Implementing Photovoltaic Inverter System with Ethernet Monitoring using C2000 Microcontrollers on Solar Explorer Kit

v 1.0, 1/31/2012

Manish Bhardwaj, Noah Leykun Brehanu
C2000 Systems and Applications Team

ABSTRACT

Energy from renewable sources such as solar and wind are gaining interest as the world's power demands increase and non-renewable resources deplete. Thus attempts are being made to raise the percentage of energy sourced from renewable sources in the grid. Photovoltaic (PV) energy sources are considered quintessential factor in increasing this percentage due to ubiquitous nature of solar power. With the renewable content growing as a percentage of the total utility power, increased control and monitoring of the power produced is required. Control and communication pose unique challenges on system designs with these energy nodes

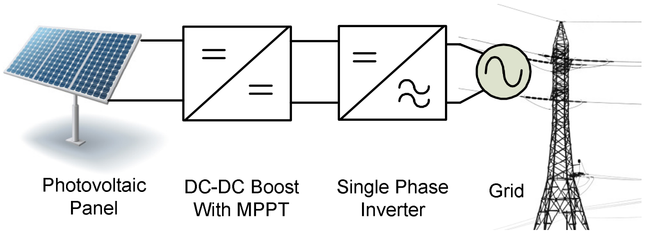


Fig 1 PV Grid Tied Inverter

Error! Reference source not found. shows a typical PV inverter system that feeds power into the gird. To extract power from the PV panel closed loop control of the different stages is desired, also for monitoring a medium like Ethernet web server that can be used to display power information and control power production is needed. The Texas Instruments Concerto family of C2000 microcontrollers, is well suited for these systems with control and communication needs. This guide uses the Solar Explorer Kit (TMDSSOLARCEXPKIT) platform to illustrate the system with a F28M35H52C control card.

Contents

1 INTRODUCTION

PV Inverter system has its unique set of challenges, to name a few: as PV panel have a non linear V vs I characteristics the PV system must make sure that it operates at the maximum power point (MPP) of this curve and feed the maximum power it can into the grid. To feed current into the grid as PV is a DC source an inverter is necessary for DC-AC conversion. Also, for the inverter to be able to feed power into the grid it must maintain a DC bus greater than the max instantaneous voltage of the grid, thus a boost stage from the PV output voltage may be required. In addition to this the current that the inverter feeds into the grid must be clean and in phase. A combination of hardware and software is used to tackle these challenges. Though the exact power stages vary depending on the PV power rating and other regulatory factors a typical PV inverter can be thought of as consisting of two power stages as shown in Fig 1:

i. DC-DC stage for boosting the voltage and tracking the MPP of the panel.

ii. DC-AC stage for conversion to AC and to feed current into the grid

In addition, with the renewable content in the grid increasing, requirements to control and monitor the power generated from distributed power generation units such as PV Inverter is also rising. The type of control and monitoring function desired from the installation are currently subject to the power ratings. For example high power central inverters communicate to the utility such that the power can be modulated dependent on the grid requirements. On the other hand in residential installation impetus is for the user to be able to see the energy production, measure the power, and if feed in tariffs are present then know the revenue the PV installation generates. Also with concept of microgrids and macrogrids gaining interest controlling and monitoring the information from the PV panel through Ethernet is becoming more popular.

However, combining closed loop control of all the power stages needed to process power from the PV panel for a PV system along with Ethernet stack on a single controller is challenging. Therefore another controller is added to perform the communication tasks in such systems typically. However, this adds additional challenges to the system design and cost. The Texas Instruments Concerto family of C2000 microcontrollers, which have dual heterogeneous core MCU's is well suited for such systems.TMS320F28M35x is a dual core MCU from Concerto family and has two different CPU cores on a single chip. One of the CPU core is the Texas Instruments C28x which is optimized for control tasks and handles the closed loop operation of the power stage whereas the other CPU is an ARM Cortex M3 which is sued to run the ethernet stack which implements the monitoring and control of the node.

1.1 Solar Explorer Kit

Solar Explorer Kit is a low voltage platform to evaluate C2000 microcontroller family of devices for renewable energy applications such as PV inverter. Fig 3 gives a block diagram of different stages present on the Solar Explorer kit that are used for the PV inverter system The input to the solar explorer kit is a 20V DC power supply which powers the controller and the supporting circuitry. A 50W solar panel can be connected to the board (Typical values Vmpp 17V, Pmax 50W). However for quick demonstration of the power processing, a PV emulator power stage is integrated on the board along with other stages that are needed to process power. The control of the PV panel is kept separate from the control of the other stages. PV is light dependent source, the PV panel emulator can be used to test PV inverter under changing lighting conditions. As the control of PV panel is executed on a separate controller a SPI link is added from the DIMM100 on the solar explorer to the PV Panel emulator controller. This simplifies the debug and demonstration. Details on the hardware and power stages present on the board can be found at:

```
controlSUITE\development_kits\
                        SolarExplorer_vx.x\~Docs\SolarExplorer_HWGuide.pdf
```

The Solar Explorer kit can be used to implement a PV inverter system by connecting the power stages as shown in Fig 2.

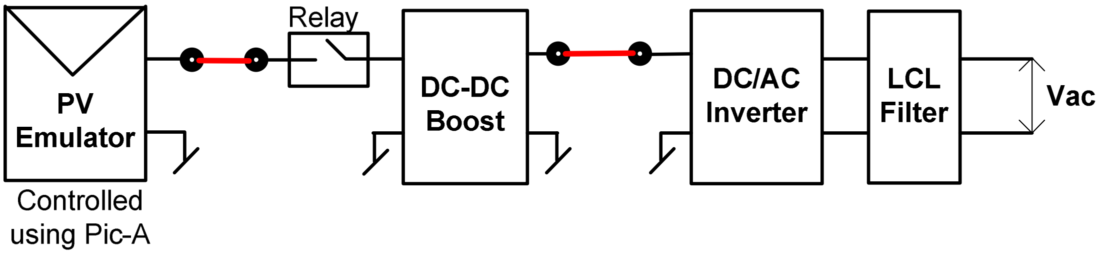


Fig 2 PV Inverter using Solar Explorer Kit

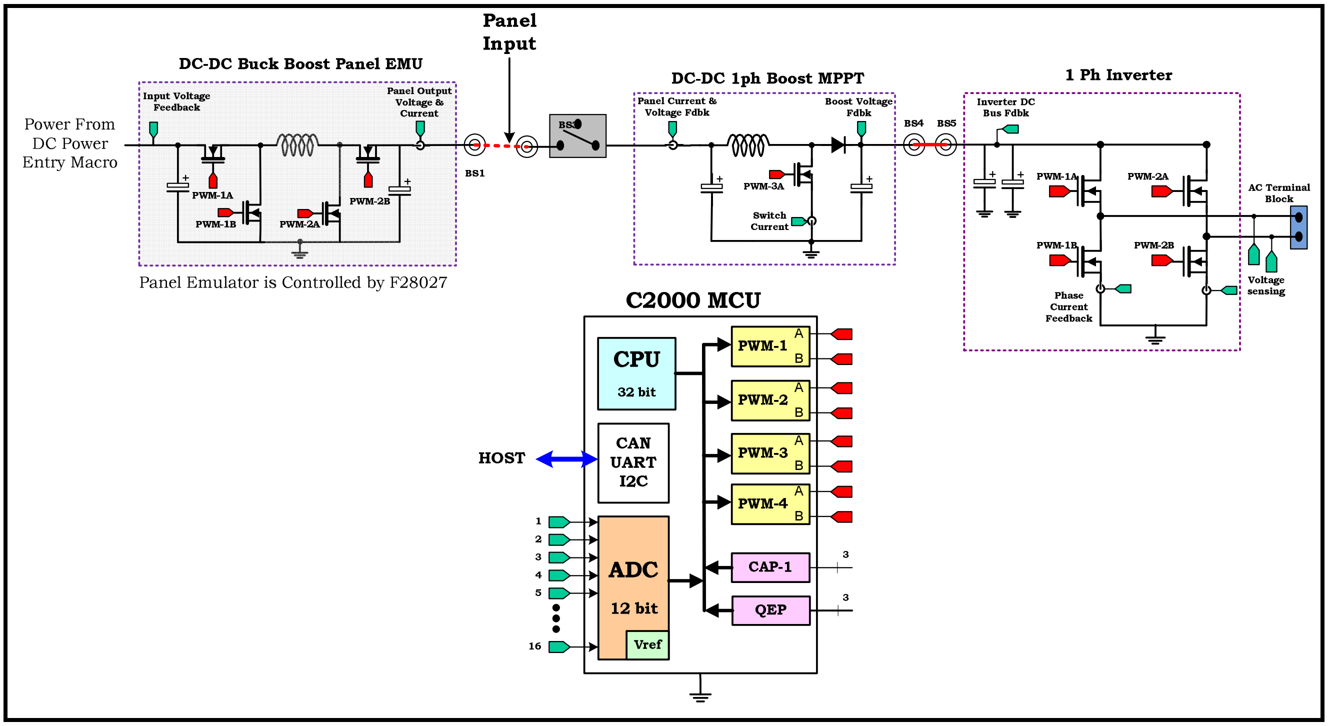


Fig 3 Solar Explorer Kit Power Stages for a typical PV Inverter System

DCDC Boost with MPPT

Input to this stage can come from Panel emulator block or externally connected solar panel. Fig 4 shows the power stage circuit implemented on solar explorer kit for this stage. Inductor L1, MOSFET switch Q1 and diode D1, together form the boost circuit. The boost circuit operates at 100 KHz. Fig 5 illustrates the control scheme for the DC-DC Boost stage with MPPT.

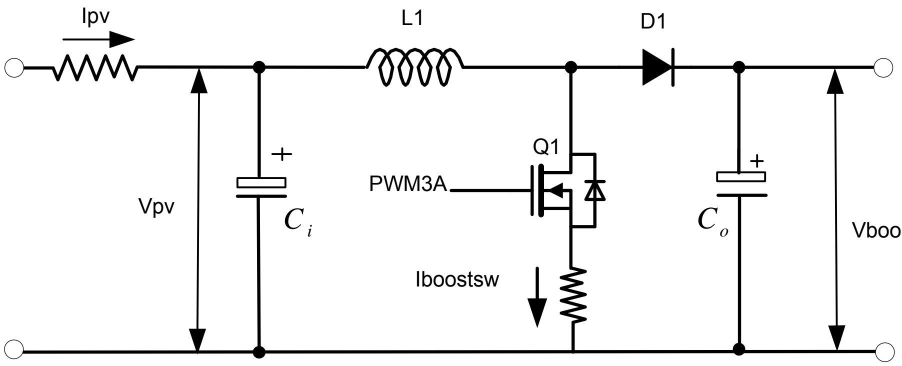


Fig 4 DC DC Boost stage power circuit

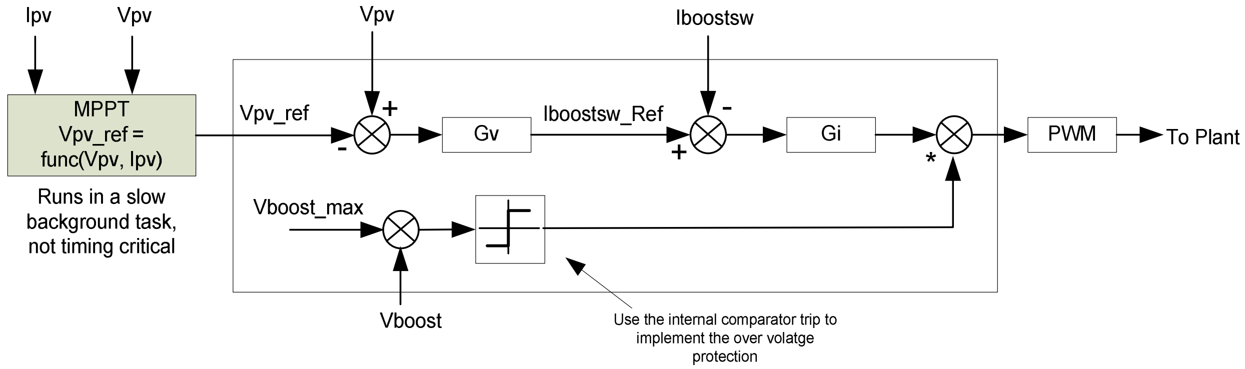


Fig 5 Control of DC-DC Boost with MPPT

DC-AC Single Phase Inverter

A full bridge inverter is used to generate single phase AC waveform. Input to this block is from DCDC Boost power stage. Inverter stage operates at 20Khz. Switches Q1, Q2, Q3, Q4 form the full bridge inverter. These, together with LCL filter, generate filtered single phase AC output.

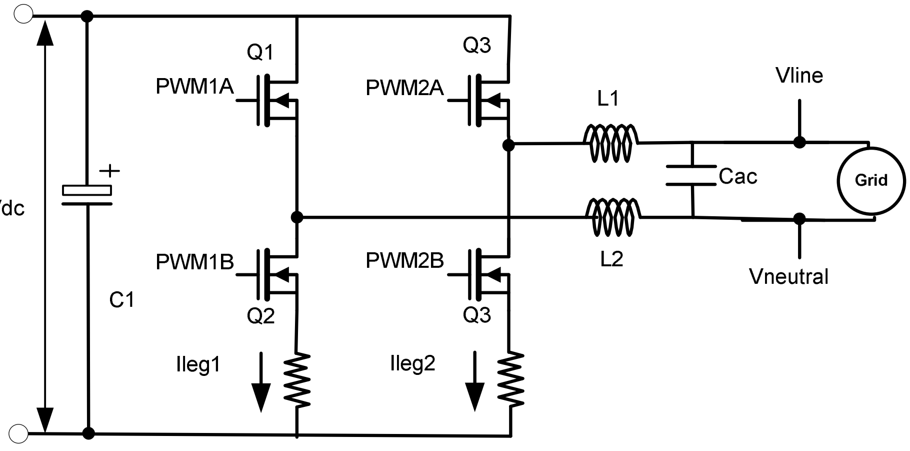


Fig 6 DCAC inverter stage power circuit

1.2 Gird Tied PV Inverter Control Diagram

Fig 7 illustrates the control scheme for a grid connected PV inverter. It is clearly noted that there are two Interrupt Service Routines (ISRs) one for closed loop control of the DC-DC stage(50Khz, every alternate switching period) and other for the closed loop control of the inverter stage(20Khz). Functions such as MPPT, gird synchronization are also required. All these key functions are implemented on the F28035 MCU for the Solar Explorer Kit.

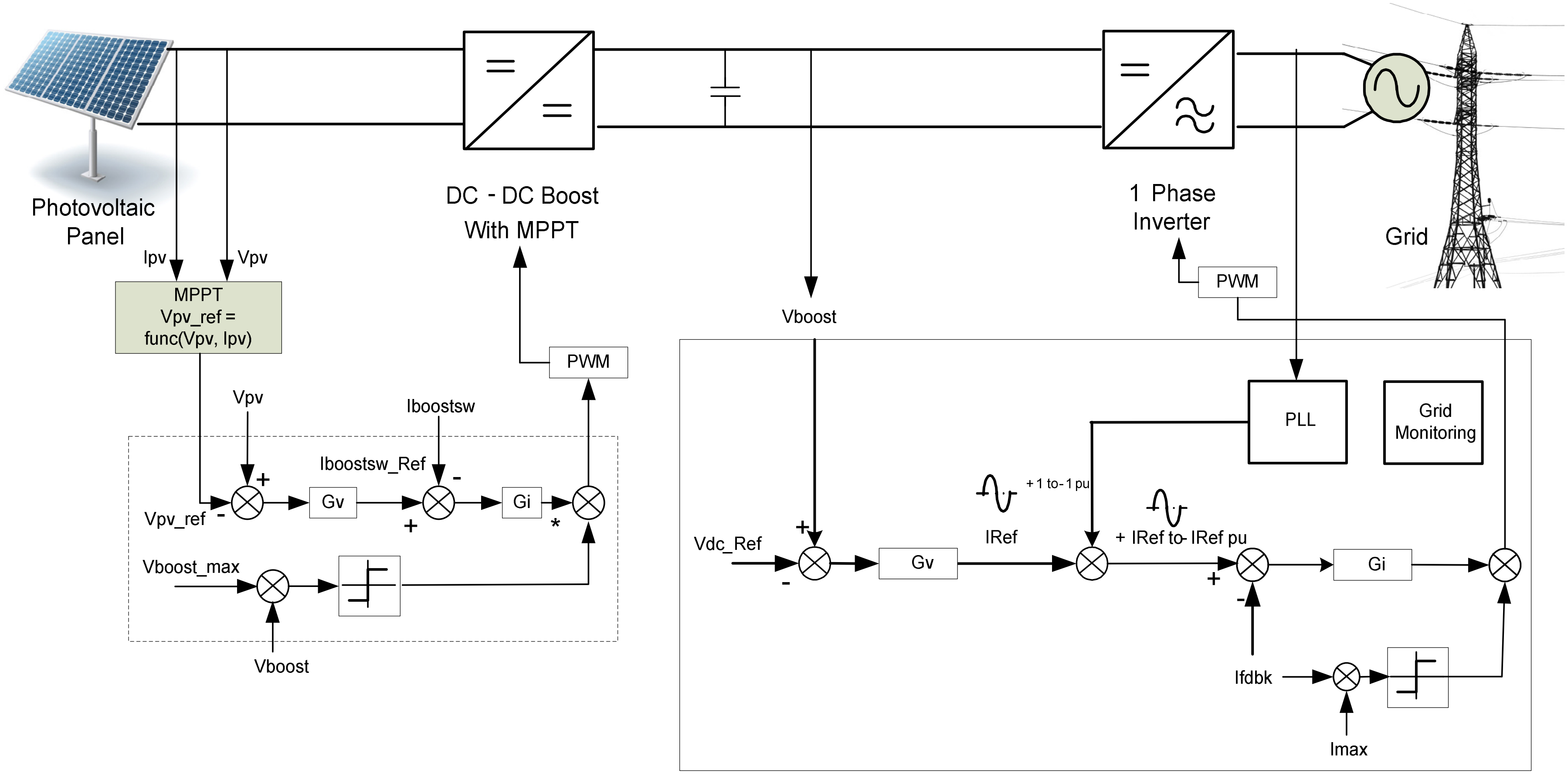


Fig 7 Control of Grid Tied PV Inverter System

1.3 Concerto Dual Core MCU

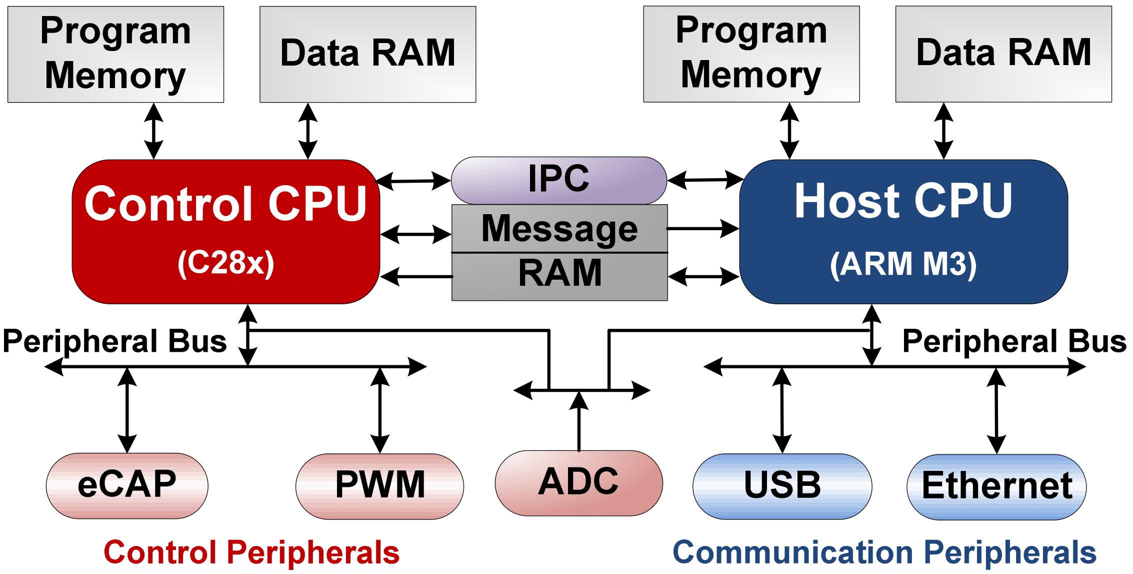


Fig 8 Concerto F28M35x Dual Heterogeneous Core MCU

Fig 8 shows the architecture of the chip. The control peripherals such as pulse width modulators, time capture modules etc are mapped to the C28x. The communication peripherals such as Ethernet and USB are mapped to the M3 core. A single analog to digital converter is shared by both the processors. Having ADC access by both cores with different ISA's adds features of redundancy in the system, for safety critical checks. M3 is the master in the dual core MCU and has control of all IO's at boot time. It then sets up the clocks and assigns IO's to the C28x i.e. the control engine.

InterProcessor Communication

The split of control and communication tasks and peripherals between the two cores provides a clean system partitioning. Communication between the two processors is handled using an Inter Processor Communication (IPC) peripheral which can configure and acknowledge interrupts on either core. Data is exchanged between the two cores with means of shared RAMs. The shared RAMs allow read accesses from the two cores but only one core has privileges to write. Fig 9 shows the software structure used to pass messages and commands between the two CPU's through the shared RAM's. The IPC peripheral can configure interrupts for the data to be read immediately and provide acknowledgement.

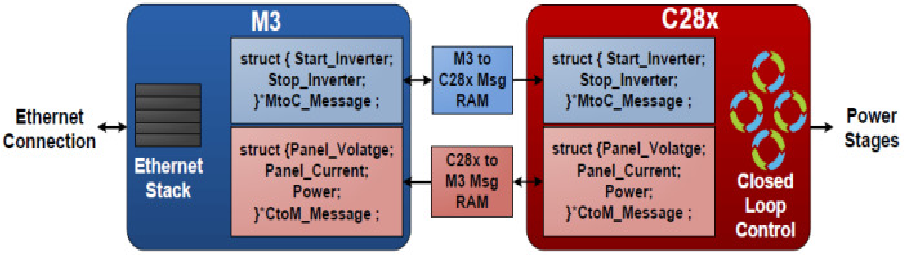

Ethernet Connection
M3
Ethernet Stack
struct { Start_Inverter; Stop_Inverter; }*MtoC_Message ;
struct {Panel_Volatge; Panel_Current; Power; }*CtoM_Message ;
M3 to C28x Msg RAM
C28x to M3 Msg RAM
C28x
struct { Start_Inverter; Stop_Inverter; }*MtoC_Message ;
struct {Panel_Volatge; Panel_Current; Power; }*CtoM_Message ;
Closed Loop Control
Power Stages

Fig 9 InterProcessor Communication Structure

2 Software & Control Description

This section describes the details of PV Inverter control and software for the Solar Explorer kit.

2.1 Project Framework

Dual core F28M35x based PV inverter system consists of two projects, one for the C28x core and the other one for the M3 core. The two projects can be found at

```
controlSUITE\development_kits\SolarExplorer_v1.0\SolarExplorer_PVInverter_F28M35Hx
                                                                             \C28x
                                                                             \M3
```

As M3 is the master the M3 project ,Fig 10, is responsible for setting up the clocks, assigning peripheral to the C28x as needed, sending commands to the C28x and monitoring information from the C28x. M3 is also responsible for the Ethernet stack and running a web server which enables access to the PV Inverter information remotely. On the solar explorer kit, the M3 also uses the SSI peripheral to communicate to the F28027, which controls the panel emulator stage, to change the lighting condition etc to change characteristic of the PV panel emulator.

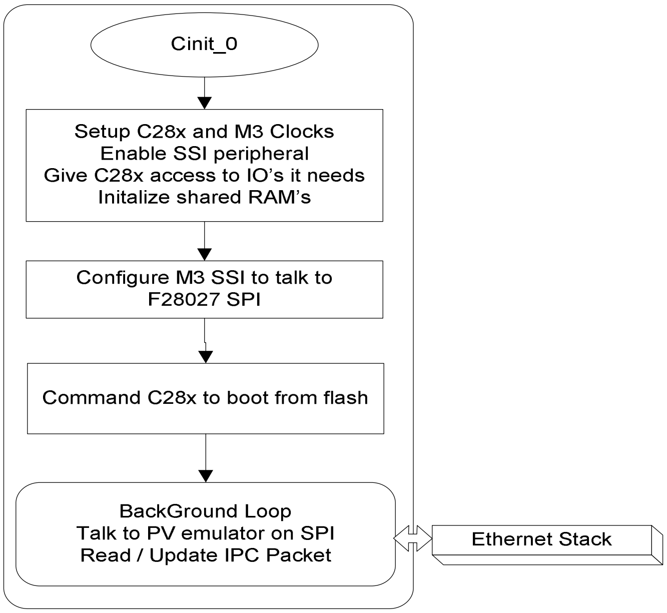


Fig 10 M3 Project Software Flow

The C28x project is responsible for running the closed loop control of the power stage. As shown in Fig 7 PV inverter control requires two real time ISR's. One is the for the closed loop control of the DC-DC stage and the other for the closed loop control of the DC-AC stage. The C2000 Solar Explorer Kit project makes use of the "C-background/C-ISR/ASM-ISR" framework, Fig 11. The fast ISR (100kHz), controlling DC-DC Boost stage, runs in assembly environment using the Digital Power Library and slower ISR (20kHz), controlling DC-AC inverter, is run from the C environment. This DC-AC ISR is made interruptible by the DC-DC ISR. 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 key framework C files used in the project are:

SolarExplorer-Main.c – this file is used to initialize, run, and manage the application. In addition, this file also contains ISR for inverter stage control, MPPT algorithm execution, data logging and relay control etc. The C28x also reads the IPC packet information in the background task.

SolarExplorer-DevInit_F28M35x.c – This file contains all the initialization routines and configuration of IOs and peripherals for this application. (Note that the M3 needs to assign the IO's to the C28x for the IO assignment in this file to be effective)

SolarExplorer-Settings.h – This file contains of setting such as incremental build option and various defines for PWM frequency, ISR triggers that are used in the project framework.

SolarExplorer-Includes.h – This file contains of all the header files used by the project.

SolarExplorer-DPL-ISR.asm – This file contains 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. This is used for the fast DC-DC boost closed loop control.

Other important include files: (more details can be found in Solar Library documentation)
SPLL_1ph.h – This files contains code for calculating grid angle while connected to the grid and used in inverter stage control.
SineAnalyzer_diff.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 runtime section which executes at 10kHz rate.
mppt_incc.h: This file contains code for incremental conductance algorithm used for tracking MPPT.
mppt_pno.h: This file contains code for perturb and observe algorithm used for tracking MPPT.

Fig 11 gives the structure of the PV inverter software for the C28x, with the main background loop, the DC-DC ISR and the DC-AC ISR.

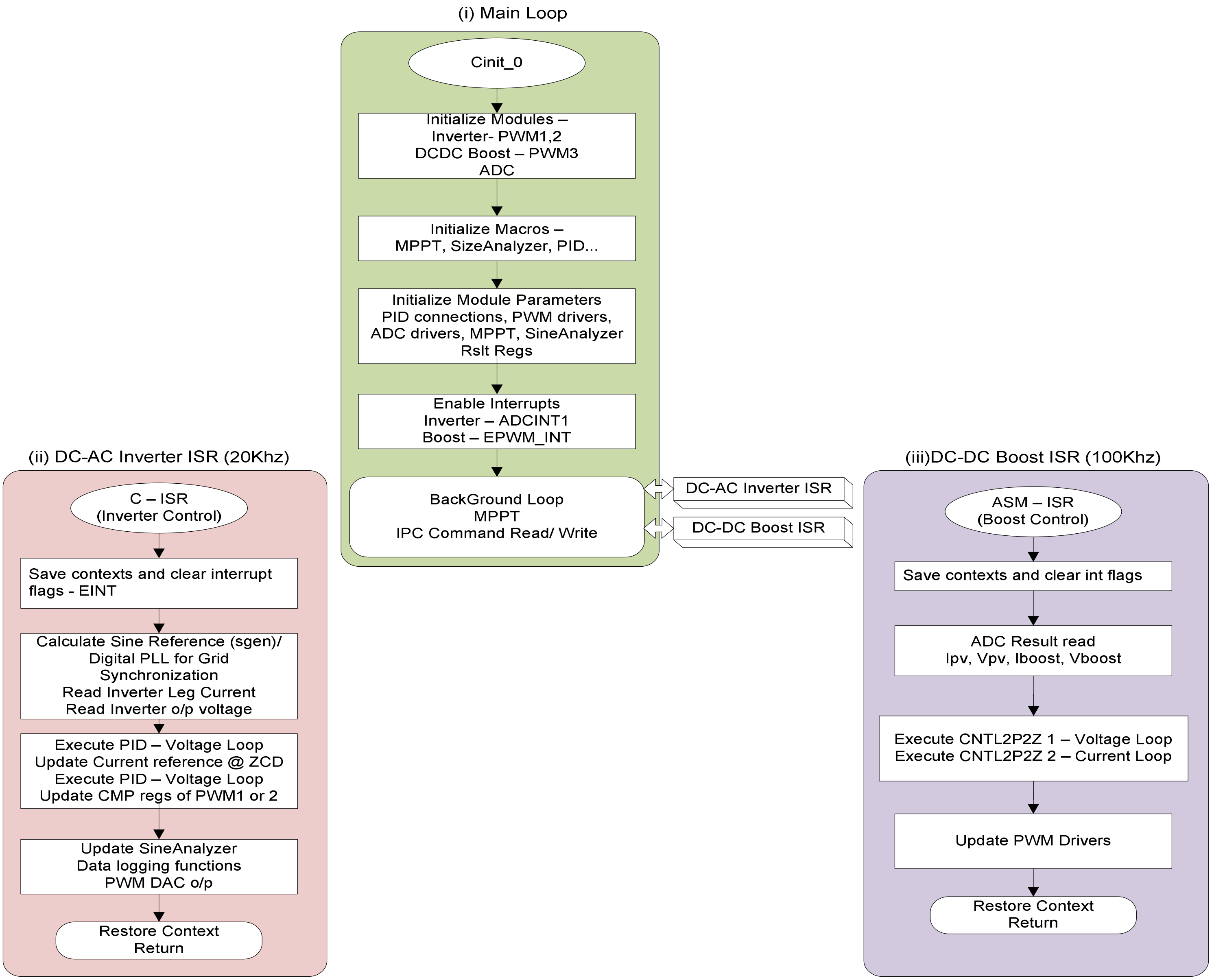


Fig 11 C28x PV Inverter Software structure (i) Main loop (ii) Inverter stage ISR (iii) DCDC Boost stage ISR

2.2 Project Dependencies & Resources

Hardware Kit	: TMDSSOLARCEXPKIT [R5]
Control Card	: F28M35x Rev1.2 or later
Software IDE	: CCSv4.2.4 or later

Control Suite Dependencies

Device Support (F28035 Header Files)	: `controlSUITE\device_support\f28m35x\v100`
IQMath Library	: `controlSUITE\libs\math\IQmath\v160`
SGEN Lib	: `controlSUITE\libs\dsp\SGEN\v101`
Digital Power Library	: `controlSUITE\app_libs\digital_power\f28M35x_v1.0`
Solar Library	: `controlSUITE\app_libs\solar\v1.0\float`
Drivers	: `controlSUITE\app_libs\drivers\v1.0\F28M35x`

The guide assumes that the user has already read the following documents related to the kit:

`controlSUITE\development_kits\SolarExplorer\~Docs\SolarExplorer_HWGuide.pdf`

The above documents discuss the kit's hardware features and power stages.

2.3 Control Description

Fig 7 shows the control of a grid tied PV inverter, which comprises of closed loop control of the boost and closed loop control of the inverter. Following sections gives details of the software flow for these two modules.

DC-DC Boost with MPPT Control Software

To get the most energy out of the solar panel, panel needs to be operated at its maximum power point. Maximum power point however is not fixed due to the non linear nature of the PV cell and changes with temperature, light intensity etc. Thus different techniques are used to track maximum power point of the panel like Perturb and Observe, incremental conductance algorithms. These techniques try to track the maximum power point of the panel under given operating conditions and are thus referred to as Maximum Power Point Tracking (MPPT) techniques/algorithms. The Solar Explorer kit has a front-end boost converter to boost the input voltage from the solar panel to a suitable level for the inverter and track the MPP, Fig 4.

The control of the stage is described in Fig 5. To track the MPP, input voltage (V_{pv}) and Input Current (I_{pv}) are sensed. The boost converter is a traditional single phase converter with a single switching MOSFET Q1. The duty cycle of the PWM output driving the Q1 MOSFET switch determines the amount of boost imparted is the controlled parameter. The MPPT is realized using nested control loops, an outer voltage loop that regulates input DC voltage (V_{pv}) and an inner current loop that controls the current of the boost stage. Increasing the current reference of the boost, i.e. current drawn through the boost loads the panel and hence results in the panel output voltage drop. Therefore the sign for the outer voltage compensator reference and feedback are reversed. The current and voltage controllers are executed at a rate of 50 kHz (half of the PWM switching frequency) while the MPPT controller is executed at a much slower rate ~ 10Hz. It is noted from Fig 5 that the boost stage output voltage is not being controlled through software. Boost output voltage however is regulated by the DC-AC inverter, which modulates the current drawn by the inverter to keep this voltage regulated. However, for protection the output of the boost is connected to ADC pin with and internal comparator that can be used to trip the PWM to the DC-DC stage in case of over voltage. Note on Concerto the internal comparator output is first mapped to an IO on the analog subsystem (), and this is then input to IO on the C28x which is configured as a cycle by cycle interrupt source. The connection from the analog to the C28x IO is fixed on the concerto control card. The code below, found in SolarExplorer-DevInit_F28M35x.c configures the comparator output to be brought out on GPIO129.

```
//  GPIO-129 - PIN FUNCTION = --COMP1OUT--
GpioG2CtrlRegs.GPEMUX1.bit.GPIO129 = 3;   //0=GPIO, 1=ePWM1B, 2=HRCAP1IN, 3=COMP1OUT
```

The GPIO129 is routed on the concerto control card to GPIO35, which is then configured as a trip source number ten in the file SolarExplorer-Main.c.

```
//Map Trip input 10 (COMP1OUT) to PF3_GPIO35
GpioTripRegs.GPTRIP10SEL.bit.GPTRIP10SEL =35;
```

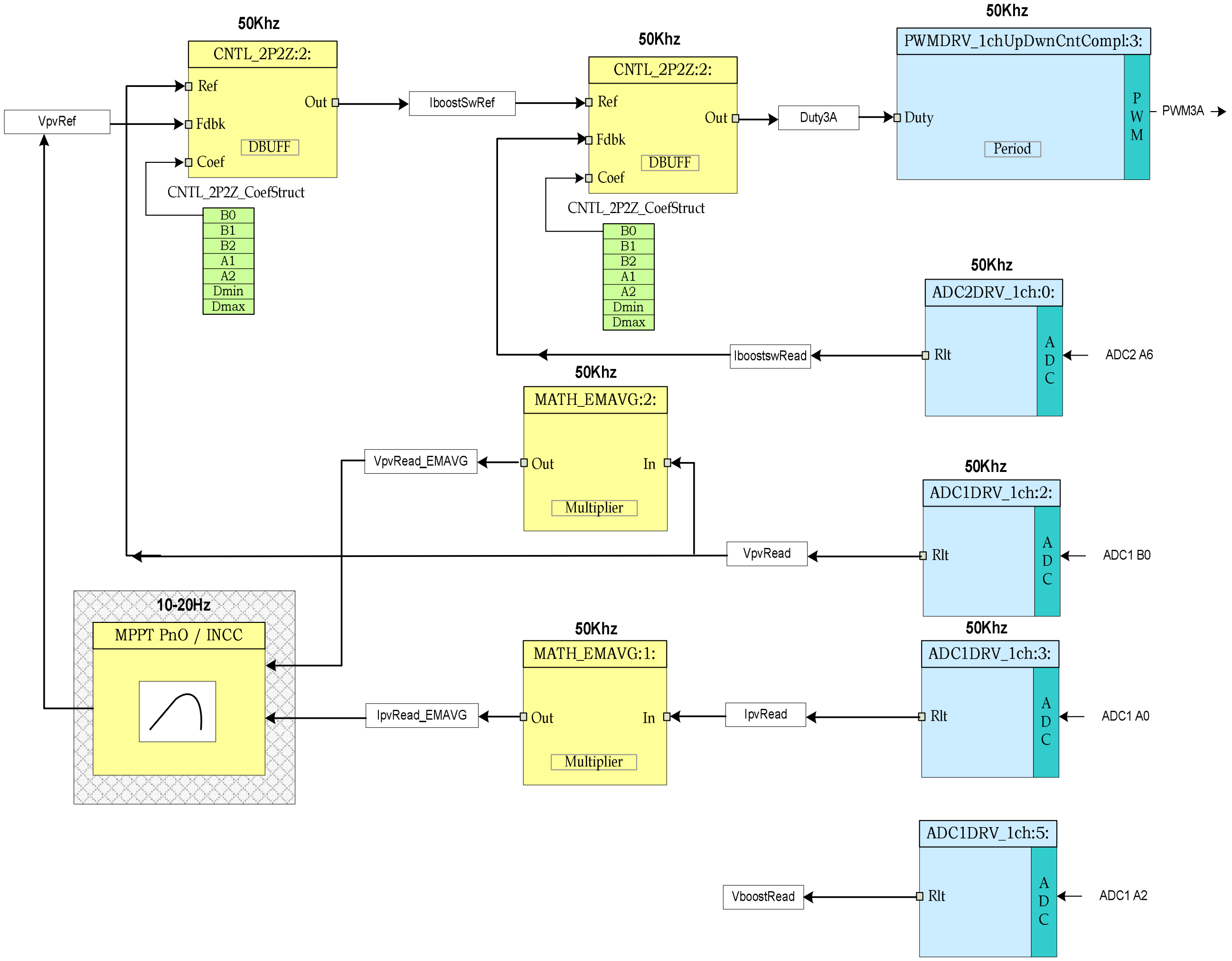


Fig 12 DC-DC 1ph Boost with MPPT Software Diagram

As switching rate of the DC-DC stage is fairly high 100Khz, the control ISR for the DC-DC is implemented in an optimized assembly ISR (ASM – ISR) which uses components from the Digital Power Library. In the PV inverter project the DC-DC ISR is invoked every alternate switching cycle, this is done because PV panel output does not change very fast. Fig 12 gives the software diagram for the DC-DC stage using the optimized blocks from the digital power library.

The ADC result registers are read by the ADCnDRV_1ch block and converted to normalized values and stored in variables I_{pvRead}, V_{pvread}, $I_{boostswread}$ and $V_{boostread}$. Two 2-pole 2-zero controllers(CNTL_2P2Z) are used to close the inner DC-DC boost current loop and the outer input voltage loop. MPPT algorithm provides reference input voltage to the boost stage to enable panel operation at maximum power point. The sensed input voltage is compared with the voltage command (V_{pvref}), generated by MPPT controller, in the voltage control loop. The voltage controller output is then compared with the output current ($I_{boostswread)}$ feedback in the current controller. Current loop controller's output decides the amount duty to be imparted to the PWM so as to regulate the input voltage indirectly. The PWMDRV_1ch_UpDwnCntCompl block is used to drive the DC-DC stage. Panel current and voltage are filtered using the MATH_EMAVG block, this is done to remove any noise on the panel current and voltage sensing that may confuse the MPPT algorithm.

Notice the color coding for the software blocks. The blocks in 'dark blue' represent the hardware modules on the C2000 controller. The blocks in 'blue' are the software drivers for these modules. Blocks in 'yellow' are the controller blocks for the control loop. Although a 2-pole 2-zero controller is used here, the controller could very well be a PI/PID, a 3-pole 3-zero or any other controller that can be suitably implemented for this application. Similarly for MPP tracking, users can choose to use a different algorithm.

Code snippet below shows the Input/Output connections between the different blocks used from the Digital Power Library to implement the DC-DC Boost MPPT control software, this can directly be related to the control diagram above.

```
PWMDRV_1ch_UpDwnCntCompl_Duty3 = &Duty3A;

ADC2DRV_1ch_Rlt0=&IboostswRead;

ADC1DRV_1ch_Rlt2=&VboostRead;
ADC1DRV_1ch_Rlt3=&IpvRead;
ADC1DRV_1ch_Rlt5=&VpvRead;

// MATH_EMAVG1 block connections
MATH_EMAVG_In1=&IpvRead;
MATH_EMAVG_Out1=&IpvRead_EMAVG;
MATH_EMAVG_Multiplier1=0.001;

// MATH_EMAVG2 block connections
MATH_EMAVG_In2=&VpvRead;
MATH_EMAVG_Out2=&VpvRead_EMAVG;
MATH_EMAVG_Multiplier2=0.001;

//connect the 2P2Z connections, for the inner current Loop
CNTL_2P2Z_Ref2 = &IboostSwRef;
CNTL_2P2Z_Out2 = &Duty3A;
CNTL_2P2Z_Fdbk2= &IboostswRead;
CNTL_2P2Z_Coef2 = &CNTL_2P2Z_CoefStruct2.b2;

//connect the 2P2Z connections, for the outer Voltage Loop
CNTL_2P2Z_Ref1 = &VpvRead;
CNTL_2P2Z_Fdbk1 = &VpvRef;
CNTL_2P2Z_Out1 = &IboostSwRef;
CNTL_2P2Z_Coef1 = &CNTL_2P2Z_CoefStruct1.b2;
```

The Run time ISR is found in the DPL-ISR.asm file, which consists of just calling the run time macros from the digital power library. The MPPT algorithm is called from a background task in the background C framework.

DC-AC Single Phase Inverter Control Software

Inverter stage gets input from the DC-DC boost stage as shown in Fig 3 and the inverter converts DC into AC. Inverter can be operated in off-gird or grid-tie configuration.

For a full bridge inverter it can be noted that when using unipolar modulation the current fed is given by the equation:

$$\Delta i_{grid} = \frac{(V_{dc} - v_{grid}).D}{Z_{LCL}(F_{sw})} + \frac{(0 - v_{grid})(1 - D)}{Z_{LCL}(F_{sw})} = \frac{V_{dc} * D - v_{grid}}{Z_{LCL}(F_{sw})}$$

Where D is the duty cycle.

It is clear from the equation that for the inverter to be able to feed current into the grid the Vdc must always be greater than the max grid voltage. Also it’s known from Fig 5 that the DC bus is not regulated by the DC-DC boost stage. Therefore the Inverter stage software uses nested control loops – an outer voltage loop and an inner current loop. Voltage loop generates the reference command for the current loop, as increasing the current command will load the stage and hence cause a drop in the DC bus voltage the sign for reference and the feedback are reversed. The current command is then multiplied by the AC angle to get the instantaneous current reference. In case of “off-grid” configuration sine reference is generated using SGEN library function which provides the angle value whereas for grid connected software PLL provides the grid angle. The instantaneous current reference is then used by the current compensator along with the feedback current to provide duty cycle for the full bridge inverter. The outer voltage loop is only run at ZCD of the AC to prevent any distortion in the current. Fig 13 and Fig 14 show the software diagram for the inverter control under grid-tie and off-grid configurations

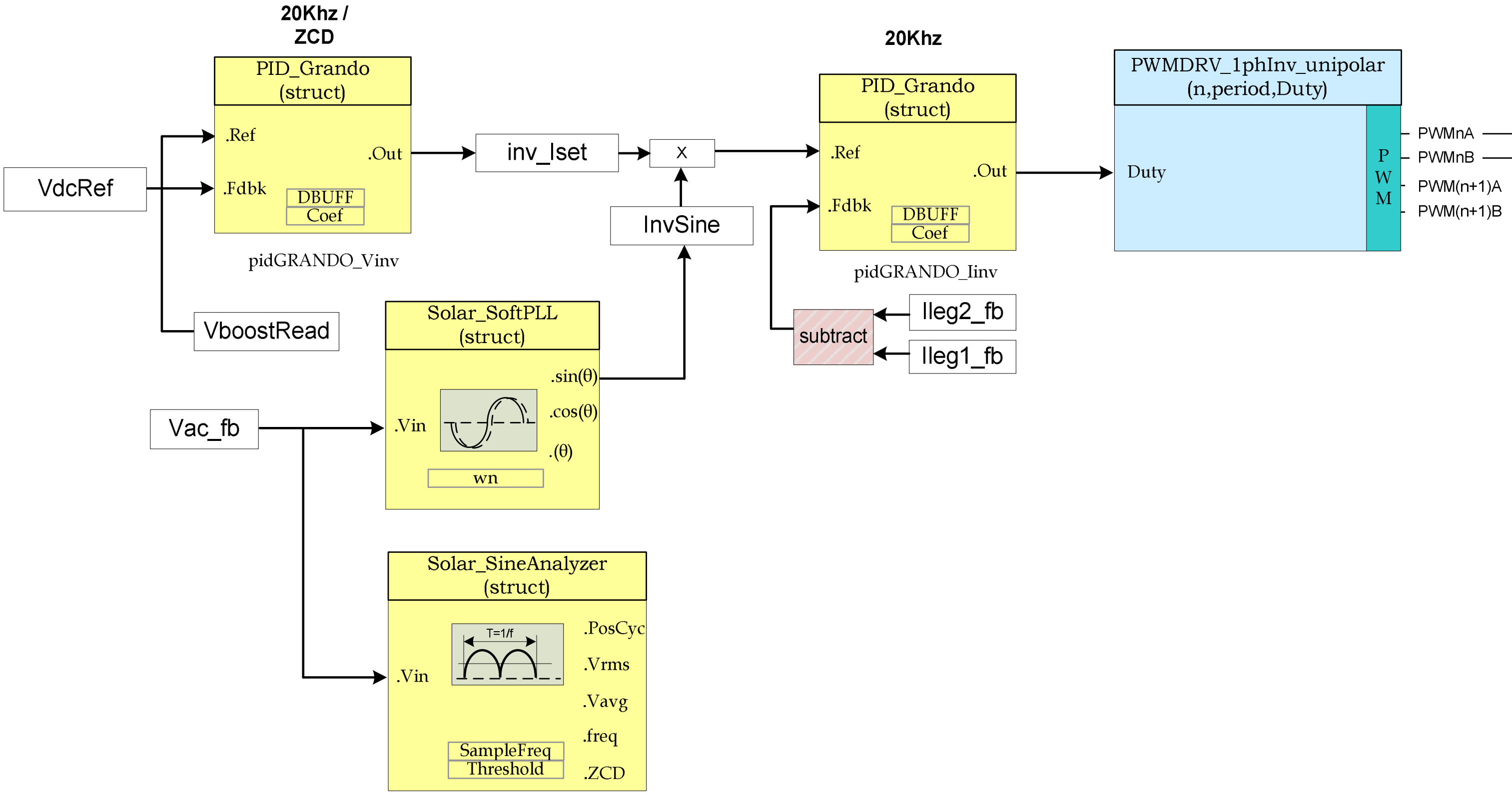


Fig 13 Closed Loop Current Control for DC-AC with grid connection

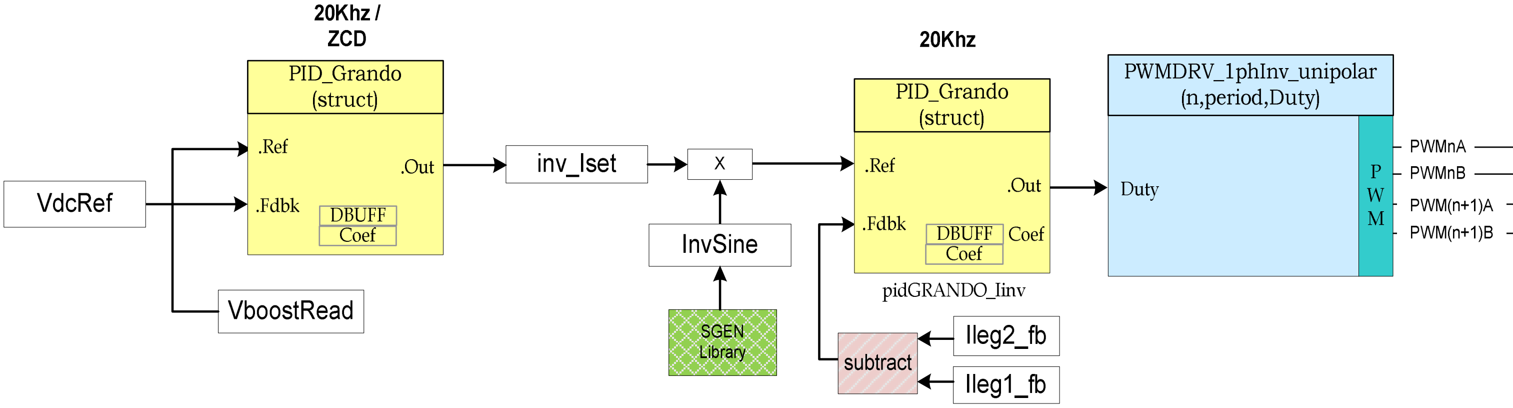


Fig 14 Closed Loop Current Control for DC-AC for off grid system

2.5 DC-DC and DC-AC Integration

As shown in Fig 7 PV inverter control requires two real time ISR's one is the for the closed loop control of the DC-DC stage(100Khz) and the other for the closed loop control of the DC-AC stage(20Khz). The peripheral i.e. ADC and PWM's on the C2000 device family have been designed to integrate multi frequency control loops and guarantee sampling at correct instances of the PWM waveform. However to best utilize the ADC it needs to be guaranteed that the multi rate ISRs do not conflict for the ADC resource at any instance. For this the phase shift mechanism of the PWM's on the ePWM peripheral is employed. Fig 15 illustrates the timing diagram for configuring the EPWM for the inverter and the boost stage and the synchronization mechanism used to avoid ADC conflicts.

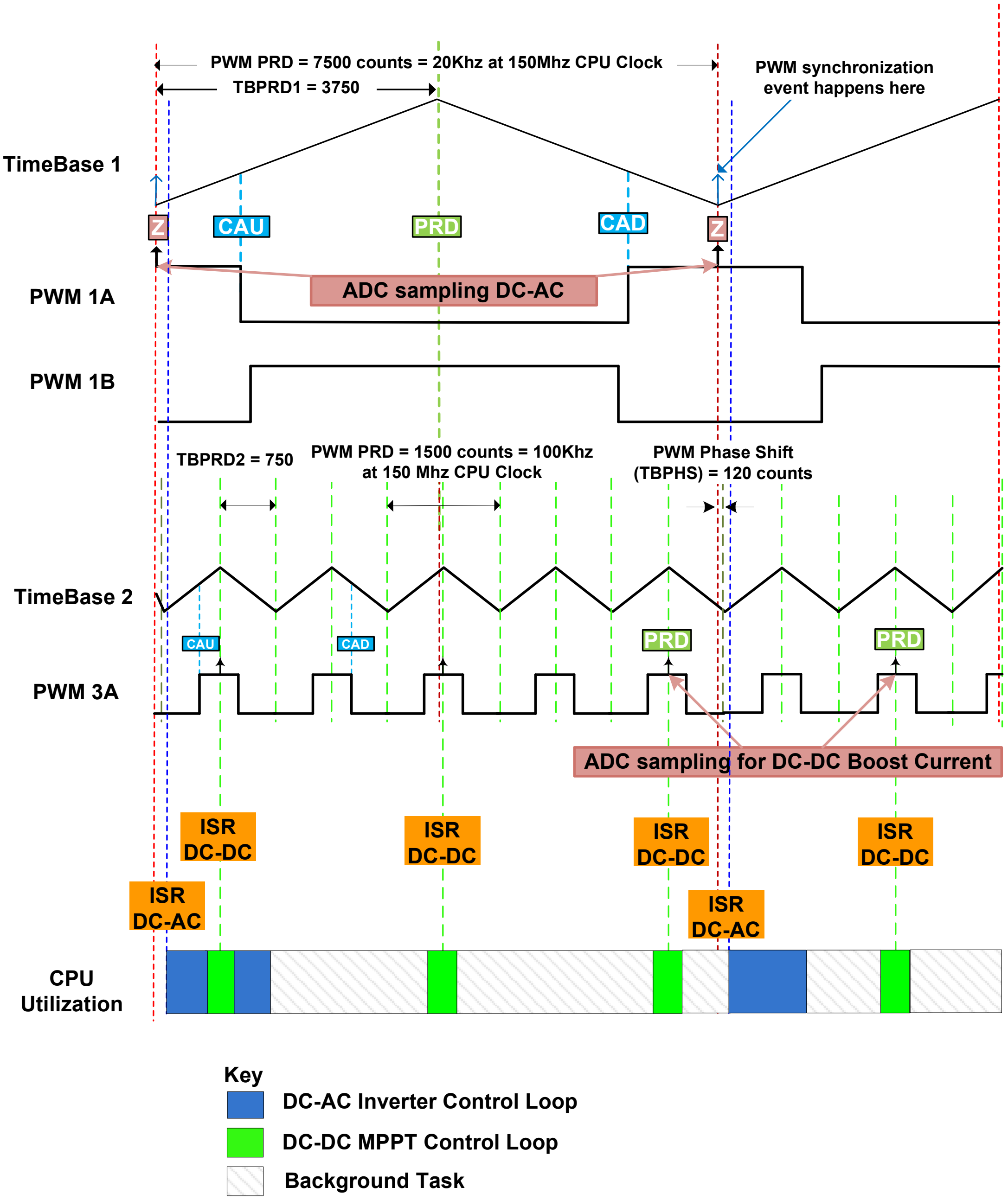

PWM PRD = 7500 counts = 20Khz at 150Mhz CPU Clock
TBPRD1 = 3750
PWM synchronization event happens here
TimeBase 1
Z
CAU
PRD
CAD
Z
PWM 1A
ADC sampling DC-AC
PWM 1B
TBPRD2 = 750
PWM PRD = 1500 counts = 100Khz at 150 Mhz CPU Clock
PWM Phase Shift (TBPHS) = 120 counts
TimeBase 2
CAU
CAD
PRD
PRD
PWM 3A
ADC sampling for DC-DC Boost Current
ISR DC-DC
ISR DC-DC
ISR DC-DC
ISR DC-DC
ISR DC-AC
ISR DC-AC
CPU Utilization
Key
DC-AC Inverter Control Loop
DC-DC MPPT Control Loop
Background Task

Fig 15 Timing diagram for Boost and Inverter Integration

The DC-DC control loop is given priority and the DC-DC ISR can be serviced from the DC-AC ISR through nesting of interrupts.

3 Hardware Platform Setup

3.1 HW Setup Instructions

Note: Do not power up the board before you have verified these settings!
Before starting the labs the user must make sure the following settings are correct. The user must ensure that these settings are valid on the board as depicted in

1) Make sure nothing is connected to the board, and no power is being supplied to the board.

2) Insert the controlCARD into the [Main]-J1 controlCARD connector if it is not already installed.

3) Do the following switch settings on the controlCARD:
    a. Populate the mini jumpers on the connectivity header in the MII section between A and B

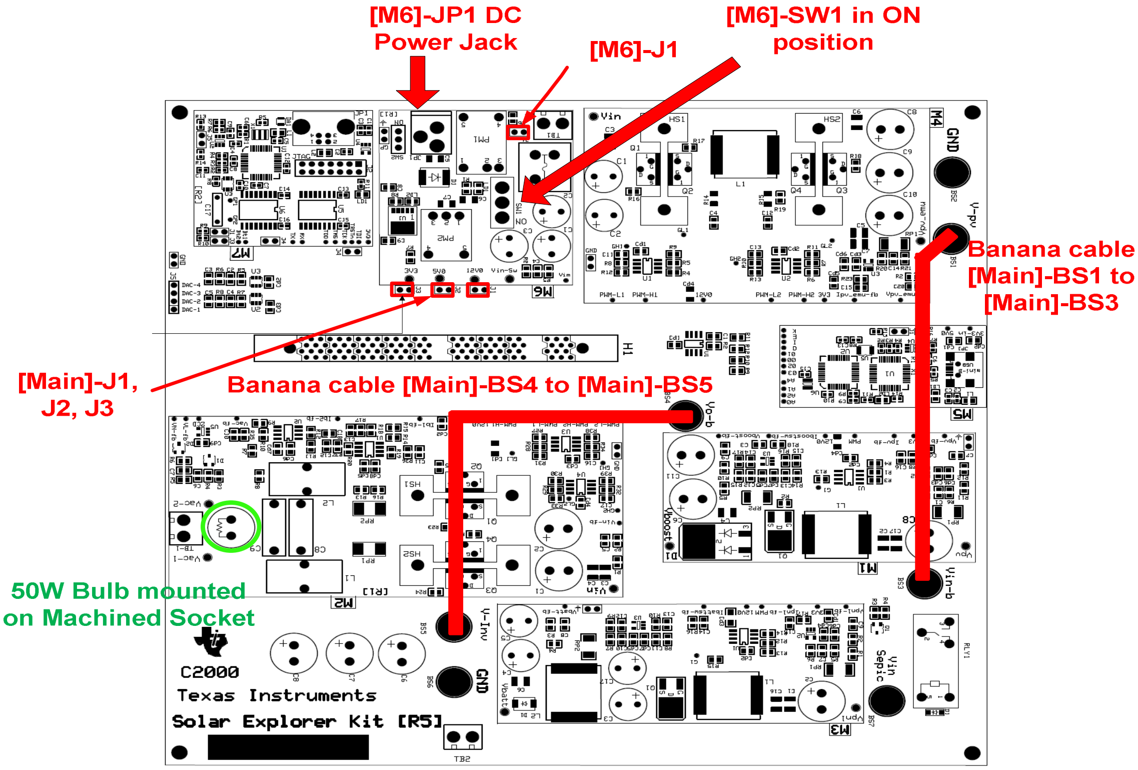


Fig 16 Hardware setting Solar Explorer Kit

4) The guide assumes that the TMS320F28027 microcontroller present in the M4 macro is pre-flashed with the panel emulator code with which the kit is shipped.

5) Connect a banana cable between [Main]-BS1 and [Main]-BS3

6) Connect a banana cable between [Main]-BS4 and [Main]-BS5

7) Verify the 50W bulb is in the bulb holder in the [M2] macro.

8) Make sure the [Main]-J1, [Main]-J2, [Main]-J3 and [M6]-J1 jumpers are populated. Verify [M6]-SW1 is in on position.

9) Connect a USB cable (mini to A Cable) from iso USB connector on the control card to the host computer. LD4 on the control card will light up indicating that the USB is powered.

10) Now connect the DC power supply shipped with the kit to [M6]-JP1, and turn on [M6]-SW2. [M5]-LD2 will start blinking indicating the PV emulator code is running on the emulator. Turn off [M6]-SW2.

3.2 Software Setup

Installing Code Composer and controlSUITE

1. If not already installed, please install Code Composer v4.x from the DVD included with the kit.

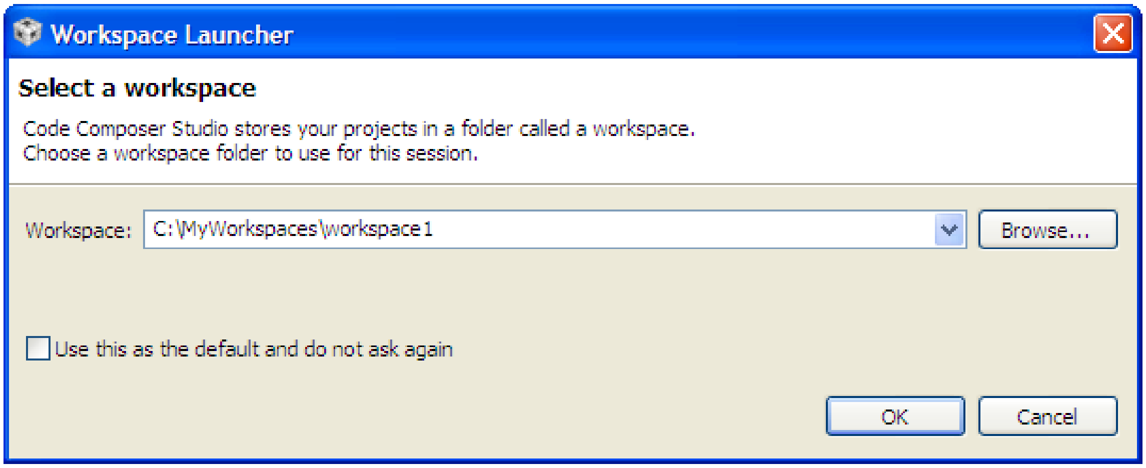
2. Go to http://www.ti.com/controlsuite and run the controlSUITE installer. Select to install the “SolarExplorer” software and allow the installer to also download all automatically checked software.

Setup Code Composer Studio to Work with SolarExplorer kit

3. Open “Code Composer Studio v4”.

4. Once Code Composer Studio opens, the workspace launcher may appear that would ask to select a workspace location,: (please note workspace is a location on the hard drive where all the user settings for the IDE i.e. which projects are open, what configuration is selected etc. are saved, this can be anywhere on the disk, the location mentioned below is just for reference. Also note that if this is not your first-time running Code Composer this dialog may not appear)

   - Click the “Browse…” button
   - Create the path below by making new folders as necessary.
   - “C:\Documents and Settings\My Documents\ CCSv4_workspaces\ProjectWorkspace”
   - Uncheck the box that says “Use this as the default and do not ask again”.
   - Click “OK”.


Workspace Launcher

Select a workspace

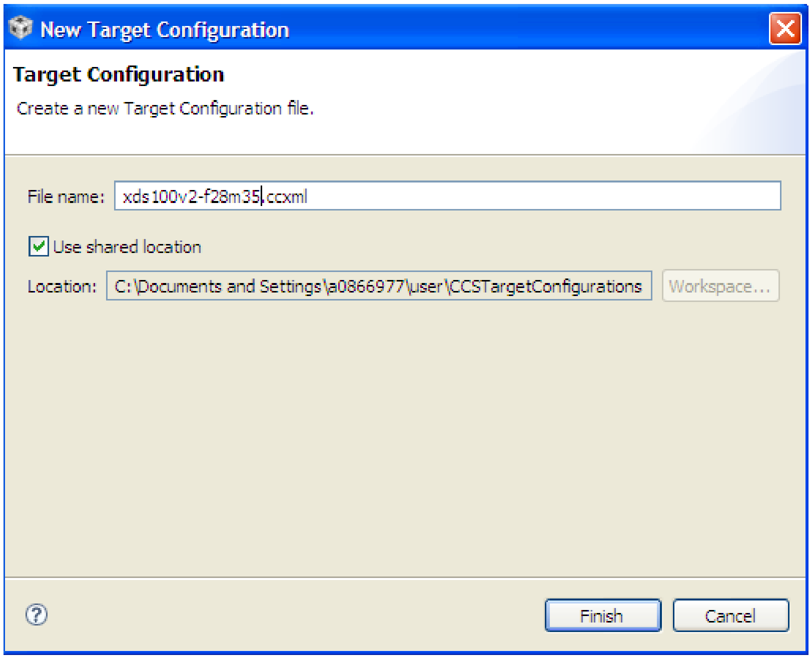
Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\MyWorkspaces\workspace1

□ Use this as the default and do not ask again

Fig 17 Open new workspace

5. Next we will configure Code Composer to know which MCU it will be connecting to. Click “Target -> New Target Configuration…”. Name the new configuration xds100v2-f28M35x.ccxml. Make sure that the “Use shared location” checkbox is checked and then click Finish.


New Target Configuration

Target Configuration

Create a new Target Configuration file.

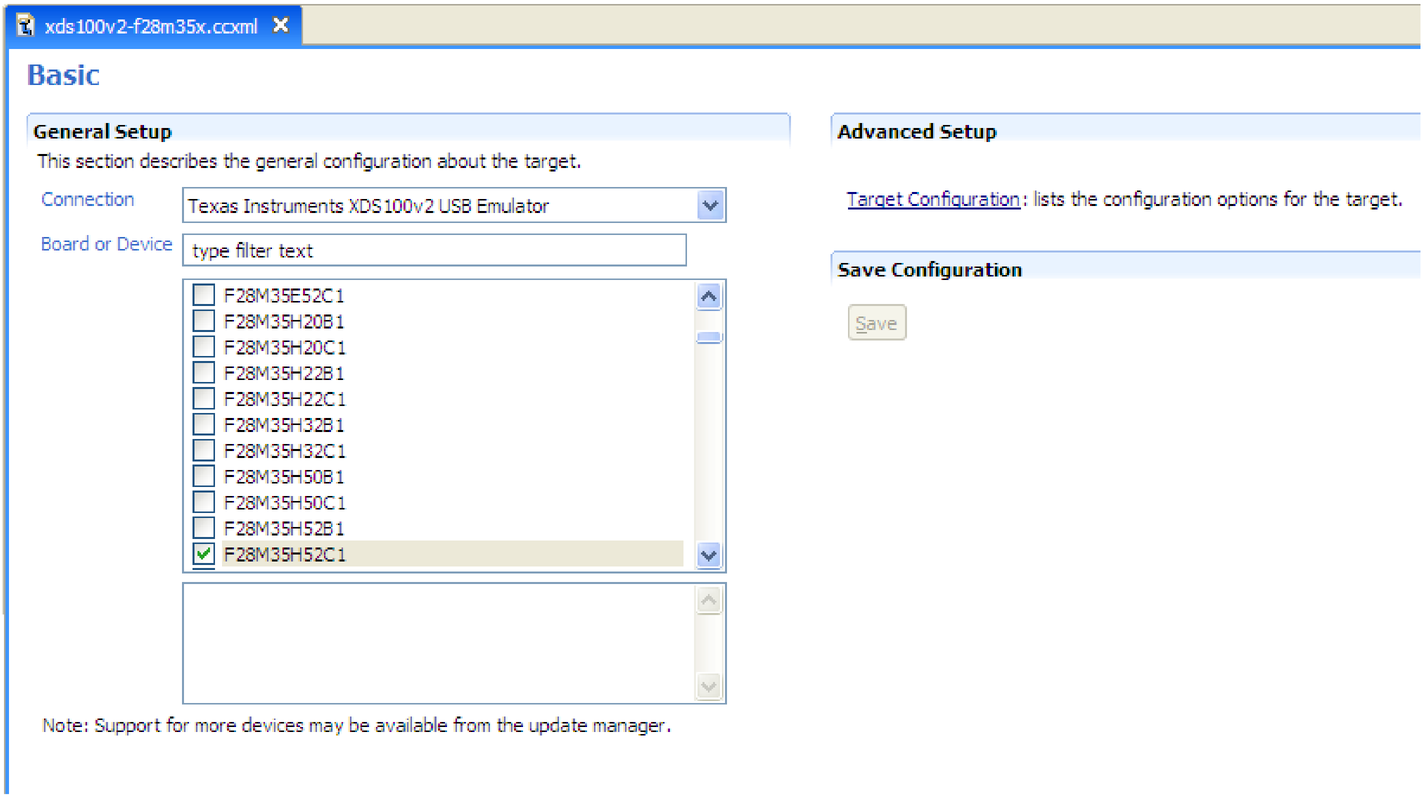
File name: xds100v2-f28m35.ccxml

☑ Use shared location

Location: C:\Documents and Settings\a0866977\user\CCSTargetConfigurations

Fig 18 New Target Configuration for XDS100 v2 and F28M35x MCU

6. A new tab will now appear. Select and enter the options as shown:
   - Connection – Texas Instruments XDS100v2 USB Emulator
   - Device – TMS320F28M35H52C1
   - Click Save
   - Close the xds100v2-f28m35x.ccxml tab



xds100v2-f28m35x.ccxml

Basic

General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS100v2 USB Emulator

Board or Device: type filter text

- ☐ F28M35E52C1
- ☐ F28M35H20B1
- ☐ F28M35H20C1
- ☐ F28M35H22B1
- ☐ F28M35H22C1
- ☐ F28M35H32B1
- ☐ F28M35H32C1
- ☐ F28M35H50B1
- ☐ F28M35H50C1
- ☐ F28M35H52B1
- ☑ F28M35H52C1

Note: Support for more devices may be available from the update manager.

Advanced Setup

Target Configuration: lists the configuration options for the target.

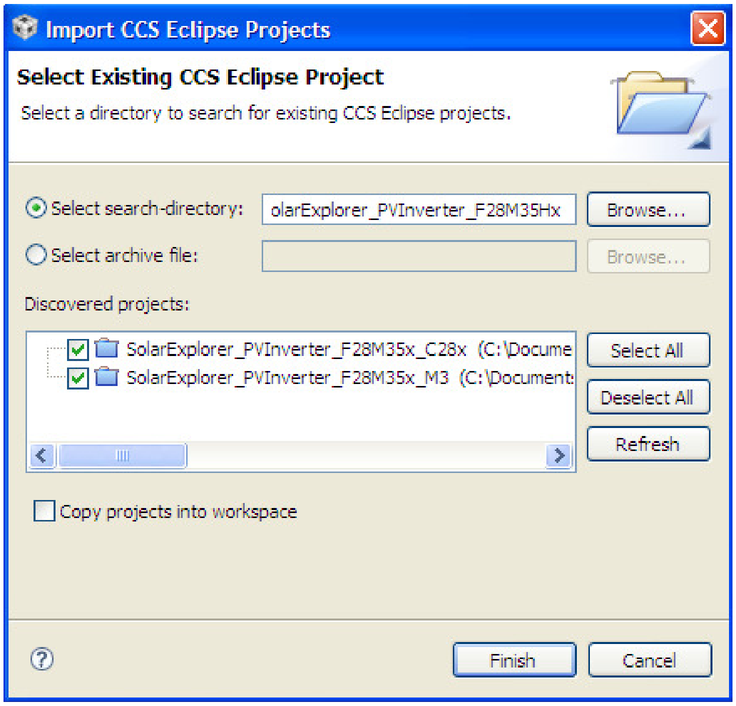
Save Configuration

Fig 19 Target Configuration Settings

7. Assuming this is your first time using Code Composer, the xds100v2-f28m35x configuration is now set as the default target configuration for Code Composer. Please check this by going to“View->Target Configurations”. In the “User Defined” section, right-click on the xds100v2-f28m35x.ccxml file and select “Set as Default”. This tab also allows you to reuse existing target configurations and link them to specific projects.

8. Add projects into your current workspace by clicking “Project->Import Existing CCS/CCE Eclipse Project”.

   ▪ Select the root directory of the SolarExplorer. This will be: “\controlSUITE\development_kits\SolarExplorer_v1.0\SolarExplorer_PVInverter_F28M35x”



Import CCS Eclipse Projects

Select Existing CCS Eclipse Project

Select a directory to search for existing CCS Eclipse projects.

● Select search-directory: olarExplorer_PVInverter_F28M35Hx

○ Select archive file:

Discovered projects:

☑ SolarExplorer_PVInverter_F28M35x_C28x (C:\Docume

☑ SolarExplorer_PVInverter_F28M35x_M3 (C:\Document:

□ Copy projects into workspace

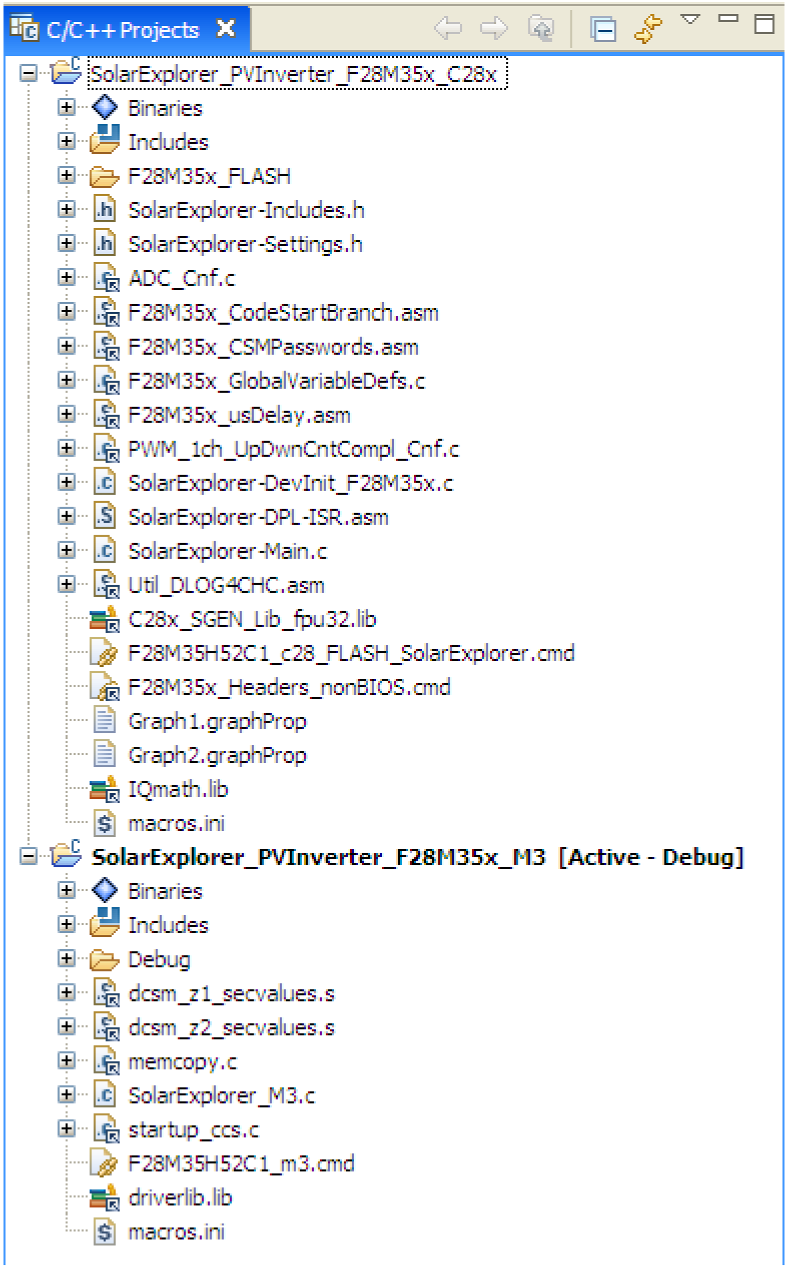
Fig 20 Adding F28M35x PV Inverter project to the workspace

- Two projects, one for the C28x and other for the M3 need to be imported into the workspace. Make sure both are checked and click ‘Finish’.

Configuring a Project

9. Expand the file structure of the project you would like to run from the C/C++ Projects tab. Right-click on this project’s name and select “Set as Active Project”, if this is not already the case.

10. Fig 21 shows the project in the CCSv4 C/C++ Project tab, it shows all the key files used in the project.



```
C/C++ Projects
SolarExplorer_PVInverter_F28M35x_C28x
    Binaries
    Includes
    F28M35x_FLASH
    SolarExplorer-Includes.h
    SolarExplorer-Settings.h
    ADC_Cnf.c
    F28M35x_CodeStartBranch.asm
    F28M35x_CSMPasswords.asm
    F28M35x_GlobalVariableDefs.c
    F28M35x_usDelay.asm
    PWM_1ch_UpDwnCntCompl_Cnf.c
    SolarExplorer-DevInit_F28M35x.c
    SolarExplorer-DPL-ISR.asm
    SolarExplorer-Main.c
    Util_DLOG4CHC.asm
    C28x_SGEN_Lib_fpu32.lib
    F28M35H52C1_c28_FLASH_SolarExplorer.cmd
    F28M35x_Headers_nonBIOS.cmd
    Graph1.graphProp
    Graph2.graphProp
    IQmath.lib
    macros.ini
SolarExplorer_PVInverter_F28M35x_M3  [Active - Debug]
    Binaries
    Includes
    Debug
    dcsm_z1_secvalues.s
    dcsm_z2_secvalues.s
    memcopy.c
    SolarExplorer_M3.c
    startup_ccs.c
    F28M35H52C1_m3.cmd
    driverlib.lib
    macros.ini
```

Fig 21 PV Inverter project in C/C++ tab

4 PV Inverter Project

This project is divided into simplified incremental builds to run smaller subsystems of increasing complexity until the target system is built up completely. This makes it easier to learn and get familiar with the board and the software. This approach is also good for debugging/testing boards. The various build options are shown below. To select a particular build option, set the appropriate value in the BUILD setting, found in the *SolarExplorer-settings.h.* Once the build option is selected, compile the complete project by selecting rebuild-all compiler option. Next chapters provides more details on how to run each of the build options.

Following are the build options supported on Solar Explorer kit for the C28x project.

Build 1: Illustrates closed current loop control of the Inverter Stage

Build 2: Illustrates MPPT & DC bus regulation along with closed current loop control of the Inverter Stage with a Bulb Load at the putput of the inverter, and locally generated sine reference.

Build 3: Illustrates the grid connection of the PV inverter along with MPPT , DC Bus regulation and closed loop current control of the inverter, a resistive load must be used (not shipped with the kit) for this build.

The M3 project is also in form of two incremental builds

Build 1: M3 project with SSI comms and IPC

Build 2: M3 project with SSI comms and IPC, with Ethernet (This build uses the Crosshairs tool for constructing the Interface on the host)

4.1 C28x BUILD = 1, M3 BUILD = 1

Inverter Current Control

Objective:
The objectives of this build are, (1) evaluate PWM and ADC software driver modules, (2) verify MOSFET gate driver circuit, voltage and current sensing circuit, (3) Closed loop current control for the Inverter power stage, (4) become familiar with the operation of Code Composer Studio (CCS). Under this build the system runs in closed-loop current mode control for the Inverter power stage and DCDC boost stage is not active. Steps required for building and running a project are explained next.

Overview:

The software in Build1 has been configured so that the user can quickly evaluate the PWM driver module, ADC drivers and closed loop current mode control for the inverter by viewing the related waveforms on a scope. User can also observe the effect of changing the current reference – as the closed loop control operation takes effect to track to the set current reference command. The user can adjust the current reference command (inv_Iset) from CCS watch window. The user can also evaluate the ADC driver module by viewing the ADC sampled data in the watch window. The *Sine Analyzer* block calculates the RMS voltage and frequency of the output voltage. It also provides additional information related to the zero crossing points, average value and the cycle information.

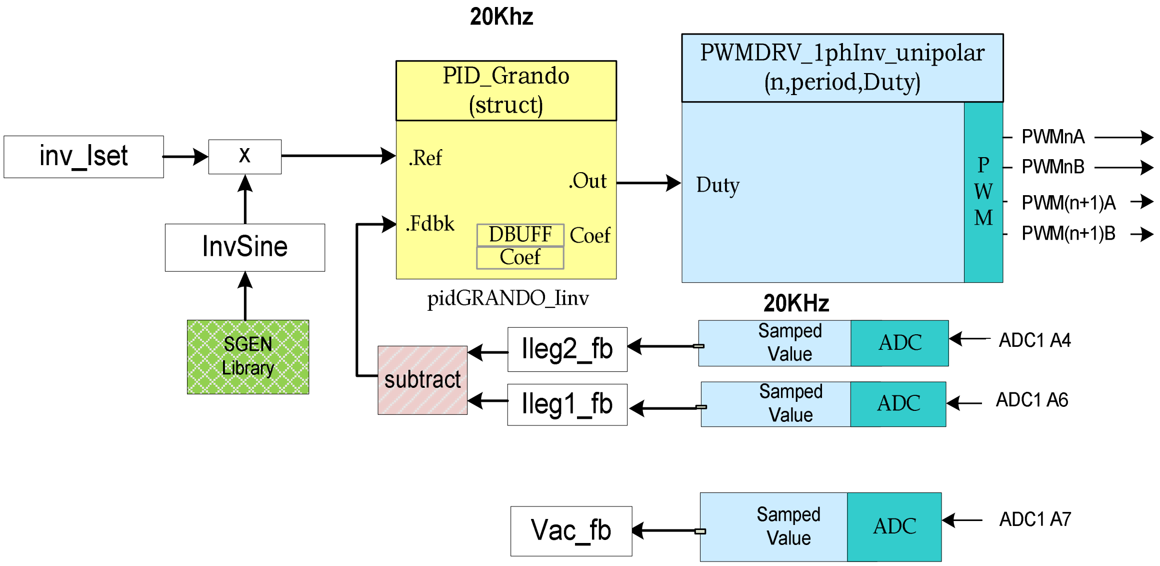


Fig 22 Build 1 Control Software Diagram

The PWM and ADC driver macro instantiations are executed inside the *SolarExplorer-Main.c* file. Fig 22 shows the software blocks used in this build. The quantities that are sensed and fed back to the MCU include - (1) the two inverter leg currents (Ileg1_fb, Ileg2_fb) sampled at appropriate time instance and the difference saved as inv_meas_cur_diff_inst, (2) Output AC voltage (Vac_FB) and (3) the DC bus voltage (VboostRead), not used in this build.

PWM 1ph Inverter Unipolar Modulation: Four PWM's are needed to drive a full bridge inverter. PWM_1phInv_unipolar_CNF(n,period,deadband_rising,deadband falling) macro, found in the driver library

controlSUITE\libs\app_libs\drivers\v1.0\F28M35x

is used to configure the PWM1 and 2 for unipolar modulation where PWM1A and PWM1B drive one leg of the inverter switches while ePWM2A and ePWM2B drive the other leg. 20Khz switching frequency is used for the inverter. Fig 23 illustrates the modulation scheme in detail.

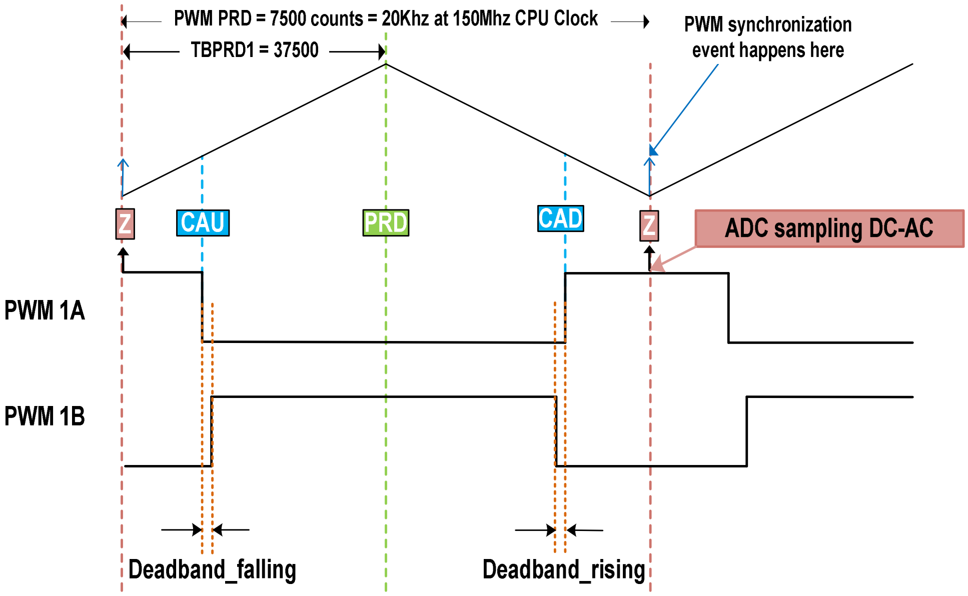

PWM PRD = 7500 counts = 20Khz at 150Mhz CPU Clock
TBPRD1 = 37500
PWM synchronization event happens here
Z
CAU
PRD
CAD
Z
ADC sampling DC-AC
PWM 1A
PWM 1B
Deadband_falling
Deadband_rising

Fig 23 PWM generation and ADC sampling for inverter control

As switched current are sensed for the inverter current sampling is done at the midpoint of the PWM ON pulse. The ADC module is configured to use SOCA of ePWM1 such that, SOCA is triggered at TBCNT1 = ZERO event. All other conversions such as AC voltage and DC Bus sampling is also completed using this SOCA trigger.

Fig 24 shows how unipolar modulation is used to switch the four PWM's in positive and negative half cycle of the grid.

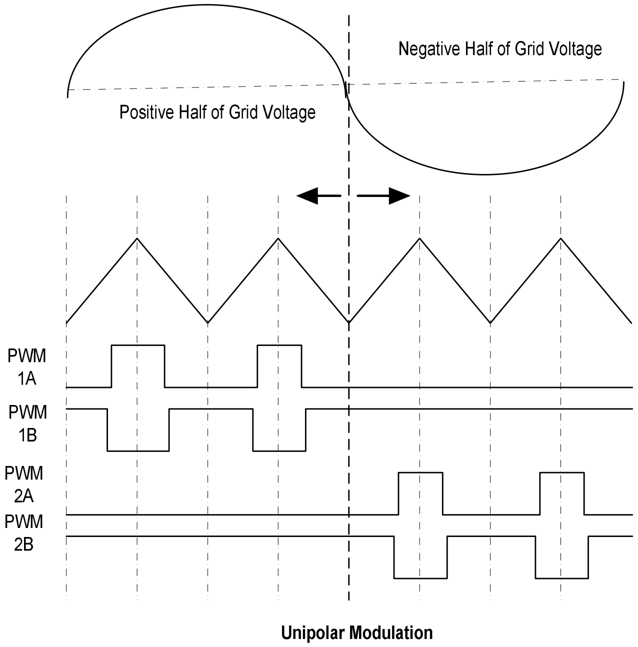


Fig 24 PWM control of inverter using unipolar modulation technique

The ADC gives a 12 bit result, as the inverter quantities are AC a few lines of code in the ISR implement the level shifting and offset removal of AC line half cycle (positive & negative half cycles) current values obtained from Ileg1-fb and Ileg2-fb and the Vac_FB signal. The inverter current feedback is then given to the pidGRANDO_Iinv and the output is fed as duty to the PWMDRV macro. Sign of the duty command is used to identify the output switching cycle (positive or negative cycle) and configures the appropriate PWM output to switch the inverter leg (ePWM1 or ePWM2) to switch either leg1 or leg2 determining the AC line half cycle at the output.

SGEN Module

The signal generator module from SGEN library (refer to SGEN library documentation at \controlSUITE\libs\dsp\SGEN) is used for SINE generation. The frequency of the generated signal is reciprocal of the time it takes for successive overflow of modulo counter, which in turn commensurate with the step value added to the counter. Thus by changing the step value, one can precisely control the frequency. The step value is not directly commanded to vary the frequency, instead the modulation of frequency is performed using the normalized variable “freq” which is normalized to the maximum frequency. The maximum required frequency is predetermined based on the application requirement and it set by initializing the “step_max” input. Thus, the normalized variable “freq” allows the user to control the frequency of the signal between 0 to maximum frequency.

Code snippet below shows how to initialize sgen module to generate 50Hz sine wave.

```
/* Signal Generator module initialisation           */
      sgen.offset=0;
      sgen.gain=0x7fff;        /* gain=1 in Q15      */
      sgen.freq=0x14F8CF92;    /* freq = (Required Freq/Max Freq)*2^31 */
                               /* = (50/305.17)*2^31 = 0x14f8cf92 */
      sgen.step_max=0x3E7FB26;/* Max Freq= (step_max * sampling freq)/2^32 */
                               /* =(0x3E7FB26*20k)/2^32 = 305.17 */
      sgen.phase=0x80000000;   /* Phase=(required Phase)/180 in Q31 format   */
                               /*      =  (+90/180) in Q31 = 8000h           */
```

PID GRANDO Module

The PID_grando module implements a basic summing junction and PID control law with the following features (refer to solar library documentation at …\controlSUITE\libs\app_libs\solar):

- Programmable output saturation
- Independent reference weighting on proportional path
- Independent reference weighting on derivative path
- Anti-windup integrator reset
- Programmable derivative filter

All input, output and internal data is in single precision floating point format.

Procedure

Assuming the hardware and setup as described in the
3.1 HW Setup Instructions and *3.2 Software Setup* are done,

1. Turn on [M6]-SW2. [M6]-LD1, [M6]-LD2 and [M5]-LD1 will glow green indicating bias power is being generated. [M5]-LD2 will start blinking indicating the PV emulator code is running on the emulator.

2. Now in the C/C++ project tab right click on the M3 project name and make it the active project.

3. Open SolarExplorer_M3.c and verify that the build level for the M3 project is 1.

4. Also notice the IPC message structure definition, two packets are defined one which is used to get diagnostic information from the C28x to the M3, this packet is thus called CtoM_Message and is pointed to the start of the C28x to M3 message RAM(C28x Writable M3 Readable). The other packet consists of commands from the M3 to C28x, like to start the inverter and stop the inverter are is stored in the MtoC_Message which is pointed to the start of the M3 to C28x Message RAM(C28x Readable M3 Writable).

```
#define M3_MTOC_PASSMSG 0x2007F800
#define M3_CTOM_PASSMSG 0x2007F000

// Variables that need to shared with C28x and M3
// m3 owned memory region
struct MtoC_Message {
      int InvStart;
      int InvStop;
      float LightCommand;
};

volatile struct MtoC_Message *MtoC_Message1_Ptr;

//C28x owned memory region
struct CtoM_Message {
      int InverterStatus;
      float PanelPower;
      float PanelPower_Theoretical;
      float PanelVoltage;
      float PanelCurrent;
      float BoostVoltage;
};

volatile struct CtoM_Message  *CtoM_Message1_Ptr;
```

……………………..

```
MtoC_Message1_Ptr= (void *)M3_MTOC_PASSMSG;
CtoM_Message1_Ptr= (void *)M3_CTOM_PASSMSG;
```

5. As the M3 is the master of the system, it has control over all the IO’s at boot time, it then needs to give access to the C28x. Inspect the SolarExplorer_m3.c to find these assignment of the IO’s to the C28x.

```
// Give C28 control of GPIOs it needs
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
```

6. Also as the M3 is the master it instructs the C28x to boot from flash. Locate the following IPC command in the SolarExplorer_M3.c which does this.

```
IPCMtoCBootControlSystem(CBROM_MTOC_BOOTMODE_BOOT_FROM_FLASH);
```

7. Click Project → “Rebuild All” button and watch the tools run in the build window.

8. Now right click on the C28x project and and select it as the ‘Active Project’. Now Open and inspect *SolarExplorer-DevInit_F28M35x.c* by double clicking on the filename in the project window. DeviceInit() function enables the peripheral clocks and starts the ADC, it also configured the GPIO Mux for the C28x however for this muxing scheme to be effective the IO’s must be assigned by the M3 master to the C28x.

9. Open and inspect *SolarExplorer-Main.c*. Notice the call made to *DeviceInit()* function and other variable initialization. Also notice code for different incremental build options, the SGEN initialization, PID initialization, SineAnalyzer initialization etc.

10. Locate and inspect the code in the main file under initialization code. Observe macros used for EPWM module initialization (*PWM_1phInv_unipolar_CNF)* and ADC module initialization (*ADC_CNF)* blocks. This is common for all incremental builds.

11. Also locate and inspect the following code in the main file under initialization code. This is where the ADC channels for different feedback signals are assigned and the start-of-conversion triggers are programmed. This is also common for all incremental builds. As Boost and Inverter are run at different frequencies and from different PWM, the PWM synchronization feature is used to avoid any ADC conflicts as shown in Fig 15. Note on Concerto the ADC subsystem triggers are different from the peripheral triggers, and for the peripheral triggers to trigger ADC SOC proper selection for the ADC subsystem trigger must be done. Concerti has 2 ADC's , the sense signals are mapped to both, the code below illustrates how the ADC mapping is done.

```
#define Ileg1_fb        Adc1Result.ADCRESULT0
#define Ileg2_fb        Adc1Result.ADCRESULT1
#define Ipv_FB          Adc1Result.ADCRESULT2
#define Vpv_FB          Adc1Result.ADCRESULT3
#define Vac_FB          Adc1Result.ADCRESULT4
#define Vboost_FB       Adc1Result.ADCRESULT5

#define Iboostsw_FB     Adc2Result.ADCRESULT0
#define LIGHT_FB        Adc2Result.ADCRESULT1

// ADC Channel Selection
ChSel1[0] = 4;    //ADC1 SOC 0 -> Ileg1-fb   -> ADC1-A4 on F28M3x Ccard
ChSel1[1] = 6;    //ADC1 SOC 1 -> Ileg2-fb   -> ADC1-A6 on F28M3x Ccard
ChSel1[2] = 0;    //ADC1 SOC 2 -> Ipv-fb     -> ADC1-A0 on F28M3x Ccard
ChSel1[3] = 8;    //ADC1 SOC 3 -> Vpv-fb     -> ADC1-B0 on F28M3x Ccard
ChSel1[4] = 7;    //ADC1 SOC 4 -> Vac-fb     -> ADC1-A7 on F28M3x Ccard
ChSel1[5] = 2;    //ADC1 SOC 5 -> Vboost-fb  -> ADC1-A2 on F28M3x Ccard
// ADC Trigger Selection
TrigSel1[0] = 6;  // ADC1 SOC 0 -> Ileg1-fb
TrigSel1[1] = 6;  // ADC1 SOC 1 -> Ileg2-fb
TrigSel1[2] = 5;  // ADC1 SOC 2 -> Ipv-fb
TrigSel1[3] = 5;  // ADC1 SOC 3 -> Vpv-fb
TrigSel1[4] = 6;  // ADC1 SOC 4 -> Vac-fb
TrigSel1[5] = 5;  // ADC1 SOC 5 -> Vboost-fb

// ADC Sub system Trigger 1 be used for the Boost Control
// ADC Sub system Trigger 2 be used for the inverter control
// Configure the CIB triggers (Note this needs to be done before the ADC SOC
//trigger selection)
EALLOW;
// EPWM3SOCA to TRIGGER 1 of the analog subsystem
AnalogSysctrlRegs.TRIG1SEL.all = ADCTRIG_EPWM3_SOCA;
// EPWM1SOCA to Trigger 2 of the analog subsystem
AnalogSysctrlRegs.TRIG2SEL.all = ADCTRIG_EPWM1_SOCA;
EDIS;

ADC_CNF(1, ChSel1, TrigSel1, ACQPS1);

// ADC Channel Selection
ChSel2[0] = 6;    //ADC2 SOC 0 -> Iboostsw-fb-> ADC2-A6 on F28M35x Ccard
```

```
    ChSel2[1] = 0;    //ADC2 SOC 1 -> Vlight-fb  -> ADC2-A0 on F28M35x Ccard

    // ADC Trigger Selection
    TrigSel2[0] = 5;  // ADC2 SOC 0 -> Iboostsw-fb
    TrigSel2[1] = 6;  // ADC2 SOC 2 -> Vlight-fb

    ADC_CNF(2, ChSel2, TrigSel2, ACQPS2);
```

12. Navigate to the portion of the code which has inverter ISR code (Inv_ISR()) inside *SolarExplorer-Main.c*. Inside the ISR, code for multiple build options can be found. Notice that for Build 1 InvSine parameter is calculated by sgen block. Macro used for driving PWM outputs is *PWMDRV_1phInv_unipolar.* Open this macro and observe how the PWM outputs are driven for the inverter stage.

Build and Load the Project

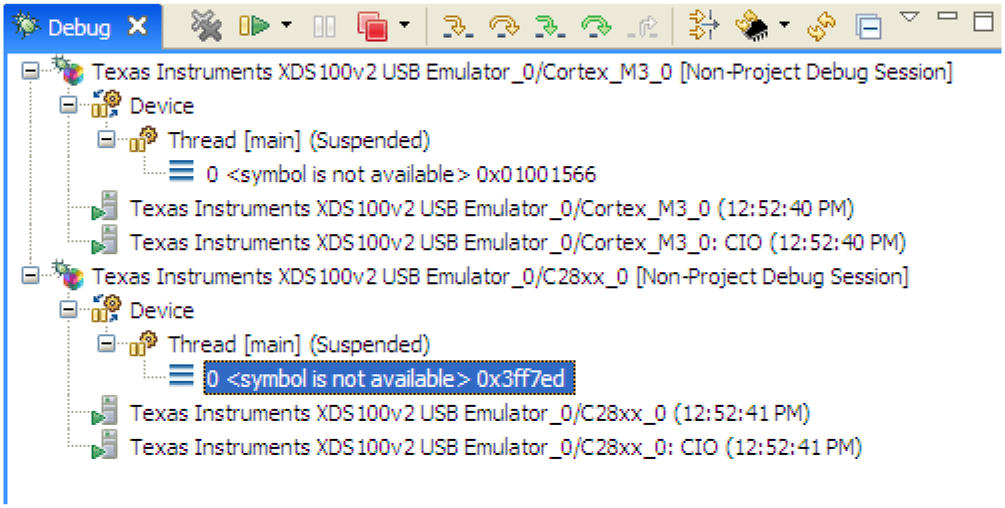
13. Select the incremental build option as 1 in the *SolarExplorer-Settings.h* file.

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

14. Click Project → “Rebuild All” button and watch the tools run in the build window.

15. Click View -> Target Configuration, under user defined right click on xdsv100v2-f28m35x.ccxml and click lauch selected configuration. CCS will now switch to the debug perspective and two controllers will appear in the debug window.

16. Now in the Debug window right click on the M3 and click connect Target. Do same for the C28x. The debug view should look like below. As M3 is the master, and C28 the slave, it is generally better because connecting the M3 first allows CCS to halt the debugger and prevents the M3 from going ahead and running whatever code may be flashed into the M3 flash.



```
Texas Instruments XDS100v2 USB Emulator_0/Cortex_M3_0 [Non-Project Debug Session]
    Device
        Thread [main] (Suspended)
            0 <symbol is not available> 0x01001566
    Texas Instruments XDS100v2 USB Emulator_0/Cortex_M3_0 (12:52:40 PM)
    Texas Instruments XDS100v2 USB Emulator_0/Cortex_M3_0: CIO (12:52:40 PM)
Texas Instruments XDS100v2 USB Emulator_0/C28xx_0 [Non-Project Debug Session]
    Device
        Thread [main] (Suspended)
            0 <symbol is not available> 0x3ff7ed
    Texas Instruments XDS100v2 USB Emulator_0/C28xx_0 (12:52:41 PM)
    Texas Instruments XDS100v2 USB Emulator_0/C28xx_0: CIO (12:52:41 PM)
```

Fig 25 Debug View F28M35x

17. Now select the M3 tab in the debug window and now click Target-> Load Program, click Browse Project and select the M3 project and click ok. This will flash the M3 code on concerto.

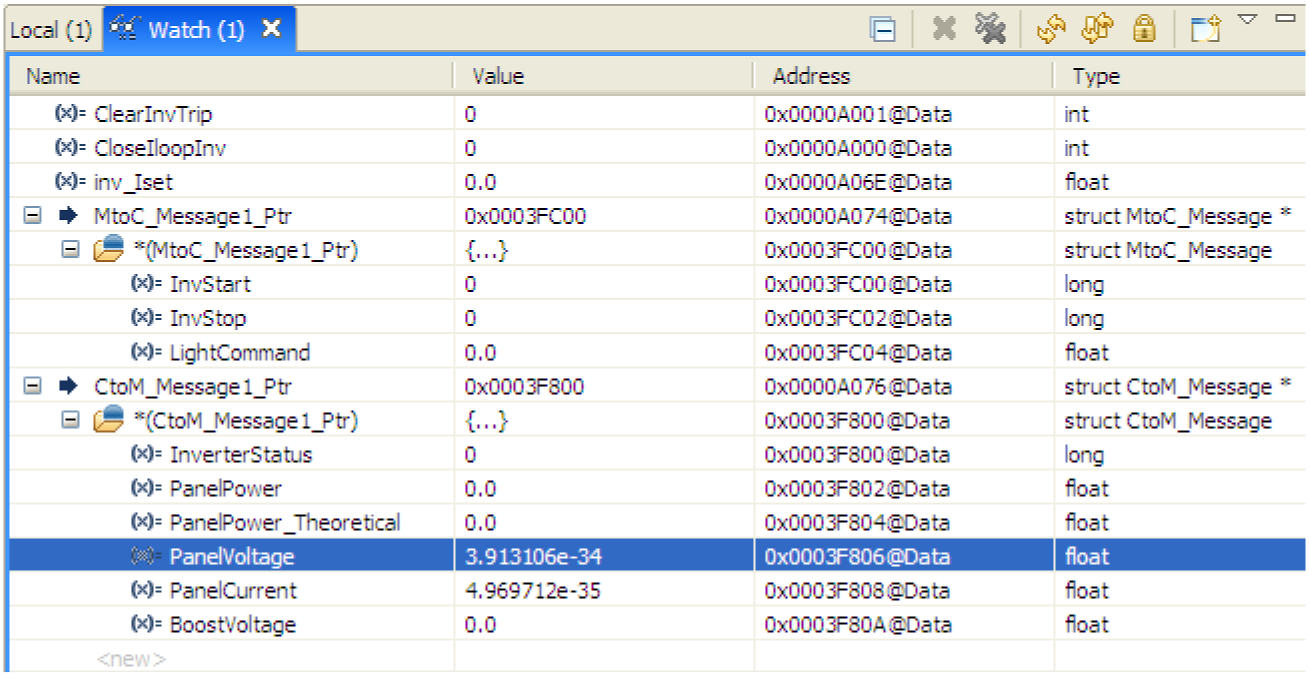
18. Then select the C28x tab and repeat the step above with the C28x project. This will flash the C28x project.

19. Once flashed, reset the C28x by clicking .

20. Now select the M3 in the debug tab and reset the M3 by clicking , then restart the M3 by . You will now be at the start of the M3 Main(), the project is now ready to be debugged.

Debug Environment Windows

21. 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 Fig 26.



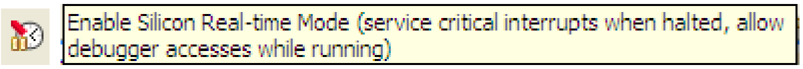
Name	Value	Address	Type
ClearInvTrip	0	0x0000A001@Data	int
CloseIloopInv	0	0x0000A000@Data	int
inv_Iset	0.0	0x0000A06E@Data	float
⊟ MtoC_Message1_Ptr	0x0003FC00	0x0000A074@Data	struct MtoC_Message *
⊟ *(MtoC_Message1_Ptr)	{...}	0x0003FC00@Data	struct MtoC_Message
InvStart	0	0x0003FC00@Data	long
InvStop	0	0x0003FC02@Data	long
LightCommand	0.0	0x0003FC04@Data	float
⊟ CtoM_Message1_Ptr	0x0003F800	0x0000A076@Data	struct CtoM_Message *
⊟ *(CtoM_Message1_Ptr)	{...}	0x0003F800@Data	struct CtoM_Message
InverterStatus	0	0x0003F800@Data	long
PanelPower	0.0	0x0003F802@Data	float
PanelPower_Theoretical	0.0	0x0003F804@Data	float
PanelVoltage	3.913106e-34	0x0003F806@Data	float
PanelCurrent	4.969712e-35	0x0003F808@Data	float
BoostVoltage	0.0	0x0003F80A@Data	float
<new>			

Fig 26 Snap shot of CCS Watch Window 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. However note this feature is only available on the C28x and not on M3.

22. From the step 20, the C28x must be in the reset and M3 code should be at the beginning of the main function, now run the M3 code by clicking . With M3 running select the C28x tab and put a breakpoint at the beginning of the C28x Main(), and hit run . The C28x code will now stop at the breakpoint. The user can now enable real time mode on the C28x by clicking button



Enable Silicon Real-time Mode (service critical interrupts when halted, allow debugger accesses while running) .

23. 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.
24. Click on Continuous Refresh buttons for the watch view.

Run the Code

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

26. Now select the M3 tab again and halt the processor . In the watch view, check the value of MtoC_Message1_Ptr->LightCommand, it will read 0.0, this is the light command for the panel emulator. As DC-DC Boost stage is not switched in this level it passes the input of the panel voltage with a diode drop. Thus Gui_Vboost which is the DC bus for the inverter as the DC-DC boost output is connected to the DC-AC inverter, should read close to the panel voltage unless some parasitic voltage is already present on the capacitors. Change this value to 0.3, and run and then halt the M3 for the value to take effect. Observe the CtoM_Message1_Ptr->PanelPowerTheoretical and CtoM_Mesage1_Ptr->PanelVoltage. For 0.3 light Command they will read 10.8 W and 8 Volts respectively.

27. Now select the C28X tab, as the processor has real time mode no need to halt and re run the C28x, the user can change the *inv_Iset in watch window* and set it to 0.2. This variable sets the current command for the inverter closed loop operation.

28. Add *ClearInvTrip, CloseIloopInv* flag to the watch window. Write “1” to both these variables, to turn on the inverter and to enable inverter operation in closed loop. ClearInvTrip will return back to ‘0’ value once the command to clear the initial trip is acknowledged.

29. Observe inverter output on the oscilloscope by probing the test point on the board using a scope probe at (Vac-FB, PWM-H1 and PWM-H2 on [M2]),

30. Fig 27 shows a captured plot of these signals along with the output voltage Vac-FB. Note that one PWMs are modulated in unipolar fashion, with one set H/L PWMs switching in one half cycle and other set in the other half cycle.

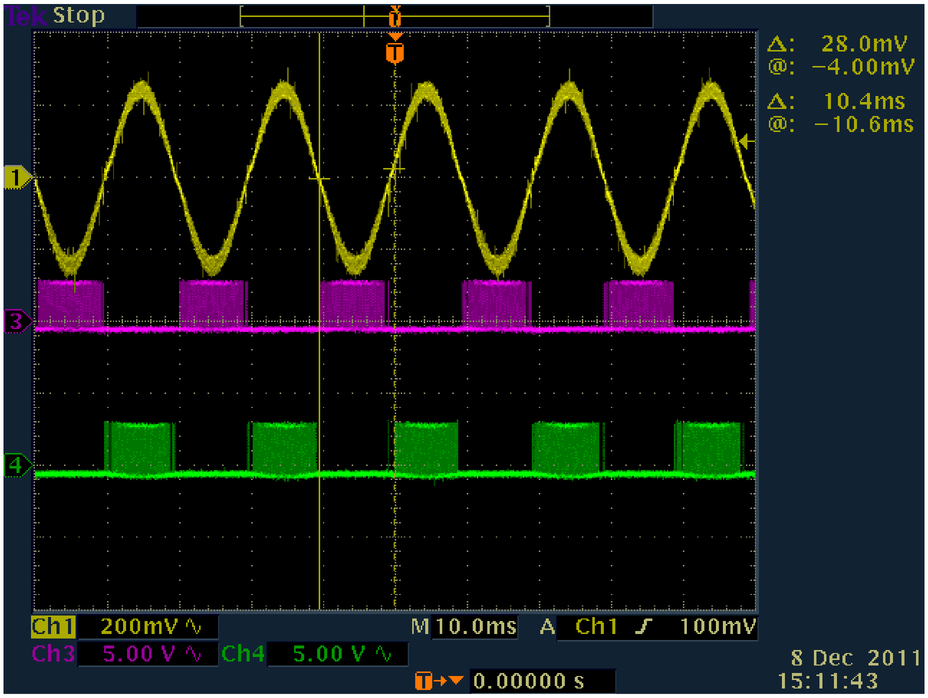


Fig 27 Unipolar Modulation for full bridge inverter, Ch1: Vac-FB, Ch3: PWM-H1, Ch4: PWM-H2

31. Monitor inverter leg currents, marked as Ileg1-fb and Ileg2-fb on the board, on oscilloscope to ensure that the current feedback is fed to the controller as expected. Fig 28 shows oscilloscope capture of these waveforms.

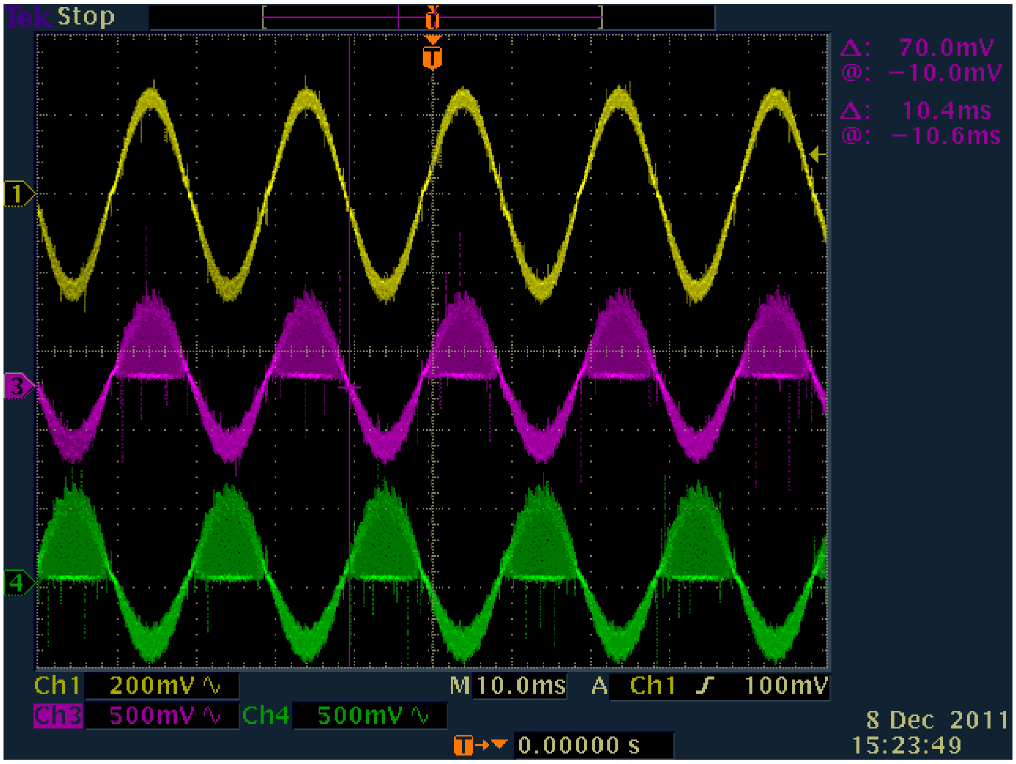


Fig 28 Inverter Current Measurement

32. Now connect the scope probes to the PWM-DAC outputs to verify closed loop operation of the inverter, [Main]-J5, PWMDAC1,2 are connected to plot the instantaneous reference and measured current respectively.

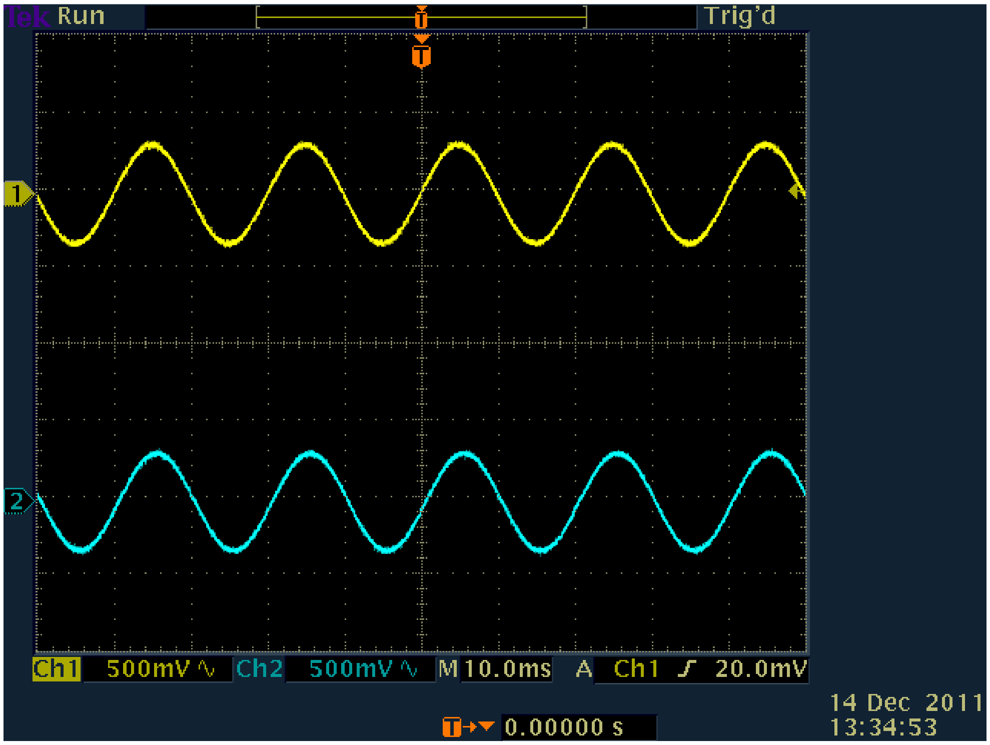


Fig 29 Closed Loop Inverter Operation, Ch1 : Current Reference , Ch2: Measured Current

33. User can now slowly, in incremental steps of 0.05, change inv_Iset value from 0.2 to 0.4 in the watch window. User can observe that the closed current loop operation is followed..

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

34. User can also vary the load resistance and see the effect on the output voltage. In Build 1, inverter is operating in closed loop current mode. Hence, current loop will always try to track inverter output current to the set reference command (inv_Iset). As the load connected is resistive – output voltage is expected to vary such that current is tracked to set reference.

35. Write 0 to inv_Iset in the watch window from the C28x side, now move to the M3 side and write a zero to LightCommand (you will have to halt and run the M3 for any changes/updates in the watch window to occur). Now halt the M3. Go to the C28x and halt the processor and take it out of real-time mode by clicking on . Finally reset the MCU .

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

End of Exercise

4.2 C28x BUILD = 2, M3 BUILD = 1

Inverter Current Control with DC Bus Regulation & DC-DC Boost MPPT Control

Objective:
The objectives of this build are, (1) Understand closed loop control of the Boost stage (2) Check MPP tracking under varying panel conditions (3) Verify full system operation with DCDC Boost and inverter working in tandem with command passed from the M3 side through the message RAM's. Under this build, the system runs in complete closed-loop mode for the Inverter power stage and DCDC boost stage. System is not connected to the grid. Steps required for building and running a project are explained next.

Overview:
Build 2 focuses on closed loop control of the inverter and boost stages along with MPP tracking. For inverter control, in addition to closed loop current control in Build 1, outer voltage loop is added. Voltage loop in the inverter stage is responsible for setting the reference command for the current loop. Current reference reflects the amount of power that can be transferred to the load depending on the instantaneous DC bus voltage. Details of the control mechanism can be found under section 2.3 Control Description of this document. DC bus reference voltage (Vdcref), for the voltage loop, is set to 30V for this Build.

Inverter stage feedback quantities are same as Build 1 – in addition, boost output voltage (Vboost_FB) is used for the outer voltage loop of the inverter as shown in Fig 31. PWM configuration for the inverter remains the same as in Build 1 for the inverter stage. For the Boost stage, PWM signal is generated at a frequency of 100 kHz, and is configured to operate in up-down count mode, PWMDRV_1ch_UpDwnCntCompl module from the digital power library is used to configure the Boost PWM, Fig 30.

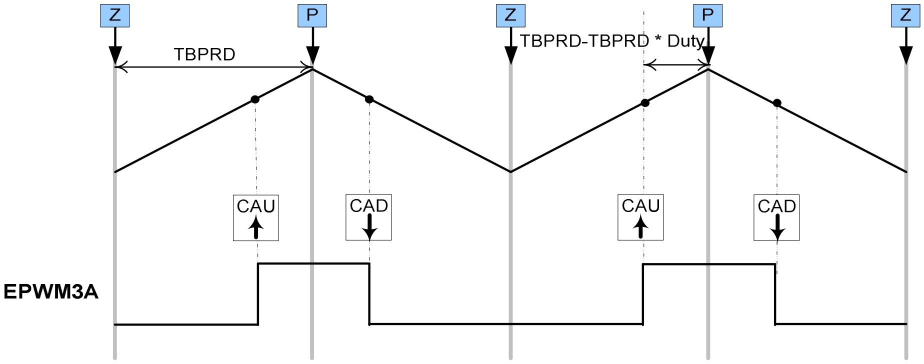


Fig 30 DC-DC Boost PWM using PWMDRV_1ch_UpDwnCntCompl Configuration

Boost switch current is sampled at the midpoint of the PWM ON pulse i.e. TBCTR=TBPRD. The panel current and voltage are sensed at zero event, this uniformly distributes the ADC loading. The ISR trigger frequency is half that of the PWM switching frequency as shown, this is done because the PV panel conditions don't change very frequently. The PWM time bases for DC-DC Boost and DC-AC inverter are synchronized to avoid conflict on the A/D converter. The DC-DC Boost loop runs as 50Khz and the inverter loop runs at 20Khz. Note that Inverter ISR is made interruptible by the DC-DC Boost interrupt and the code jumps to the Boost ISR and then returns back to the inverter ISR, as shown in Fig 15.

Sampled values of panel voltage and current (Ipv, Vpv) are passed on to MATH_EMAVG block, a software module that performs exponential moving average over data stored. Averaged values of Ipv, Vpv are then passed on to MPPT algorithm. This version of the software uses Incremental Conductance algorithm for maximum power point tracking. Both Incremental conductance(mppt_incc.h) and Perturb and Observe(mppt_pno.h) are available are library components in controlSUITE under solar library (…\controlSUITE\libs\app_libs\solar) directory. User can choose to use either of these algorithms to evaluate performance and measure MPPT efficiency. MPPT algorithm generates reference voltage command for the outer voltage loop of the boost stage.

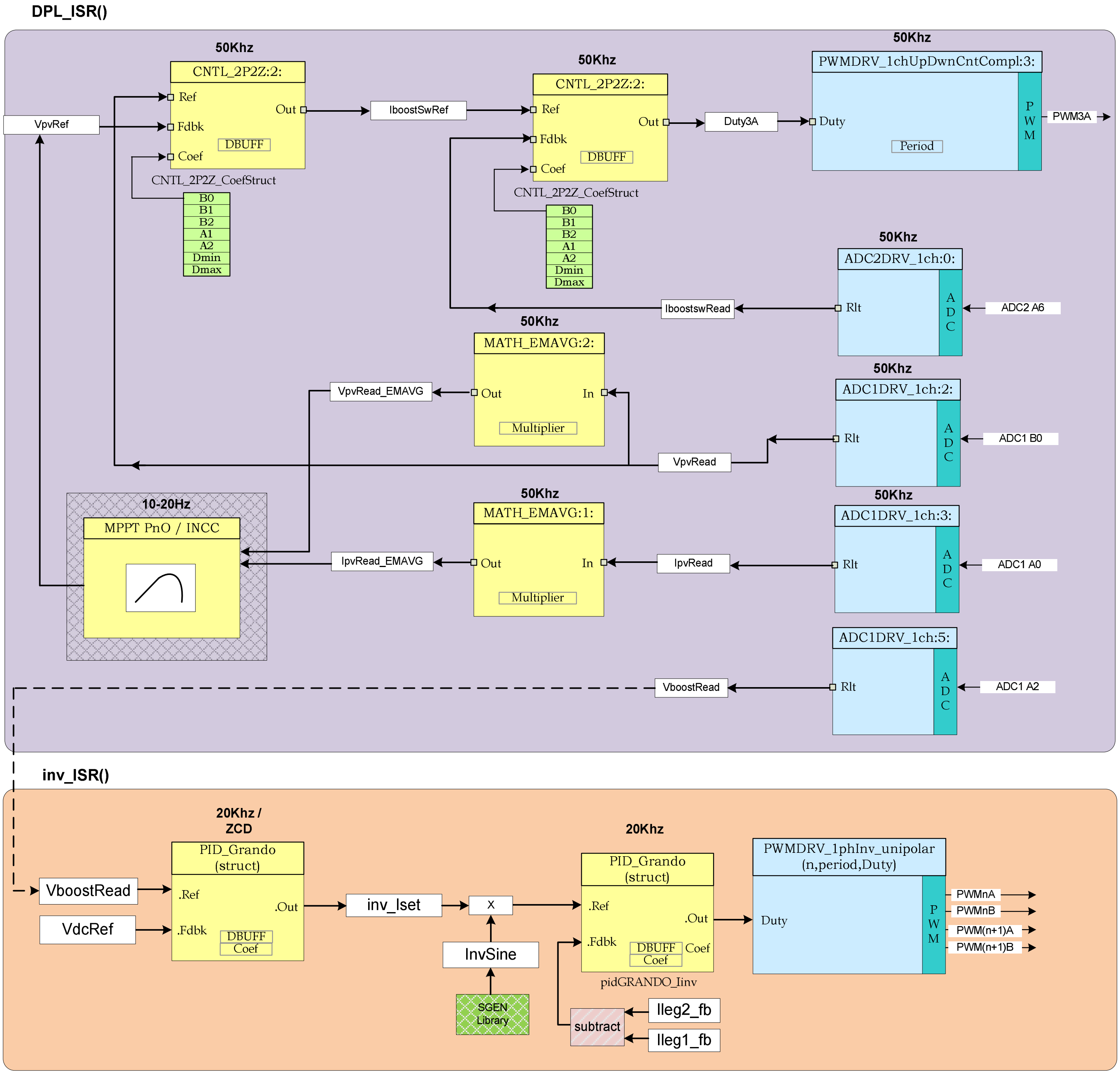


Fig 31 Software Diagram for Build 2

Procedure

Assuming the user if following from the Build-1 the hardware setting as described in the
3.1 HW Setup Instructions and *3.2 Software Setup* will already be done,

1. Turn on [M6]-SW2, if not already on. [M6]-LD1, [M6]-LD2 and [M5]-LD1 will glow green indicating bias power is being generated. [M5]-LD2 will start blinking indicating the PV emulator code is running on the emulator.

2. The connection of the digital power library blocks to implement the DC-DC Boost with MPPT is illustrated in 2.3 Control Description -> DC-DC Boost with MPPT Control Software . Further Locate and inspect the following code in the main file under initialization code (*SolarExplorer-Main.c*). This is where coefficient buffers for 2P2Z modules used in Boost stage control software are initialized. Also, look for mppt module initialization code in the same file.

```
      // Coefficients for Outer Voltage Loop
      // PID coefficients & Clamping - Current loop
      Dmax_V  = 0.9;
      Pgain_V = 0.015;
      Igain_V = 0.00005;
      Dgain_V = 0.0;

      // Coefficient init
      CNTL_2P2Z_CoefStruct1.b2   =Dgain_V;                            // B2
      CNTL_2P2Z_CoefStruct1.b1   =(Igain_V-Pgain_V-Dgain_V-Dgain_V);  // B1
      CNTL_2P2Z_CoefStruct1.b0   =(Pgain_V + Igain_V + Dgain_V);      // B0
      CNTL_2P2Z_CoefStruct1.a2   =0.0;                                // A2 = 0
      CNTL_2P2Z_CoefStruct1.a1   =1.0;                                // A1 = 1
      CNTL_2P2Z_CoefStruct1.max  =Dmax_V;                             //Clamp Hi
      CNTL_2P2Z_CoefStruct1.min  =0.0;                                //Clamp Min

      // Coefficients for Inner Current Loop
      // PID coefficients & Clamping - Current loop

      Dmax_I  = 0.9;
      Pgain_I = 0.015;
      Igain_I = 0.00005;
      Dgain_I = 0.0;

      // Coefficient init
      CNTL_2P2Z_CoefStruct2.b2   =Dgain_I;                            // B2
      CNTL_2P2Z_CoefStruct2.b1   =(Igain_I-Pgain_I-Dgain_I-Dgain_I);  // B1
      CNTL_2P2Z_CoefStruct2.b0   =(Pgain_I + Igain_I + Dgain_I);      // B0
      CNTL_2P2Z_CoefStruct2.a2   =0.0;                                // A2 = 0
      CNTL_2P2Z_CoefStruct2.a1   =1.0;                                // A1 = 1
      CNTL_2P2Z_CoefStruct2.max  =Dmax_I;                             //Clamp Hi
      CNTL_2P2Z_CoefStruct2.min  =0.0;                                //Clamp Min

      // Initialize the net variables
      Duty3A =0.0;
      VboostRead=0.0;
      IboostswRead=0.0;
      VpvRef=0.9; // to increase current, we need to reduce VpvRef, thus initailize it
with a high value.
      IboostSwRef=0.0;
      Duty3A_fixed=0.0;
```

```
    VpvRead_EMAVG=0.0;
    IpvRead_EMAVG=0.0;

    // MPPT testing related code
    // mppt incc
    mppt_incc1.IpvH = 0.0001;
    mppt_incc1.IpvL = -0.0001;
    mppt_incc1.VpvH = 0.0001;
    mppt_incc1.VpvL = -0.0001;
    mppt_incc1.MaxVolt = 0.9;
    mppt_incc1.MinVolt = 0.0;
    mppt_incc1.Stepsize = 0.005;
    mppt_incc1.mppt_first=1;
    mppt_incc1.mppt_enable=0;

    //mppt pno
    mppt_pno1.DeltaPmin = 0.00001;
    mppt_pno1.MaxVolt = 0.9;
    mppt_pno1.MinVolt = 0.0;
    mppt_pno1.Stepsize = 0.005;
```

---

3. Also open the *SolarExplorer-DPL_ISR.asm* and inspect its contents. This file consists of the real time ISR code for the DC-DC Boost, notice the macros used for ADC result reading, execution of control code (2P2Z modules) and PWM drivers used for driving the boost PWM switch.

```
_DPL_ISR:
   ; context save
   ADC1DRV_1ch 2
   ADC1DRV_1ch 3
   ADC2DRV_1ch 0
   ADC1DRV_1ch 5
   CNTL_2P2Z 1
   CNTL_2P2Z 2
   PWMDRV_1ch_UpDwnCntCompl 3          ; PWM3A
   MATH_EMAVG 1
   MATH_EMAVG 2
   ;context restore
   IRET
```

4. Now go to task B1 in the Main.c file and inspect the MPPT usage. Task B1 is repeatedly called from the Main.c background framework, the frequency at which this task is called is determined by the value written to the CpuTimer1Regs.PRD.all register in the beginning of the main code. The task consists of both the usage of mppt_incc and mppt_pno algorithms, the user may comment may choose to use either. An LED is also blinked on the controller when the MPPT task is enabled. The MPPT execution is further slewed in software by variable MPPT_slew. MPPT_incc is used as default in this lab exercise.

```
// MPPT routine
mppt_incc1.Ipv = IpvRead_EMAVG; //IpvRead;
mppt_incc1.Vpv = VpvRead_EMAVG; //VpvRead;

mppt_incc_MACRO(mppt_incc1);

VpvRef_MPPT = mppt_incc1.VmppOut;

mppt_pno1.Ipv = IpvRead_EMAVG; //IpvRead;
```

```
mppt_pno1.Vpv = VpvRead_EMAVG; //VpvRead;

mppt_pno_MACRO(mppt_pno1);

// VpvRef_MPPT = mppt_pno1.VmppOut;
```

5. For build 2, software goes through a state machine, listed below, before turning on the inverter output. This state machine is described in task B3.
   - Inverter State ==0 , wait for the command to start production of power (Gui_InvStart==1)
   - Inverter State ==1 , Check if panel voltage is available, i.e. Vpv > 3V, if true enable MPPT
   - Inverter State ==2 , Now as the MPPT algorithm kicks in the capacitors between the DC-DC and DC-AC will start storing the energy from the panel and the voltage of these caps will rise, Now check if Gui_Vboost > 30V and enable inverter closed loop operation to regulate the DC bus and the current on.
   - Inverter State ==3, The inverter is now ON and producing power, the power delivered will change as the Gui_LightCommand value is changed. Now the system waits for stop command (Gui_InvStop==1), if stopped, trip all PWM's, shut down MPPT and return to state 0, reset all values

Build and Load the Project

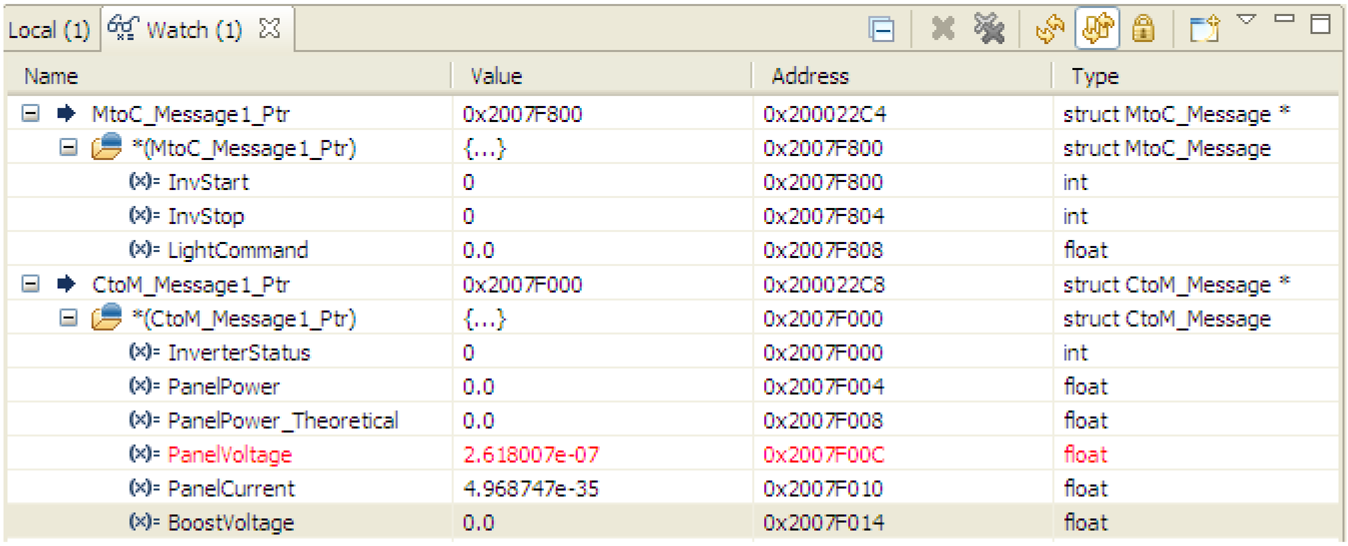
6. Open *SolarExplorer-Settings.h* and set build option as 2.

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

8. If the debug session is still on you may be asked to just reload the updated image on the C28x, otherwise launch the xds100v2-f28m35x target configuration and connect to the C28x and M3 and load the flash images. Once image is loaded on both M3 and the C28x follow the sequence below

   a. Reset C28x
   b. Reset M3
   c. Restart M3
   d. Run M3
   e. Run the C28x
   f. Now halt the M3 code (C28x is running)

9. Setup the Debug Environment Window to observed the variables from the M3 side, no debug is performed from the C28x side in this build, if the user desires the real time can be enabled as was illustrated in the BUILD=1 of the C28x. Also note on the M3 side as there is no real time mode, the user must halt, update values and run the code and then halt again to see changes in the value.



Name	Value	Address	Type
MtoC_Message1_Ptr	0x2007F800	0x200022C4	struct MtoC_Message *
*(MtoC_Message1_Ptr)	{...}	0x2007F800	struct MtoC_Message
InvStart	0	0x2007F800	int
InvStop	0	0x2007F804	int
LightCommand	0.0	0x2007F808	float
CtoM_Message1_Ptr	0x2007F000	0x200022C8	struct CtoM_Message *
*(CtoM_Message1_Ptr)	{...}	0x2007F000	struct CtoM_Message
InverterStatus	0	0x2007F000	int
PanelPower	0.0	0x2007F004	float
PanelPower_Theoretical	0.0	0x2007F008	float
PanelVoltage	2.618007e-07	0x2007F00C	float
PanelCurrent	4.968747e-35	0x2007F010	float
BoostVoltage	0.0	0x2007F014	float

Fig 32 Build 2 CCS Watch Window setup

Run the Code

10. Write a value of 0.2 to MtoC_Message1_Ptr->LightCommand, in the watch window. This value determines the curve that the panel emulator uses for interpolation of the V vs I curve under different lighting conditions. Any changes to this value are communicated through SSI/SPI to the F28027 controller that implements the PV panel emulator. A table with V vs I characteristic is stored corresponding to the LightCommand value of (1.0) and the points are linearly interpolated for a different luminance level.

$$V_{pv\_ref\_G2} = \frac{G2}{G1} * V_{pv\_ref}$$

The Fig 33 shows the panel emulator characteristic with changing luminance, note these are not exactly the same as a real panel but are useful for a quick illustration of MPPT and Solar Powered Inverter.

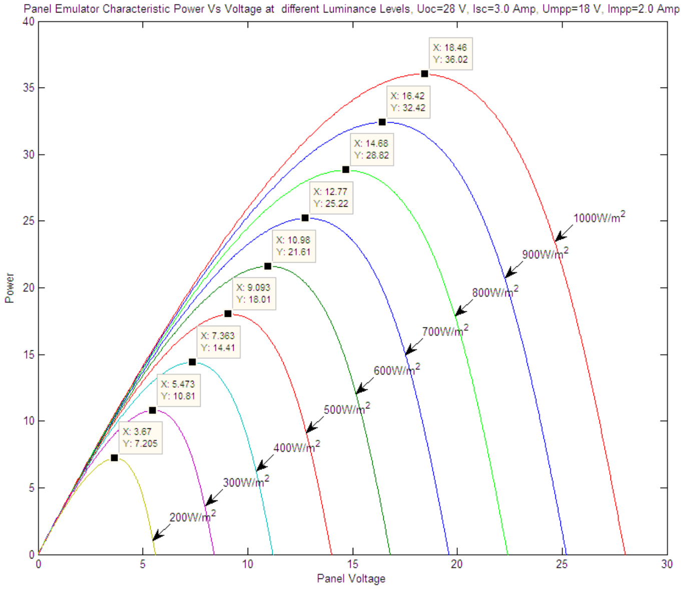


Fig 33 Panel Emulator Characteristic

Table 1 Panel Emulator MPP voltage and power with Changing Luminance

Luminance Ratio (w.r.t 1000W/m^2)	**Pmpp =(Pmax * Luminance Ratio) Watts**	**Vmpp (Volts)**
1.0 = 1000 W/m^2	36.02	18.46
0.9 = 900W/m^2	32.42	16.42
0.8 = 800W/m^2	28.82	14.68
0.7 = 700W/m^2	25.22	12.77
0.6= 600W/m^2	21.61	10.98
0.5=500W/^2	18.01	9.093
0.4=400W/m^2	14.41	7.363
0.3=300W/m^2	10.81	5.473
0.2=200W/m^2	7.205	3.67

11. In the watch window notice the value of CtoM_Message1_Ptr->PanelPower_Theoretical which will be (7.205) corresponding to the table listed above.

12. Now change the value of MtoC_Message1_Ptr→InvStart to 1, the state machine will now kick in. MPPT will be enabled and the DC bus will start rising and as it reached 30V the inverter close loop operation will be started. The state can also be monitored by watching the variable PVInverterState from the C28x side in the watch window. Note the CtoM_Message1_Ptr->PanelPower, which shall now be close to the theoretical value. This shows the tracking of the MPPT from a cold starts.

13. Now change the MtoC_Message1_Ptr->LightCommand set to 0.4, the CtoM_Message1_Ptr-PanelPower_Theoretical will now change to (14.41), as MPPT is on the CtoM_Message1_Ptr->PanelPower is tracked close to 14.41W and CtoM_Message1_Ptr->PanelVoltage reaches close to 7.363V. Now the system is operating at Maximum Power Point for the given LightCommand as shown in the table above, illustrating MPPT tracking uder varying lighting condition.

14. The user can now change the value of the LightCommand in increments or decrements of 0.1 between (0.2) to (0.8) and under each condition ensure that the system is tracked to the MPP. Once the LightCommand is changed it takes a few seconds for the system to track to the new Maximum Power Point under the changed Luminance conditions. Check the observed results with the expected results from Table 1 Panel Emulator MPP voltage and power with Changing Luminance.

    If LightCommand greater than 0.8 is desired the user must use a power supply of a higher rating than what is supplied with the kit. This can be done by connecting an external power supply at [M6]-TB1 and depopulating the jumper [M6]-J1.

15. MPPT efficiency can be measured using different metrics; one is **Static Efficiency** which is defined as how close the panel power is to the theoretical power that can be extracted from the panel under given conditions. **Dynamic efficiency** is determined by the speed with which MPP is achieved under varying light conditions.

Screen shots of system parameters under various dynamic MPPT test are below:

Channel 1: Vpv
Channel 4: Vboost
Channel 2: Iac_measured

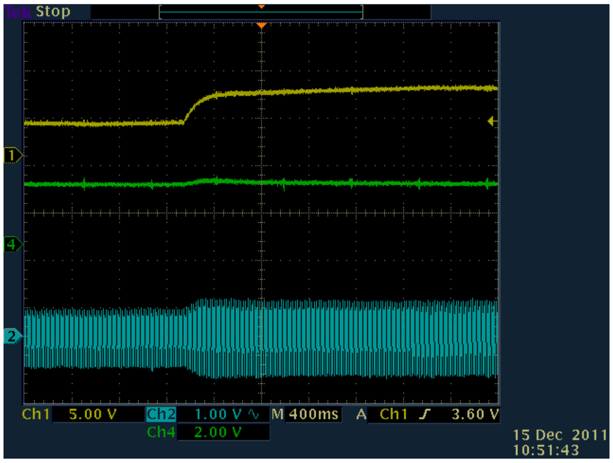


Fig 34 Dynamic MPPT Luminance 0.2 -> 0.4

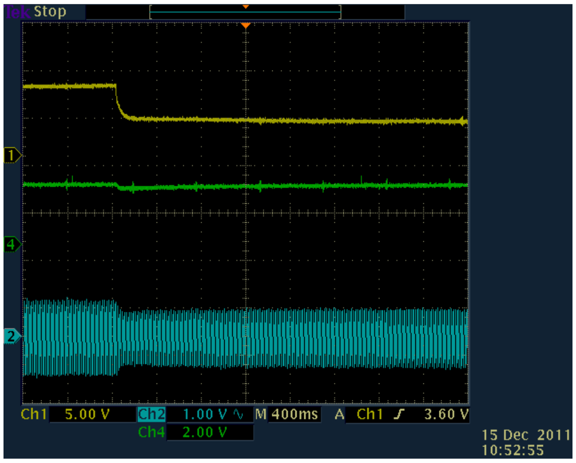


Fig 35 Dynamic MPPT 0.4 -> 0.2

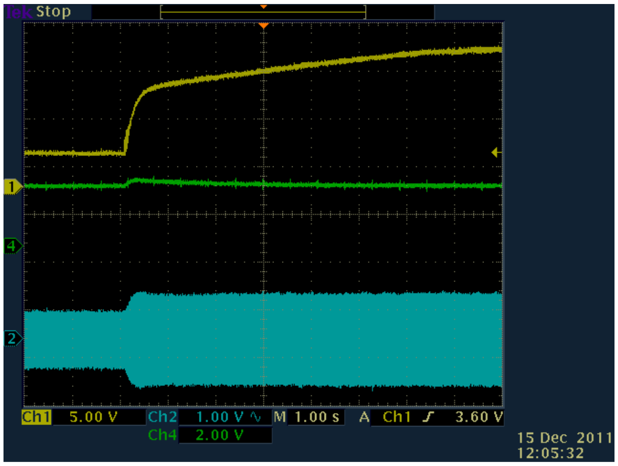


Fig 36 Dynamic MPPT Luminance 0.2 -> 0.8

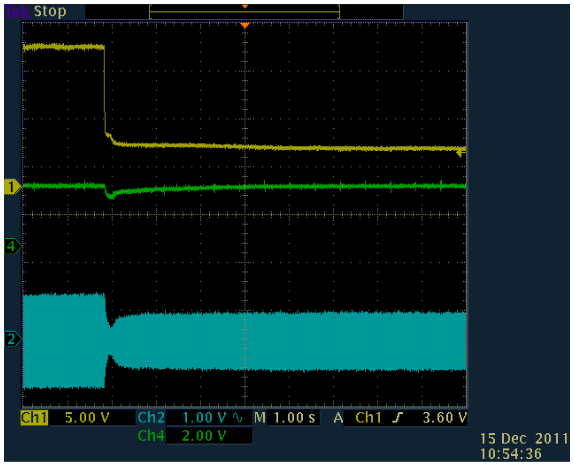


Fig 37 Dynamic MPPT Lumimance 0.8 -> 0.2

16. To end this exercise, write ‘1’ to MtoC_Message1_Ptr→InvStop in the watch window from the M3 side, now write a ‘0.0’ to MtoC_Message1_Ptr->LightCommand. Now halt the M3 and the C28x ⏸, take the C28x out of realtime mode if enabled by clicking on .

17. Close CCS debug session by clicking on Terminate Debug Session ⏹ (Target->Terminate all).

18. Switch off the power to the board by moving [M6]-SW2 to off position.

End of Exercise

4.3 C28x BUILD = 3, M3 BUILD = 1

Inverter Current Control with DC Bus Regulation & DC-DC Boost MPPT Control & Grid Sync

Objective:
The objective of the build is to understand the operation of grid-tied PV Inverter system. Inverter closed current loop along with DC bus regulation is run along with MPPT with DC-DC Boost stage. Software PLL is used for grid synchronization and calculating the phase of the grid.

Overview:
The inverter stage on the Solar Explorer kit is rated for low voltages, hence to test connection with the grid a step down transformer is used. Also the bulb used in build 1 and 2, as it has a very low resistance, is replaced with a resistive load of 15 Ohms. Fig 38 shows the hardware setup for the BUILD 3. The AC Grid is stepped down to 14V AC; this can be checked using a multimeter. At this voltage the power consumed by the load is, $P_{ac} = \frac{V_{ac}^2}{R_l}$, when the inverter is off all the power to the load is provided by the grid current i.e. i_{grid}. As the inverter starts producing power the load current is shared by the PV inverter and the grid i.e. $i_{Rl} = i_{grid} + i_{pv}$. As the power produced from the inverter increases the entire load current is supplied by the inverter and the surplus is fed back into the grid. This is observable from the phase of the grid current w.r.t to the grid voltage.

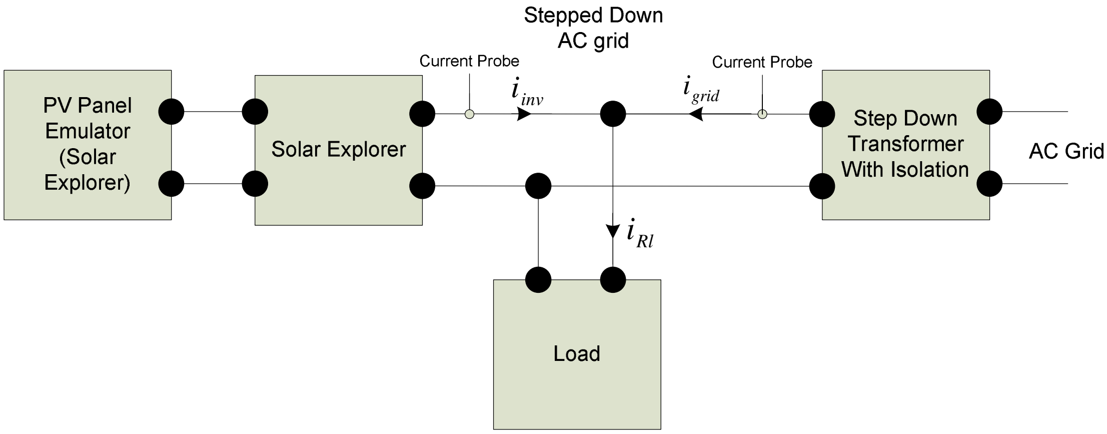


Fig 38 Hardware Setup for Grid Connection Test with SolarExplorer

Control for this build is similar to Build 2, with an addition of software PLL block integration which is used to determine the phase of the grid instead of the SGEN block. Sine analyzer block is added to determine the RMS voltage of the grid and ZCD determination.

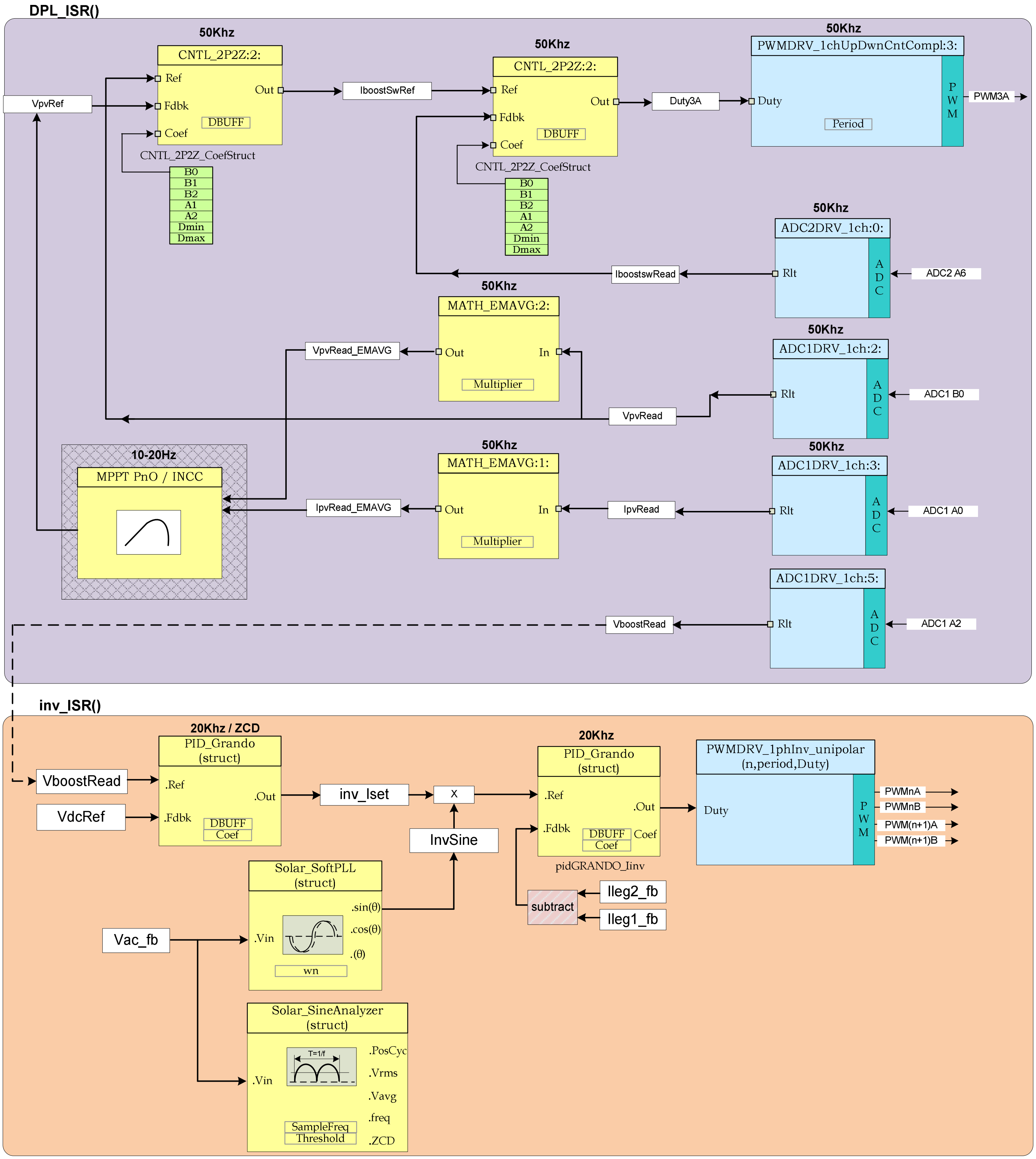


Fig 39 Software Diagram for Build 3

Procedure

Following sections detail the hardware setup and steps to connect CCS perform objectives for build 3.

Hardware set up for Build 3:

- Assuming the hardware setting for build 2 is present, and the board is not powered. Take out the bulb from the socked located in [M2] macro (**Caution**, the bulb may be hot!).
- Fig 40 shows the hardware settings for the Build 3 test.
- First move the autotransformer to 0 setting and make connection with the resistive load and the solar explorer board with the gird as shown in the Fig 40.
- Verify SW1 on [M6] is in on position. Now turn on power to the board [M6]SW2 to ON position.

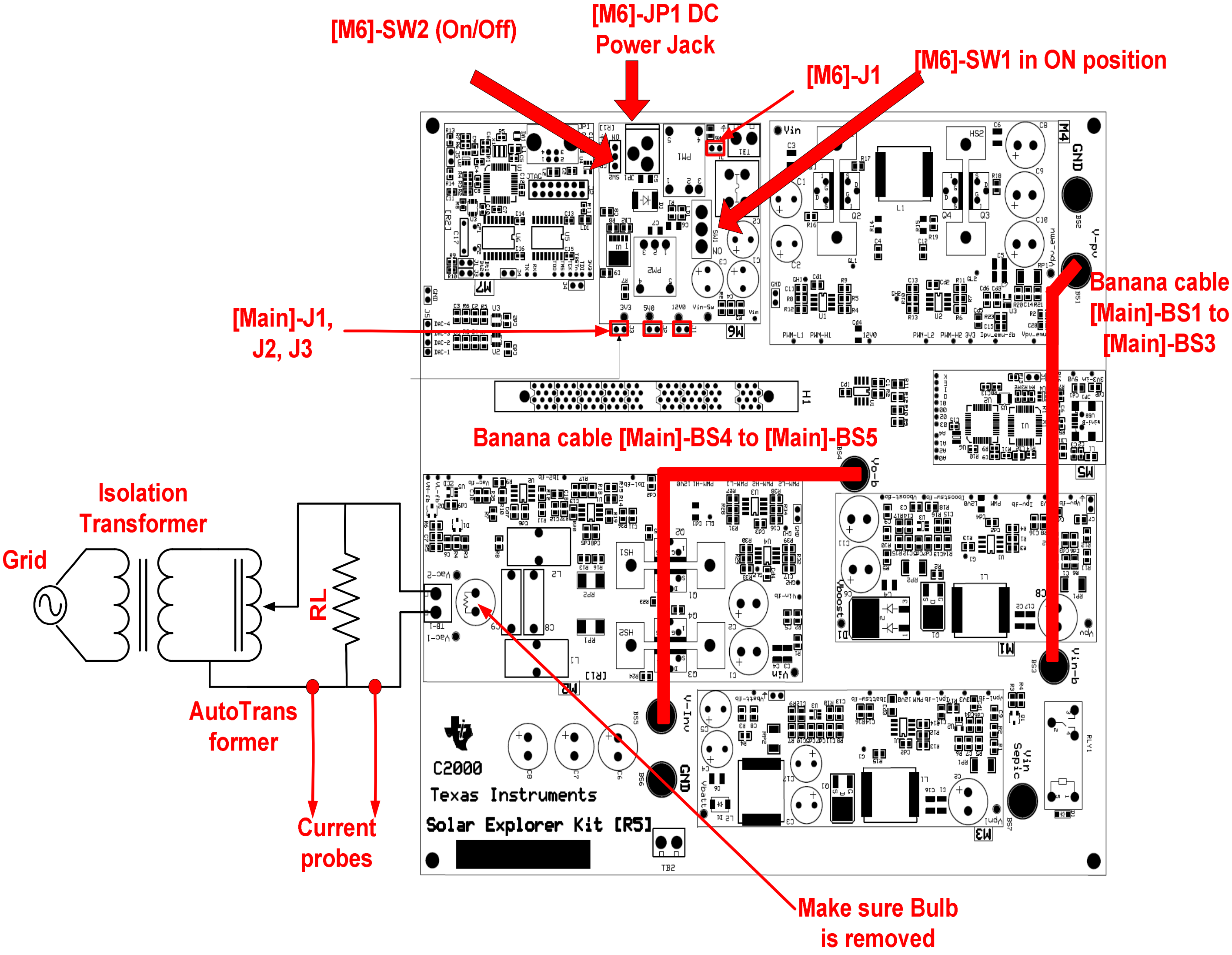


Fig 40 Hardware Setting for Build 3

Start CCS and Open a Project

1. Locate and inspect the following code in the main file (*SolarExplorer-Main.c)* under Inv_ISR() for Build 3. This is where the software PLL is instantiated. PLL takes grid voltage reference (Vac_FB) and generates sine of the grid angle (spll1.Mysin[0]).

```
// PLL Start
spll1.AC_input=((float)Vac_FB*0.000244140625-0.5)*2.0;

SPLL_1ph_MACRO(spll1);

InvSine     =(spll1.sin[0]);

//    Voltage loop
pidGRANDO_Vinv.term.Fbk = (0.75); // 30V/ 39.97
pidGRANDO_Vinv.term.Ref = VboostRead; //Ref=VDC/sqrt(2) at full modulation index

if (CloseVloopInv==1 && sine_mainsV.ZCD==1)
{
      PID_GR_MACRO(pidGRANDO_Vinv);
      inv_Iset=pidGRANDO_Vinv.term.Out;
}
```

2. For build 3, software goes through a state machine, listed below, before turning on the inverter output. This state machine is described in task B3.
   - Inverter State ==0 , Wait for inverter start command (InvStart==1)
   - Inverter State ==1 , Check grid voltage, i.e. Vrms Real > _IQ15(12)
   - Inverter State ==2 , Check if panel voltage is available, i.e. Vpv > 3V, if true enable MPPT
   - Inverter State ==3, Now as the MPPT algorithm kicks in the capacitors between the DC-DC and DC-AC will start storing the energy from the panel and the voltage of these caps will rise, Now check if Vboost > 30V and enable inverter closed loop operation to regulate the DC bus and the current on.
   - Inverter State ==4, Wait for the current command to be of significant value (this is primarily done as a safety measure), then clear the initial inverter trip.
   - Inverter State ==5, The inverter is now ON and producing power, the power delivered will change as the LightCommand value is changed. This will also change the power sourced from the grid. The system waits for stop command (Gui_InvStop==1), if stopped, trip all PWM's, shut down MPPT and return to state 0, reset all values

Build and Load the Project

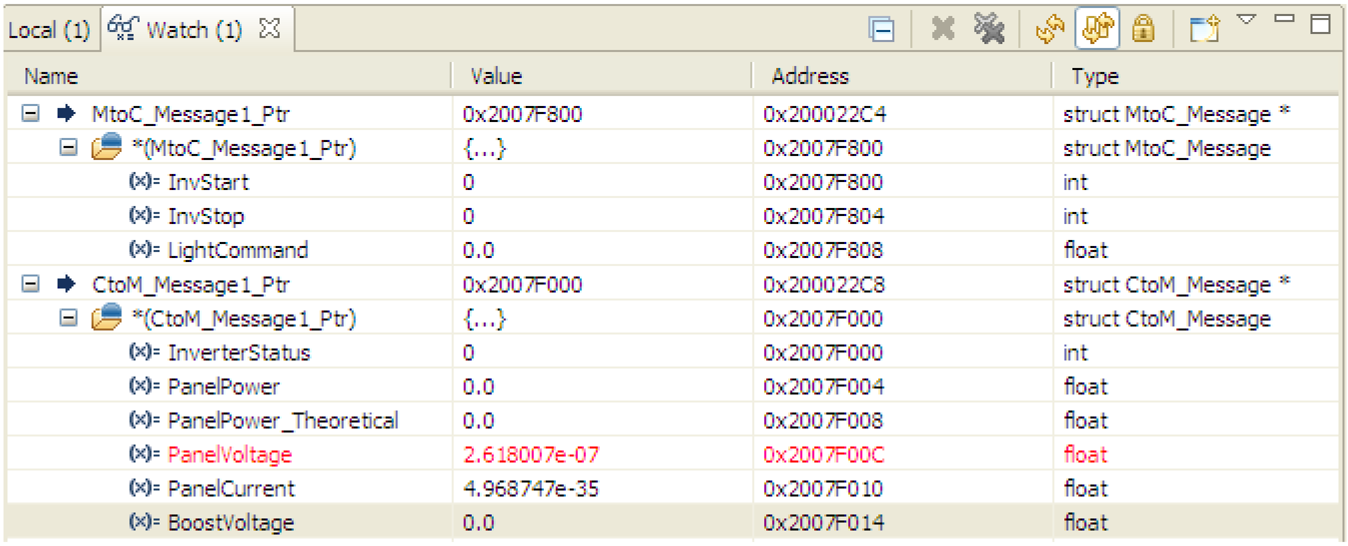
3. Select the incremental build option as 3 in the *SolarExplorer-Settings.h* file.

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

4. Launch the xds100v2-f28m35x target configuration and connect to the C28x and M3 and load the updated flash images. Once image is loaded on both M3 and the C28x follow the sequence below

   a. Reset C28x
   b. Reset M3
   c. Restart M3
   d. Run M3
   e. Run the C28x
   f. Now halt the M3 code (C28x is running)

5. Setup the Debug Environment Window to observed the variables from the M3 side, no debug is performed from the C28x side in this build, if the user desires the real time can be enabled as was illustrated in the BUILD=1 of the C28x. Also note on the M3 side as there is no real time mode, the user must halt, update values and run the code and then halt again to see changes in the value.



Name	Value	Address	Type
⊟ MtoC_Message1_Ptr	0x2007F800	0x200022C4	struct MtoC_Message *
⊟ *(MtoC_Message1_Ptr)	{...}	0x2007F800	struct MtoC_Message
(x)= InvStart	0	0x2007F800	int
(x)= InvStop	0	0x2007F804	int
(x)= LightCommand	0.0	0x2007F808	float
⊟ CtoM_Message1_Ptr	0x2007F000	0x200022C8	struct CtoM_Message *
⊟ *(CtoM_Message1_Ptr)	{...}	0x2007F000	struct CtoM_Message
(x)= InverterStatus	0	0x2007F000	int
(x)= PanelPower	0.0	0x2007F004	float
(x)= PanelPower_Theoretical	0.0	0x2007F008	float
(x)= PanelVoltage	2.618007e-07	0x2007F00C	float
(x)= PanelCurrent	4.968747e-35	0x2007F010	float
(x)= BoostVoltage	0.0	0x2007F014	float

Fig 41 Build 3 CCS Watch Window setup

Run the Code

6. Now make the following connection to the oscilloscope to observe different variables in the subsequent exercises.

Channel 1: Grid Voltage (measured PWMDAC)
Channel 2: PLL lock (calculated PWMDAC)
Channel 3: Grid Current (measured with current probe on the connector from step down transformer to load)
Channel 4: Inverter current (measured using current probe placed on the connector from TB1 to load) or PWMDAC

7. **Checking Grid Synchronization**: Now as the code is running, first thing to check is if the PLL is operating correctly. For this two channel from the PWMDAC are used (PWMDAC1 and PWMDAC2), on which the SPLL out and the input AC voltage measured are plotted. Note as autotransformer is currently at zero setting the SPLL will be getting reset again and & again as there is no AC to lock to. Now gradually increase the AC voltage by moving the auto transformer position from the zero position. The SPLL out will be locked to the output voltage of the inverter. The waveform will be similar to as shown in figure below

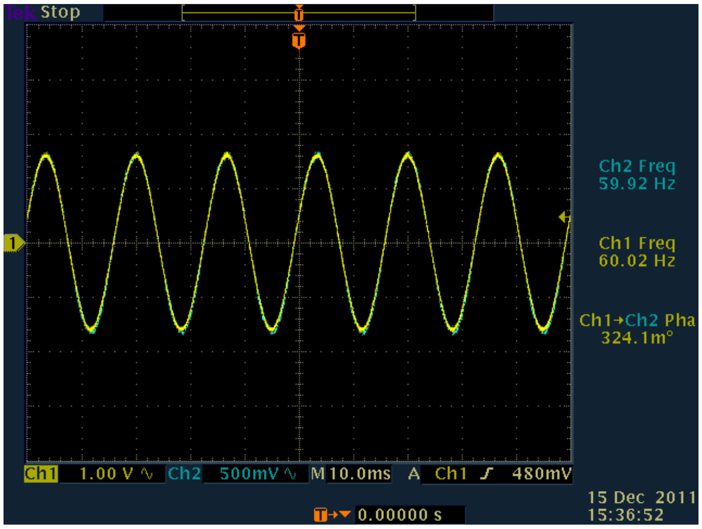


Fig 42 Grid Synchronization using SPLL

8. Move the autotransformer to the point such that the VrmsReal reads close to (14.0).

9. **Load Power = Power From Inverter + Power From Grid**, Now for a load resistance of 15 Ohms and Grid Voltage (Vrms) of 14V, load current requirement would be Irms 0.933Amps. Total Power dissipated in the load is 13W. At the start the Gui_LightCondition has a value of 0.2 which corresponds to theoretical power maximum of 7.2W. Therefore once the inverter is turned on, under these conditions, inverter will deliver close to 7.2W (Luminance Ratio of 0.2), and remaining power requirement for the load will be provided from the grid.

   Write MtoC_Message1_Ptr->LightCondition value to 0.2 in the watch window, and watch the CtoM_Message1_Ptr->PanelVoltage increase (note you will have to do this from the M3 side, and run and halt the processor again for any change you make to be effective). Now change the value of MtoC_Message1_Ptr->InvStart to 1, the state machine will now kick in. MPPT will be enabled and the DC bus will start rising and as it reaches 30V the inverter close loop operation will be started. The state can also be monitored by watching the variable PVInverterState in the watch window from the C28x side if needed. Note the CtoM_Message1_Ptr->PanelPower, which shall now be close to the theoretical value CtoM_Message1_Ptr->PanelPower_Theoretical. This shows the tracking of the MPPT from a cold start. Fig 43 Shows the current waveform with the inverter and the grid current, as they are almost in phase this shows the currents from the two sources i.e. the grid and the PV inverter sum together to provide the required current for the load.

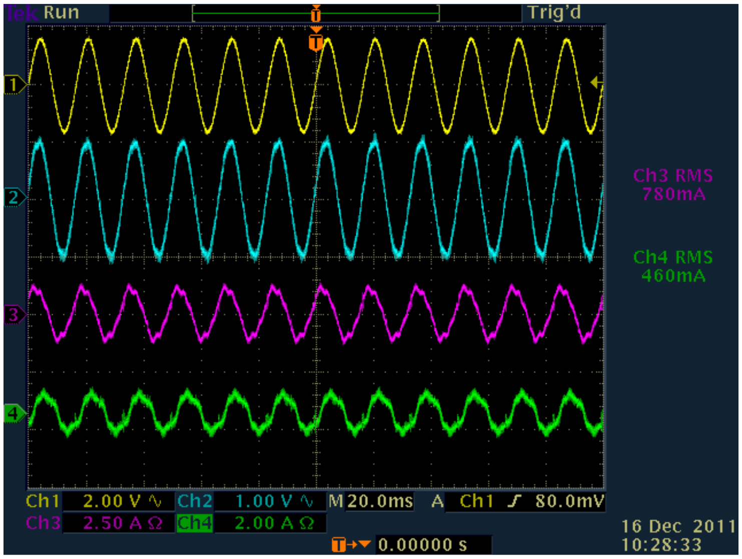


Fig 43 Load Power = Power From Inverter + Power From Grid, Vrms ~15-16V, Rl ~15 Ohms

10. **Power From Inverter = Load Power + Power Fed Into the Grid**, Now change the MtoC_Message1_Ptr>LightCommand to 0.8, The CtoM_Message1_Ptr->PanelPower_Theoretical will now change to (28.7), as MPPT is on the PanelPower is tracked close to 28.7W. Now as the inverter has surplus power that, it starts feeding power into the grid. This reversal in power flow is marked by the phase change in the grid current Fig 44 w.r.t the inverter current as compared to the previous case of Fig 43.

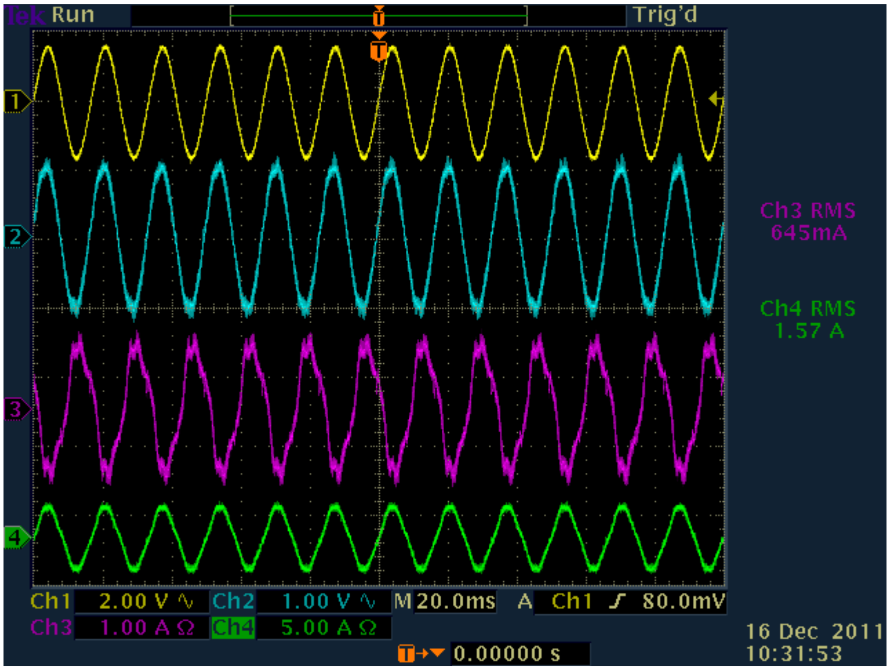


Fig 44 Power From Inverter = Load Power + Power Fed Into the Grid, Vrms ~15-16V, Rl ~15 Ohms

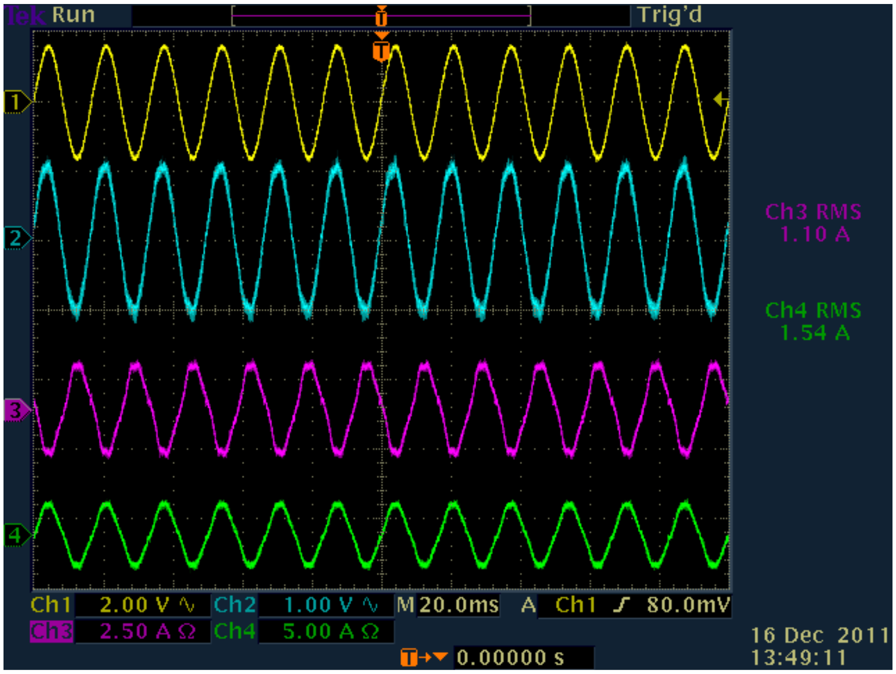


Fig 45 Load changed(reduced) w.r.t. Fig 44,
Note the power from the inverter remains the same, the power fed to the grid increases, Vrms ~15-16V, Rl ~30 Ohms

11. The user can now change the value of the LightCommand in increments or decrements of 0.1 between (0.2) to (0.8) and under each condition ensure that the system is tracked to the MPP and how the grid current changes as each level. Once the LightCommand is changed it takes a few seconds for the system to track to the new Maximum Power Point under the changed Luminance conditions. Check the observed results with the expected results from Table 1 Panel Emulator MPP voltage and power with Changing Luminance.

12. To end this exercise, write '1' to MtoC_Message1_Ptr→InvStop in the watch window from the M3 side, now write a '0.0' to MtoC_Message1_Ptr->LightCommand. Now halt the M3 and the C28x , take the C28x out of realtime mode if enabled by clicking on .

13. Close CCS debug session by clicking on Terminate Debug Session (Target->Terminate all).

14. Switch off the power to the board by moving [M6]-SW2 to off position.

End of Exercise

4.4 C28x BUILD = 2, M3 BUILD = 2

Inverter Current Control with DC Bus Regulation & DC-DC Boost MPPT Control and Monitoring and Control Over Ethernet

This build is for user’s inspection only. The build implements a PV inverter system with Ethernet diagnostic capability built into it. Crosshairs engine is also integrated in the software and Crosshairs Interface designer is used to implement the GUI over Ethernet. This is the same GUI that is used in the QSG with Concerto.

Note: Rebuilding the project and using with the QSG GUI will not work as the crosshairs GUI is linked from the *.out file and any recompile can change the location of variables in the out file. Crosshairs Interface designer is needed to customize the GUI similar to QSG. Alternatively the user can refer to web server examples found in the device header file to create a simple diagnostics page for the control and monitoring of the PV Inverter.