Digitally Controlled HV Solar MPPT DC-DC Converter Using C2000 Piccolo Microcontroller

CCS User Guide
Version 1.0
October 2011

Abstract

This document presents the implementation details of a digitally controlled DC-DC converter that is used as a front end converter for solar inverter (DC-AC) application. It implements an isolated DC-DC stage with maximum power point tracking (MPPT) algorithm in order to utilize the full capacity of a 500W solar panel. It maintains its input voltage at the reference set point generated by the MPPT algorithm and delivers power to a downstream DC-AC inverter when connected across its output. The DC-AC inverter transfers the power from the DC-DC stage to an emulated grid connected across its own output. A C2000 Piccolo-B control card and a 500W isolated DC-DC stage EVM are used to implement the complete DC-DC system.

This EVM comes with a Piccolo-B control card and not the Piccolo-A card. However, a Picocolo-A control card can also be used to implement full control of the EVM.

1 Introduction

Photovoltaic (PV) systems based on solar energy offer an environmentally friendly source of electricity. A key feature of such PV system is the efficiency of conversion at which the power converter stage can extract the energy from the PV arrays and deliver to the load. The maximum power point tracking (MPPT) of the PV output for all sunshine conditions allows reduction of the cost of installation and maximizes the power output from the PV panel. Therefore, a DC-DC converter employing some MPPT algorithm is generally used as a front-end converter to efficiently extract the PV output power and convert the PV output voltage to a high voltage DC bus. The DC-DC converter, depending on the system requirement, can use either an isolated power stage or a non-isolated stage. The high voltage bus from the DC-DC converter is then fed to power the DC-AC inverter that eventually supplies the load and connects to the grid.

This C2000 MPPT DC-DC EVM uses an isolated DC-DC stage as is shown in Fig 1. It consists of two DC-DC stages. These are, (1) a 2-ph interleaved boost converter and, (2) an isolated half bridge LLC resonant converter.

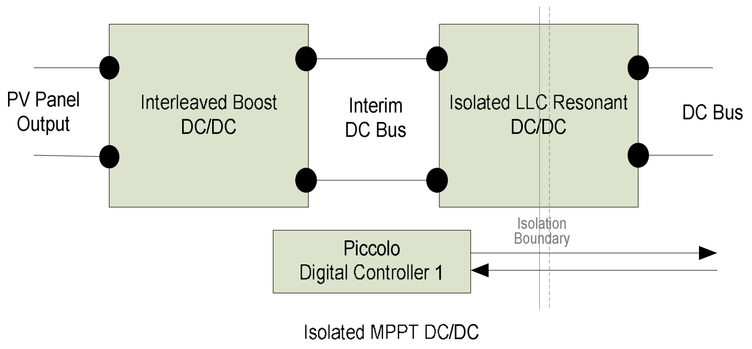


Figure 1. Isolated DC-DC Converter Block Diagram

The DC-DC converter draws dc current from the PV panel such that the panel operates at its maximum power transfer point. This requires maintaining the panel output, i.e., the DC-DC converter input at a level determined by the MPPT algorithm. This is implemented in the 2-ph interleaved boost converter stage. The isolated LLC resonant converter simply provides high frequency isolation for the DC-DC stage.

A C2000 piccolo microcontroller with its on-chip PWM, ADC and analog comparator modules is able to implement complete digital control of such MPPT DC-DC system.

1.1. DC-DC Stage Implementation

Fig 1.1 illustrates a C2000 based MPPT DC-DC converter control system. The PV panel output voltage, Vpv, is applied to the 2-ph interleaved boost stage.

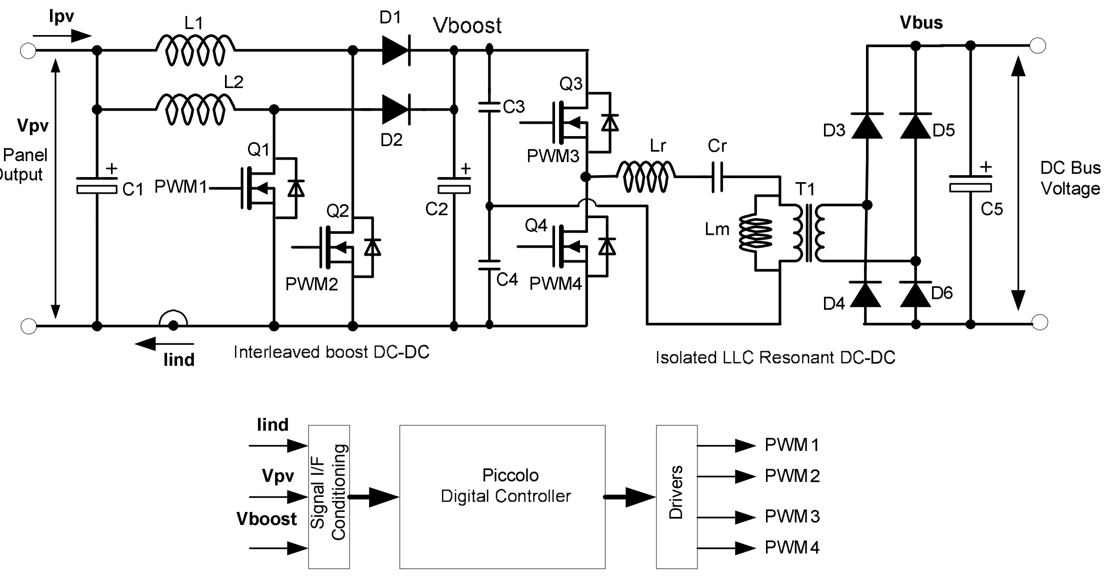


Figure 1.1 MPPT DC-DC Converter Control using C2000 Micro-controller

Inductor L1, MOSFET Q1, and diode D1 together form one of the boost stages while, L2, Q2, and D2 form the other. A capacitor C2 at the boost converter output acts as an energy reservoir and provides boost voltage to the resonant LLC stage.

The H bridge LLC resonant stage consists of MOSFETs Q3~Q4, input capacitors C3~C4, resonant inductor Lr, resonant capacitor Cr, transformer T1, output rectifiers D3~D6 and output capacitor C5. This stage has a voltage ratio of 1 and provides the isolation between the primary and secondary side.

Figure 1.1 indicates all the interface signals needed for full control of this DC-DC converter using a C2000 micro-controller (MCU). The MCU controls the hardware using three feedback signals and four PWM outputs. The signals that are sensed and fed back to the MCU include the panel output voltage (Vpv) and the boost output voltage (Vboost) and the total boost inductor currents (I_{ind}). These sensed signals are used to implement the voltage and current control loops for the DC-DC boost stage. The interleaved boost DC/DC topology is chosen to boost the variable DC output to a fixed DC bus voltage. The main reason for using this topology is the wide input voltage variation. The PWM signals for the power switches Q1 and Q2 is phase-shifted by 180 degrees. This helps reduce the ripple in the PV panel current.

The LLC stage runs at open loop with its PWM frequency set to be the same as the resonant frequency. The Piccolo controller shares the common ground with the primary side of the LLC stage and there is no isolated feedback to the controller from the LLC secondary output terminals. Thus the LLC is run under open loop and, therefore, it is necessary to maintain a voltage conversion factor of 1. This is achieved by making (1) the LLC PWM frequency the same as the resonant frequency and, (2) by maintaining a minimum load of about 10W across the LLC output.

Figure 1.2 shows the DC-DC interleaved boost converter control loops. This uses current mode control. However, the goal is to control the PV panel output (Vpv) which is the input to the DC-DC stage. This allows the PV panel (array) operates at its maximum power point at all time. Input current is regulated by adjusting the duty cycles of the power switches Q1 and Q2. Input voltage is regulated by adjusting the input current. A Maximum Power Point Tracking algorithm described in the next section is responsible for determining the set point (Vpv_ref) for the PV panel voltage. Notice that the input voltage control loop works quite differently compared to conventional feedback used in output voltage control. Under this control scheme, when the PV panel voltage (Vpv) tends to go higher than the reference panel voltage (Vpv_ref) set by the MPPT algorithm, the control loop increases the panel current command (reference current for inner current loop Iind_ref) and thereby controls the panel voltage at its reference level (Vpv_ref). When the panel voltage tends to go lower than the reference, the control loop reduces the panel current command in order to reestablish the panel voltage to its reference level.

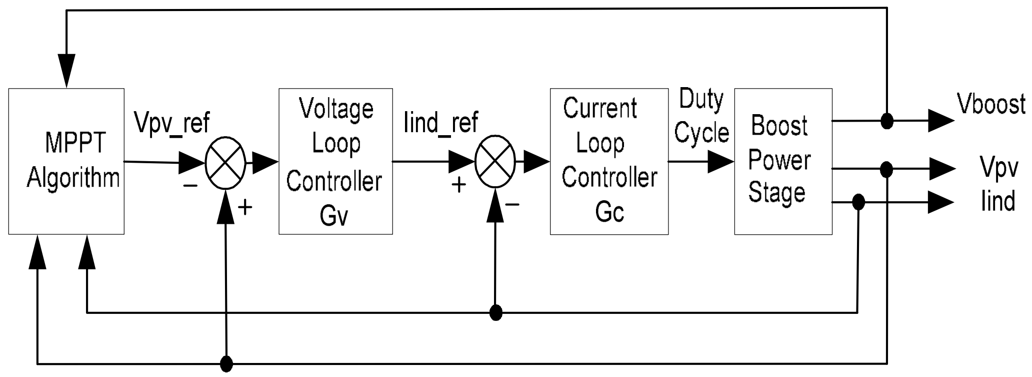


Figure 1.2 MPPT DC-DC Converter Control Loops

The panel voltage Vpv, sensed through one of the ADC channels, is compared against the reference voltage Vpv_ref set by the MPPT algorithm. The resulting error signal Ev is then input the voltage loop controller Gv which regulates the panel voltage at the reference level. The voltage controller Gv has the form of a two pole two zero (2P2Z) compensator. The output of Gv is the reference current command for the inner inductor current loop. The average value of the inductor current is the panel current Ipv. Therefore, by controlling the average value of the inductor current the current controller Gc essentially controls the panel current.

This reference current command Iind_ref for the current control loop is compared against the feedback inductor current Iind sensed through another ADC channel. The resulting current error signal is then input the current loop controller Gc which generates the boost converter PWM duty ratio command d for the boost switches Q1 & Q2.

In addition to implementing the voltage and current loop controllers, C2000 MCU also monitors the boost output voltage for over voltage protection. The ADC channel that monitors the boost voltage has an internal analog comparator with user programmable threshold. This threshold for the comparator is set by use of an internal 10-bit DAC. Whenever the DC bus voltage reaches an upper limit corresponding to the user programmable comparator threshold, the comparator initiates a pulse by pulse duty limit for the boost PWM signals. This limits the boost inductor current and hence the boost bus voltage to its desired upper limit.

C2000 MCU also generates two PWM outputs to drive the isolated LLC stage. This stage is run in an open loop fashion with unity voltage conversion ratio (voltage gain). This means the boost voltage and the LLC output voltage Vbus is almost equal. However, this requires a small minimum load of about 10W across Vbus (16kohm at 400V). With no load connected across Vbus and the boost output voltage set to 400V, the LLC stage gain might be higher than 1, resulting in high voltage across LLC output (Vbus). ***The user must prevent this condition by always maintaining a minimum load resistor of about 16kOhm across the LLC output (Vbus).***

All the time critical functions related to the DC-DC control loops 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. DC-DC Electrical Specifications

Following lists the key highlights of the C2000 MPPT DC-DC EVM.

- Panel Voltage: 200V (Min) to 300V (Max)
- 400Vdc Output
- 500 Watts Output Power
- Full Load efficiency greater than 94%

2 Software overview

2.1 Software Control Flow

The CCS project for C2000 MPPT DC-DC mostly makes use of the "C-background/ASM-ISR" framework. The main fast ISR (50kHz) 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. Also, a third ISR runs from C environment at a much slower frequency to implement the LIN(local interconnect network) based communication with the DC-AC inverter stage. The frequency of the LIN interrupt is set by the inverter at 50Hz.

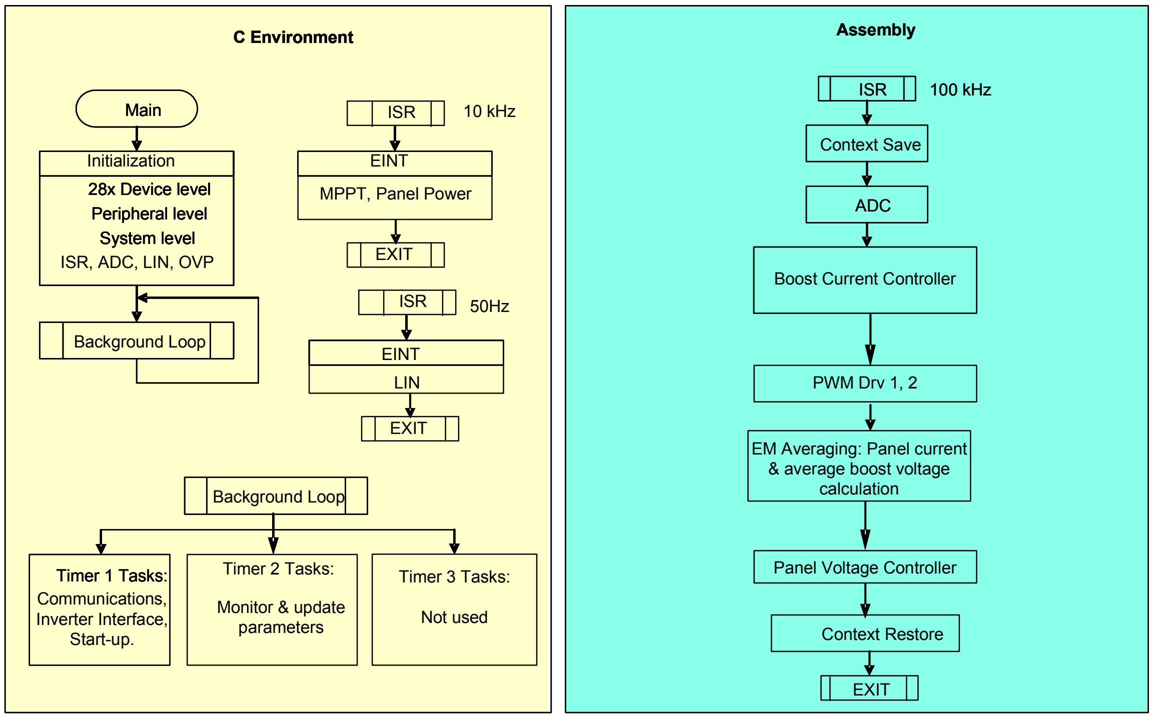


Figure 2.1.1. MPPT DC-DC Software flow diagram

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, control calculations, and PWM updates. The slower ISRs in the C environment implement the MPPT algorithm to calculate the reference PV panel voltage and establish LIN communication with the inverter stage. Fig 2.1.1 depicts the general software flow for this project.

The key framework C files used in this project are:

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

HV_Solar_DC-DC -DevInit_F2803x.c – The control card (2803x) used in the MPPT DC-DC 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:

HV_Solar_DC-DC-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 (50kHz) as the PWM time-base(100kHz) used to trigger it.

The slow ISR that runs at 10kHz consists of two files. The user selects one of the two files to implement the MPPT algorithm.

Mppt_incc.h – this file contains code for calculating the panel voltage for maximum power point tracking using the incremental conductance method. This file has an initialization section (one time execute) and a run-time section which executes at 10kHz rate.

Mppt_pno.h – this file contains code for calculating the panel voltage for maximum power point tracking using the perturb and observe method. This file has an initialization section (one time execute) and a run-time section which executes at 10kHz rate.

The second slow ISR that runs at 50Hz consists of one file.

SolarHv_DCDC_Lin.C – this file contains code for establishing LIN communication with the inverter stage.

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

These power library modules may have both a C and an assembly component. In this project, five 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
PWM_CompPairDB_Cnf.C		
	MATH_EMAVG_INIT n	MATH_EMAVG 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)

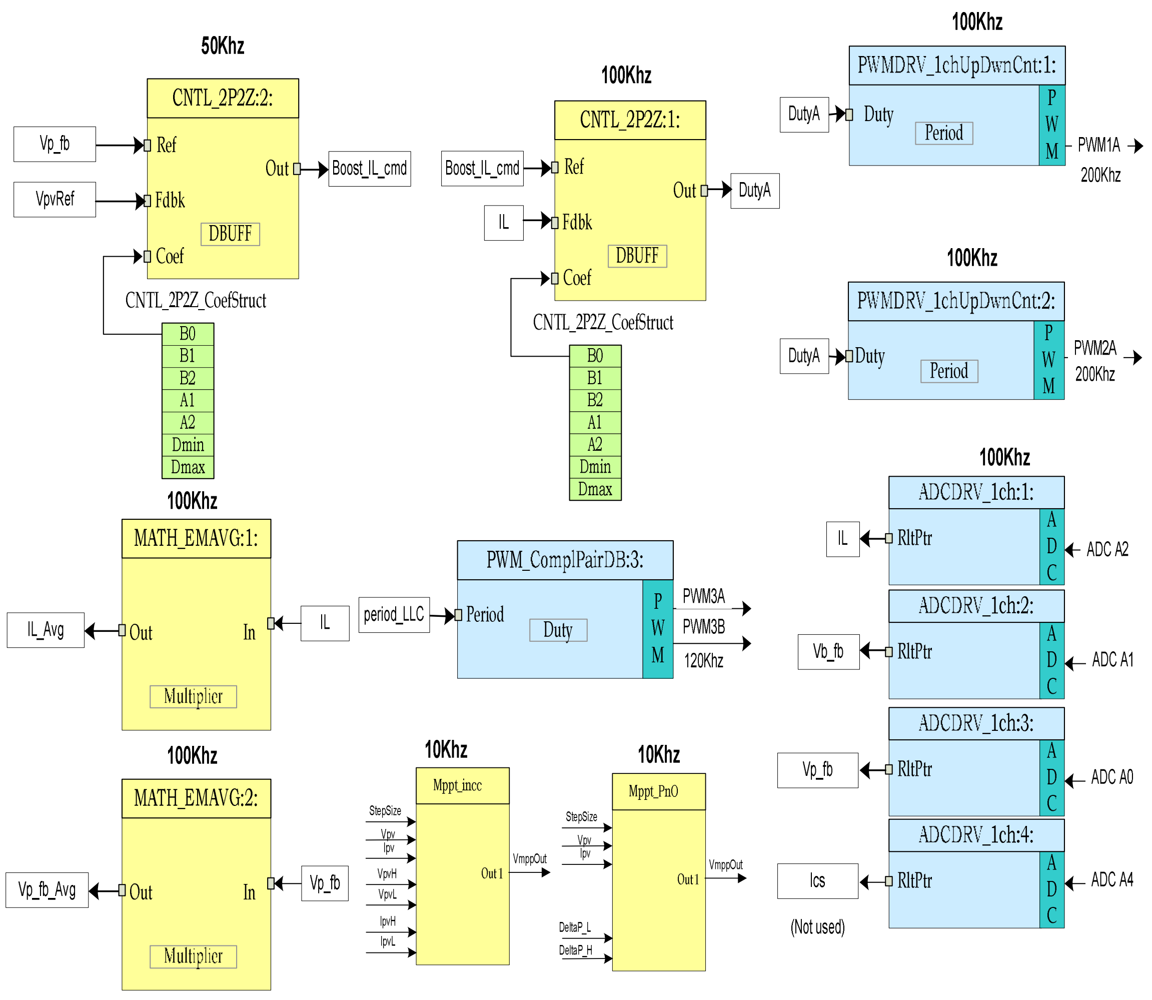


Figure 2.1.2 Software blocks

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.

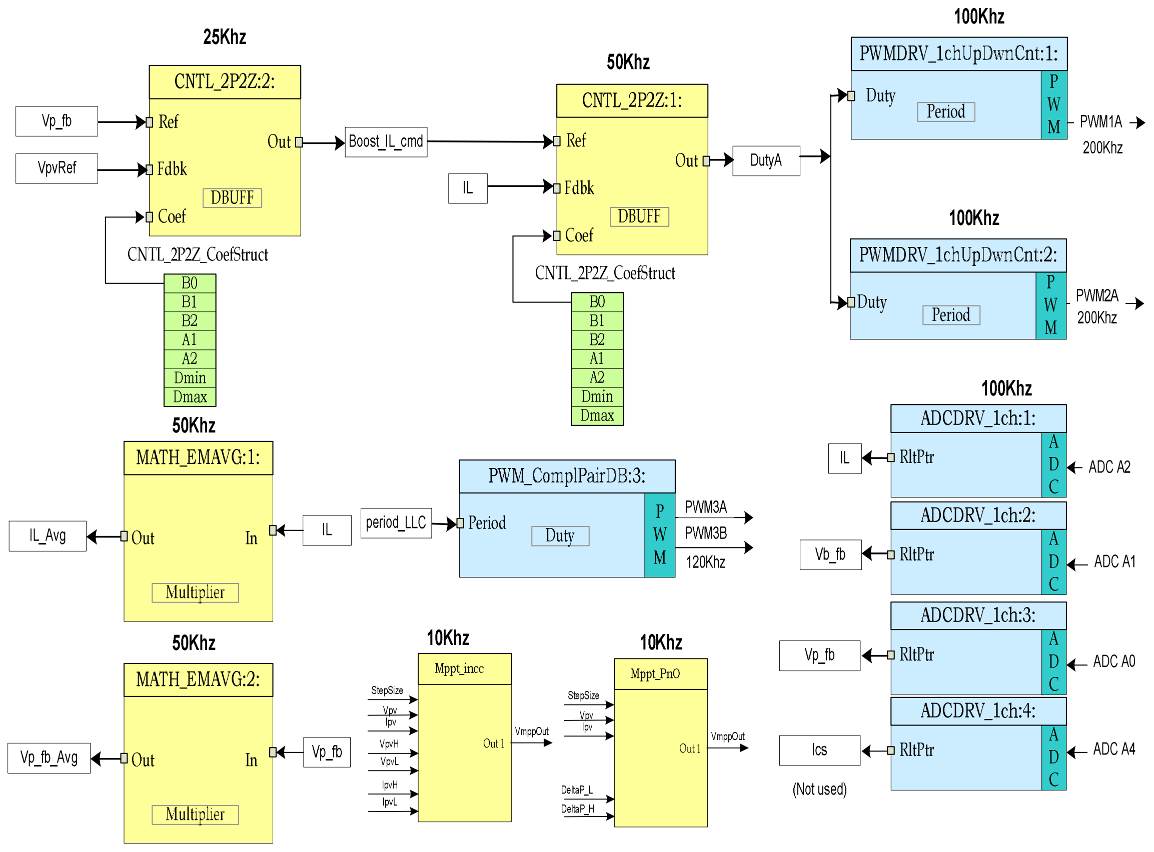


Figure 2.1.3. Software Control Flow

As mentioned in section 1.1 the MPPT DC-DC system is controlled by two feedback loops. The outer voltage loop maintains the panel voltage at the level calculated by the MPPT algorithm, while a faster inner current loop control the average boost inductor current. 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 50kHz (half of the PWM switching frequency) while the voltage controller is executed at 25kHz 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 MPPT DC-DC 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 INCR_BUILD, found in the *HV_Solar_DC-DC-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	
INCR_BUILD = 1	Open loop check for boost and LLC action and ADC feedback (Check sensing circuitry)
INCR_BUILD = 2	Open voltage loop and closed current loop control of boost
INCR_BUILD = 3	Closed voltage and current loop control of boost with MPPT

Table 2.2.1 Incremental build options for MPPT DC-DC

3 Procedure for running the incremental builds

All software files related to this C2x controlled MPPT DC-DC 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\MPPT DC-DC_v1.0\MPPT DC-DC. 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 a PV panel emulator with appropriate power rating should be used to power the unit. Before power is applied to the board a voltmeter and an appropriate resistive or electronic load should be attached to the output. This will discharge the bus capacitor quickly when the PV power is turned off. 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.

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

3.1 Build 1: Open loop boost with ADC measurements

➢ Objective

The objectives of this build are, (1) evaluate MPPT DC-DC 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 CCS 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 DC-DC 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 BOOST switches are obtained from ePWM module 1 & 2. ePWM1A drives one of the BOOST switches while ePWM2A drives the other. The two PWM signals for the two LLC stage switches are obtained from ePWM module 3. ePWM3A drives the upper(high side) LLC switch while ePWM3B drives the low side switch.

The quantities that are sensed and fed back to the MCU include, (1) the panel voltage (Vp_fb), (2) the combined BOOST inductor current (I_L), and (3) the boost DC bus voltage (V_{b_fb}). These quantities are read using the ADC driver module and are indicated in Fig 3.1.1. The fourth channel for the ADC driver macro is not used in this application. The ADC driver module converts the 12-bit ADC result to a 32bit Q24 value.

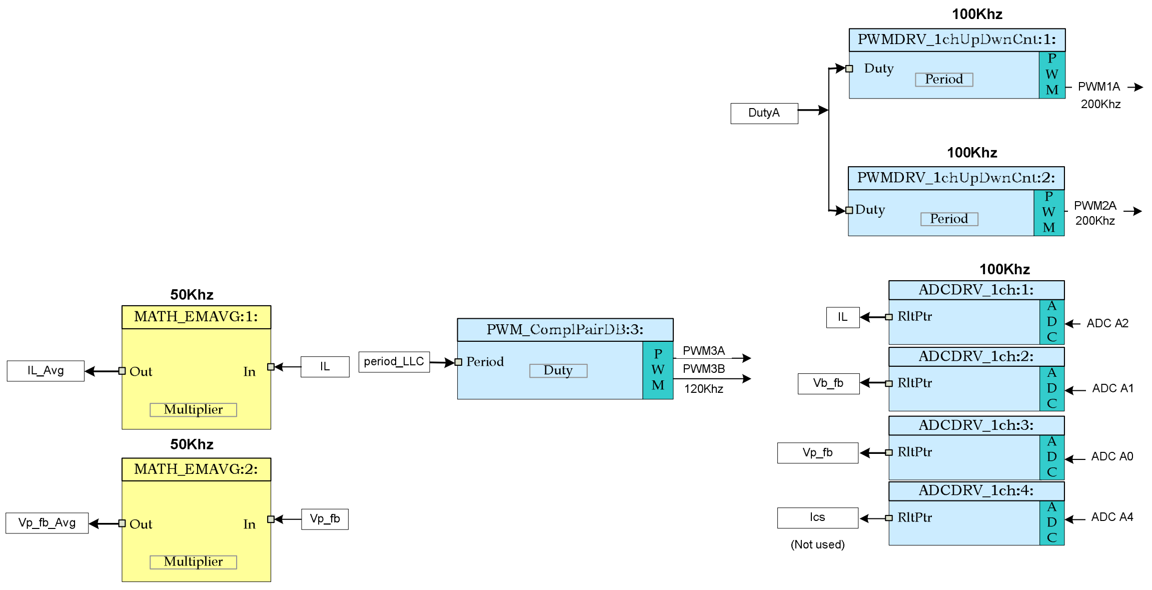


Figure 3.1.1. Build 1 software blocks

The boost PWM signals are generated at a frequency of 100 kHz i.e. a period of 10 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 10us is equivalent to 600 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 300 (period register value) will give a total PWM period value of 600 counts (i.e. 10 us).

The LLC PWM signals are generated at a frequency of 120 kHz i.e. a period of 8.33 us. With the controller operating at 60MHz, one count of the time base counter of ePWM3 corresponds to 16.6667ns. This implies a PWM period of 8.33us is equivalent to 500 counts of the time base counter (TBCNT3). The ePWM3 module is configured to operate in up-down count mode. This means a time base period value of 250 (period register value) will give a total PWM period value of 500 counts (i.e. 8.33 us).

BOOST inductor current is sampled at the midpoint of the PWM1A ON pulse since the sampled value represents the average inductor current under CCM (continuous conduction mode) condition. Under DCM condition this sampled current value represents a fraction of the average inductor current.

The other two voltage signal conversions are 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.

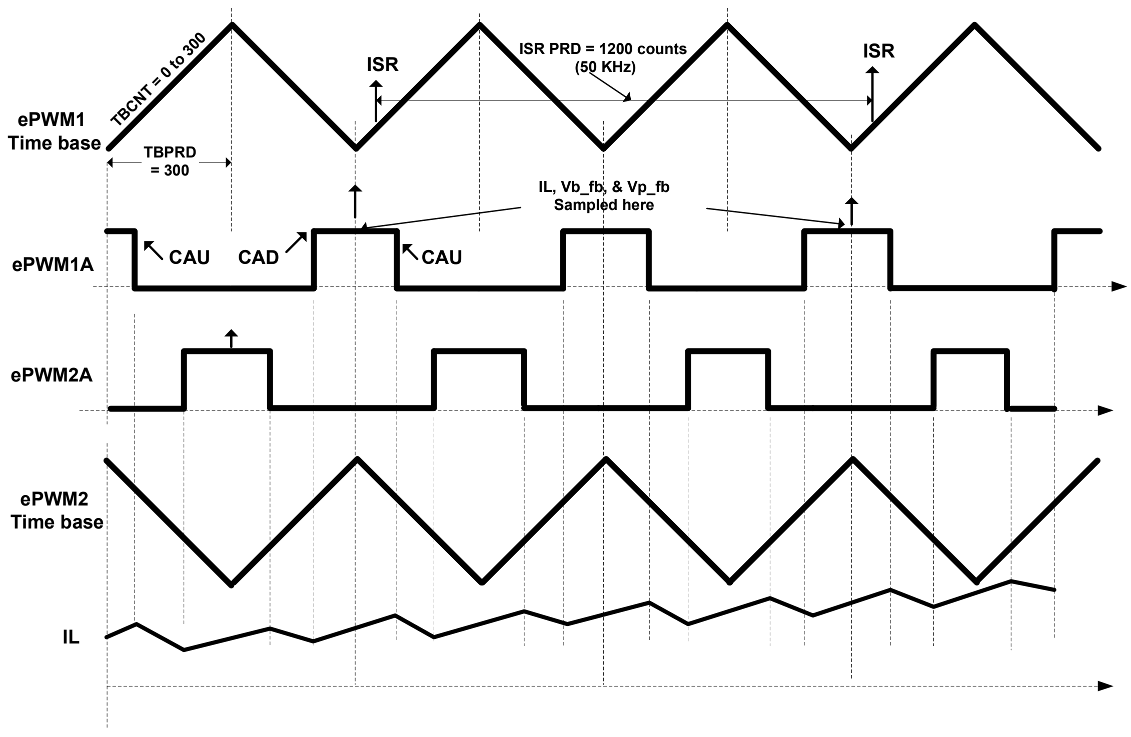


Figure 3.1.2. PWM generation and ADC sampling

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 also configured in a similar manner with 180 phase shift with respect to ePWM1A.

The CMPA value is derived from the input "BOOSTDuty" (Q24 variable) command.

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 3 ADC results are read in the ISR by executing the ADC driver module from the 50kHz 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 MPPT DC-DC EVM. This OVP applies only for the boost output and not for the LLC stage output. Since the Piccolo controller is on the primary side of the isolation it has no knowledge of the isolated LLC output. ***Therefore, the user must connect a minimum load of about 10W across the LLC output in order to make the open loop LLC output follow its input voltage, i.e., the output voltage from the boost stage.*** The minimum load across the LLC output helps maintain the LLC stage voltage conversion factor of 1. This way the max LLC output will also be limited by the max boost output which in turn is protected by the OVP mechanism.

The sensed boost stage 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 404V. This threshold parameter is programmed inside the file ***HV_SOLAR_DC_DC-Main.h***. This is done by the following initialization of the on chip DAC reference voltage.

Comp3Regs.DACVAL.bit.DACVAL = 808;

Since the 10 bit DAC full scale value represents a max voltage of 511V, the init value of 808 represents the OV threshold of 404V.

In case of an OV condition the PWM outputs are duty limited pulse by pulse 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.

➢ **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\MPPT DC-DC_v1.0\MPPT DC-DC** directory. Make sure that under the Projects tab MPPT DC-DC 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:

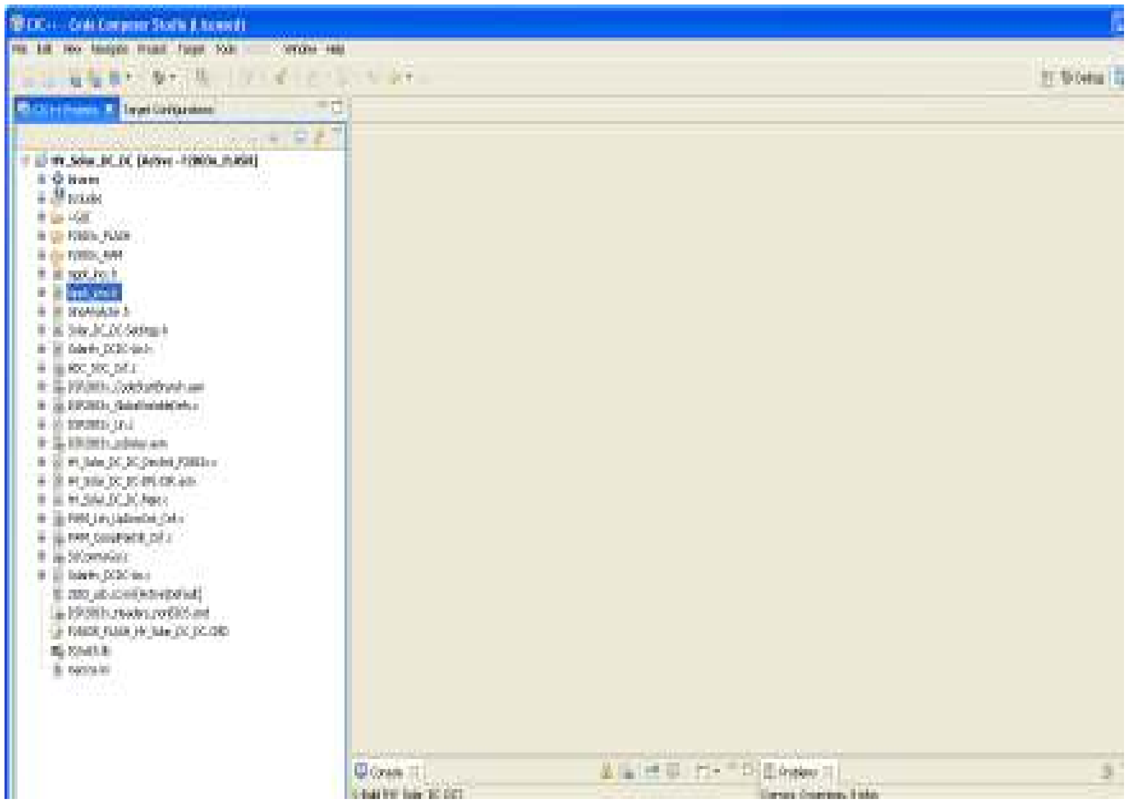


Figure 3.1.3. CCS Project Window

Device Initialization, Main, and ISR Files

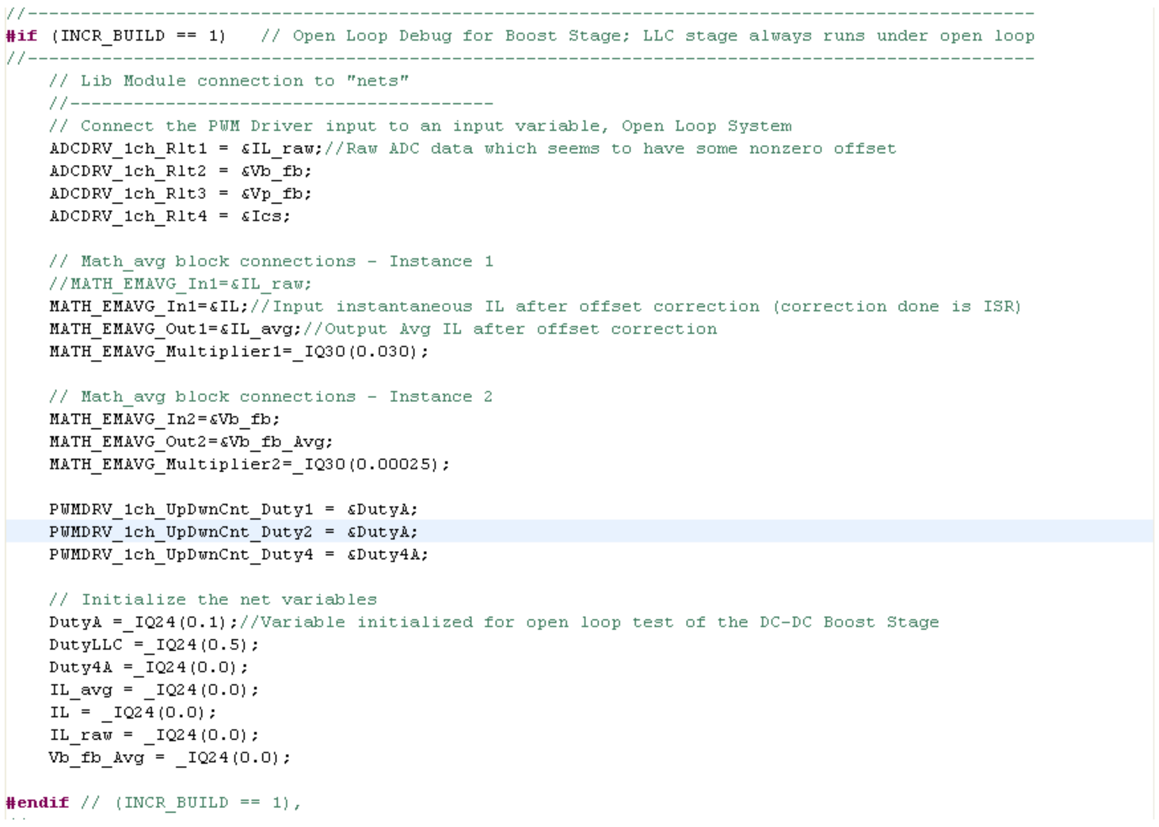
Note: ***DO NOT*** make any changes to the source files − ***`ONLY INSPECT`***

4. Open and inspect *SOLAR_DC_DC-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.

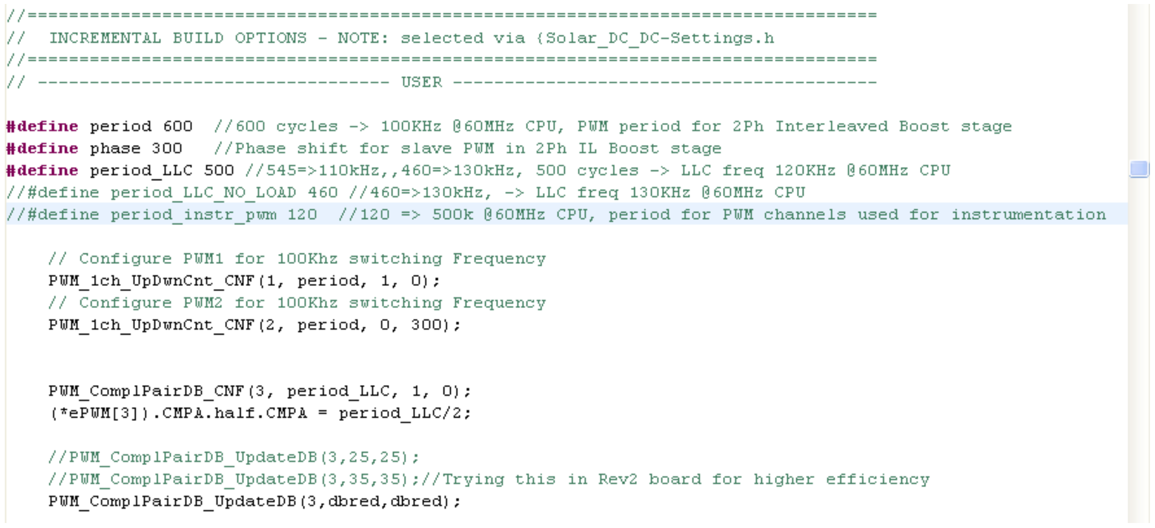
5. Open and inspect *SOLAR_DC_DC -Main.c*. Notice the call made to *DeviceInit()* function and other variable initialization. Also notice code for different incremental build options, the ISR intialization and the background for(;;) loop.

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.



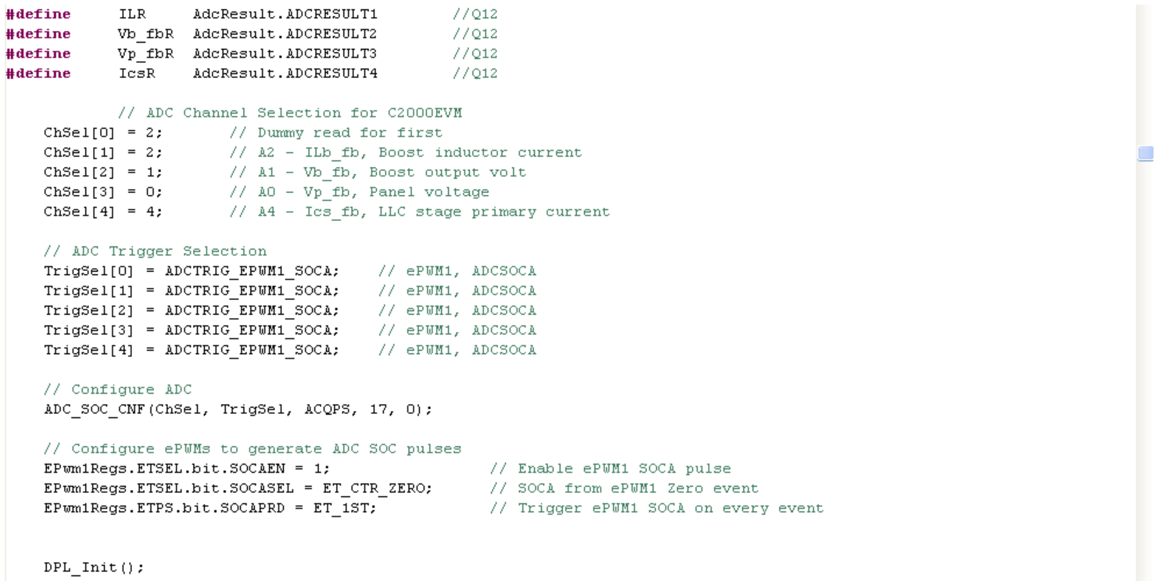
```
//---------------------------------------------------------------------------------------------
#if (INCR_BUILD == 1)   // Open Loop Debug for Boost Stage; LLC stage always runs under open loop
//---------------------------------------------------------------------------------------------
    // Lib Module connection to "nets"
    //----------------------------------------
    // Connect the PWM Driver input to an input variable, Open Loop System
    ADCDRV_1ch_Rlt1 = &IL_raw;//Raw ADC data which seems to have some nonzero offset
    ADCDRV_1ch_Rlt2 = &Vb_fb;
    ADCDRV_1ch_Rlt3 = &Vp_fb;
    ADCDRV_1ch_Rlt4 = &Ics;

    // Math_avg block connections - Instance 1
    //MATH_EMAVG_In1=&IL_raw;
    MATH_EMAVG_In1=&IL;//Input instantaneous IL after offset correction (correction done is ISR)
    MATH_EMAVG_Out1=&IL_avg;//Output Avg IL after offset correction
    MATH_EMAVG_Multiplier1=_IQ30(0.030);

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

    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 of the DC-DC Boost Stage
    DutyLLC =_IQ24(0.5);
    Duty4A =_IQ24(0.0);
    IL_avg = _IQ24(0.0);
    IL = _IQ24(0.0);
    IL_raw = _IQ24(0.0);
    Vb_fb_Avg = _IQ24(0.0);

#endif // (INCR_BUILD == 1),
```

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 600 and internally calculates the period register value of 300.



```
//==============================================================================
//  INCREMENTAL BUILD OPTIONS - NOTE: selected via {Solar_DC_DC-Settings.h
//==============================================================================
// ---------------------------------- USER ----------------------------------------

#define period 600  //600 cycles -> 100KHz @60MHz CPU, PWM period for 2Ph Interleaved Boost stage
#define phase 300   //Phase shift for slave PWM in 2Ph IL Boost stage
#define period_LLC 500 //545=>110kHz,,460=>130kHz, 500 cycles -> LLC freq 120KHz @60MHz CPU
//#define period_LLC_NO_LOAD 460 //460=>130kHz, -> LLC freq 130KHz @60MHz CPU
//#define period_instr_pwm 120  //120 => 500k @60MHz CPU, period for PWM channels used for instrumentation

    // Configure PWM1 for 100Khz switching Frequency
    PWM_1ch_UpDwnCnt_CNF(1, period, 1, 0);
    // Configure PWM2 for 100Khz switching Frequency
    PWM_1ch_UpDwnCnt_CNF(2, period, 0, 300);


    PWM_ComplPairDB_CNF(3, period_LLC, 1, 0);
    (*ePWM[3]).CMPA.half.CMPA = period_LLC/2;

    //PWM_ComplPairDB_UpdateDB(3,25,25);
    //PWM_ComplPairDB_UpdateDB(3,35,35);//Trying this in Rev2 board for higher efficiency
    PWM_ComplPairDB_UpdateDB(3,dbred,dbred);
```

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     ILR     AdcResult.ADCRESULT1        //Q12
#define     Vb_fbR  AdcResult.ADCRESULT2        //Q12
#define     Vp_fbR  AdcResult.ADCRESULT3        //Q12
#define     IcsR    AdcResult.ADCRESULT4        //Q12

            // ADC Channel Selection for C2000EVM
    ChSel[0] = 2;       // Dummy read for first
    ChSel[1] = 2;       // A2 - ILb_fb, Boost inductor current
    ChSel[2] = 1;       // A1 - Vb_fb, Boost output volt
    ChSel[3] = 0;       // A0 - Vp_fb, Panel voltage
    ChSel[4] = 4;       // A4 - Ics_fb, LLC stage primary current

    // 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


    DPL_Init();
```

8. Open and inspect *SOLAR_DC_DC-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 ***SOLAR_DC_DC-Settings.h*** file.

   Note: Whenever you change the incremental build option in ***SOLAR_DC_DC-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

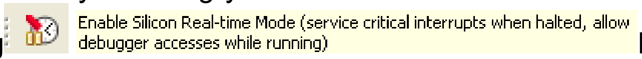
It is standard debug practice to watch local and global variables while debugging code. There are various methods for doing this in Code Composer Studio, such as memory views and watch views. If a watch view did not open when the debug environment was launched, open a new *watch view* and add various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

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.

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 boost converter.

19. Apply a resistive load to the DC-DC EVM output terminal (10~100W).

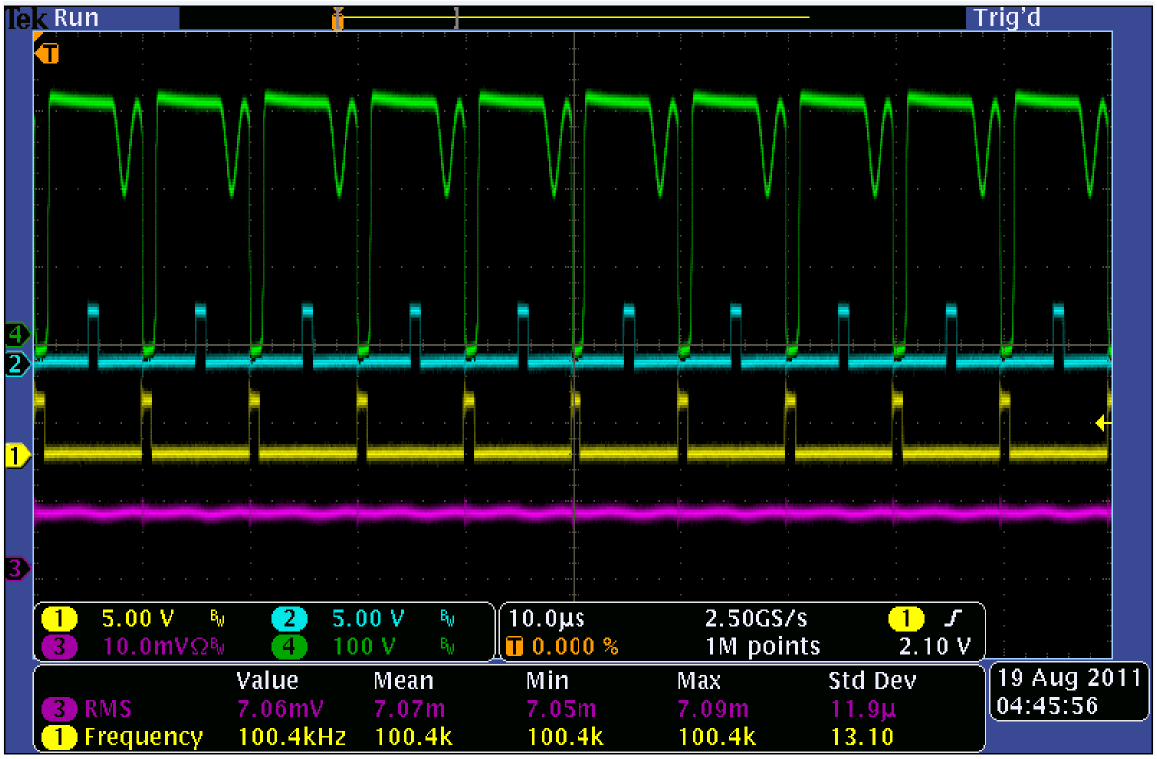
20. Use a high voltage isolated DC supply to power the DC-DC EVM. Measure and verify the boost DC bus voltage corresponding to applied input voltage and the duty ratio.

21. Use DutyA to slowly change the duty from the watch window. The boost converter output voltage should change accordingly and this, in turn, will change the EVM output.

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

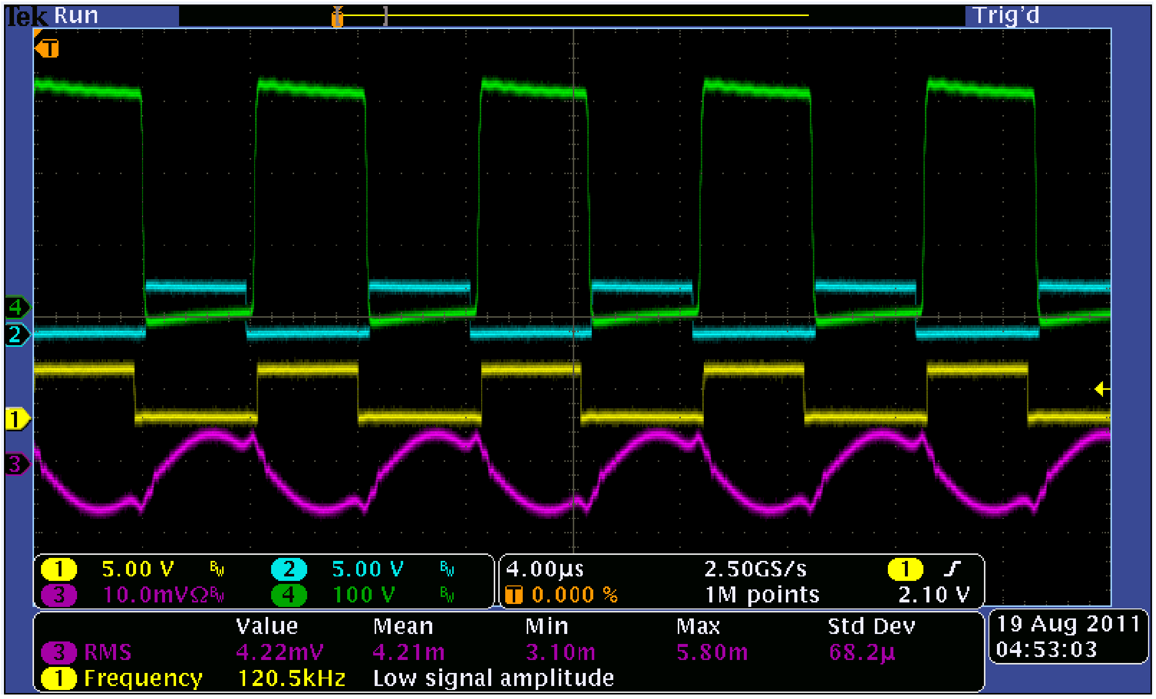
22. Add the other variables such as, Vb_fb, Vp_fb and verify the different ADC results in the watch view.

23. The following oscilloscope captures show two PWM outputs (Ch1 & Ch2), DC source input current (Ch3) and Boost MOSFET drain to source voltage (Ch4) when the output DC bus load is 1K ohm, input DC voltage is 250V and the set duty ratio is about 10%. The PWM frequency is measured as 100kHz.



	Value	Mean	Min	Max	Std Dev
3 RMS	7.06mV	7.07m	7.05m	7.09m	11.9µ
1 Frequency	100.4kHz	100.4k	100.4k	100.4k	13.10

24. The following oscilloscope captures show two PWM outputs (Ch1 & Ch2) for the LLC stage. In this figure Ch4 represents LLC stage primary switch node voltage and Ch3 is the LLC primary current when the boost input is 250V, boost output (LLC input) is 300V, LLC output is 307V, LLC output load is 1K ohm and the boost duty ratio is set to about 10%.



	Value	Mean	Min	Max	Std Dev
3 RMS	4.22mV	4.21m	3.10m	5.80m	68.2µ
1 Frequency	120.5kHz	Low signal amplitude			

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 DC 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: MPPT DC-DC with closed current loop

➢ **Objective**

The objective of this build is to verify the operation of the MPPT DC-DC under closed current loop and open voltage loop mode. Since the voltage loop is open, there is no MPPT operation under this build.

➢ **Overview**

Fig 3.2.1 shows the software blocks used in this build. Notice that 1 additional software block compared to the Build 1 diagram (Figure 3.1.1). This block is shown in Figure 3.2.1 as *CNTL_2P2Z:1*. It represents a two pole two zero (2p2z) controller and is used for the current control loop. 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 50 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.

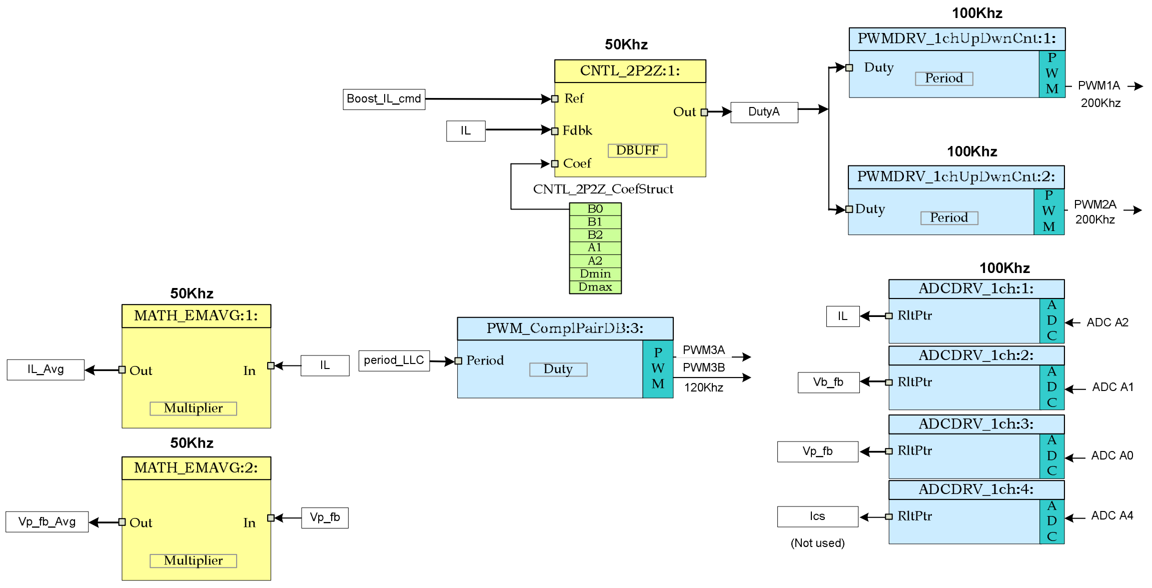


Figure 3.2.1. Build 2 software blocks

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 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.

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*, normally comes from the voltage loop controller output. But, in this build there is no voltage loop controller and so the variable Boost_IL_cmd is used to change the reference current under user control. 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) blocks shown in Figure 3.2.1 calculate the average of the boost inductor current and the panel voltage. The average inductor current represents the panel current and is used to implement the MPPT algorithm.

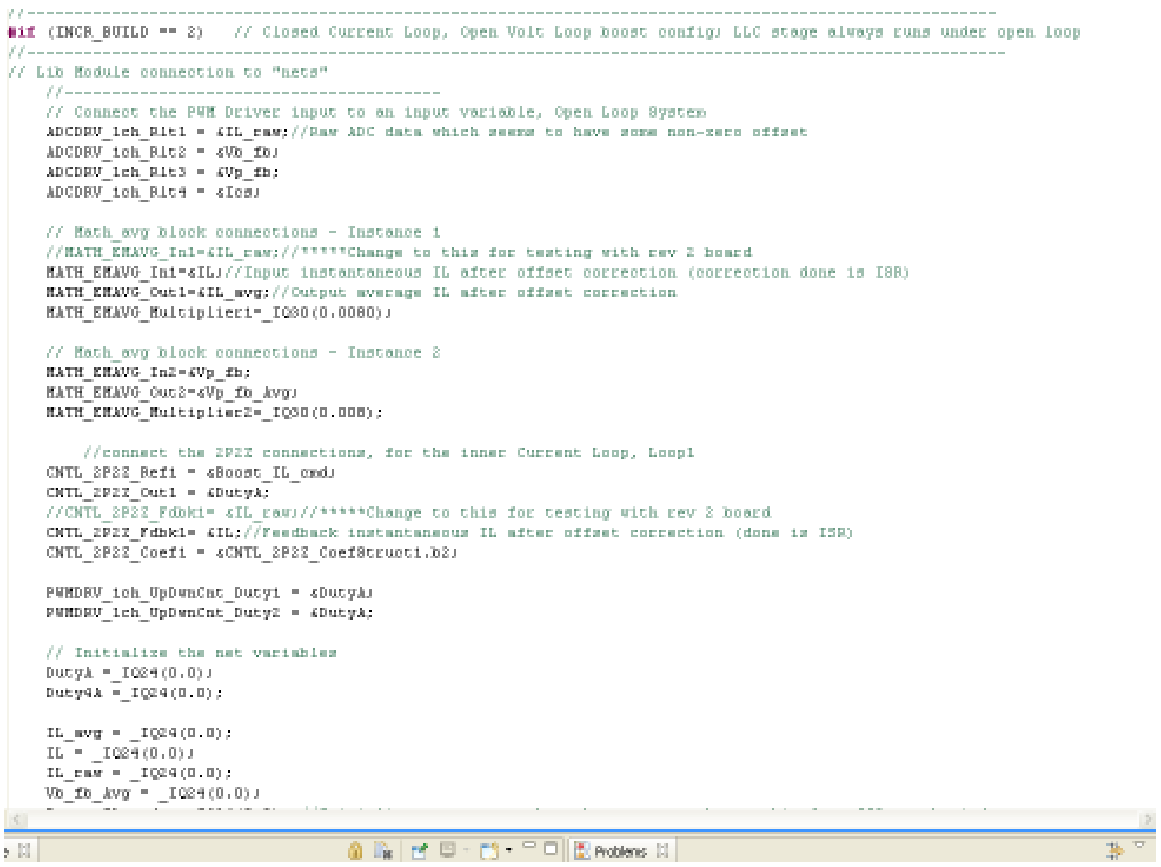
➢ **Procedure**

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, Open Volt Loop boost config; LLC stage always runs under open loop
//----------------------------------------------------------------------------------------------------
// Lib Module connection to "nets"
    //----------------------------------------
    // Connect the PWM Driver input to an input variable, Open Loop System
    ADCDRV_1ch_Rlt1 = &IL_raw;//Raw ADC data which seems to have some non-zero offset
    ADCDRV_1ch_Rlt2 = &Vb_fb;
    ADCDRV_1ch_Rlt3 = &Vp_fb;
    ADCDRV_1ch_Rlt4 = &Ios;

    // Math_avg block connections - Instance 1
    //MATH_EMAVG_In1=&IL_raw;//*****Change to this for testing with rev 2 board
    MATH_EMAVG_In1=&IL;//Input instantaneous IL after offset correction (correction done is ISR)
    MATH_EMAVG_Out1=&IL_avg;//Output average IL after offset correction
    MATH_EMAVG_Multiplier1=_IQ30(0.0080);

    // Math_avg block connections - Instance 2
    MATH_EMAVG_In2=&Vp_fb;
    MATH_EMAVG_Out2=&Vp_fb_Avg;
    MATH_EMAVG_Multiplier2=_IQ30(0.008);

        //connect the 2P2Z connections, for the inner Current Loop, Loop1
    CNTL_2P2Z_Ref1 = &Boost_IL_cmd;
    CNTL_2P2Z_Out1 = &DutyA;
    //CNTL_2P2Z_Fdbk1= &IL_raw;//*****Change to this for testing with rev 2 board
    CNTL_2P2Z_Fdbk1= &IL;//Feedback instantaneous IL after offset correction (done is ISR)
    CNTL_2P2Z_Coef1 = &CNTL_2P2Z_CoefStruct1.b2;

    PWMDRV_1ch_UpDwnCnt_Duty1 = &DutyA;
    PWMDRV_1ch_UpDwnCnt_Duty2 = &DutyA;

    // Initialize the net variables
    DutyA =_IQ24(0.0);
    Duty4A =_IQ24(0.0);

    IL_avg = _IQ24(0.0);
    IL = _IQ24(0.0);
    IL_raw = _IQ24(0.0);
    Vb_fb_Avg = _IQ24(0.0);
```

1) Open and inspect ***SOLAR_DC_DC-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 ***SOLAR_DC_DC-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().

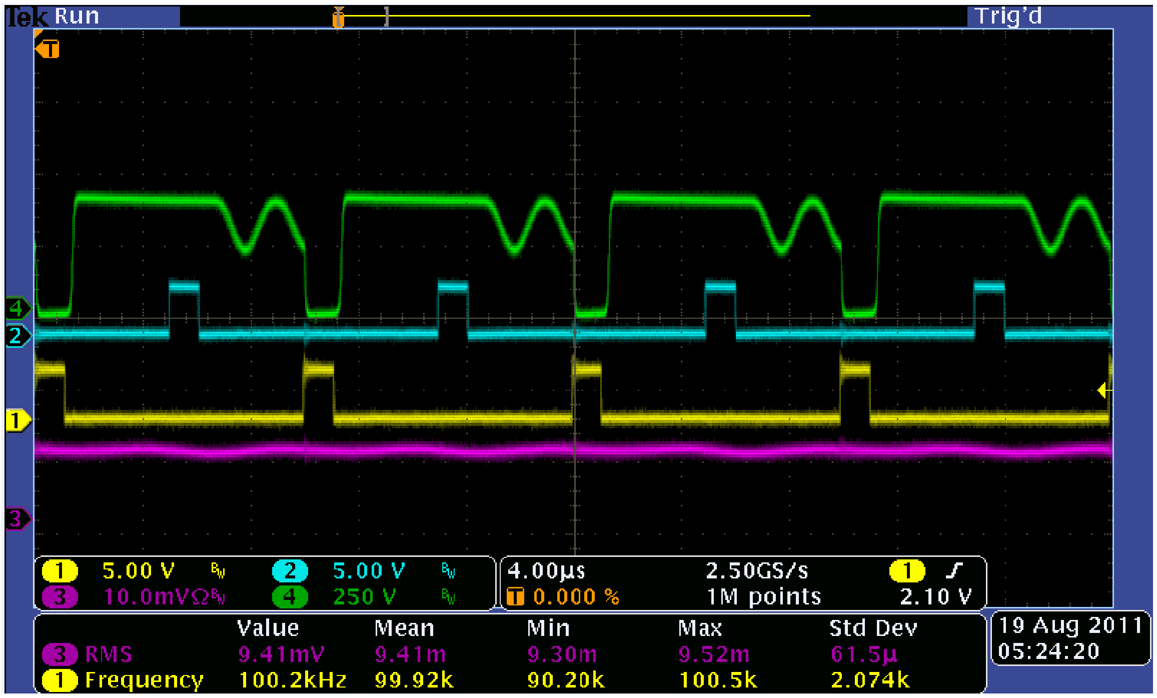
   Note: Whenever you change the incremental build option in ***SOLAR_DC_DC-Settings.h*** always do a "Rebuild All"

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 *Boost_IL_cmd* 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) Connect an appropriate resistive load across the DC-DC output. For example, an 1.0Kohm resistor of 400W rating can be used. This will provide a load of 160W at 400V bus voltage.

6) Slowly apply DC power to the board from an isolated DC source. Monitor the DC-DC EVM output voltage as the input voltage is raised slowly to 300V. Slowly adjust (in steps of 0.01) the value for *Boost_IL_cmd* to set the output voltage to about 385V. Use an oscilloscope with voltage and current probes to observe the input voltage, input current, boost MOSFET voltage, LLC primary current and the PWM outputs. With a 300V boost input and 1.0kohm resistive load when the boost output voltage is set to 373V you should see the LLC output voltage (EVM output) of 385V. The following scope plot is captured under this condition. Here Ch1 and Ch2 show the boost PWM outputs. Ch3 is the boost input current and Ch4 is the voltage across the boost MOSFET.



	Value	Mean	Min	Max	Std Dev
3 RMS	9.41mV	9.41m	9.30m	9.52m	61.5µ
1 Frequency	100.2kHz	99.92k	90.20k	100.5k	2.074k

7) Increase *Boost_IL_cmd* slightly (in steps of 0.01) and observe the bus voltage settles to a higher value. Increasing *Boost_IL_cmd* increases the magnitude of the current reference signal and the bus voltage will rise. Therefore, apply caution and set the overvoltage protection threshold to a value less than 400V.

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: MPPT DC-DC with closed voltage and current loop

➢ **Objective**

The objective of this build is to verify the operation of the complete MPPT DC-DC 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 MPPT DC-DC voltage loop control. This voltage loop controller is executed at 25kHz rate which is half the rate for current loop. The output from this control block drives the input node *Boost_IL_cmd* of the current controller block.

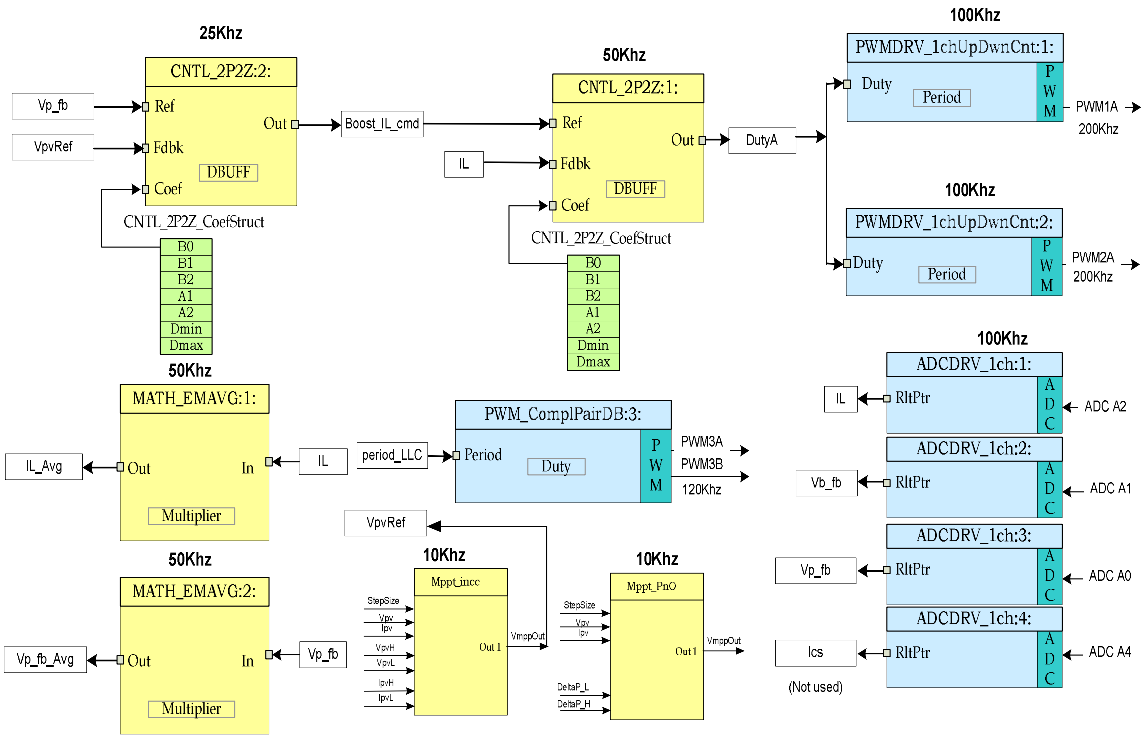


Figure 3.3.1. Build 3 software blocks

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.

Figure 3.3.1 also shows two additional blocks implementing two different MPPT algorithms that are used in this EVM. The default code setting uses incremental conductance algorithm for the MPPT and so the output from this MPPT block is connected to the feedback terminal (Fdbk) of the voltage loop controller. These MPPT blocks are run from a 10kHz ISR.

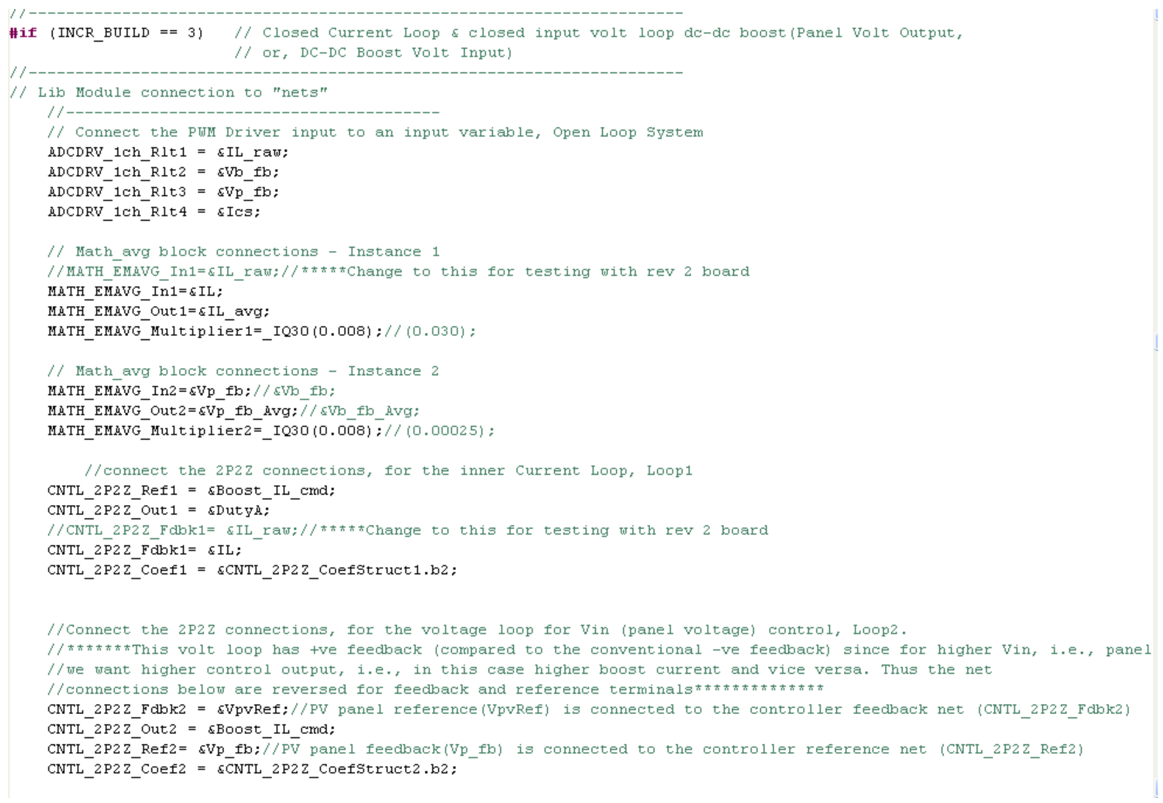
➢ **Procedure**

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 input volt loop dc-dc boost(Panel Volt Output,
                        // or, DC-DC Boost Volt Input)
//----------------------------------------------------------------------
// Lib Module connection to "nets"
    //---------------------------------------
    // Connect the PWM Driver input to an input variable, Open Loop System
    ADCDRV_1ch_Rlt1 = &IL_raw;
    ADCDRV_1ch_Rlt2 = &Vb_fb;
    ADCDRV_1ch_Rlt3 = &Vp_fb;
    ADCDRV_1ch_Rlt4 = &Ics;

    // Math_avg block connections - Instance 1
    //MATH_EMAVG_In1=&IL_raw;//*****Change to this for testing with rev 2 board
    MATH_EMAVG_In1=&IL;
    MATH_EMAVG_Out1=&IL_avg;
    MATH_EMAVG_Multiplier1=_IQ30(0.008);//(0.030);

    // Math_avg block connections - Instance 2
    MATH_EMAVG_In2=&Vp_fb;//&Vb_fb;
    MATH_EMAVG_Out2=&Vp_fb_Avg;//&Vb_fb_Avg;
    MATH_EMAVG_Multiplier2=_IQ30(0.008);//(0.00025);

        //connect the 2P2Z connections, for the inner Current Loop, Loop1
    CNTL_2P2Z_Ref1 = &Boost_IL_cmd;
    CNTL_2P2Z_Out1 = &DutyA;
    //CNTL_2P2Z_Fdbk1= &IL_raw;//*****Change to this for testing with rev 2 board
    CNTL_2P2Z_Fdbk1= &IL;
    CNTL_2P2Z_Coef1 = &CNTL_2P2Z_CoefStruct1.b2;


    //Connect the 2P2Z connections, for the voltage loop for Vin (panel voltage) control, Loop2.
    //*******This volt loop has +ve feedback (compared to the conventional -ve feedback) since for higher Vin, i.e., panel
    //we want higher control output, i.e., in this case higher boost current and vice versa. Thus the net
    //connections below are reversed for feedback and reference terminals*************
    CNTL_2P2Z_Fdbk2 = &VpvRef;//PV panel reference(VpvRef) is connected to the controller feedback net (CNTL_2P2Z_Fdbk2)
    CNTL_2P2Z_Out2 = &Boost_IL_cmd;
    CNTL_2P2Z_Ref2= &Vp_fb;//PV panel feedback(Vp_fb) is connected to the controller reference net (CNTL_2P2Z_Ref2)
    CNTL_2P2Z_Coef2 = &CNTL_2P2Z_CoefStruct2.b2;
```

1) Open and inspect ***SOLAR_DC_DC-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 ***SOLAR_DC_DC-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().

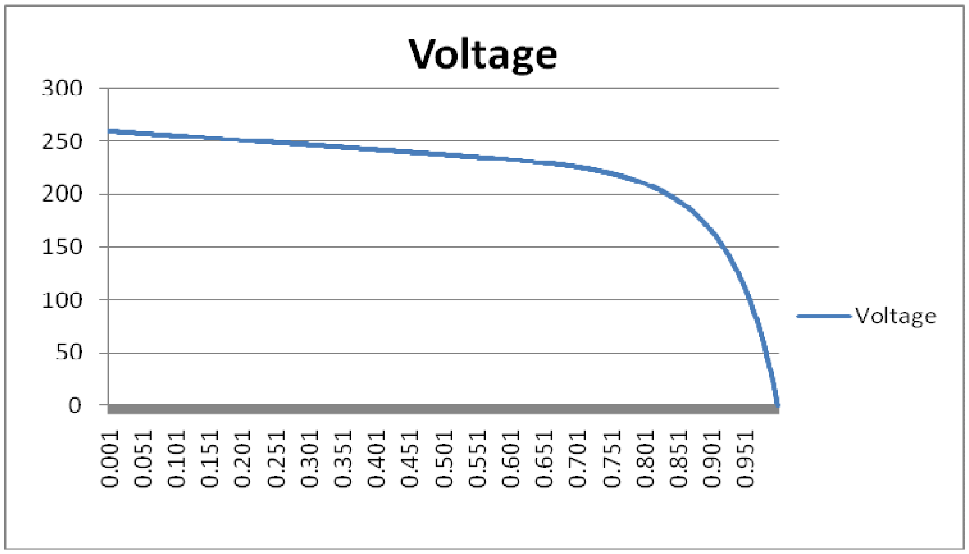
   Note: Whenever you change the incremental build option in ***SOLAR_DC_DC-Settings.h*** always do a “Rebuild All”

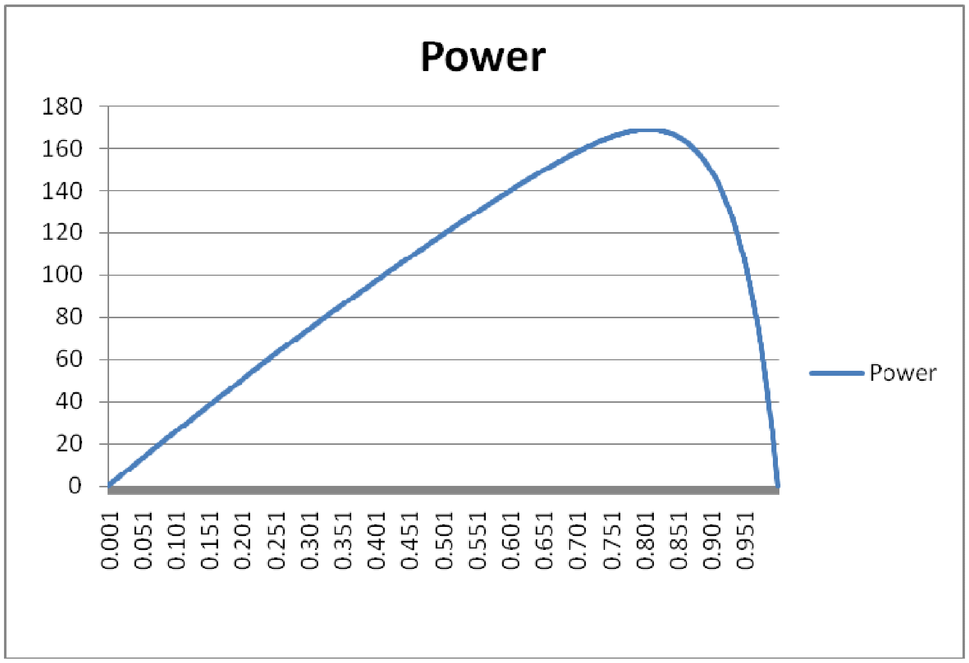
3) Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.

4) In the watch view, add four variables inverter_connected, *Start_DC_DC*, *Vp_fb,and Vb_fb*. Set the Q-format for the last two variables (Vp_fb & Vb_fb) to Q24. These two variables represent the boost input and output voltages respectively. These will slowly increase as the DC-DC starts up when PV panel emulator power is applied and the MPPT is turned on. To start MPPT from CCS (under this build) first, the user needs to modify the code as explained in step 6 below and reload the program to flash memory. Then, from CCS watch window, the user will set the variables *Start_DC_DC* to 1 and *inverter_connected* to 0.

5) Configure a solar panel emulator (200V to 300V, 500W max) to provide input power to the EVM. Configure the panel emulator to emulate the following solar panel characteristics, connect it to the EVM input but **do not turn on panel power at this time.**

Example Panel Emulator Parameters		
Voc	Open circuit panel voltage	**260V**
Vmpp	Panel voltage for max power point tracking (MPPT)	**220V**
Impp	Panel current for max power point tracking (MPPT)	**0.75A**
Isc	Short circuit panel current	**1A**

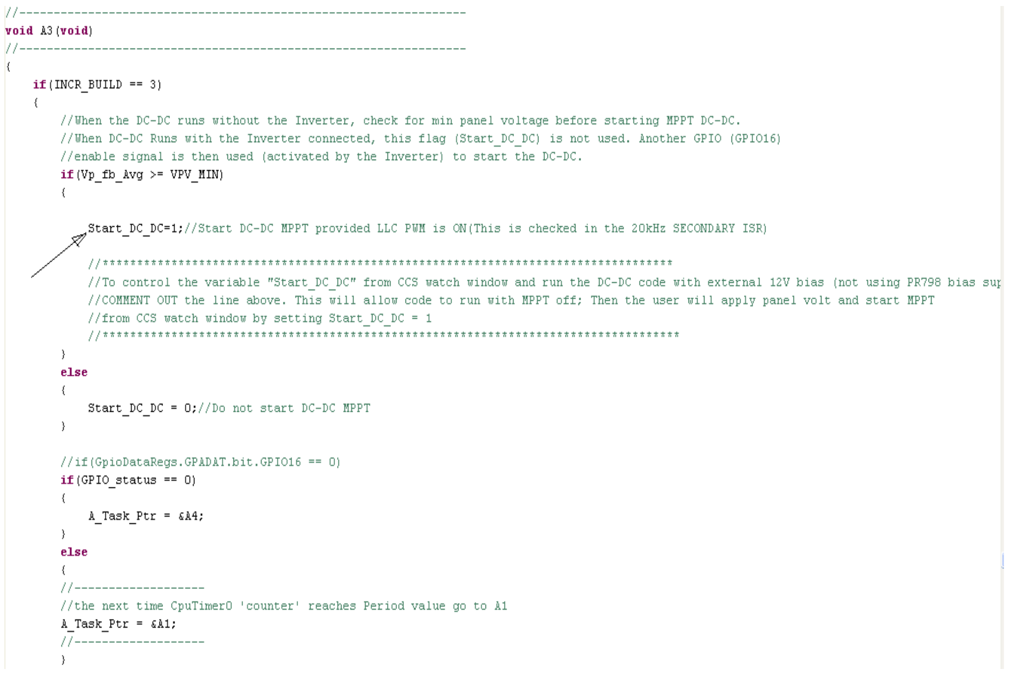




Connect an appropriate resistive load to the EVM output terminals (Vo-R & GND terminals). As in the example above, if the panel emulator is configured to supply 165W of power at MPPT point, then select a load resistor value of 970 ohm so that the EVM output voltage is limited to about 400V (P = 400*400/970 ≈ 165W). A smaller resistor will also work as long as the output voltage does not fall below 350V. This means that the smallest resistor that can be chosen for this load set up (165W) is about 742 ohm (R = 350*350/165 = 742 ohm). A resistor value larger than 970 ohm will cause output voltage higher than 400V for this load set up. **This output overvoltage condition must be prevented by choosing the maximum resistor value of 970 ohm for this load set up of 165W.** It is recommended that the resistor with a power rating > 200W is used for this load setting.

6) Starting the MPPT algorithm (in build 3) from CCS watch window, with the PV panel emulator power applied to the EVM input, will require changing one line of code as explained below. Without this change the code automatically starts the MPPT algorithm when (1) the minimum panel input voltage is applied and, (2) from the GUI the user set the variable inverter_connected to 0. When the EVM is shipped, the default code loaded in the flash memory allows this GUI based power up in standalone mode. Therefore, this code modification is needed **only if** the user wants to start the MPPT from the CCS watch window.

   Open the CCS project file HV_Solar_DC_DC-Main.c and locate the code where the variable *Start_DC_DC* is set to 1 as shown below.

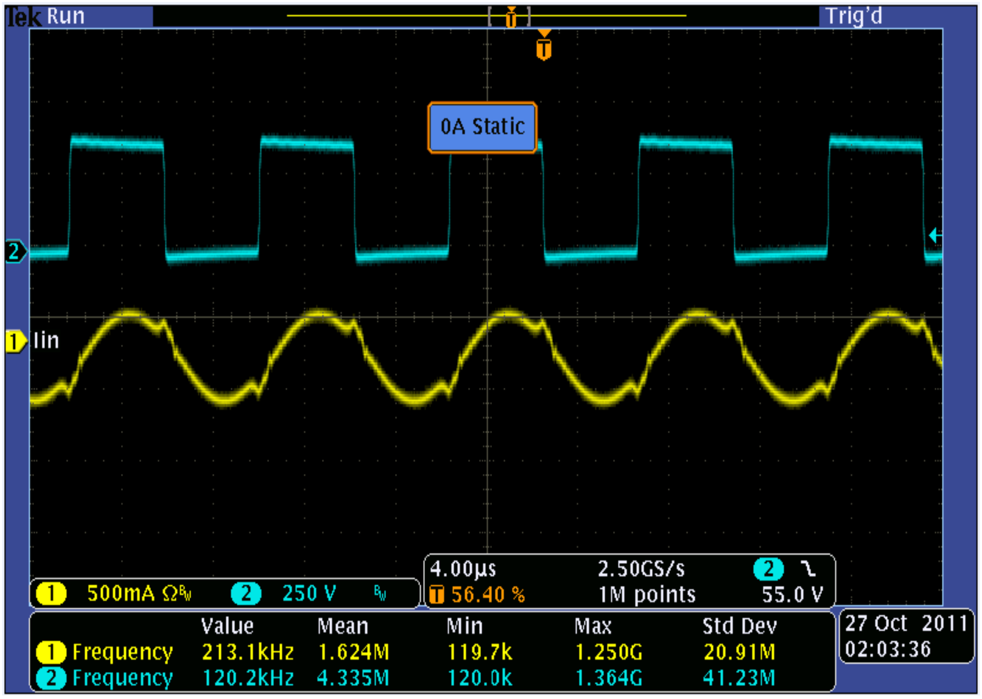


```
//------------------------------------------------------------------
void A3(void)
//------------------------------------------------------------------
{
    if(INCR_BUILD == 3)
    {
        //When the DC-DC runs without the Inverter, check for min panel voltage before starting MPPT DC-DC.
        //When DC-DC Runs with the Inverter connected, this flag (Start_DC_DC) is not used. Another GPIO (GPIO16)
        //enable signal is then used (activated by the Inverter) to start the DC-DC.
        if(Vp_fb_Avg >= VPV_MIN)
        {

            Start_DC_DC=1;//Start DC-DC MPPT provided LLC PWM is ON(This is checked in the 20kHz SECONDARY ISR)

            //*********************************************************************************
            //To control the variable "Start_DC_DC" from CCS watch window and run the DC-DC code with external 12V bias (not using PR798 bias sup
            //COMMENT OUT the line above. This will allow code to run with MPPT off; Then the user will apply panel volt and start MPPT
            //from CCS watch window by setting Start_DC_DC = 1
            //*********************************************************************************
        }
        else
        {
            Start_DC_DC = 0;//Do not start DC-DC MPPT
        }

        //if(GpioDataRegs.GPADAT.bit.GPIO16 == 0)
        if(GPIO_status == 0)
        {
            A_Task_Ptr = &A4;
        }
        else
        {
        //-------------------
        //the next time CpuTimer0 'counter' reaches Period value go to A1
        A_Task_Ptr = &A1;
        //-------------------
        }
```

Comment out this one line of code, save the file, recompile and reload the project in the flash memory. Follow steps 1 through 3 (as described before) and run the code. Enable the real time mode and then Click on Continuous Refresh buttons for the watch view. The variables *Start_DC_DC* will be set to 0 and *inverter_connected* will be set to 1.

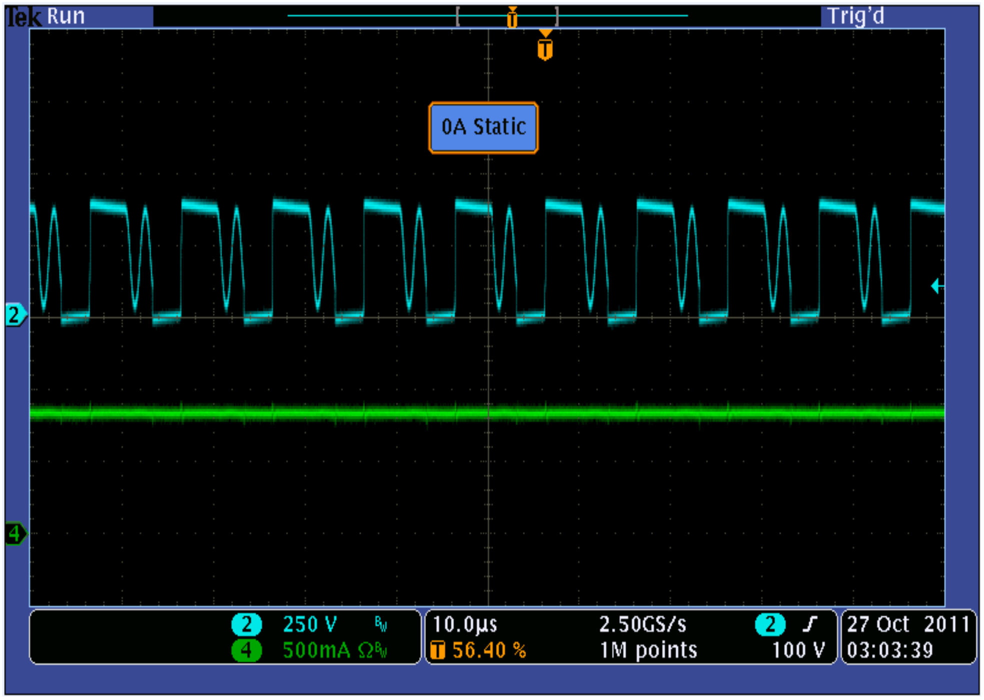
7) Use a voltmeter to monitor the DC bus voltage across the EVM output. Now turn on the PV panel emulator with the setup described in step 5. At this point the MPPT will still remain off and so the EVM input should be around 260V and the output voltage, with a 1kohm load resistor, will also be at the same level. From the CCS watch window now set the variables *inverter_connected* to 0 and *Start_DC_DC* to 1. This will start the MPPT algorithm and the EVM output will rise to around 400V. With the MPPT turned on and the input voltage loop controller connected, the panel output voltage (i.e., the boost input voltage) will be around 220V. The EVM will deliver 165W of panel power (as described in step 5) at the MPPT reference voltage of 220V.

Use an oscilloscope to capture the switch node voltage and transformer primary current from the LLC stage under this operating condition. This is shown in figure below where Ch2 represents the LLC primary switch node voltage and Ch1 represents the LLC primary current.



	Value	Mean	Min	Max	Std Dev
1 Frequency	213.1kHz	1.624M	119.7k	1.250G	20.91M
2 Frequency	120.2kHz	4.335M	120.0k	1.364G	41.23M

Now use the scope probes to capture the boost stage MOSFET drain to source voltage and the PV emulator current under this operating condition. This is shown in figure below where Ch2 represents the boost MOSFET drain to source voltage and Ch4 represents the panel current.



Observe the variables on the watch window. The variable Vp_fb should show a value of about 0.4297 (=220/512) when the Q format is set to

Q24. The maximum panel voltage set by the sense resistors is about 512V that corresponds to maximum ADC input of 3.3V. Therefore, the normalized or per unit value will be about 0.4297 when the actual panel voltage is 220Vdc. The variable Vb_fb should show a value of about 0.7813 (=400/512) when the Q format is set to Q24. The maximum boost output voltage set by the sense resistors is about 512V that corresponds to maximum ADC input of 3.3V. Therefore, the normalized or per unit value will be about 0.7813 when the actual boost output voltage is 400Vdc.

8) Follow steps 26 and 27 as in section 3.1 to turn off power and reset the MCU. Undo the change in code performed in step (6), then recompile and reload the code into the flash memory for standalone operation of the EVM. Set the Piccolo control card jumpers (see Piccolo control card documentation) appropriately such that the device can boot from FLASH.

End of Exercise

References

For more information please refer to the following guides:

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

  ..\controlSUITE\development_kits\HV_SOLAR_DC_DC\~Docs\QSG_HV_SOLAR_DC_DC_GUI_Rev1.0.pdf

- **MPPT DC-DC_Rel-1.0-HWdevPkg** – A folder containing various files related to the Piccolo-B controller card schematics and the MPPT DC-DC schematic.

  ..\controlSUITE\development_kits\HV_SOLAR_DC-DC\HV_SOLAR_DC-DC_HWDevPkg

- **F28xxx User's Guides**

  http://www.ti.com/f28xuserguides