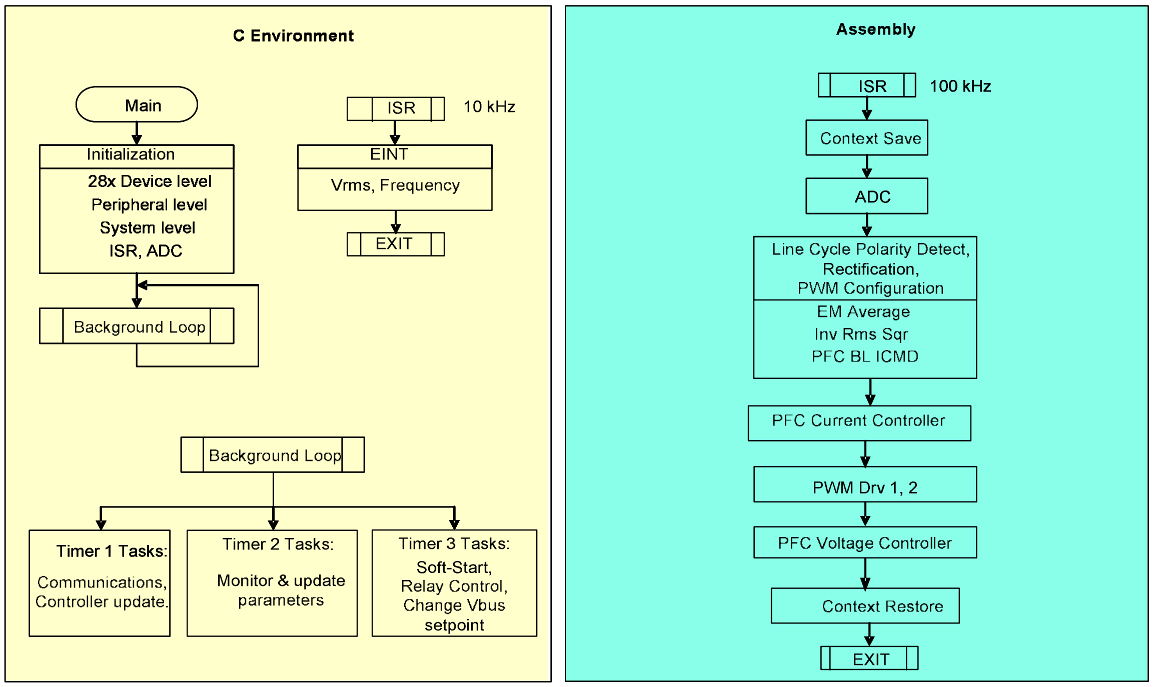


Figure 2.1.1. BL PFC Software flow diagram

The project 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 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 voltage and frequency of the input line voltage. Fig 2.1.1 depicts the general software flow for this project.

The key framework C files used in this project are:

BridgelessPFC-Main.c – this file is used to initialize, run, and manage the application.

BridgelessPFC-DevInit_F2802x.c or, *BridgelessPFC-DevInit_F2803x.c* – Depending on the control card (2802x or 2803x respectively) used in the BL PFC EVM one of these files will be in the CCS project. 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:

BridgelessPFC-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 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:

C configure function	**ASM initialization macro**	**ASM run-time macro**
PWM_1ch_UpDwnCnt_Cnf.c	PWMDRV_1ch_UpDwnCnt_INIT n	PWMDRV_1ch_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_BL_ICMD_INIT n	PFC_BL_ICMD n
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

Table 2.1.1 Library Modules

The assembly modules can also be represented graphically as below. (Figure 2.1.2)

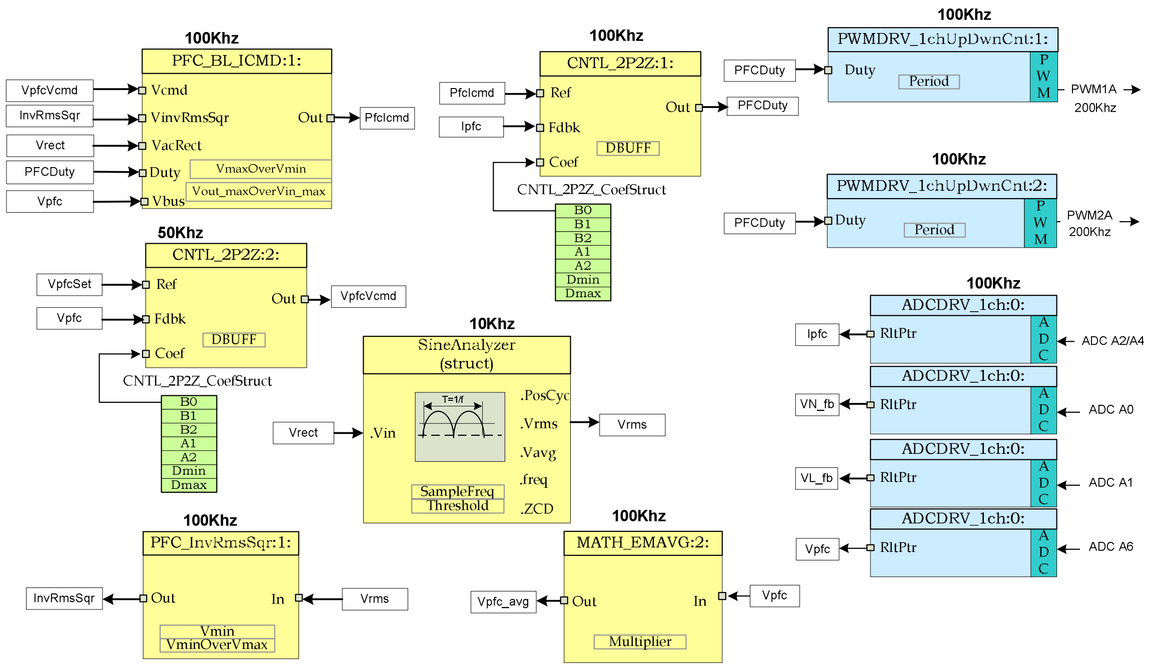


Figure 2.1.2 Software blocks

Note the color coding used for the modules in Fig 2.1.2. 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 Fig 2.1.3. 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.

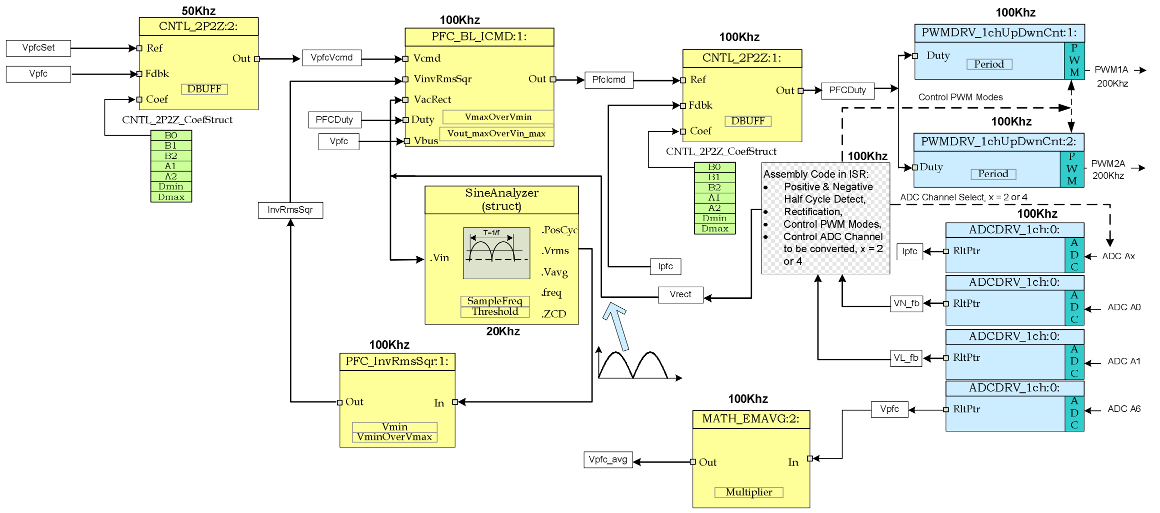


Figure 2.1.3. Software Control Flow

As mentioned in section 1.1 the BL PFC system is controlled by two feedback loops. The outer voltage loop regulates the DC bus voltage, while a faster inner current loop wave shapes the input current in order to maintain a high input power factor. Fig 2.1.3 also gives the rate at which the software modules 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 at 50kHz rate.

2.2 Incremental Builds

This project 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 BL PFC hardware. This approach also simplifies the task of debugging and testing the boards.

The build options are shown below. To select a particular build option set the macro INCR_BUILD, found in the *BridgelessPFC-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	Open voltage loop and closed current loop control of BL PFC
INCR_BUILD = 3	Complete voltage loop and current loop control of BL PFC

Table 2.2.1 Incremental build options for PFC

3 Procedure for running the incremental builds

All software files related to this C2x controlled BL 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\BLPFC_v1.0\BLPFC***. 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 overcurrent protection implemented on the board and so the user should take appropriate measures for preventing any output short circuit condition. **The BLPFC 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 below to build and run the example included in the PFC software.

3.1 Build 1: Open loop boost with ADC measurements

- **Objective**

The objectives of this build are, (1) evaluate BL PFC PWM and ADC software driver modules, (2) verify MOSFET gate driver circuit, voltage and current sensing circuit, (3) become familiar 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.

- **Overview**

The software in Build1 has been configured so that the user can quickly evaluate the PWM driver module 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 by viewing the ADC sampled data in the watch window.

The PWM and ADC driver macro instantiations are executed inside the _DPL_ISR. Fig 3.1.1 shows the software blocks used in this build. The two PWM signals for the two PFC switches are obtained from ePWM module 1 & 2. ePWM1A drives one of the PFC switches while ePWM2A drives the other.

The quantities that are sensed and fed back to the MCU include, (1) the line and neutral voltages (VL_fb, VN_fb), (2) the two PFC switch currents (I_{sw1}, I_{sw2}) each sampled at appropriate line half cycle and then saved as Ipfc, and (3) the DC bus voltage ($V_{\mathbf{pfc}}$). These quantities are read using the ADC driver module and are indicated in Fig 3.1.1. The ADC driver module converts the 12-bit ADC result to a 32bit 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 selection of the appropriate PFC switch current as Ipfc. These lines of code also configures the appropriate PWM output (PWM1A, PWM2A) to operate either in PWM mode or in forced ON mode depending on the input AC line half cycle.

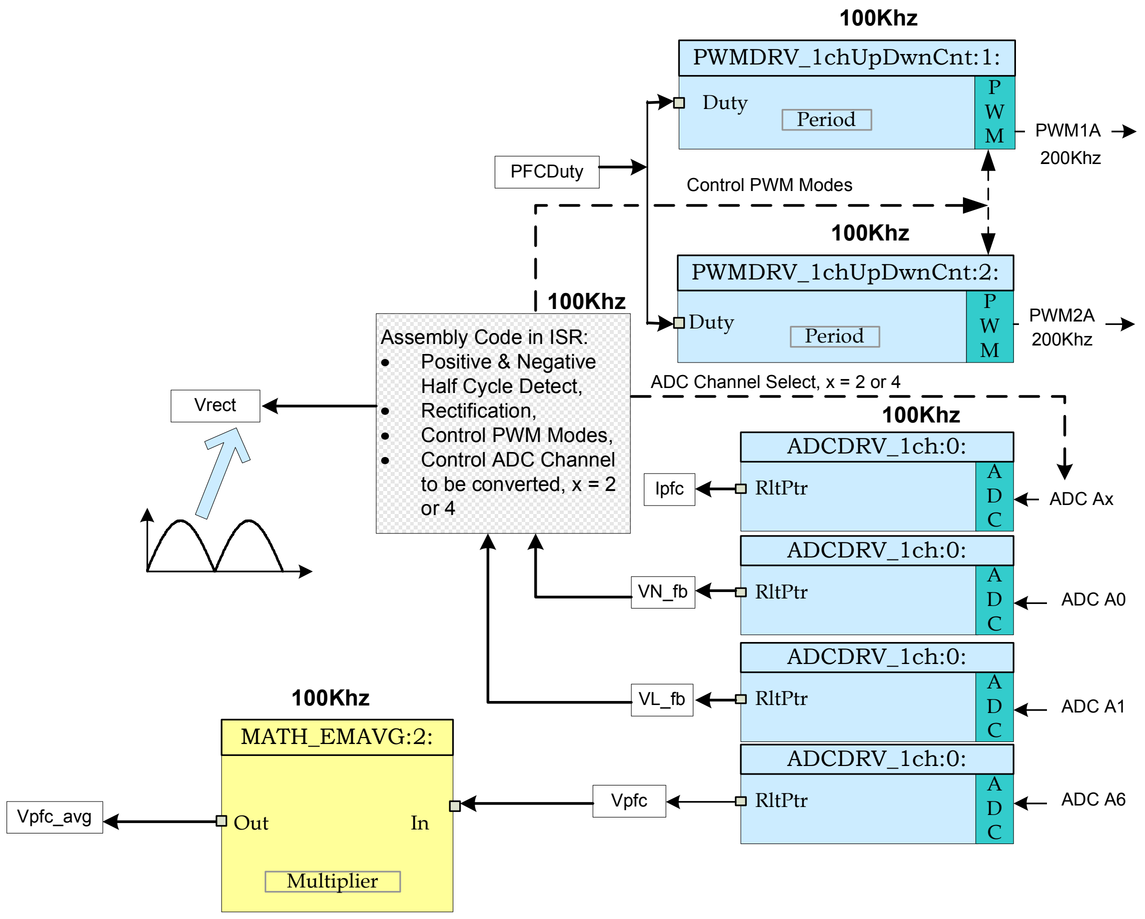


Figure 3.1.1. Build 1 software blocks

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, TBCNT2). The ePWM1 and ePWM2 modules are configured to operate in up-down count mode as shown in Fig 3.1.2. 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 switch 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 this sampled switch current value also represents a fraction of the average inductor current. This relationship between the sampled switch current and average inductor current can be derived by analyzing the related waveforms under DCM and CCM condition.

All the other voltage signal conversion is also initiated at this time. This is indicated in Fig 3.1.2. The flexibility of ADC and PWM modules on C2000 devices allow for precise and flexible ADC start of conversions. In this case ePWM1 is used as a time base to generate a start of conversion (SOC) trigger when the TBCNT1 reaches zero. A dummy ADC conversion is performed at this point in order to ensure the integrity of the ADC results.

Fig 3.1.2 also shows the PWM outputs when the input AC voltage is in positive half cycle. Corresponding switch current Isw1 is also shown as the current converted and saved as Ipfc for PFC current loop control.

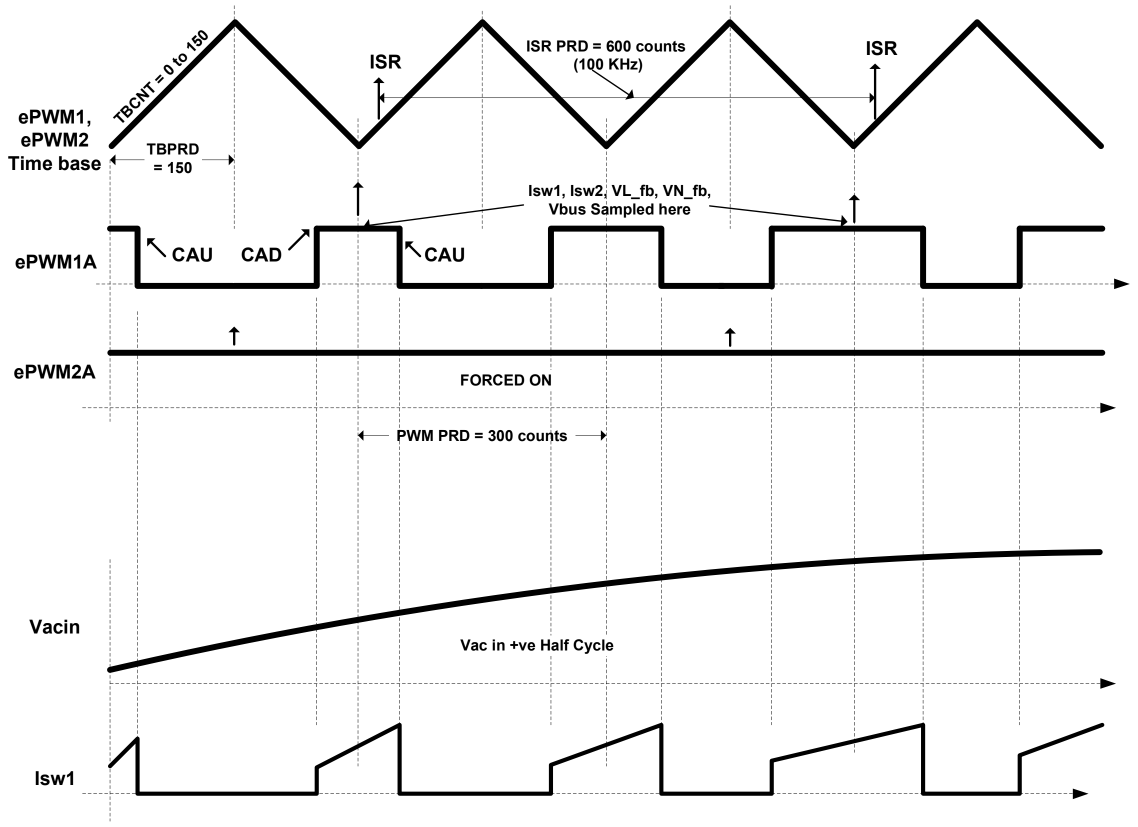


Figure 3.1.2. PWM generation and ADC sampling

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. ePWM2A is set to forced ON mode. This configuration of ePWM1A and ePWM2A happens during the positive half cycle of the input voltage. For negative half cycle of input voltage the two PWM output configurations are switched i.e., ePWM1A goes into forced ON mode and ePWM2A runs in PWM mode.

The CMPA value is derived from the input “PFCDuty” (Q24 variable) command.

Table 3.1.1 gives example CMPA values calculated for a TBPRD value of 150.

PFCDuty (Hex Values in Q24)	**CMPA = (PFCDuty/(2^{24}-1))*TBPRD**	**% Duty**
0x00200000	18	12.5
0x00800000	75	50
0x00FFFFFF	150	100

Table 3.1.1. Duty values for reference

The ADC module is configured to use SOCA of ePWM1 such that, SOCA is triggered at TBCNT1 = ZERO event. All conversions are completed using this SOCA trigger. These 5 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 EPWM1 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. This is where the *PWMDRV_1ch_UpDwnCnt* 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 3.1.2.

Protection

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

The sensed DC bus output voltage from the ADC input is compared against the overvoltage protection threshold set by the user. The default OV threshold set point is 440V. This threshold parameter is labeled as VBUS_OVP_THRSHLD inside the file ***BridgelessPFC-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.

➢ **Procedure**

Start CCS and Open a Project

Follow the steps below to execute this build:

1. Connect USB connector to the Piccolo controller board for emulation. Power up the 12V bias supply at JP1. 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 from RAM 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 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\BLPFC_v1.0\BLPFC*** directory. Make sure that under the Projects tab BLPFC 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 3.1.3:

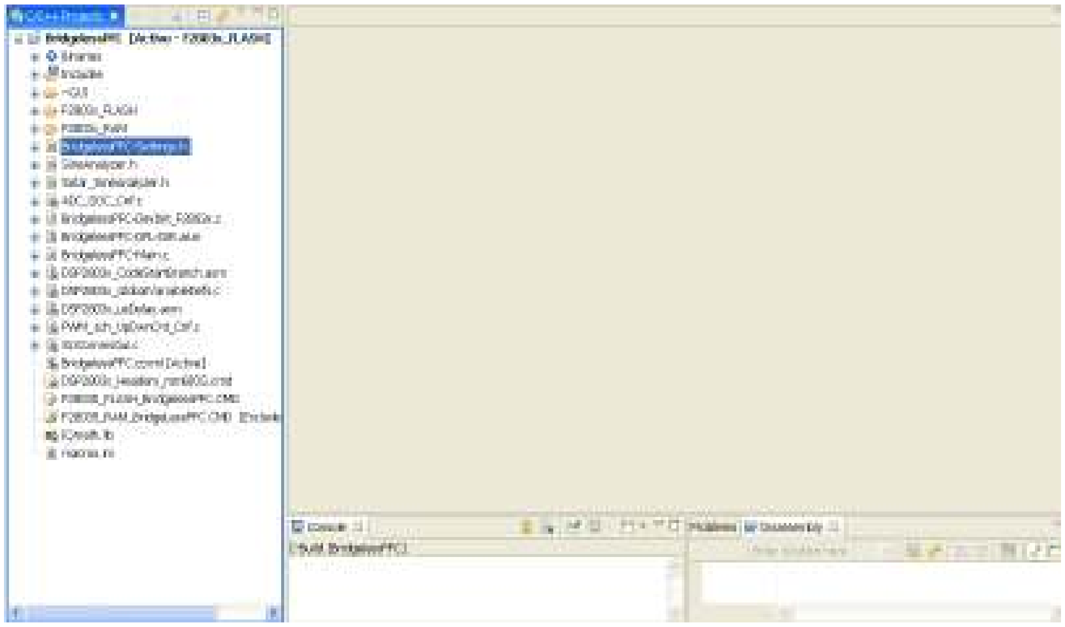

```
BridgelessPFC [Active - F2803x_FLASH]
  Binaries
  Includes
  ~GUI
  F2803x_FLASH
  F2803x_RAM
  BridgelessPFC-Settings.h
  SineAnalyzer.h
  [illegible]
  ADC_SOC_Cnf.c
  BridgelessPFC-DevInit_F2803x.c
  BridgelessPFC-DPL-ISR.asm
  BridgelessPFC-Main.c
  DSP2803x_CodeStartBranch.asm
  DSP2803x_GlobalVariableDefs.c
  DSP2803x_usDelay.asm
  PWM_1ch_UpDwnCnt_Cnf.c
  [illegible]
  BridgelessPFC.ccxml [Active]
  DSP2803x_Headers_nonBIOS.cmd
  F28035_FLASH_BridgelessPFC.CMD
  F28035_RAM_BridgelessPFC.CMD [Excluded]
  IQmath.lib
  [illegible]
```

Figure 3.1.3. CCS Project Window

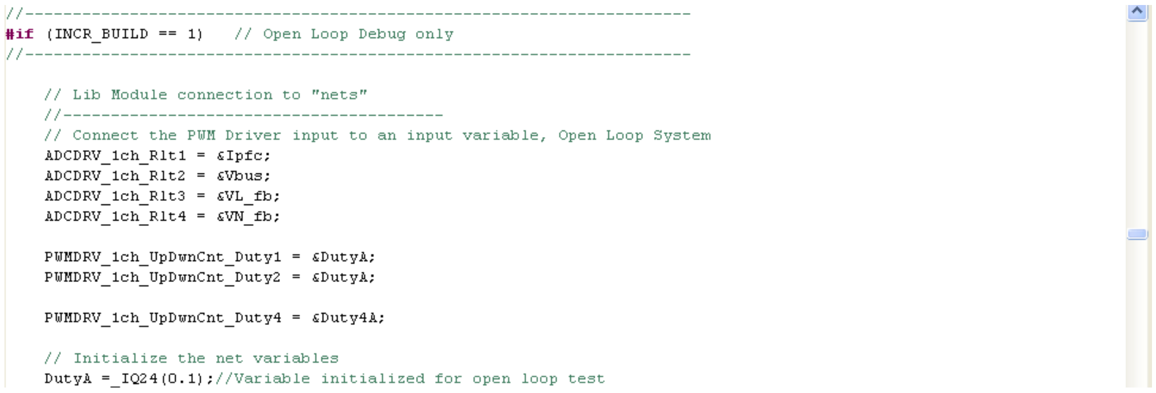
Device Initialization, Main, and ISR Files

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

4. Open and inspect *BridgelessPFC-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.

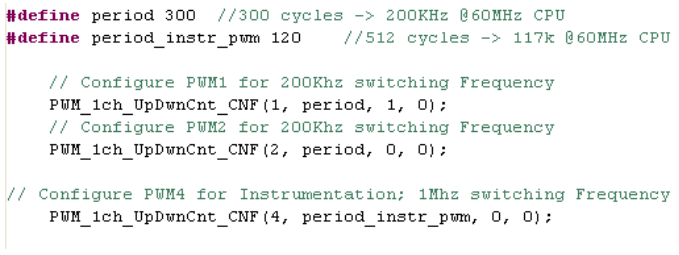
5. Open and inspect *BridgelessPFC -Main.c*. Notice the call made to *DeviceInit()* function and other variable initialization. Also notice code for different incremental build options, the ISR intialization and the background for(;;) loop.

6. Locate and inspect the following code in the main file under initialization code specific for build 1. This is where the *PWMDRV_1ch_UpDwnCnt* and *ADCDRV_1CH* blocks are connected in the control flow.



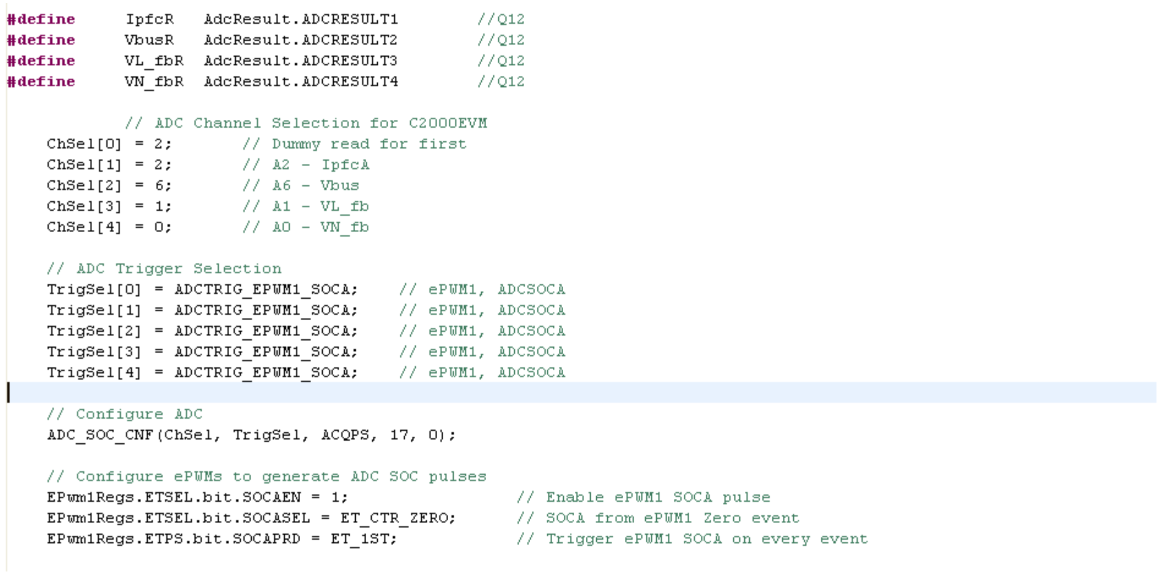
```
//----------------------------------------------------------------------
#if (INCR_BUILD == 1)   // Open Loop Debug only
//----------------------------------------------------------------------

    // Lib Module connection to "nets"
    //----------------------------------------
    // Connect the PWM Driver input to an input variable, Open Loop System
    ADCDRV_1ch_Rlt1 = &Ipfc;
    ADCDRV_1ch_Rlt2 = &Vbus;
    ADCDRV_1ch_Rlt3 = &VL_fb;
    ADCDRV_1ch_Rlt4 = &VN_fb;

    PWMDRV_1ch_UpDwnCnt_Duty1 = &DutyA;
    PWMDRV_1ch_UpDwnCnt_Duty2 = &DutyA;

    PWMDRV_1ch_UpDwnCnt_Duty4 = &Duty4A;

    // Initialize the net variables
    DutyA =_IQ24(0.1);//Variable initialized for open loop test
```

7. Locate and inspect the following code in the main file under initialization code. This is where the *PWMDRV_1ch_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.



```
#define period 300  //300 cycles -> 200KHz @60MHz CPU
#define period_instr_pwm 120    //512 cycles -> 117k @60MHz CPU

    // Configure PWM1 for 200Khz switching Frequency
    PWM_1ch_UpDwnCnt_CNF(1, period, 1, 0);
    // Configure PWM2 for 200Khz switching Frequency
    PWM_1ch_UpDwnCnt_CNF(2, period, 0, 0);

// Configure PWM4 for Instrumentation; 1Mhz switching Frequency
    PWM_1ch_UpDwnCnt_CNF(4, period_instr_pwm, 0, 0);
```

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.



```
#define     IpfcR   AdcResult.ADCRESULT1        //Q12
#define     VbusR   AdcResult.ADCRESULT2        //Q12
#define     VL_fbR  AdcResult.ADCRESULT3        //Q12
#define     VN_fbR  AdcResult.ADCRESULT4        //Q12

            // ADC Channel Selection for C2000EVM
    ChSel[0] = 2;       // Dummy read for first
    ChSel[1] = 2;       // A2 - IpfcA
    ChSel[2] = 6;       // A6 - Vbus
    ChSel[3] = 1;       // A1 - VL_fb
    ChSel[4] = 0;       // A0 - VN_fb

    // ADC Trigger Selection
    TrigSel[0] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA
    TrigSel[1] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA
    TrigSel[2] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA
    TrigSel[3] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA
    TrigSel[4] = ADCTRIG_EPWM1_SOCA;    // ePWM1, ADCSOCA

    // Configure ADC
    ADC_SOC_CNF(ChSel, TrigSel, ACQPS, 17, 0);

    // Configure ePWMs to generate ADC SOC pulses
    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
```

8. Open and inspect *BridgelessPFC-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.

Build and Load the Project

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

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

10. Click Project→"Rebuild All" button and watch the tools run in the build window.

11. 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-F28035.ccxml). Check "Use shared location" and click Finish.

12. 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.

13. Click Target→"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().

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.

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 in Figure 3.1.4

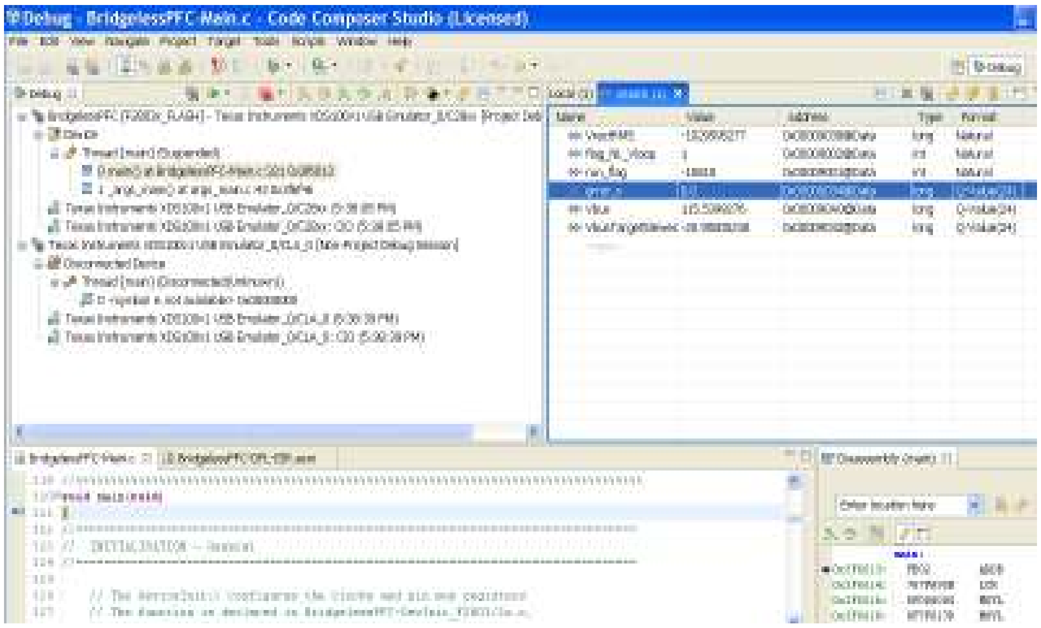

Debug - BridgelessPFC-Main.c - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help

Figure 3.1.4. CCS watch view for Build 1

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 tuning control law parameters on-the-fly, for example.

14. 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.

15. 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.

16. Click on Continuous Refresh buttons for the watch view.

Run the Code

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

18. 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.

19. Apply an appropriate resistive load to the PFC system at the DC output (10~100W).

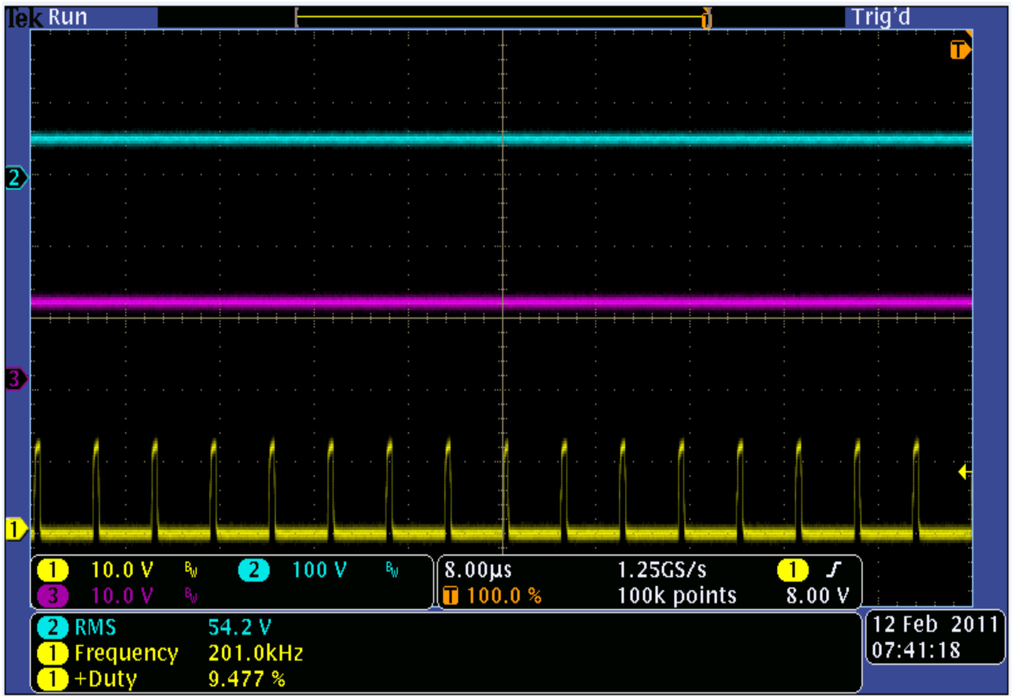
20. Apply AC Power to the board. Measure and verify the DC bus voltage corresponding to applied input voltage and the duty ratio.

21. Use DutyA to slowly change the duty from the watch window. The boost converter output voltage should change accordingly.

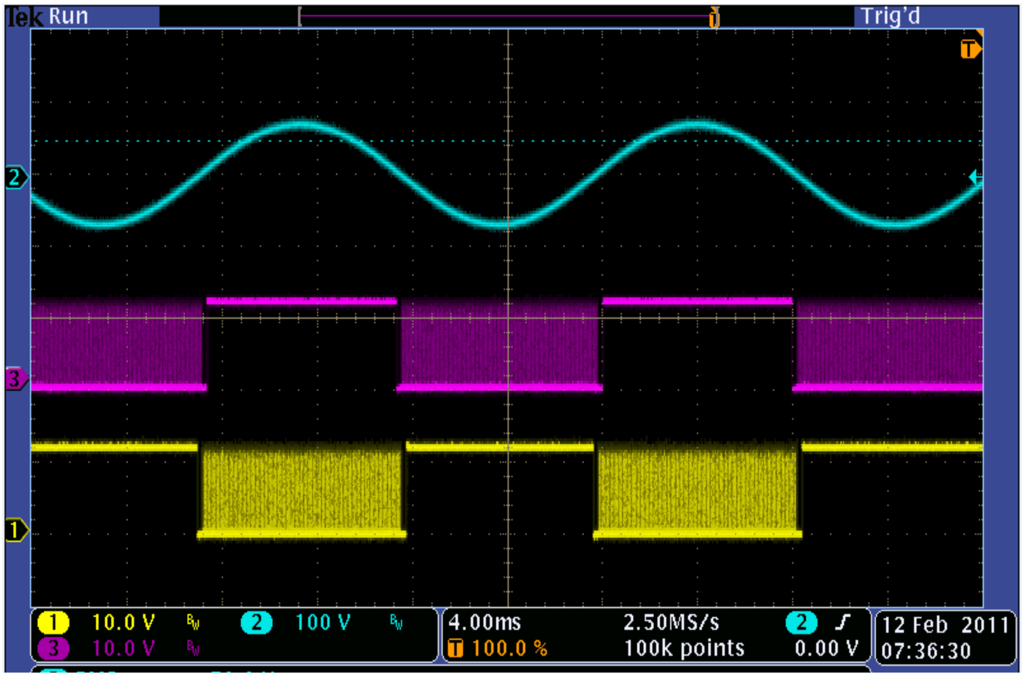
    Observe the output voltage carefully, this should not be allowed to exceed the maximum voltage rating of the board.

22. 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 20 above). In that case the PFC stage will temporarily operate in a pure dc-dc boost mode.

23. The following oscilloscope captures show two PWM outputs (Ch1 & Ch3) and the ***DC input voltage*** (Ch2) when the output DC bus load is 8K ohm and the set duty ratio is about 10%. Since the input voltage in this case is always positive (+50V in this case) PWM1A will be in PWM mode (Ch1) and PWM2A will be in forced ON mode (Ch3). The PWM frequency is also measured to be 200kHz.



24. The following oscilloscope captures show two PWM outputs (Ch1 & Ch3) and the ***AC input voltage*** (Ch2) of about 50V when the output DC bus load is 8K ohm and the set duty ratio is about 10%. With the AC input voltage PWM1A and PWM2A alternately switch between PWM mode and forced ON mode.



25. 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.

26. 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.

27. You may choose to leave Code Composer Studio running for the next exercise or optionally close CCS.

End of Exercise

3.2 Build 2: BL PFC with closed current loop

- **Objective**

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

- **Overview**

Fig 3.2.1 shows the software blocks used in this build. Notice that 4 additional software blocks are added to the Build 1 diagram (Figure 3.1.1) 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), the sensed DC bus voltage (Vpfc) and PFC PWM duty are used in the 3rd software block *PFC_BL_ICMD* to generate the reference current command *PfcIcmd* for the PFC current control loop. *PFC_BL_ICMD* block uses a 5th 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 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 3.2.1 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 Fig 3.2.1 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.

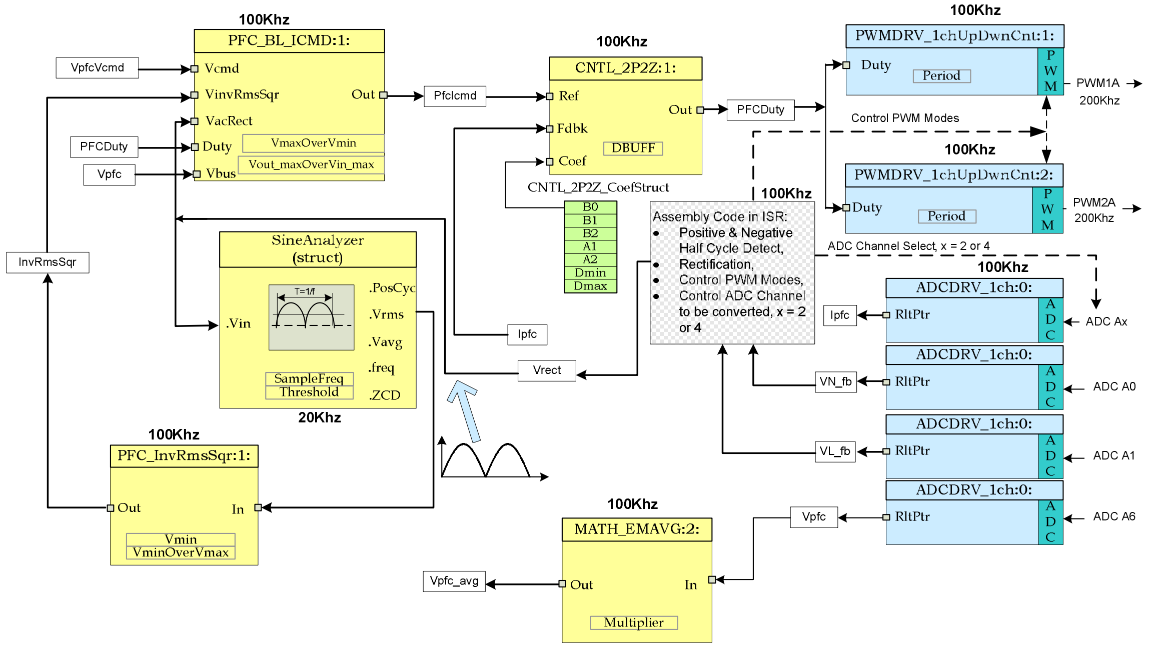


Figure 3.2.1. Build 2 software blocks

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 next section 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. To change these coefficients from the GUI environment (or from CCS watch views) they are further scaled to values ranging from 0 to 9999.

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_BL_ICMD* block as mentioned before. The z-domain transfer function for *CNTL_2P2Z* is given by:

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

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:

$$b_0 = K_p + K_i + K_d$$
$$b_1 = -K_p + K_i - 2K_d$$
$$b_2 = K_d$$

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

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

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

$a_1 = -1$ and $a_2 = 0$

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.

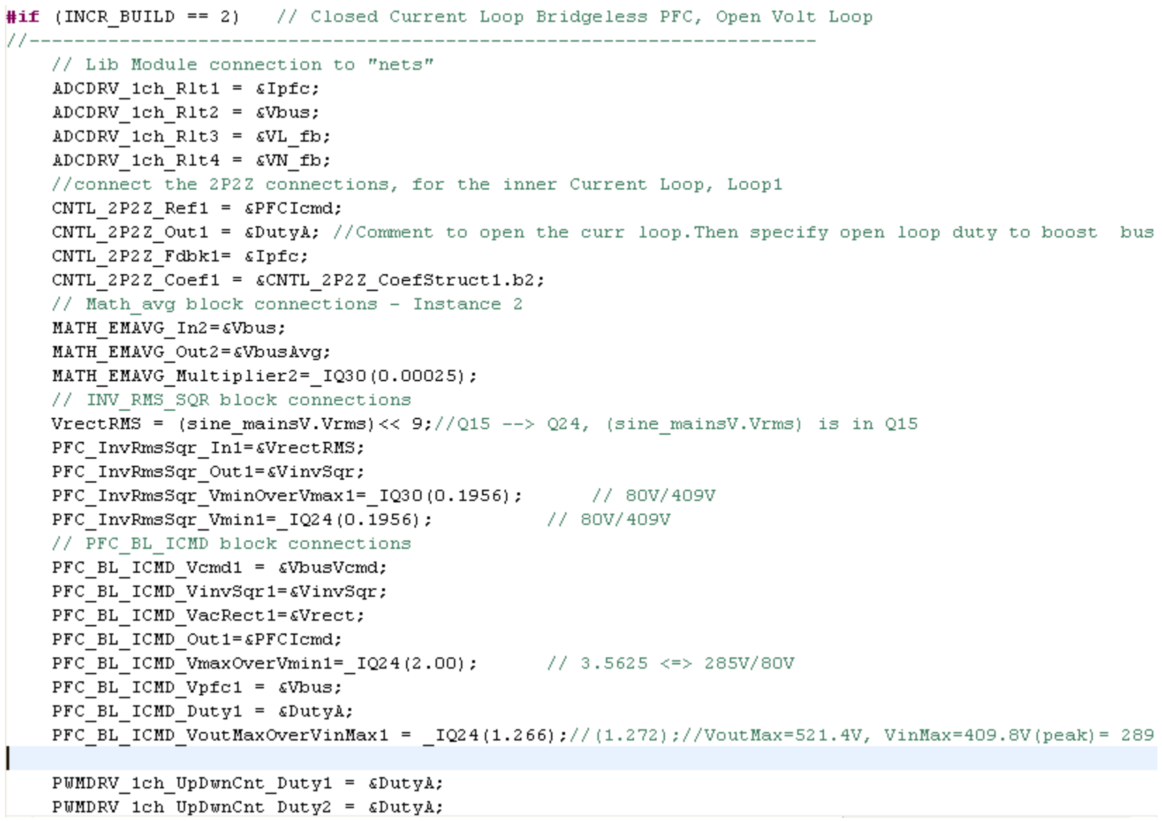
➢ **Procedure**

Build and Load Project

Follow the steps below to execute this build:

Follow steps 1 through 7 exactly as in build 1(section 3.1) except that in step 6 select build 2 option instead of build 1. Then complete step 6 as below:

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 Bridgeless PFC, Open Volt Loop
//----------------------------------------------------------------------
    // Lib Module connection to "nets"
    ADCDRV_1ch_Rlt1 = &Ipfc;
    ADCDRV_1ch_Rlt2 = &Vbus;
    ADCDRV_1ch_Rlt3 = &VL_fb;
    ADCDRV_1ch_Rlt4 = &VN_fb;
    //connect the 2P2Z connections, for the inner Current Loop, Loop1
    CNTL_2P2Z_Ref1 = &PFCIcmd;
    CNTL_2P2Z_Out1 = &DutyA; //Comment to open the curr loop.Then specify open loop duty to boost  bus
    CNTL_2P2Z_Fdbk1= &Ipfc;
    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_BL_ICMD block connections
    PFC_BL_ICMD_Vcmd1 = &VbusVcmd;
    PFC_BL_ICMD_VinvSqr1=&VinvSqr;
    PFC_BL_ICMD_VacRect1=&Vrect;
    PFC_BL_ICMD_Out1=&PFCIcmd;
    PFC_BL_ICMD_VmaxOverVmin1=_IQ24(2.00);      // 3.5625 <=> 285V/80V
    PFC_BL_ICMD_Vpfc1 = &Vbus;
    PFC_BL_ICMD_Duty1 = &DutyA;
    PFC_BL_ICMD_VoutMaxOverVinMax1 = _IQ24(1.266);//(1.272);//VoutMax=521.4V, VinMax=409.8V(peak)= 289

    PWMDRV_1ch_UpDwnCnt_Duty1 = &DutyA;
    PWMDRV_1ch_UpDwnCnt_Duty2 = &DutyA;
```

1) Open and inspect ***BridgelessPFC-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.

2) Select the Incremental build option as 2 in the ***BridgelessPFC-Settings.h*** file. Then follow steps 10 through 17 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 ***BridgelessPFC-Settings.h*** always do a "Rebuild All"

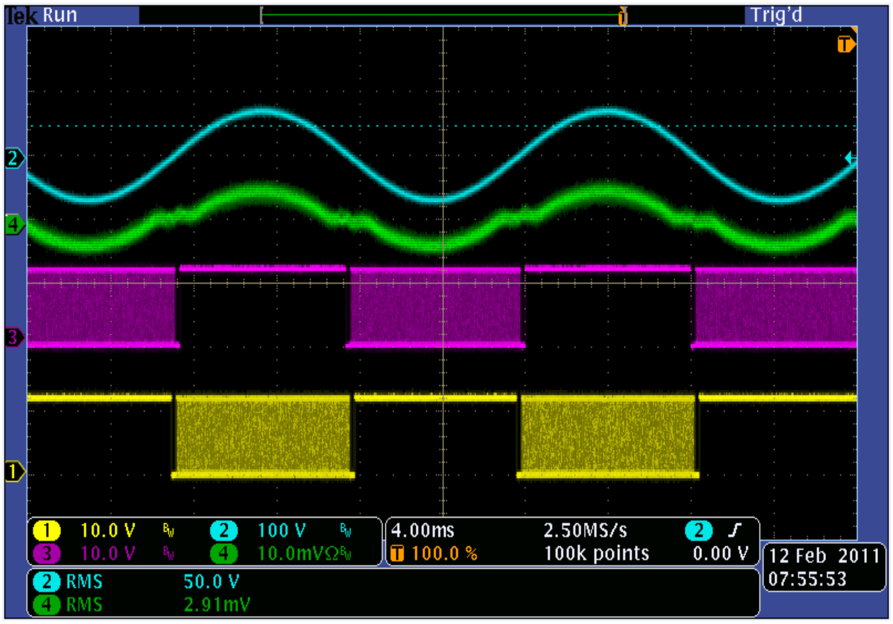
3) Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.

4) In the watch view, add the variable *VpfcVcmd* and set it to 0.05 (=838861 in Q24). This variable sets the magnitude of the reference current command for the current control loop.

5) 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 5W at 200V bus voltage.

6) 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 50V rms. The bus voltage now

should be around 200V. Adjust the value for *VpfcVcmd* to set the bus voltage to about 200V. Use an oscilloscope with voltage and current probes to observe the input voltage, input current and the PWM outputs. With a 50V rms input, 8.0kohm resistive load and bus voltage set to 200V you should see the following waveforms. Here Ch1 and Ch3 show the PWM outputs. Ch2 is the input voltage and Ch4 is the input current. With the current loop closed the input current should have the same shape of the input voltage with good power factor.



7) Increase *VpfcVcmd* slightly (in steps of 0.01) 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 a value less than 350V. This threshold parameter is labeled as VBUS_OVP_THRSHLD inside the file ***BridgelessPFC-Settings.h***. Now change the input voltage or the load resistance to see the PFC operation under current control loop.

8) Follow steps 26 and 27 as in section 3.1 to turn off power and reset the MCU.

End of Exercise

3.3 Build 3: BL PFC with closed voltage and current loop

➢ **Objective**

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

➢ **Overview**

Fig 3.3.1 shows the software blocks used in this build. Compared to build 2 in Figure 3.2.1 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 BL 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_BL_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.

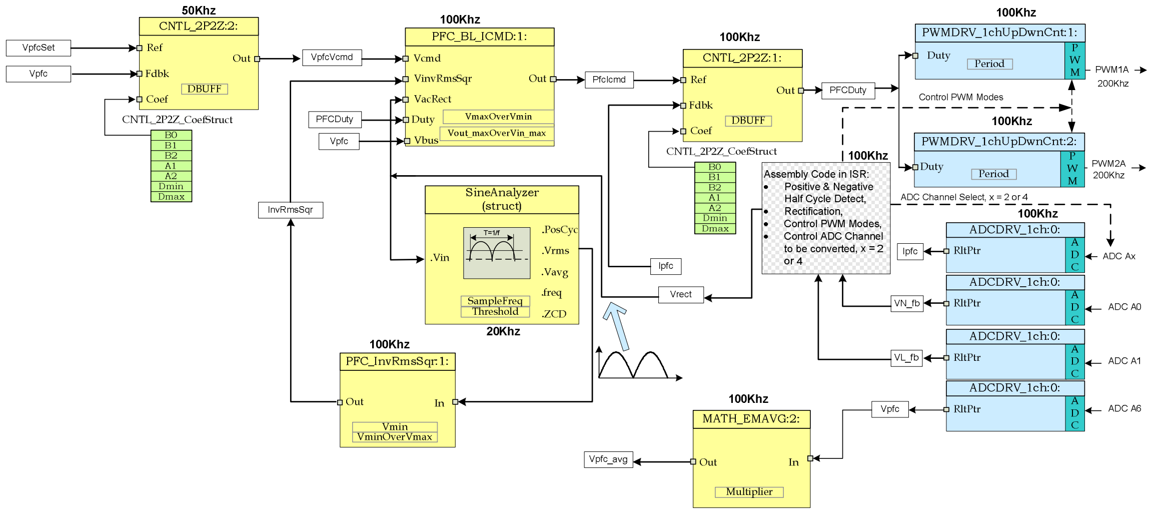


Figure 3.3.1. Build 3 software blocks

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 3.2.

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 (default setting around 100Vdc) the inrush relay control circuit is activated to bypass the inrush current control resistor. Following this the PFC action is enabled and the output DC bust slowly ramps up to the desired value of about 400Vdc. This 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.

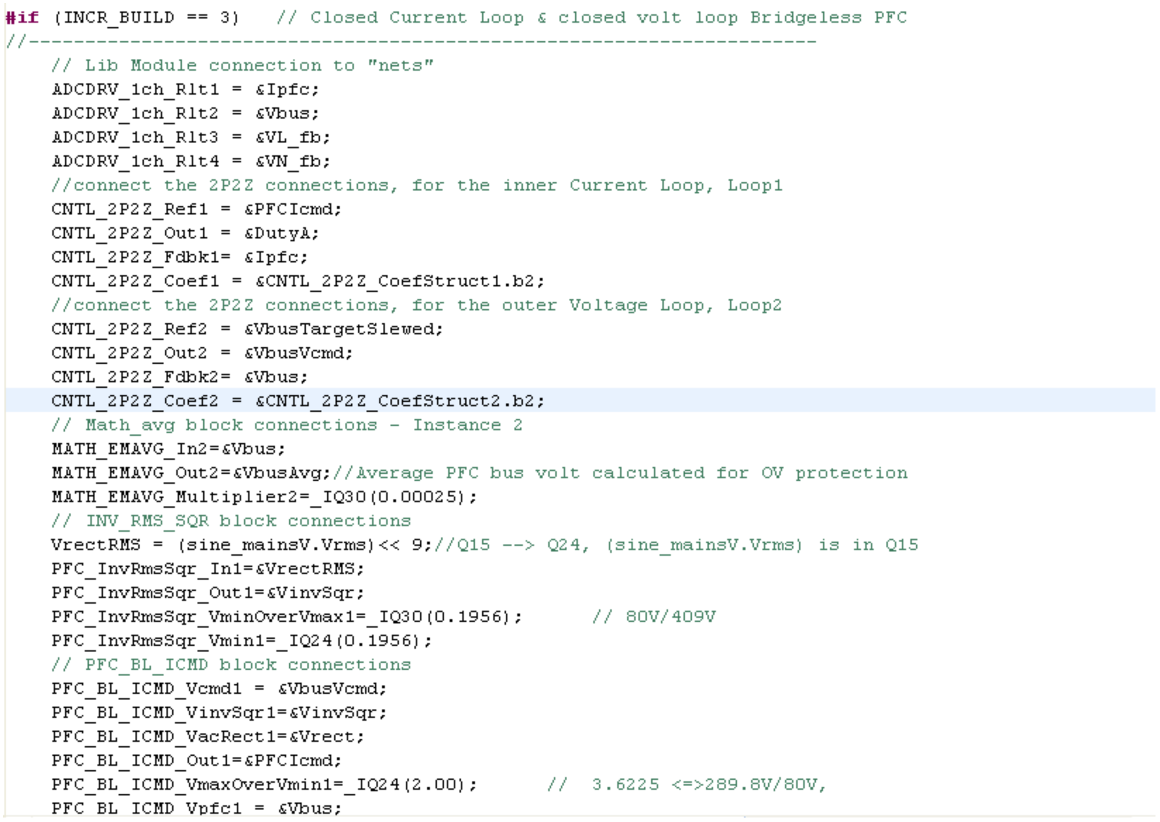
➢ **Procedure**

Build and Load Project

Follow the steps below to execute this build:

Follow steps 1 through 7 exactly as in build 1(section 3.1) except that in step 6 select build 3 option instead of build 1. Then complete step 6 as below:

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 Bridgeless PFC
//---------------------------------------------------------------------
    // Lib Module connection to "nets"
    ADCDRV_1ch_Rlt1 = &Ipfc;
    ADCDRV_1ch_Rlt2 = &Vbus;
    ADCDRV_1ch_Rlt3 = &VL_fb;
    ADCDRV_1ch_Rlt4 = &VN_fb;
    //connect the 2P2Z connections, for the inner Current Loop, Loop1
    CNTL_2P2Z_Ref1 = &PFCIcmd;
    CNTL_2P2Z_Out1 = &DutyA;
    CNTL_2P2Z_Fdbk1= &Ipfc;
    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_BL_ICMD block connections
    PFC_BL_ICMD_Vcmd1 = &VbusVcmd;
    PFC_BL_ICMD_VinvSqr1=&VinvSqr;
    PFC_BL_ICMD_VacRect1=&Vrect;
    PFC_BL_ICMD_Out1=&PFCIcmd;
    PFC_BL_ICMD_VmaxOverVmin1=_IQ24(2.00);      //  3.6225 <=>289.8V/80V,
    PFC_BL_ICMD_Vpfc1 = &Vbus;
```

1) Open and inspect ***BridgelessPFC-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.

2) Select the Incremental build option as 3 in the ***BridgelessPFC-Settings.h*** file. Then follow steps 10 through 17 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().

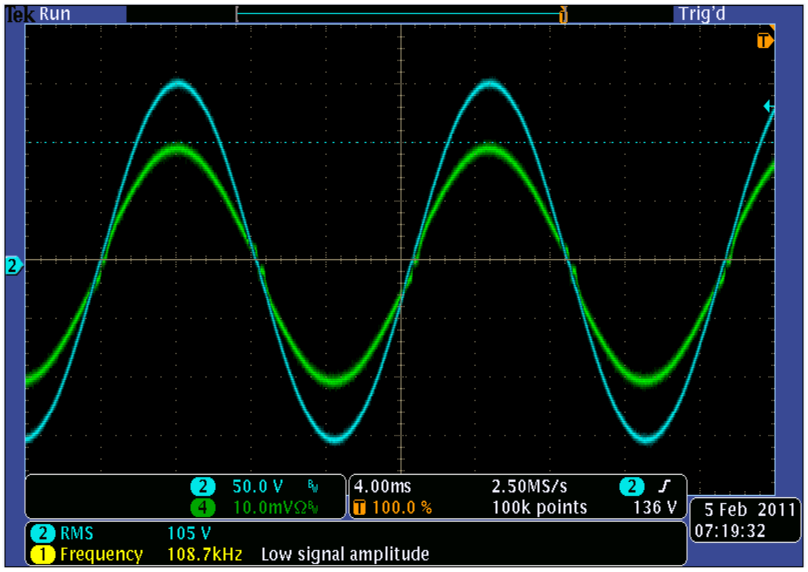
   Note: Whenever you change the incremental build option in ***BridgelessPFC-Settings.h*** always do a "Rebuild All"

3) Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.

4) In the watch view, add the variables *VbusTargetSlewed*, *Vbus* and set the Q-format to Q24. These variables represent the reference bus voltage and the feedback bus voltage respectively. These will slowly increase to the setpoint value as the PFC starts up when AC power is applied.

5) 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 20W at 400V bus voltage.

6) Configure an isolated AC source to output 110V, 60Hz, AC voltage output. Use a voltmeter to monitor the DC bus voltage. Turn on the AC source output for 110Vrms. When the DC bus voltage reaches 100V the inrush relay will activate and the bus voltage will slowly increase to 400v. Notice that *VbusTargetSlewed* and *Vbus* variables on the watch window show a value of about 0.7707 (=400/519) when the Q format is set to Q24. The maximum bus voltage set by the Vbus sense resistors is about 519V that corresponds to maximum ADC input of 3.3V. Therefore, the normalized or per unit value will be about 0.7707 when the actual bus voltage is 400Vdc. Adjust *VbusTargetSlewed* to 0.732 (=380/519) to set the bus voltage to about 380V. Use an oscilloscope with voltage and current probes to observe the input voltage, input current and the PWM outputs. With a 110V rms input, 540 ohm resistive load and bus voltage set to 400V you should see the following waveforms.

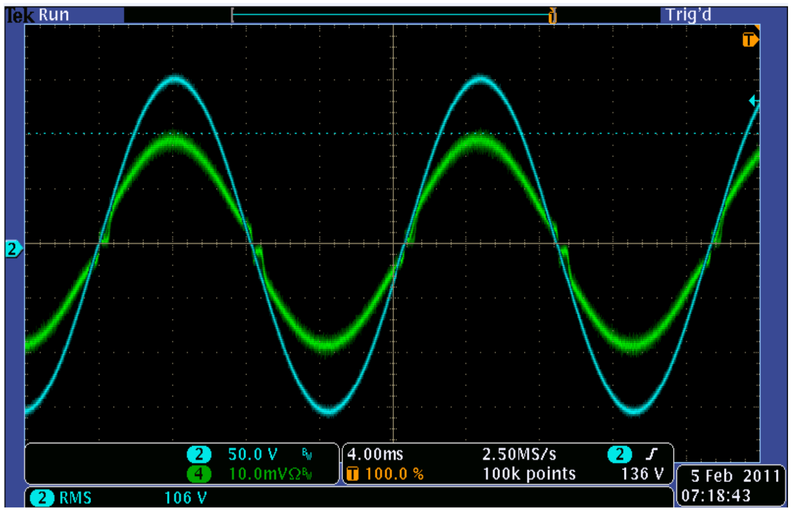


2 RMS 105 V
1 Frequency 108.7kHz Low signal amplitude

BL PFC Input Voltage and Current waveforms at 300W

Here Ch2 is the input voltage and Ch4 is the input current.

With 110V rms input, 1080 ohm resistive load and bus voltage set to 400V you should see the following waveforms.

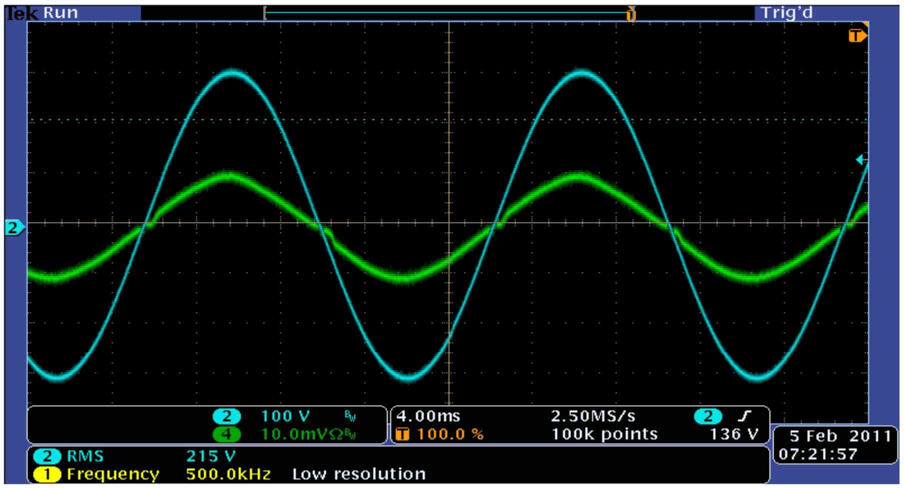


BL PFC Input Voltage and Current waveforms at 150W

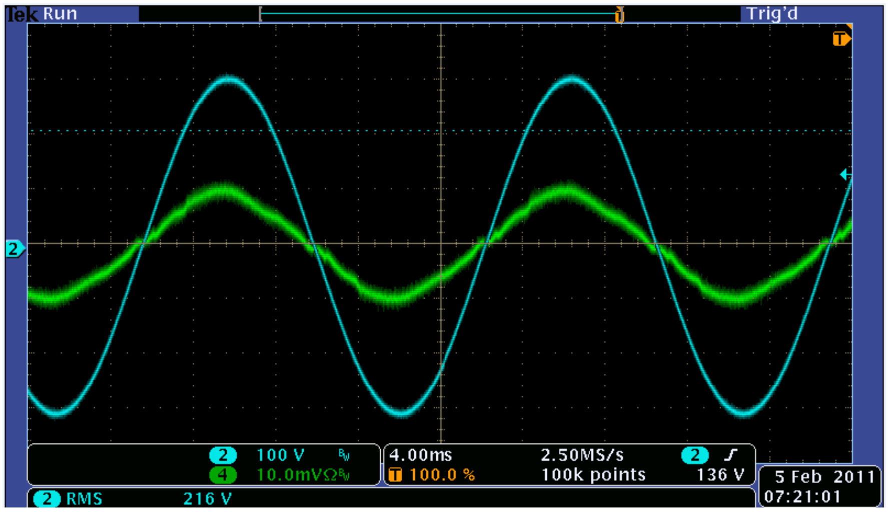
Change the input voltage (90Vrms~240Vrms) or the load resistance (0~300W) to see the PFC operation under closed current and voltage control loop.

7) Follow steps 26 and 27 as in section 3.1 to turn off power and reset the MCU.

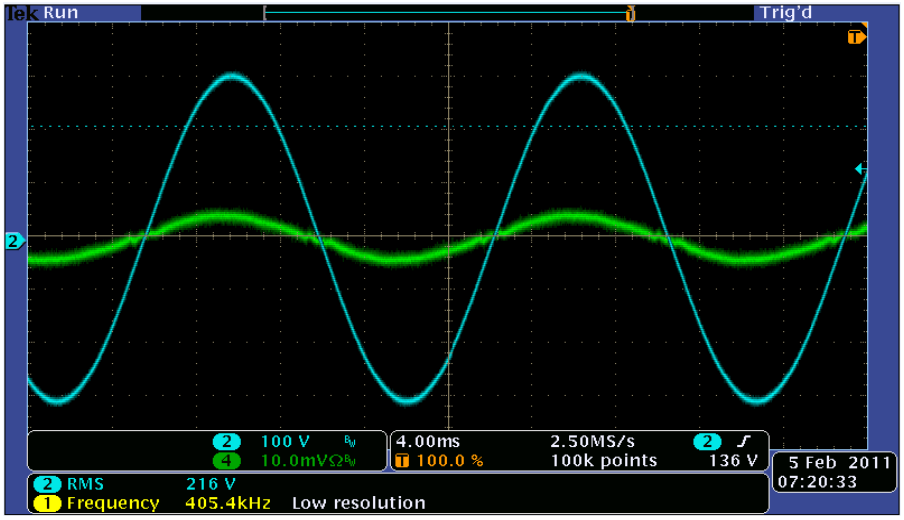
Additional Test Results



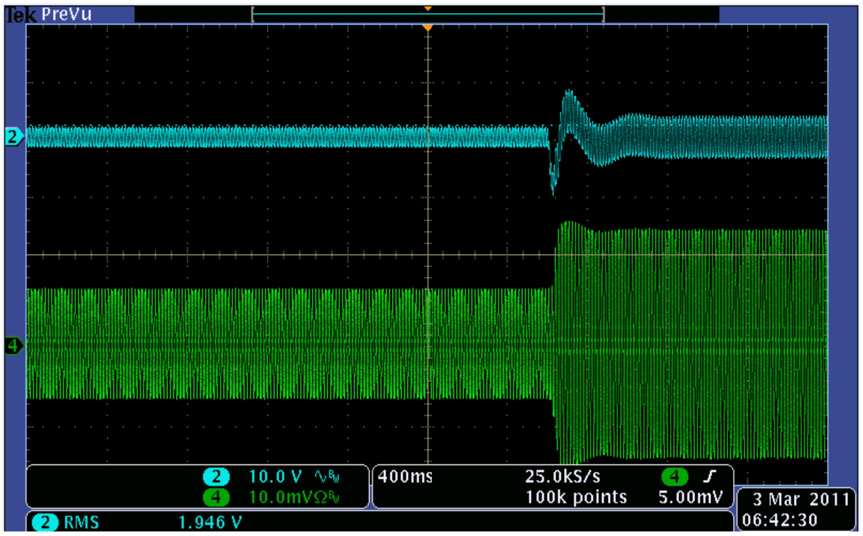
BL PFC Input Voltage and Current waveforms, Ch2 –Vin, Ch4-Iin, Vrms=220V, Vbus=400V, Pout=300W



BL PFC Input Voltage and Current waveforms, Ch2 –Vin, Ch4-Iin, Vrms=220V, Vbus=400V, Pout=150W

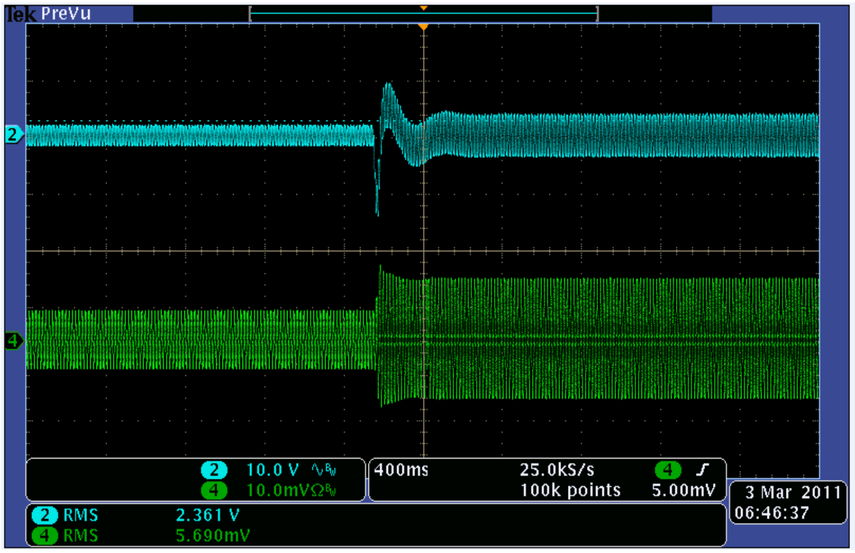


BL PFC Input Voltage and Current waveforms, Ch2 –Vin, Ch4-Iin, Vrms=220V, Vbus=400V, Pout=60W



BL PFC DC Bus Load Transient Response,

Ch2 –Vbus, Ch4-Iin, Vrms=110V, Vbus=400V, Load Step 150W~300W



BL PFC DC Bus Load Transient Response,

Ch2 –Vbus, Ch4-Iin, Vrms=220V, Vbus=400V, Load Step 150W~300W

End of Exercise

References

For more information please refer to the following guides:

- **BLPFC-GUI-QSG** – A quick-start guide for quick demo of the BLPFC EVM using a GUI interface.

  ..\controlSUITE\development_kits\BLPFC \~Docs\BLPFC-GUI-QSG.pdf

- **BLPFC_Rel-1.0-HWdevPkg** – A folder containing various files related to the Piccolo-B controller card schematics and the BL PFC schematic.

  This PFC EVM uses a Piccolo-B control card and not the Piccolo-A card just because of the absence of a RC filter in the Piccolo-A card. This RC filter is available on both the ADC channels on Piccolo-B card that sense two BL PFC switch currents. However, on Piccolo-A card only one of the two ADC channels has this RC filter. A revised version of this Picocolo-A control card can be easily used to implement full control of this BL PFC EVM.

  ..\controlSUITE\development_kits\BLPFC\BLPFC_HWDevPkg

- **F28xxx User's Guides**

  http://www.ti.com/f28xuserguides