

PSFB Control Using C2000 Microcontrollers

Hrishikesh Nene

ABSTRACT

The phase shifted full bridge (PSFB) converter is used for DC-DC conversion in various applications, for example, in telecom systems to convert a high voltage bus to an intermediate distribution voltage, typically closer to 48 V. PSFB stage provides voltage translation as well as isolation from the line voltage, since this topology includes a transformer.

This application report presents the implementation details of a digitally controlled PSFB system implemented on the high voltage phase shifted full bridge (HVPSFB) kit from Texas Instruments. This kit converts a 400 V DC input to a regulated 12 V DC output and is rated for operations up to 600W. Both peak current mode control (PCMC) and voltage mode control (VMC) implementations are described. These highly integrated microcontroller-based realizations feature adaptive zero voltage switching (ZVS) and various synchronous rectification schemes, which are discussed here. Details for generating complex gate drive waveforms, required by these control schemes, and intelligent timing control, to optimize system performance under changing operating conditions, are provided. A step-by-step guide to run the HVPSFB project is also included. A constant high system efficiency above 10% rated load, novel PCMC waveform generation based on on-chip hardware mechanisms, and simple system implementation are the highlights of this solution.

NOTE: If you would like to quickly evaluate this kit without going through the implementation details, see the accompanying quick start guide (QSG-HVPSB-Rev1.1.pdf located at www.ti.com/controlsuite) instead of this document.

Contents

1	Introduction	3
2	Functional Description	10
3	Software Overview - PCMC	15
4	Procedure for Running the Incremental Builds - PCMC	19
5	Software overview - VMC	35
6	Procedure for Running the Incremental Builds - VMC	38
7	References	51

List of Figures

1	A Phase Shifted Full Bridge Circuit.....	4
2	PSFB PWM Waveforms.....	4
3	PSFB System Block Diagram	5
4	Efficiency versus Load (PCMC and VMC Implementations)	6
5	ZVS and LVS Switching at 12A Load (a) Active to Passive Leg Transitions (ZVS), (b) Passive to Active Leg Transitions (LVS)	6
6	PCMC Transient Response (a) 0% to 80% Load Step, (b) 80% to 0% Load Step	7
7	PCMC GUI	7
8	HVPSFB Board and Controller Card	8
9	TMS320F28027(Piccolo-A) Controller Card Schematic	8

Piccolo, C2000, Code Composer Studio are trademarks of Texas Instruments.
 All other trademarks are the property of their respective owners.

10	HVPSFB Baseboard Schematic	9
11	PCMC Block Diagram.....	10
12	PCMC PWM Waveforms	11
13	VMC Block Diagram.....	12
14	VMC Waveforms.....	12
15	Effect of Fixed-Point ADC Conversion Triggering for Output Voltage Sensing	13
16	PCMC Software Flow	15
17	PCMC Software Blocks	16
18	PCMC Control Flow	17
19	Build 1 Software Blocks	20
20	C and C++ Projects	22
21	Build 2 Software Blocks	27
22	Constant Current and Constant Power Software Flowchart	30
23	Transformer Primary Voltage, Primary Sensed Current and PWM Waveforms Driving Two Diagonally Opposite Switches (Q1 and Q3).....	33
24	VMC Software Flow	35
25	VMC Software Blocks.....	36
26	VMC Control Flow	37
27	Build 1 Software Blocks	39
28	Build 2 Software Blocks	47

List of Tables

1	Library Modules.....	16
2	Incremental Build Options for PCMC	18
3	Example RAMPMAXREF and DACVAL Values.....	20
4	HVPSFB Signal Interface Reference - PCMC	21
5	Library Modules.....	35
6	Incremental Build Options for VMC	38
7	Phase Values for Reference	39
8	HVPSFB Signal Interface Reference - VMC.....	40

1 Introduction

Phase shifted full bridge (PSFB) DC-DC converters are used frequently to step down high DC bus voltages or provide isolation in medium to high power applications like server power supplies, telecom rectifiers, battery charging systems, and renewable energy systems. Traditionally, microcontrollers have been restricted to only performing supervisory or communications tasks in these systems. With the availability of high performing microcontroller devices, it is now possible to use microcontrollers for closing control loops in these systems, in addition to handling the traditional microcontroller functions. The transition to digital power control means that functions that were previously implemented in hardware are now implemented in software. In addition to flexibility, this simplifies the system considerably. These systems can implement advanced control strategies to optimally control the power stage under different conditions and also provide system level intelligence.

A PSFB converter consists of four power electronic switches (like MOSFETs or IGBTs) that form a full-bridge on the primary side of the isolation transformer and diode rectifiers or MOSFET switches for synchronous rectification (SR) on the secondary side. This topology allows all the switching devices to switch with ZVS resulting in lower switching losses and an efficient converter. In this work, ZVS for switches in the one leg of the full bridge and zero or low voltage or valley switching for switches in the other leg is achieved across the complete load range, by changing dead-times for primary side switches based on load conditions.

For such an isolated topology, signal rectification is required on the secondary side. For systems with low output voltage or high output current ratings, implementing synchronous rectification instead of diode rectification achieves the best possible performance by avoiding diode rectification losses. In this work, current double synchronous rectification is implemented on the secondary side with different switching schemes to achieve optimum performance under varying load conditions.

A DC-DC converter system can be controlled in various modes like VMC, average current mode control (ACMC) or PCMC. Implementing these different control modes for controlling the same power stage typically requires redesigning the control circuit along with some changes to the power stage sensing circuit. With a microcontroller based system, all these modes can be experimented with on the same design with minimal or no additional changes. Such a system is implemented here using VMC and PCMC control schemes.

PCMC is a highly desired control scheme for power converters because of its inherent voltage feed forward, automatic cycle by cycle current limiting, flux balancing and other advantages. Implementing PCMC for a PSFB system requires complex pulse width modulation (PWM) waveform generation with precise timing control. A new approach to this waveform generation is presented using Texas Instruments Piccolo™ series TMS320F2802x and TMS320F2803x microcontrollers without requiring any additional support circuitry. Unique programmable on-chip slope compensation hardware is used to provide appropriate slope compensation that assures open loop stability and eliminates and limits sub-harmonic oscillations at the output. For PCMC implementation with a microcontroller, the regulated output voltage is dependent on the amount of output voltage ripple, which in turn is dependent on the load. This relation is explained in detail and different solutions are suggested.

Peak efficiency greater than 95% and efficiency greater than 90% down to 10% load is achieved with the 600W PSFB system is used here.

1.1 Basic Operation

Figure 1 shows a simplified circuit of a phase shifted full bridge. MOSFET switches Q_A , Q_B , Q_C and Q_D form the full-bridge on the primary side of the transformer T1. Q_A and Q_B are switched at 50% duty and 180° out of phase with each other. Similarly, Q_C and Q_D are switched at 50% duty and 180° out of phase with each other. The PWM switching signals for leg $Q_C - Q_D$ of the full bridge are phase shifted with respect to those for leg $Q_A - Q_B$. The amount of this phase shift decides the amount of overlap between diagonal switches, which in turn decides the amount of energy transferred. D_1 , D_2 provide the diode current double rectification on the secondary, while L_o and C_o form the output filter. The inductor L_R provides assistance to the transformer leakage inductance for the resonance operation with MOSFET capacitance and facilitates ZVS. Note the two different grounds G_1 and G_2 on the two sides of transformer T1.

Figure 2 provides the switching waveforms for the system in Figure 1.

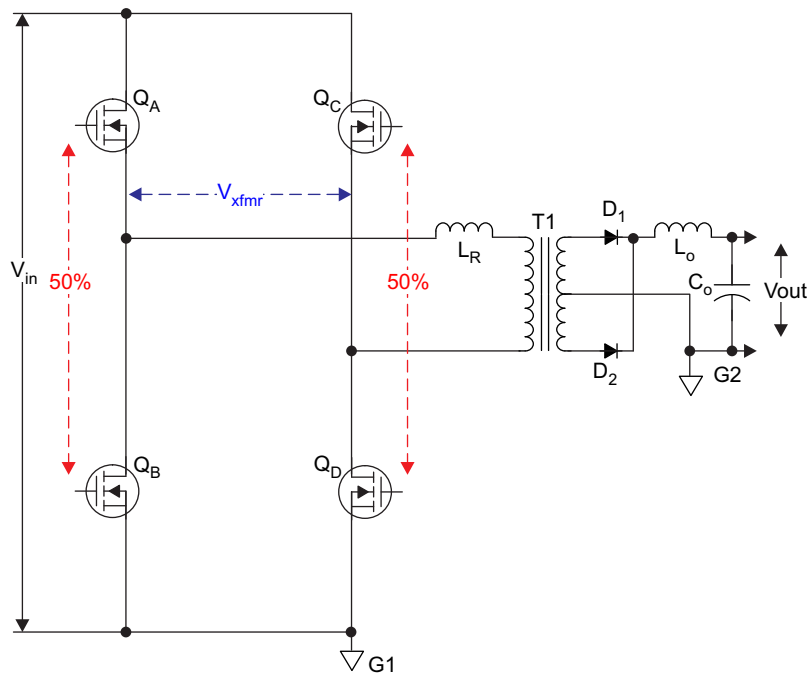


Figure 1. A Phase Shifted Full Bridge Circuit

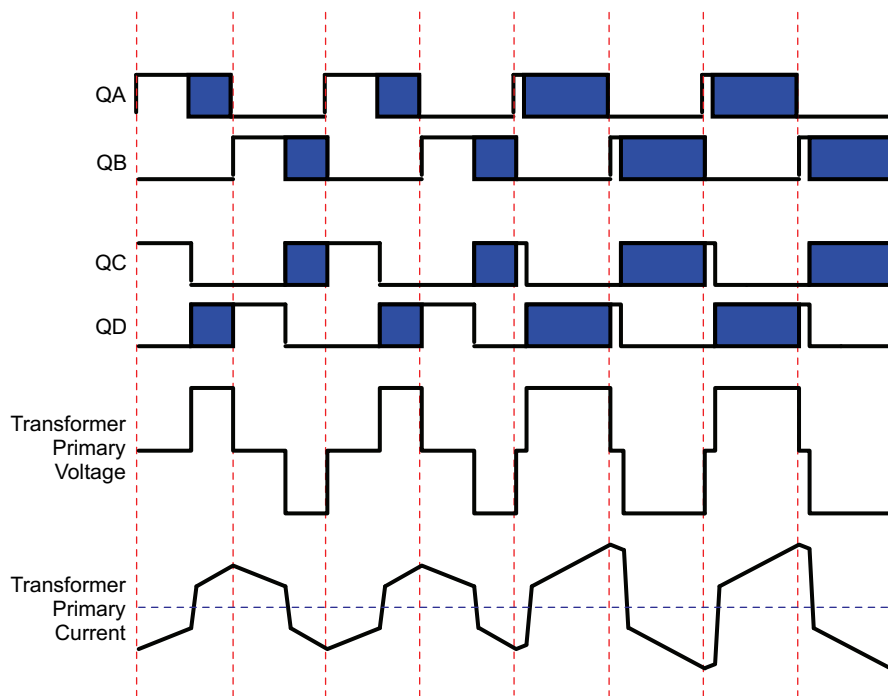


Figure 2. PSFB PWM Waveforms

1.2 PSFB Implementation on HVPSFB Board

Figure 3 shows a simplified block diagram of the PSFB circuit implemented on the HVPSFB board. Switches Q_A , Q_B , Q_C and Q_D in Figure 2 correspond to switches Q1, Q4, Q2 and Q3, respectively. Switches Q5 and Q6 are used for synchronous rectification on the secondary.

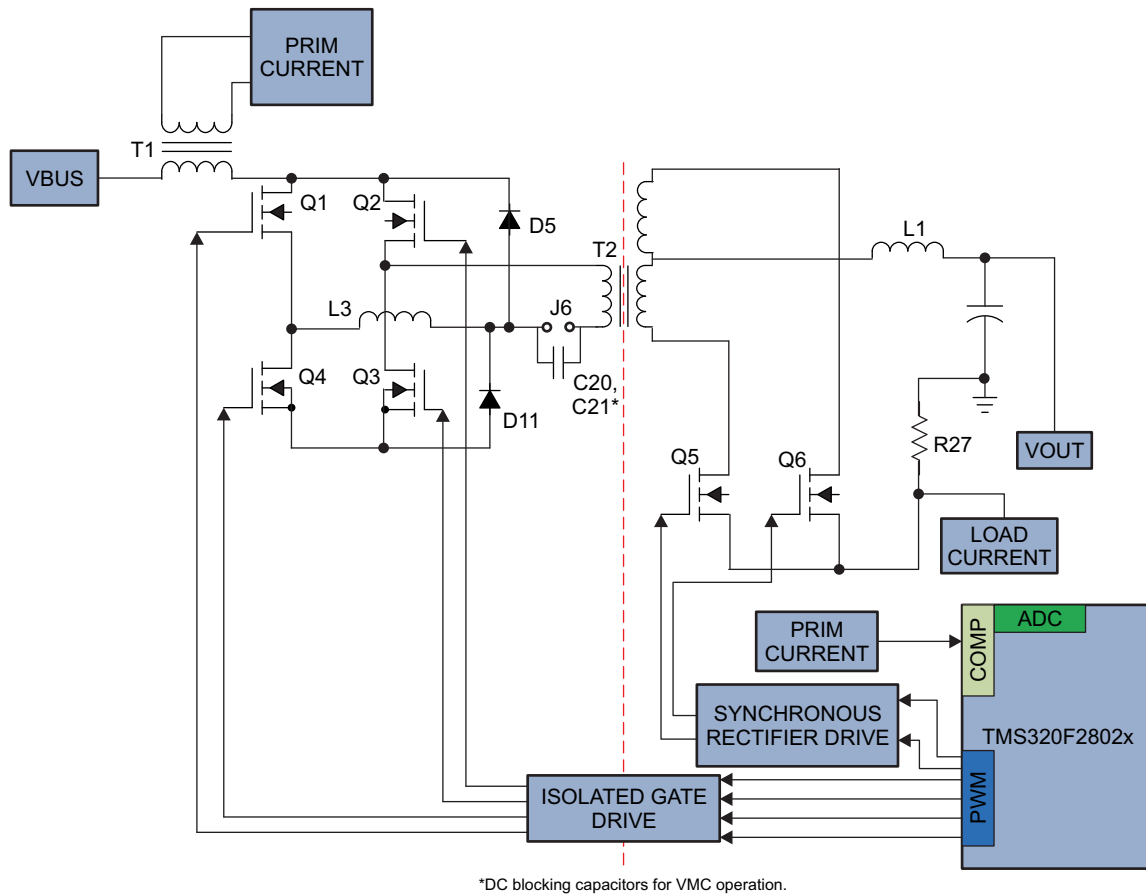


Figure 3. PSFB System Block Diagram

The control algorithm is implemented on a C2000™ microcontroller (MCU). The MCU interacts with the PSFB power stage by way of feedback signals and PWM outputs. The controller is placed on the secondary side on this design. Deciding on the placement of the controller with respect to the isolation boundary is a crucial step when designing an isolated DC-DC system. For systems that have multiple output rails or handle many signals and control loops on the secondary side or communicate with other systems in the application (on the secondary), placing the controller on the secondary side is more advantageous.

Phase shift between PWM signals driving the two legs of the full bridge determines the amount of energy transferred to the load. This phase shift is the controlled parameter.

MCU achieves DC – DC conversion by controlling this phase shift so as to regulate and maintain the output voltage at the commanded level.

Controlling such a system in different operation modes requires generating complex PWM drive waveforms along with fast and efficient control loop calculations. This is made possible on C2000 microcontrollers by advanced on-chip control peripherals like PWM modules, analog comparators with digital analog converter (DAC) and slope compensation hardware and 12-bit high speed ADCs coupled with an efficient 32-bit CPU. A detailed description of the software algorithm is provided in the following sections

1.3 HVPSFB Kit Highlights

The following is a list of key features of the HVPSFB kit:

- 400 V DC input (370 Vdc to 410 Vdc operation), 12 V DC output
- Peak efficiency greater than 95%. Above 90% efficiency down to 10% load.
- 50 Amp (600 Watt) rated output
- Phase shifted full-bridge (PSFB) circuit topology
- 100 KHz switching frequency
- PCMC with no external support circuitry for PCMC function
- Multiple synchronous rectification (SR) switching schemes
- Adaptive zero voltage switching (ZVS) and LVS across the complete load range
- Efficient graphic user interface (GUI) that allows fast and easy system tuning for optimal performance
- Fault protection: input UV and OV, over-current, output UV (CC and CP mode)
- Constant current (CC) and constant power (CP) functions
- Optional voltage mode control (VMC)

Figure 4 through Figure 6 provides some results obtained using this kit.

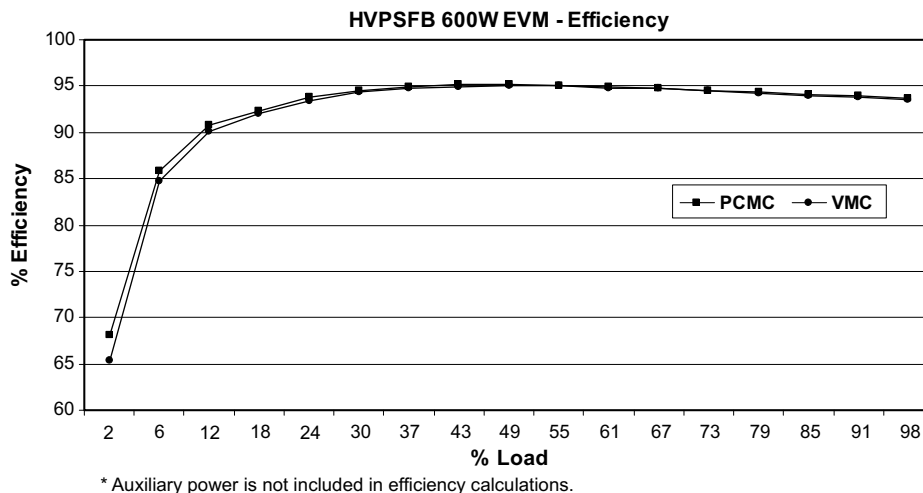


Figure 4. Efficiency versus Load (PCMC and VMC Implementations)

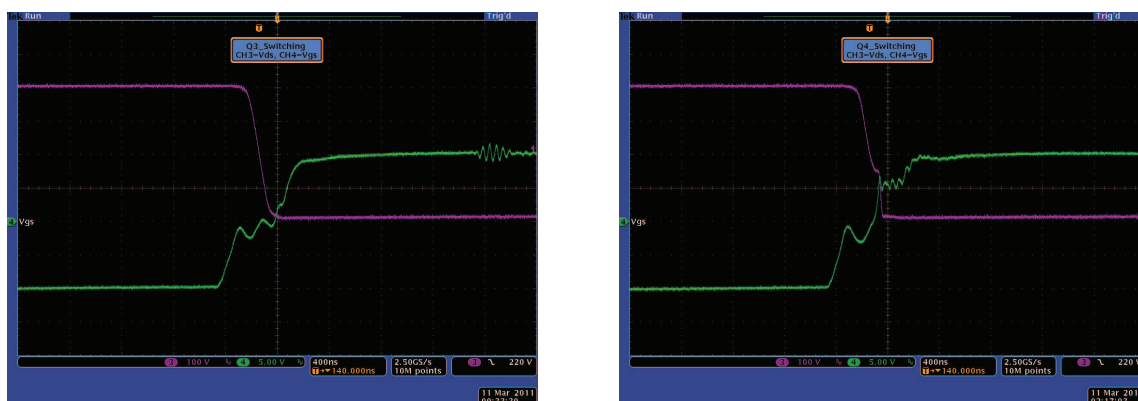


Figure 5. ZVS and LVS Switching at 12A Load (a) Active to Passive Leg Transitions (ZVS), (b) Passive to Active Leg Transitions (LVS)

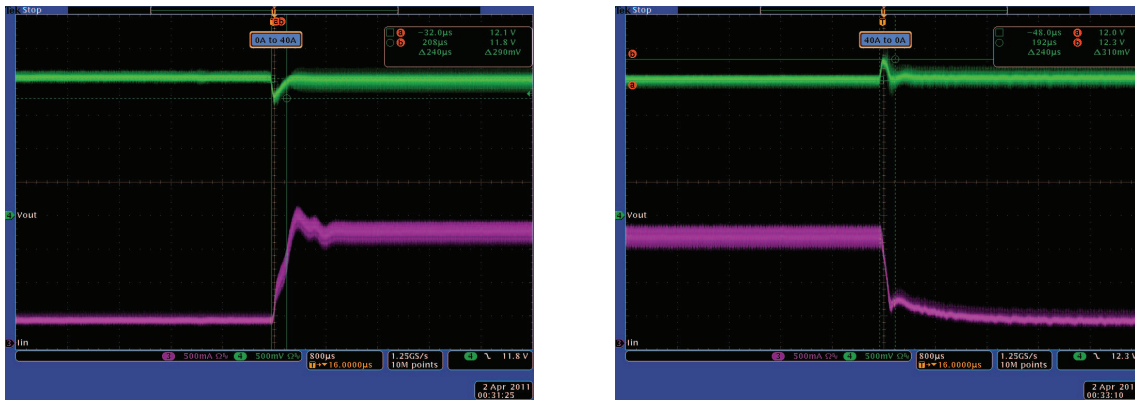


Figure 6. PCMC Transient Response (a) 0% to 80% Load Step, (b) 80% to 0% Load Step

ZVS and LVS switching are achieved across the complete operating range. System efficiency of more than 90% is obtained for all loads greater than 10% rated load, while the peak efficiency is greater than 95%. Output peak deviations of less than 3% of rated output and settling times less than 250 μ s are achieved for 80% step change in load. PCMC step response looks like a damped first order system. This C2000 MCU-based (TMS320F2802x) implementation provides an ability to generate, and control, complex gate drive waveforms required for PCMC and VMC control schemes while still providing a level of intelligence unique to digitally controlled solutions.

Figure 7 provides a snapshot of the PCMC GUI included in the software package accompanying this kit. A separate GUI for VMC implementation is also included.

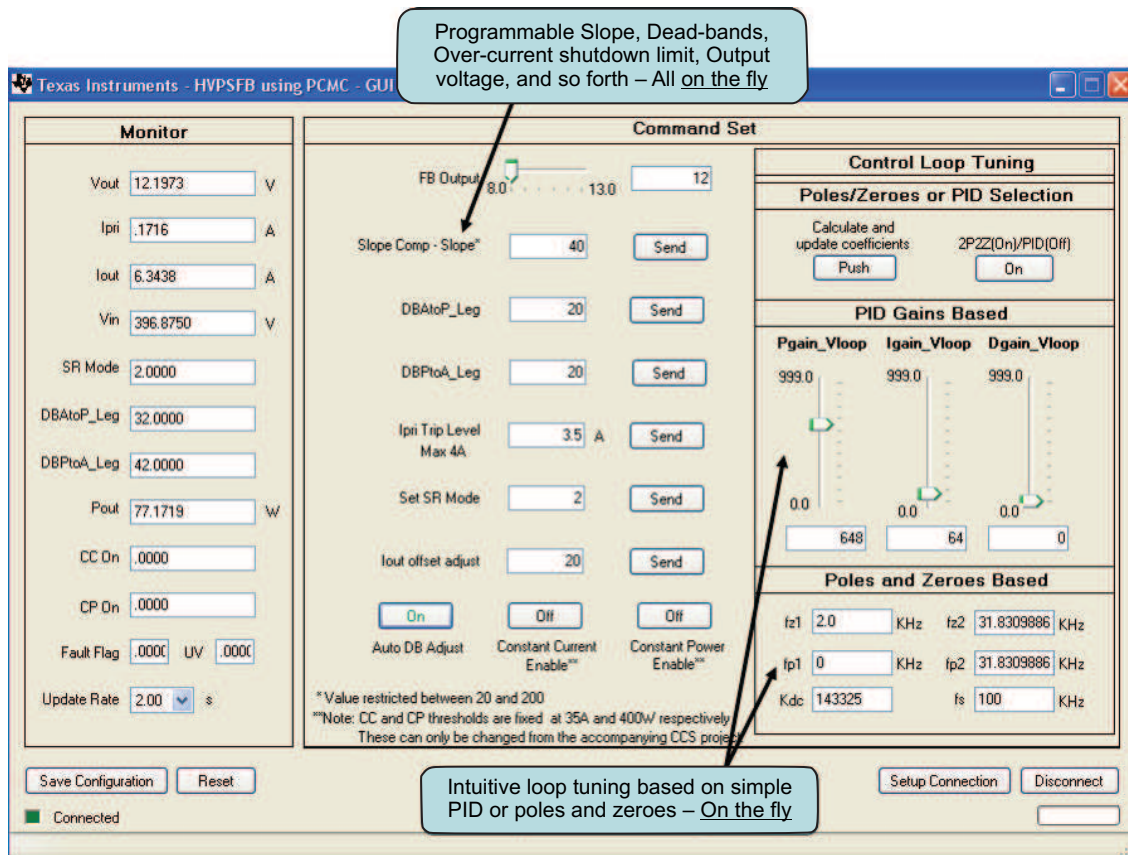


Figure 7. PCMC GUI

1.4 Identifying Key Components on the Board

Some of the key components on the actual hardware are shown in [Figure 8](#).

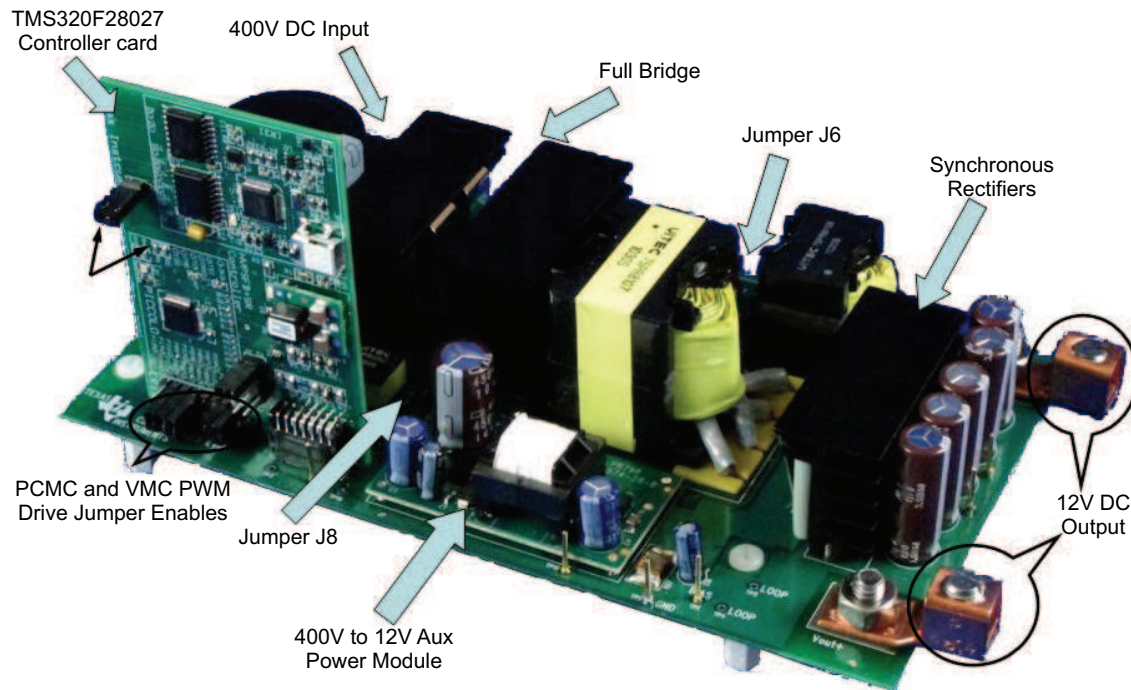


Figure 8. HVPSFB Board and Controller Card

[Figure 9](#) and [Figure 10](#) provide the schematic diagram for the Piccolo-A controller card and the HVPSFB baseboard.

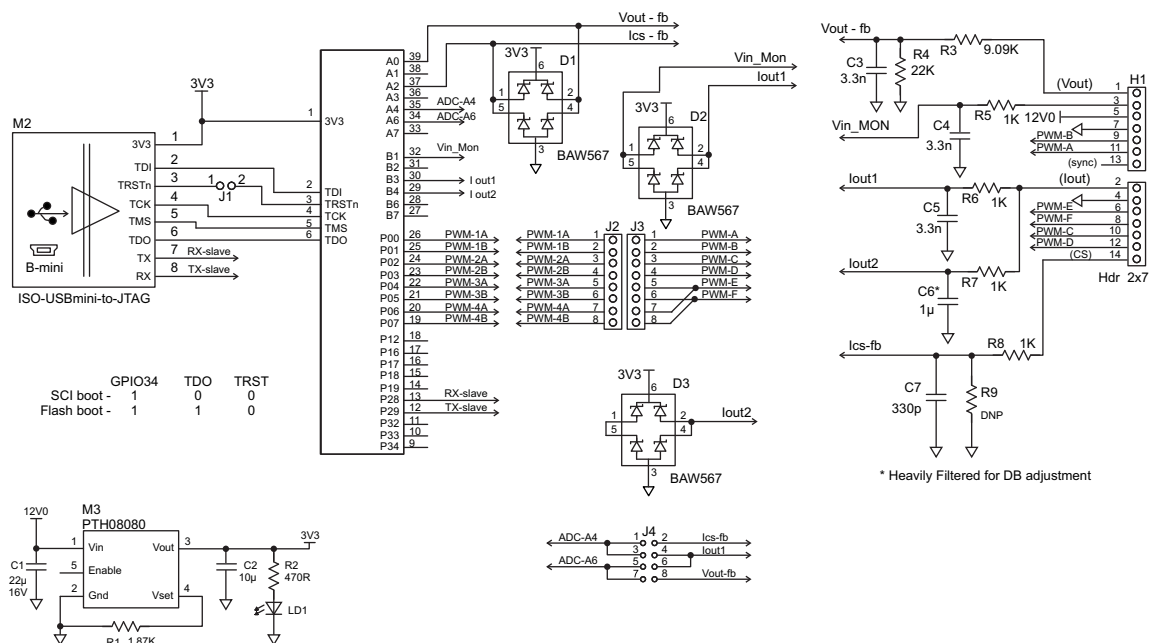


Figure 9. TMS320F28027(Piccolo-A) Controller Card Schematic

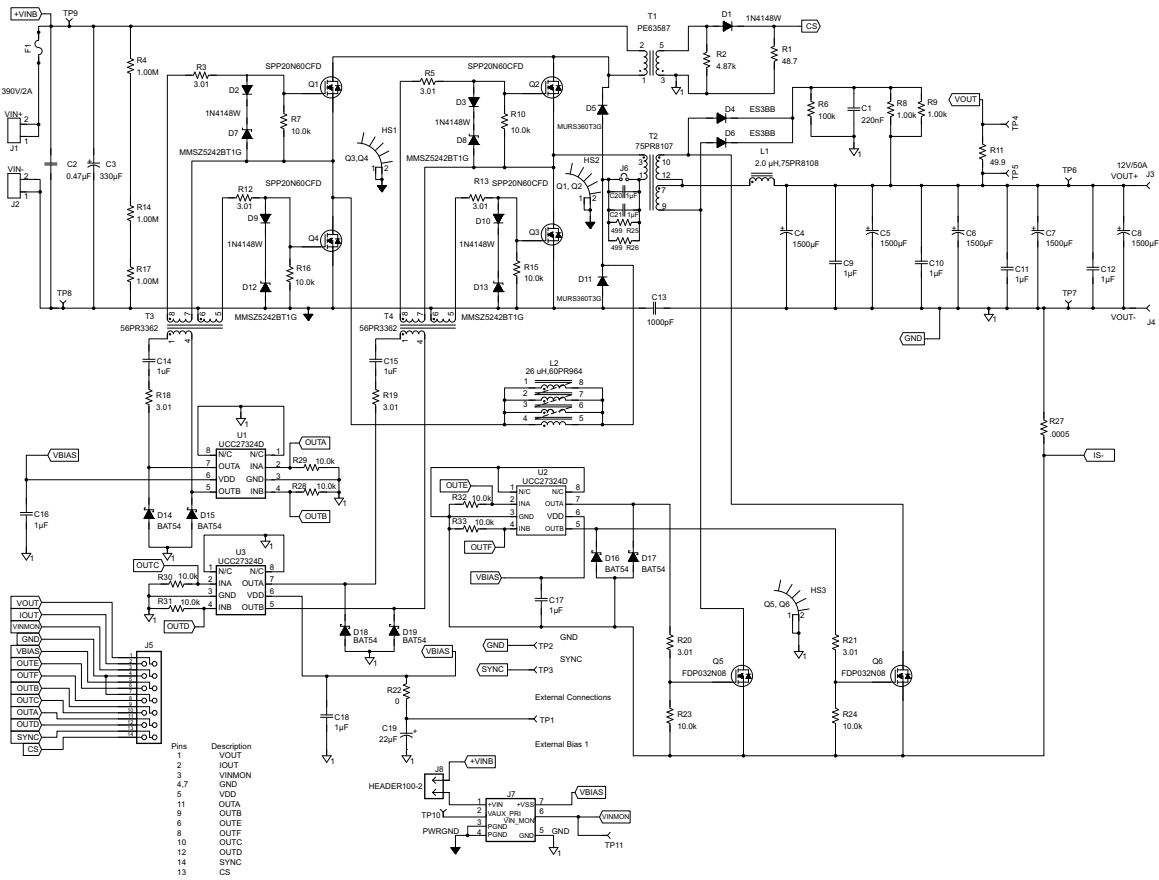


Figure 10. HVPSFB Baseboard Schematic

2 Functional Description

2.1 Peak Current Mode Control (PCMC)

Implementing PCMC for a PSFB system requires complex PWM waveform generation with precise timing control. The Piccolo family of devices from Texas Instruments feature advanced on-chip control peripherals that make this implementation possible without any external support circuitry for this purpose. These peripherals include on-chip analog comparators, DAC, advanced PWM resources and unique programmable on-chip slope compensation hardware.

Figure 11 provides a block diagram representation of the PCMC implementation. The transformer primary current is compared with the peak current reference calculated by the voltage loop using the on-chip comparator 1. As shown in Figure 12, in every half of the switching cycle when the transformer primary current reaches the commanded peak reference value, one of the PWM waveforms driving the switches (Q2 and Q3) is 'Reset' immediately ending the power transfer phase. The PWM waveform driving the other switch in the same leg is 'Set' after a programmable dead-time (dead-band) window. The appropriate slope compensation is also applied that adds a ramp with a programmable negative slope to the peak reference current signal. The 'Resetting' and 'Setting' action of the PWMs in one leg results in a phase shift between PWM signals driving the two legs. The amount of this phase shift and, thereby, the overlap between diagonal switches is dependent on the amount of peak reference current. The higher the peak reference current, the longer the overlap between diagonal switches and, thereby, the more energy transferred to the secondary. The controller regulates the output by controlling this energy transfer by way of controlling the peak reference current value. Therefore, this peak reference current is the controlled parameter.

An important feature of this implementation is that the same peak reference current command is used for both halves of the switching cycle under all operating conditions. This provides optimal flux balance for the transformer primary reducing any chances of saturation.

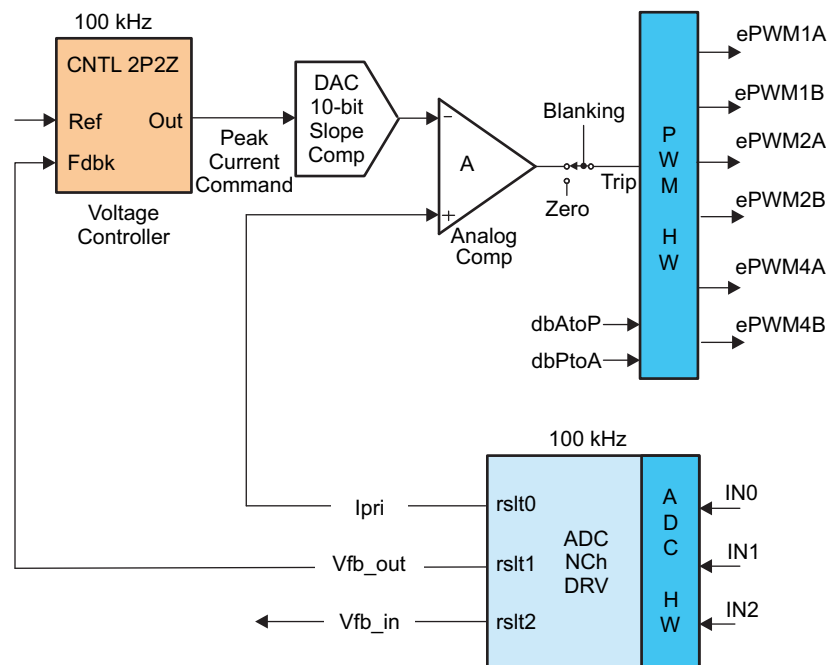


Figure 11. PCMC Block Diagram

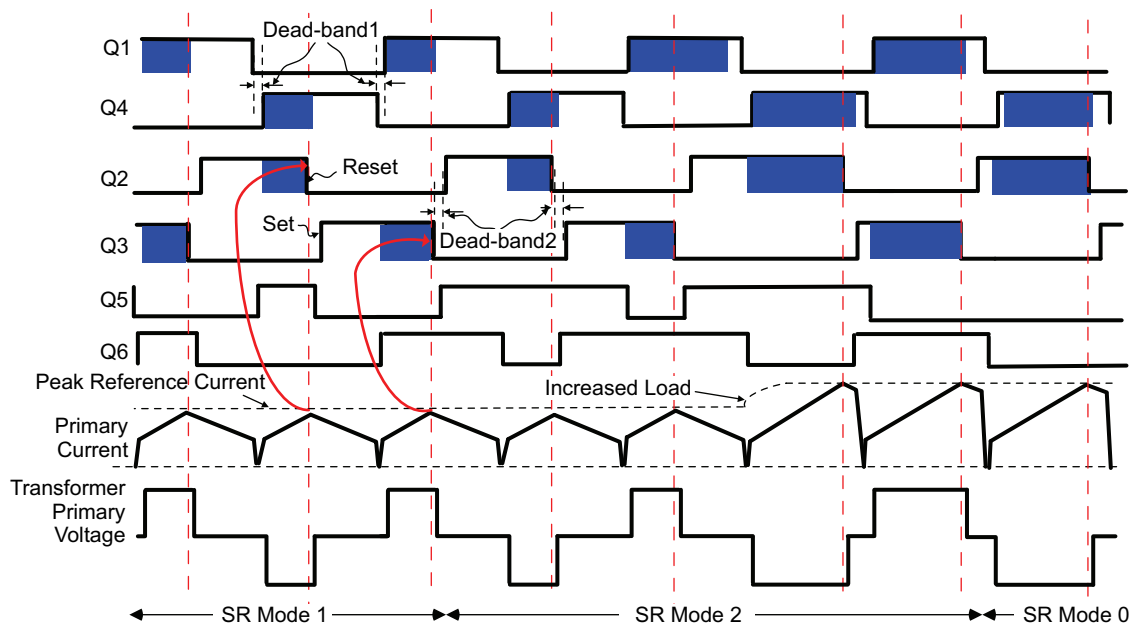


Figure 12. PCMC PWM Waveforms

2.2 Voltage Mode Control (VMC)

In the VMC implementation, switches in each leg are driven with complementary PWM signals of fixed (50%) duty cycle and frequency. As shown in [Figure 13](#), the controller directly drives and controls the phase shift of PWM signals driving switches in one leg of the bridge with respect to the ones driving switches in the other leg. This phase shift dictates the amount of overlap between diagonally opposite switches, which is clear in [Figure 14](#). The longer the overlap between diagonal switches, the longer the amount of time the input voltage is imposed across the transformer primary winding and, thereby, the larger the amount of energy transferred to the secondary. The controller regulates the output by controlling this energy transfer by way of directly controlling the phase shift between the PWM signals driving the two legs. Therefore, this phase shift is the controlled parameter. It should be noted that with VMC implementation, there is a need to include a DC blocking capacitor in the transformer primary to avoid possible transformer saturation from flux imbalance over time. Therefore, jumper J6 in [Figure 8](#) should be removed for the VMC implementation.

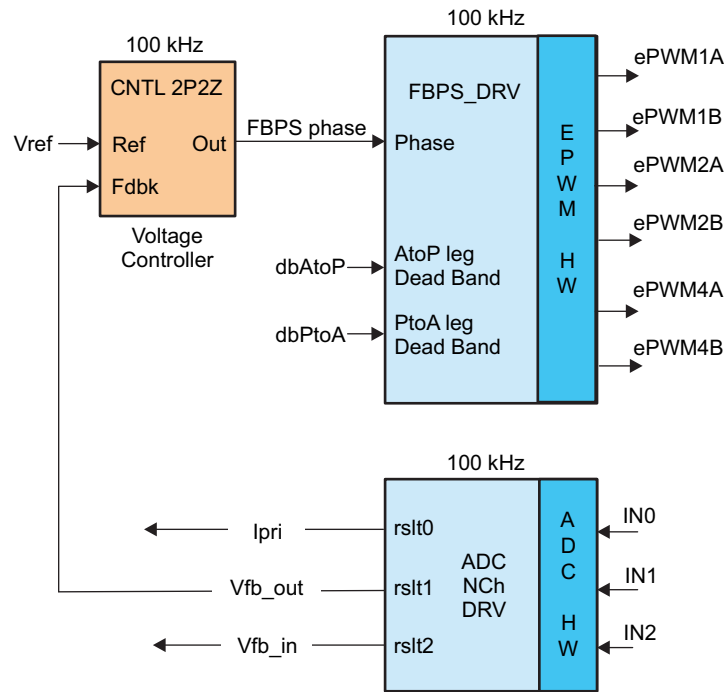


Figure 13. VMC Block Diagram

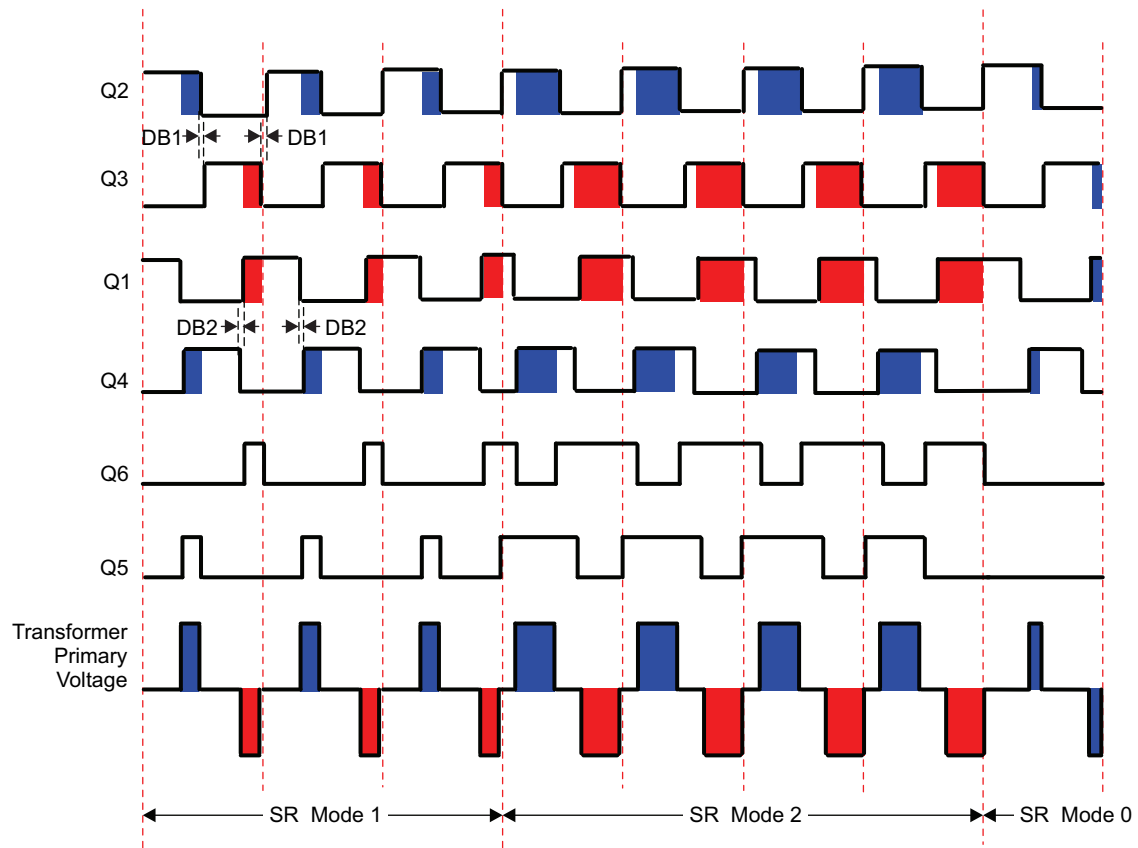


Figure 14. VMC Waveforms

2.3 Zero Voltage Switching (ZVS) or Low Voltage Switching (LVS)

PSFB DC-DC converters make use of parasitic elements in the circuit to ensure zero voltage across the MOSFET switches before turning them ON, providing soft switching. This considerably reduces the amount of switching losses associated with hard switching.

For the system discussed here, switching transitions for switches in the Q2- Q3 leg end the power transfer interval. Therefore, this leg is called the 'Active to Passive' leg. When transitions occur for switches in this leg, current in the primary winding is close to its maximum magnitude for that half PWM switching cycle. The reflected load current aids the circulating energy in the primary circuit during this time, which makes it possible for voltage across switches in this leg to approach zero volts. It is possible to achieve ZVS for switches in this Q2-Q3 leg across the complete load range. It should be noted that as the load decreases the amount of dead-time needs to be increased to achieve or approach ZVS.

Switching transitions for switches in the Q1- Q4 leg start the power transfer interval. Therefore, this leg is called the 'Passive to Active' leg. During these switching transitions, primary current decreases. It crosses zero current value and changes direction. This results in lower available energy for ZVS. In fact, for operations under low load conditions, voltage across these switches may not go to zero before turning them on. Switching losses can be kept to a minimum by turning these switches ON at a time when the voltage across them is at a minimum. This is called valley switching or low voltage switching (LVS). As the load changes, the time at which the switch should be turned on to achieve LVS changes, requiring dead-time adjustment similar to the Q2-Q3 leg switches.

2.4 Synchronous Rectification

Synchronous rectifiers can work in one of the following three modes at any given time:

- Mode 0: This is the classical diode current doubler mode achieved by keeping synchronous rectifiers turned OFF. It is useful for very low load operations where synchronous rectifier switching losses are greater than the power savings obtained by synchronous rectification.
- Mode 1: In this mode, the synchronous rectifier switches behave like ideal diodes. This mode is useful when operating at very low to low loads, typically when burst mode is being used. In this mode, synchronous rectifier MOSFETs are ON only when the corresponding diagonal bridge drive signals overlap.
- Mode 2: Useful for all other load conditions. In this mode, synchronous rectifier MOSFETs are OFF only when the corresponding opposite diagonal bridge drive signals overlap.

Figure 12 and Figure 14 depict waveforms generated for driving the synchronous rectifier switches in these modes. It is important to implement mode transitions seamlessly without any glitches or anomalies on the PWM outputs even during large load transients or sudden phase shift change commands to ensure safe operation of the system.

2.5 Output Voltage Regulation With Changing Load

The regulated output voltage is affected by changing load conditions. This problem exists because of ripple on the output voltage that appears at two times the switching frequency. At low loads, peak-to-peak voltage ripple is relatively small, while at higher loads it increases considerably. If the ADC conversions are triggered at a fixed point within a switching cycle, as shown in Figure 15, the sensed analog-to-digital converter (ADC) voltage result is smaller at high loads (for the same average voltage).

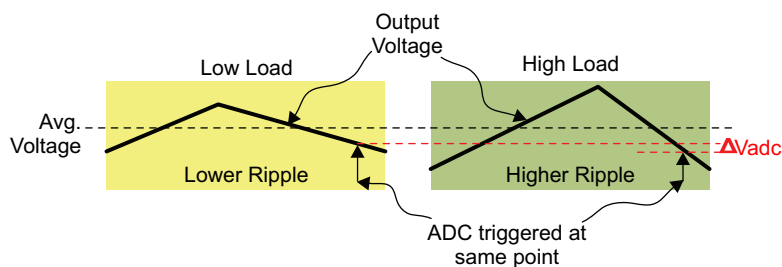


Figure 15. Effect of Fixed-Point ADC Conversion Triggering for Output Voltage Sensing

Therefore, for the same average output voltage, a smaller output voltage is seen by the controller under high load operation. The controller compensates for this ΔV_{adc} , which results in an increase in the actual output voltage value as the load increases. A few possible solutions to minimize this effect are suggested here.

- Start of ADC conversion can be triggered at an appropriate time in every half PWM switching cycle so as to directly sense the average output voltage. With PCMC implementation, duty cycle is not pre-determined and this trigger point is unknown. This method works very well with VMC.
- Average output voltage can be calculated at a slow rate and an outer slower loop can be used to adjust voltage reference or feedback. However, this may affect the dynamic performance.
- The average output voltage can be calculated on a cycle-by-cycle basis by over-sampling it over one or two ripple cycles. Since the average output voltage is calculated over a full ripple cycle, any effects from higher or lower peak-to-peak ripple are avoided. Also, since the average is calculated in one or two ripple cycles and used for the next PWM switching cycle, the dynamic behavior and control loop performance do not suffer much. With this method multiple ADC conversions are required within a single PWM half cycle. This is the method recommended here for PCMC implementation. In this implementation the output voltage is oversampled eight times within a PWM switching cycle.
- Increased filtering at the ADC input can attenuate the ripple and reduce ΔV_{adc} . However, this affects system dynamic performance and achievable loop bandwidth, while the output voltage behavior still remains the same.

3 Software Overview - PCMC

3.1 Software Control Flow

The HVPSFB_PCMC project makes use of the “C-background/ASM-ISR” framework. It uses C-code as the main supporting program for the application, and is responsible for all system management tasks, decision making, intelligence, and host interaction. The assembly code is strictly limited to the interrupt service routine (ISR), which runs all the critical control code and typically this includes ADC reading, control calculations, and PWM and DAC updates. Figure 16 depicts the general software flow for this project.

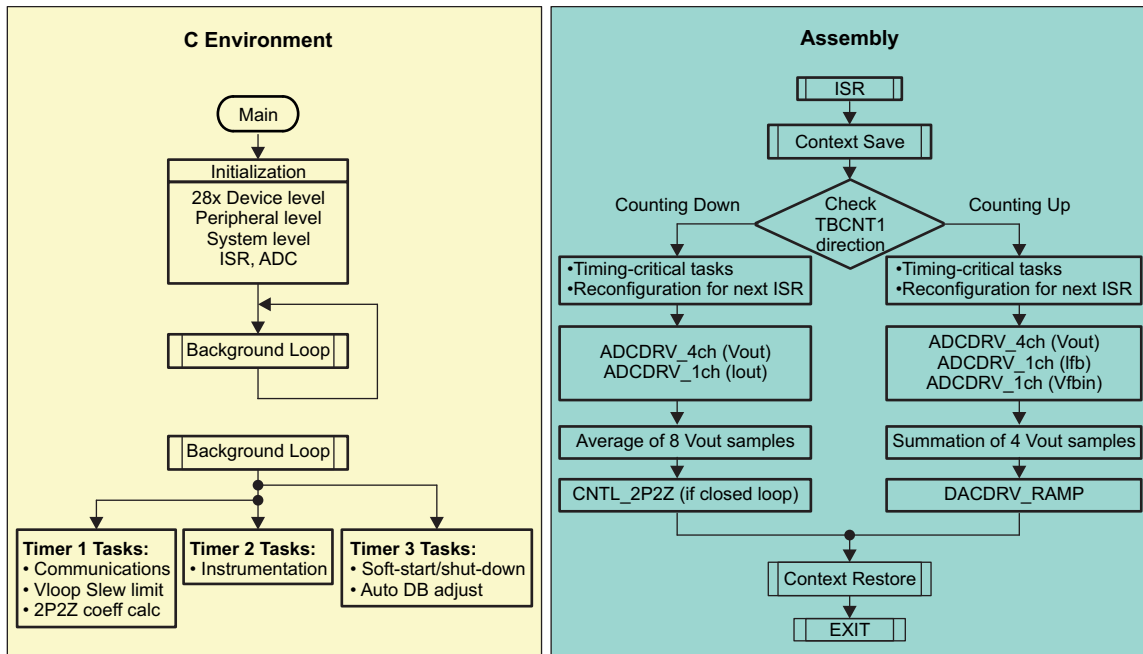


Figure 16. PCMC Software Flow

The key framework C files used in this project are:

- *HVPSFB-Main.c* – This file is used to initialize, run, and manage the application. This is the “brains” behind the application.
- *HVPSFB-DevInit.c* – This file is responsible for a one time initialization and configuration of the microcontroller, and includes functions such as setting up the clocks, phase-locked loop (PLL), general-purpose input/output (GPIO), and so forth.

The ISR consists of a single file:

- *HVPSFB-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 twice the PWM switching frequency.

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

Library modules may have both a C and an assembly component. In this project, the following library modules are used. The C and corresponding assembly module names are listed in [Table 1](#).

Table 1. Library Modules

C Configure Function	ASM Initialization Macro	ASM Run-Time Macro
DAC_Cnf.c	DACDRV_RAMP_INIT n	DACDRV_RAMP n
ADC_SOC_Cnf.c	ADCDRV_4CH_INIT m,n,p,q	ADCDRV_4CH m,n,p,q
ADC_SOC_Cnf.c	ADCDRV_1CH_INIT n	ADCDRV_1CH n
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

The control blocks are also represented graphically as shown in [Figure 17](#).

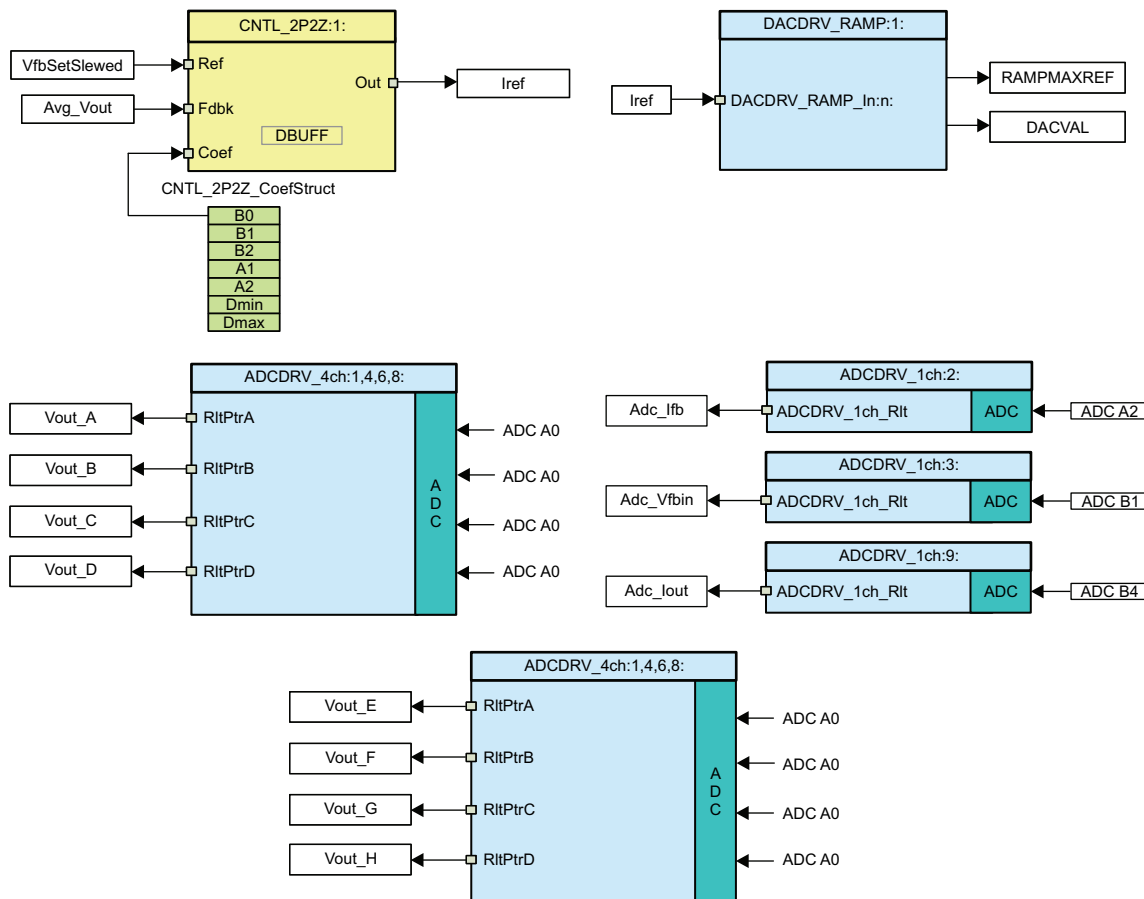


Figure 17. PCMC Software Blocks

Notice the color coding for modules in Figure 17. Blocks in 'dark blue' represent hardware modules on the C2000 microcontroller. 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 and PID, a 3-pole 3-zero or any other controller that can be suitably implemented for this application. Such a modular library structure makes it convenient to visualize and understand the complete system software flow as shown in Figure 18. It also allows for easy use and additions and 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.

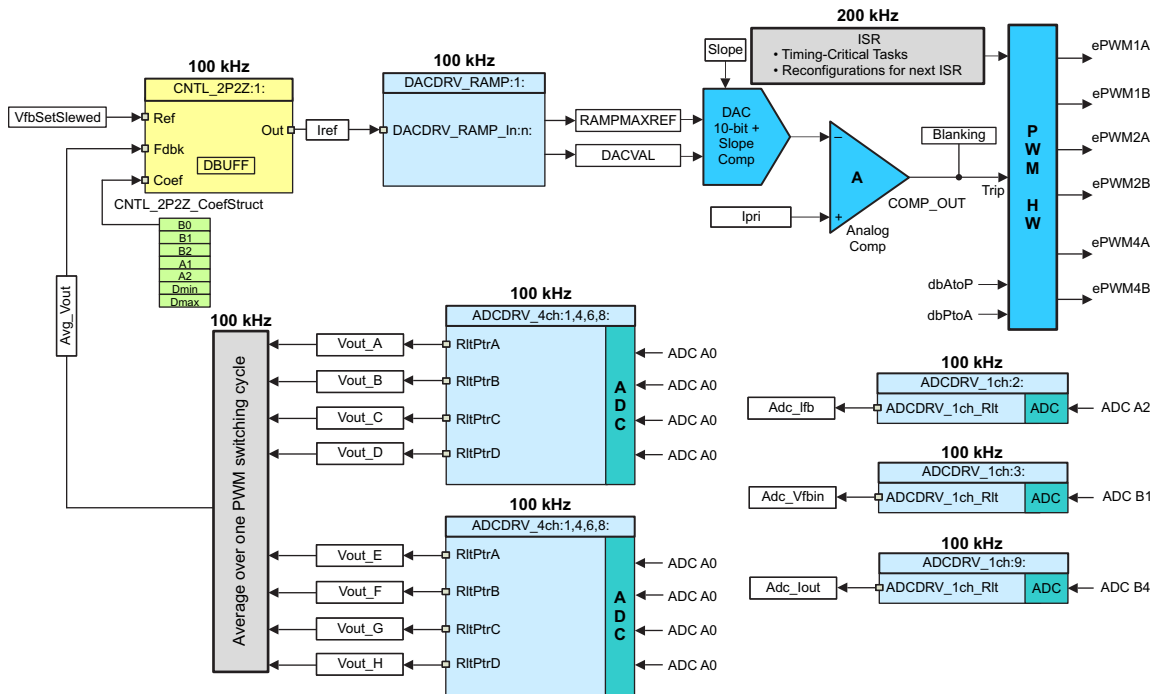


Figure 18. PCMC Control Flow

The system is controlled by two feedback loops: an outer voltage loop (implemented using the software control block) and an inner peak current loop (implemented using the on-chip analog comparator) DAC and PWM hardware resources. Figure 18 also gives the rate at which control blocks are executed. For example, the voltage loop controller is executed at a rate of 100 kHz (same as the PWM frequency). Note that the interrupt service routine (ISR) is executed at twice the PWM frequency. The following explains the control implemented above.

Output voltage is sampled at four points, equally separated from each other in time, in each half of the PWM switching cycle. The average (*Avg_Vout*) of these output voltage samples is calculated over one PWM cycle. *Avg_Vout* is compared with slewed version (*VfbSetSlewed*) of the voltage reference command (*Vref*) in the voltage controller. The voltage controller output is then translated to an appropriate DAC command. This is the peak current reference command that eventually dictates the amount of phase overlap between the two legs of the full bridge to regulate the output voltage. *Slope* value for the slope compensation mechanism can be programmed from outside the control flow in slower system level tasks. A default slope of at least 0.04 V/ μ s is provided. For more details, see the calculations provided in [1], [2] and [5]. The on-chip analog comparator compares the transformer primary current with the slope compensated peak current reference. The comparator output directly drives the PWM hardware on the device. Some time critical configuration code is executed inside the ISR, which is triggered twice in one PWM cycle. The *dbAtoP* and *dbPtoA* parameters provide dead band values for the 'Active to Passive' and 'Passive to Active' legs of the full bridge, respectively. These are used to achieve ZVS and LVS across the load range.

3.2 Incremental Builds

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

The build options are shown in the following sections. To select a particular build option:

1. Set the macro INCR_BUILD, found in the *HVPSFB-Settings.h* file, to the corresponding build selection as shown in [Table 2](#).
2. Compile the complete project by selecting → rebuild-all compiler option, once the build option is selected.

The next section provides more details to run each of the build options.

Table 2. Incremental Build Options for PCMC

Incremental Build Options	
INCR_BUILD = 1	Peak current loop check with constant I command and open voltage loop (check PWM drive circuit and sensing circuit)
INCR_BUILD = 2	Closed current and voltage loop (full PSFB)

4 Procedure for Running the Incremental Builds - PCMC

The main source files, ISR assembly file and the project file for C framework to bring up the PSFB system, are located in the following directory (please use the latest version of the software package). Version 1.1 is the latest software version as of March 2012, which is located at www.ti.com/controlsuite ...\controlSUITE\development_kits\HVPSFB_v1.1\HVPSFB_PCMC. The projects included with this software are targeted for Code Composer Studio™ v4.

WARNING

There are high voltages present on the board. It should only be handled by experienced power supply professionals in a lab environment. To safely evaluate this board, an appropriate isolated high voltage DC source should be used. Before DC power is applied to the board, a voltmeter and an appropriate resistive or electronic load must be attached to the output. The unit should never be handled when the power is applied to it.

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

4.1 **Build 1: Peak Current Loop Check With Open Voltage Loop**

4.1.1 **Objective**

The objective of this build is to evaluate the peak current mode operation of the system, verify the DAC and ADC driver modules, verify the MOSFET driver circuit and sensing circuit on the board and become familiar with the operation of Code Composer Studio. Since this system is running open-loop, the ADC measured values are only used for instrumentation purposes in this build. Steps required to build and run a project are explored.

4.1.2 **Overview**

The software in Build1 has been configured so that you can quickly evaluate the DAC driver module by viewing various waveforms on an oscilloscope and observing the effect of change in peak current reference command on the output voltage by interactively adjusting this command from Code Composer Studio. Additionally, you can evaluate the ADC driver module by viewing the ADC sampled data in the watch view.

The DAC and ADC driver macro instantiations are executed inside the `_DPL_ISR` as mentioned in the previous section. [Figure 19](#) shows the blocks used in this build. The peak current reference command is written to the `DACDRV_RAMP` module. This module derives an appropriate 16-bit value Ramp Maximum Reference Register (RAMPMAXREF), which is the starting value of the RAMP used for slope compensation. This module also drives an appropriate 10-bit value to the DAC Value Register (DACVAL), which can be used if slope compensation is not needed or provided externally.

The on-chip analog comparator compares the transformer primary current with the slope compensated peak current reference. Comparator output is connected to the trip zone logic of the PWM modules. ePWM1 module acts as the master time-base for the system. It operates in up-down count mode, while other PWM modules operate in up-count mode. ePWM1A and ePWM1B drive Q1 and Q4 full-bridge switches, while ePWM2A and ePWM2B drive Q2 and Q3 full-bridge switches. ePWM4A and ePWM4B drive Q5 and Q6 synchronous rectifier switches. Whenever the comparator output goes high in a PWM half cycle, the ePWM2 module output (ePWM2A or ePWM2B), which was high at that instant, is immediately pulled low while the other PWM2 module output is pulled high after an appropriate dead-band window (*dbAtOP*). ePWM4A and ePWM4B outputs are driven in a similar way. These waveforms are shown in [Figure 12](#).

It should be noted that this slope compensation ramp generation, comparator action and PWM waveform generation are all hardware generated without any software involvement as denoted by the dark blue blocks in Figure 19. Some register reconfigurations are done inside the ISR in preparation for the next half of the PWM cycle. This time critical code executes at the start of the ISR and should not be re-ordered or changed. The assembly ISR_DPL_ISR routine is triggered by ePWM1. Note that the ISR trigger frequency is twice that of the PWM switching frequency.

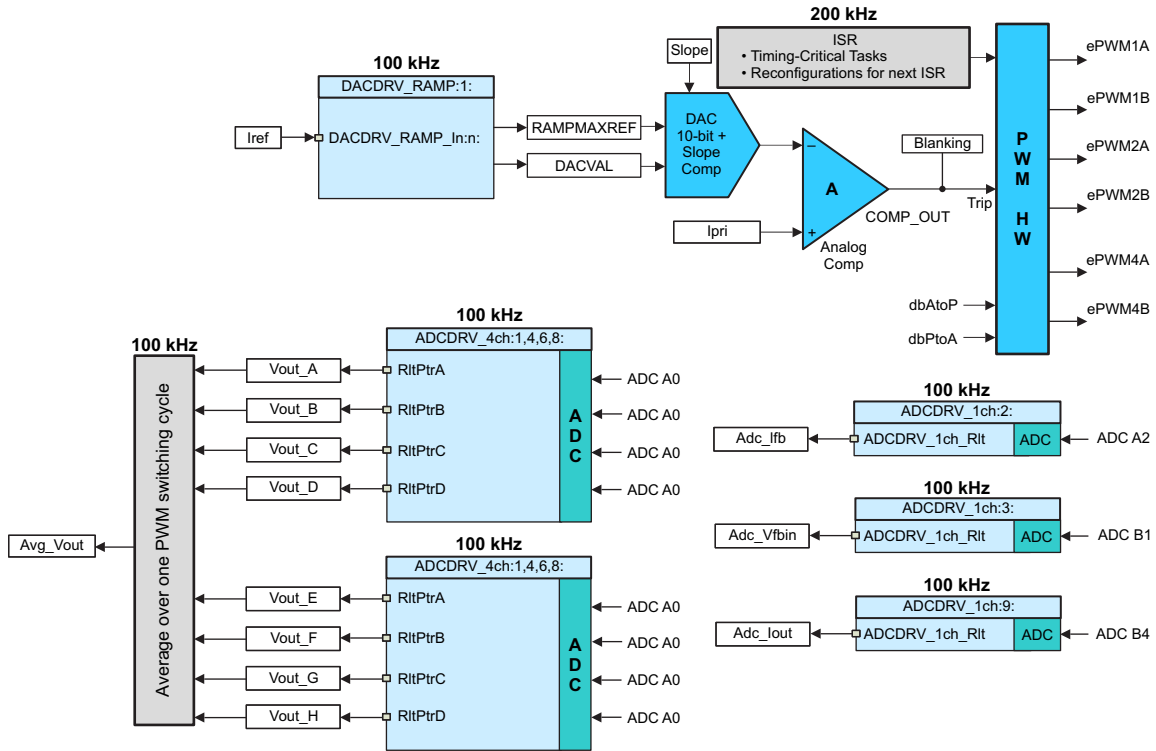


Figure 19. Build 1 Software Blocks

The RAMPMAXREF and DACVAL values are derived from the input Iref (Q24 variable) command. Table 3 gives example RAMPMAXREF and DACVAL values calculated from Iref.

Table 3. Example RAMPMAXREF and DACVAL Values

Iref (Q24)	RAMPMAXREF = (Iref/2^8)	DACVAL = (Iref /2^14)	Peak Current Reference (Max = 6.776A)
2097152d	8192	128	0.845A
8388608d	32768	512	3.39A
16776704d	65534	1023	6.77A

The ADC driver modules are used to read 12-bit ADC results and convert them to Q24 values. In every half PWM cycle PWM1 start of conversion A (SOCA), PWM3 SOCA and PWM3 start of conversion B (SOCB) are used to trigger four ADC conversions for the output voltage. These conversion trigger points are equally separated from each other in time. These four results are read in every ISR, therefore, providing a total of eight ADC conversion results of the output voltage within every PWM cycle. A few lines of code in the ISR are used to calculate the average output voltage over one PWM cycle based on these eight results.

The output voltage sensing circuit is made up of simple voltage dividers. A current transformer, with a turns-ratio of 1:100, and a sense resistor (48.7 Ω) are used to sense the full-bridge transformer primary current. The input voltage sensing is provided by auxiliary power module (J7). For details on these calculations, see the HVPSFB-Calculations.xls file located at www.ti.com/controlsuite.

4.1.3 Protection

At this stage, it is appropriate to introduce the shutdown mechanism used on this board. Here overcurrent protection is implemented for the transformer primary current using on-chip analog comparator 2. The reference trip level is set using the internal 10-bit DAC and fed to the inverting terminal of this comparator. The comparator outputs are configured to generate a one-shot trip action on ePWM1 whenever the sensed current is greater than the set limit. The flexibility of the trip mechanism on C2000 devices provides the possibilities for taking different actions on different trip events. In this project, ePWM1A and ePWM1B outputs are driven low immediately to protect the power stage. ePWM2A, ePWM2B, ePWM4A and ePWM4B outputs are then forced low from software. All outputs are held in this state until a device reset is executed. For details on these calculations, see the HVPSFB-Calculations.xls file located at www.ti.com/ctrlsuite.

An output under voltage shut down mechanism is also implemented from the software when operating under constant current or constant power modes.

Input under and over voltage lockout is also implemented in the software.

4.1.4 Resource Mapping

The key signal connections between the C2000 MCU and the HVPSFB stage are summarized in [Table 4](#). Note that PWM mapping is different between VMC and PCMC projects. Jumper (J2, J3 – PCMC and VMC PWM drive jumper enables) on the controller card need to be correctly configured for VMC mode (default jumper positions are set for PCMC operation). Here are the jumper configurations for the two modes:

- PCMC: J2(1) → J3(1), J2(2) → J3(2), J2(3) → J3(3), J2(4) → J3(4), J2(7) → J3(7), J2(8) → J3(8)
- VMC: J2(1) → J3(3), J2(2) → J3(4), J2(3) → J3(1), J2(4) → J3(2), J2(7) → J3(8), J2(8) → J3(7)

Table 4. HVPSFB Signal Interface Reference - PCMC

Signal Name	Description	Connection to C2000 Controller
ePWM-1A	PWM drive for full-bridge switch Q1	GPIO-00
ePWM-1B	PWM drive for full-bridge switch Q4	GPIO-01
ePWM-2A	PWM drive for full-bridge switch Q2	GPIO-02
ePWM-2B	PWM drive for full-bridge switch Q3	GPIO-03
ePWM-4A	PWM drive for synchronous rectifier switch Q5	GPIO-06
ePWM-4B	PWM drive for synchronous rectifier switch Q6	GPIO-07
Vout	PSFB output voltage	ADC-A0
lfb	Transformer primary current	ADC-A2 and COMP1A
lfb	Transformer primary current	ADC-A4 and COMP2A ⁽¹⁾
Vfbn	PSFB input voltage	ADC-B1
lout1	PSFB output current	ADC-B3
lout2	PSFB output current (heavily filtered)	ADC-B4

⁽¹⁾ Default jumper J4 configuration on controller card. Input is configurable by jumper J4 selection.

NOTE: The HVPSFB_PCMC and HVPSFB_VMC projects use *DSP2802x_Comp.h* and *DSP2802x_EPWM.h* header files that are located in their own project directories instead of the ones located in the *device_support* directory. The two files in the *device_support* directory will be updated at a later date of *ControlSuite*, at which time these updated files can be used with the two projects.

4.2 Procedure

4.2.1 Start Code Composer Studio and Open a Project

Use the following steps to quickly execute this build:

1. Make sure that jumpers J8 and J6 on the baseboard are populated.
2. By default, resistor R6 on the Piccolo macro of the controller card and jumper J1 are removed to enable boot from FLASH. Re-populate these two to run and program RAM or program FLASH.
3. Connect the USB connector to the Piccolo controller card for emulation. Use of an appropriate isolated DC power supply set to output around 400 V DC is recommended. The DC power supply should remain off when it is connected to J1 and J2 on the main board.
4. Use a 20AWG 600 V wire to connect the power source to J1 and J2. Make sure that polarity of this connection is correct. Apply an appropriate resistive or DC electronic load to the phase shifted full bridge system at the DC output at J3 and J4.
5. Do not turn on 400 V DC power at this time.
6. Power up the bias supply between TP1 and TP2 with around 11 V DC (this voltage must be less than 12 V).
7. Double click on the Code Composer Studio icon on the desktop.
8. Maximize Code Composer Studio to fill your screen.
9. Close the Welcome screen if it opens up. The project contains all the files and build options needed to develop an executable output file (.out), which can be run on the MCU hardware.
10. Click → Select Project → Import Existing Code Composer Studio and CCE Eclipse Project, on the menu bar. Under Select root directory → navigate to and select the `..\controlSUITE\development_kits\HVPSFB_v1.1\HVPSFB_PCMC` directory. Make sure that HVPSFB_PCMC is checked under the Projects tab.
11. Click Finish. This project invokes all the necessary tools (compiler, assembler, linker) to build the project.
12. Click the plus sign (+) to the left of Project (in the project window on the left). Your project window will look like the following graphic.

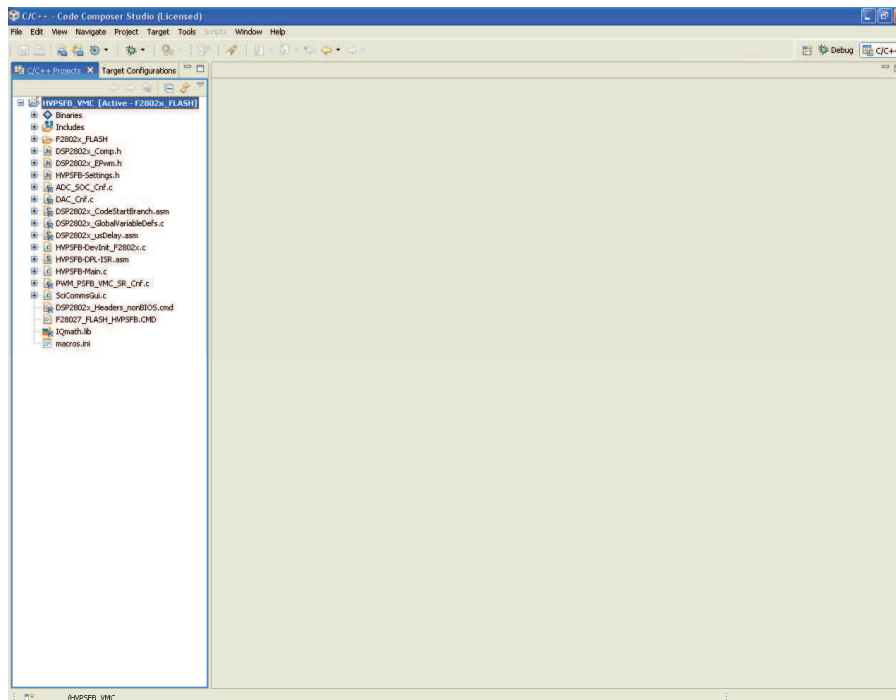


Figure 20. C and C++ Projects

4.2.2 Device Initialization, Main, and ISR Files

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

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

```
DacDrvCnf (1, 1280, 1, 2, Slope); // Comp1, DACval = 1280 (Initial), Slope compensation
is used,
// Ramp is PWM3 Synced, Initial Slope
// DAC connections
DACDRV_RAMP_In1 = &Iref; // Controls the DAC reference voltage
```

4. Locate and inspect the following code in the main file under initialization code specific for build 1. This is where the ADCDRV_4CH and multiple instantiations of ADCDRV_1CH blocks are configured, initialized, and connected in the control flow.

```
#define Vfb_outR AdcResult.ADC RESULT1 //
#define IfbR AdcResult.ADC RESULT2 //
#define Vfb_inR AdcResult.ADC RESULT3 //

#define IoutR AdcResult.ADC RESULT9 //

// Channel Selection for Cascaded Sequencer
ChSel [0] = 0; // A0 - O/P Voltage - Dummy
ChSel [1] = 0; //B // A0 - O/P Voltage
ChSel [2] = 2; // A2 - Transformer Primary Current
ChSel [3] = 9; // B1 - I/P Voltage
ChSel [4] = 0; //C // A0 - O/P Voltage

ChSel [5] = 0; // A0 - O/P Voltage - Dummy
ChSel [6] = 0; //A // A0 - O/P Voltage
ChSel [7] = 0; // A0 - O/P Voltage - Dummy
ChSel [8] = 0; //D // A0 - O/P Voltage
ChSel [9] = 11; // B3 - Iout1
ChSel [9] = 12; // B4 - Iout2

TrigSel[0] = ADCTRIG_EPWM3_SOCA; // O/P Voltage sampling triggered by EPWM3 SOCA -
Dummy
TrigSel[1] = ADCTRIG_EPWM3_SOCA; //B // O/P Voltage sampling triggered by EPWM3 SOCA
TrigSel[2] = ADCTRIG_EPWM3_SOCA; // Transformer Primary Current sampling triggered
by EPWM3 SOCA
TrigSel[3] = ADCTRIG_EPWM3_SOCA; // I/P Voltage sampling triggered by EPWM3 SOCA
TrigSel[4] = ADCTRIG_EPWM3_SOCA; //C // O/P Voltage sampling triggered by EPWM3 SOCA
TrigSel[5] = ADCTRIG_EPWM1_SOCA; // O/P Voltage sampling triggered by EPWM1 SOCA at
CTR = ZRO or PRD - Dummy
TrigSel[6] = ADCTRIG_EPWM1_SOCA; //A // O/P Voltage sampling triggered by EPWM1 SOCA at
CTR = ZRO or PRD
TrigSel[7] = ADCTRIG_EPWM3_SOCA; // O/P Voltage sampling triggered by EPWM3 SOCA at
CMPB3 - Dummy
TrigSel[8] = ADCTRIG_EPWM3_SOCA; //D // O/P Voltage sampling triggered by EPWM3 SOCA at
CMPB3
TrigSel[9] = ADCTRIG_EPWM2_SOCA; // Iout triggered by EPWM2 SOCA
EALLOW;
AdcRegs.SOCPRCTL.bit.SOCPRIORITY = 9; // SOC0-8 are high priority
EDIS;
ADC_SOC_CNF(ChSel,TrigSel,ACQPS, 16, 0); // ACQPS=8, No ADC channel triggers an interrupt
IntChSel > 15,
// Mode= Start/Stop (0)
```

```

// ADC feedback connections
ADCDRV_4ch_RltPtrA = &Adc_VavgBus[1];
ADCDRV_4ch_RltPtrB = &Adc_VavgBus[2];
ADCDRV_4ch_RltPtrC = &Adc_VavgBus[3];
ADCDRV_4ch_RltPtrD = &Adc_VavgBus[4];
ADCDRV_1ch_Rlt2 = &Adc_Ifb;
ADCDRV_1ch_Rlt3 = &Adc_VfbIn;
ADCDRV_1ch_Rlt9 = &Adc_Iout;

```

5. Open and inspect HVPSFB-DPL-ISR.asm. Notice the `_DPL_Init` and `_DPL_ISR` sections. This is where the DAC and ADC driver macro instantiation is done for initialization and runtime, respectively. Optionally, you can close the inspected files.

4.2.3 Build and Load the Project

1. Select the Incremental build option as 1 in the *HVPSFB-Settings.h* file located at www.ti.com/controlsuite.

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

2. Click the Project → "Rebuild All" button and watch the tools run in the build window.
3. Click Target → "Debug Active Project". Code Composer Studio will ask you to open a new Target configuration file if one has not already been selected. If a valid target configuration file has been created for this connection, you can go to Step 5. Type in the name of the .ccxml file for the target you will be working with (example: xds100-F28027.ccxml), in the New target Configuration Window. Check "Use shared location" and click Finish.
4. In the .ccxml file that opens up, select the Connection as "Texas Instruments XDS100v2 USB Emulator" and scroll down under the device and select "TMS320F28027".
5. Click Save.
6. Click Target → "Debug Active Project". The program will be loaded into the FLASH or RAM memory depending on the project configuration selected. This project comes in only F2802x_FLASH configuration. You should now be at the start of Main().

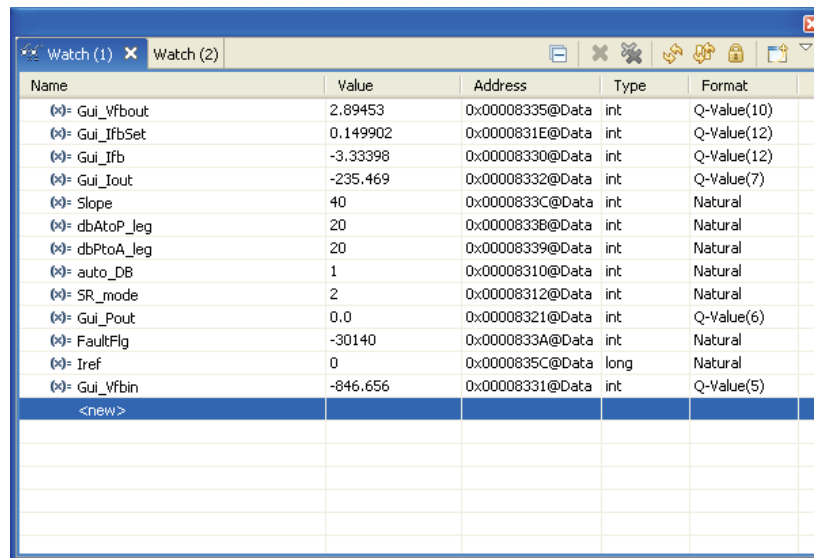
4.2.4 Debug Environment Windows

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

If a watch view did not open when the debug environment was launched, open a new watch view and add various parameters to it by following the procedure given below.

1. Click View → Watch on the menu bar.
2. Click the "Watch (1)" tab. You may add any variables to the watch view.
3. Type the symbol name of the variable you want to watch in the empty box in the "Name" column and press enter on keyboard. Be sure to modify the "Format" as needed. The watch view should look something like the following. Please note that some of the variables have not been initialized at this point in the main code and may contain some garbage values.

FaultFlg, if set, indicates an over current condition (discussed above), which shuts down the PWM outputs. PWM outputs are held in this state until a device reset (follow the proper procedure in Step 11). The *Ipri_trip* variable sets the internal 10-bit DAC reference level for the on-chip comparator 2. Please note that this is a Q15 number.



Name	Value	Address	Type	Format
Gui_Vfbout	2.89453	0x00008335@Data	int	Q-Value(10)
Gui_IfbSet	0.149902	0x0000831E@Data	int	Q-Value(12)
Gui_Ifb	-3.33398	0x00008330@Data	int	Q-Value(12)
Gui_Iout	-235.469	0x00008332@Data	int	Q-Value(7)
Slope	40	0x0000833C@Data	int	Natural
dbAtOP_leg	20	0x0000833B@Data	int	Natural
dbPtoA_leg	20	0x00008339@Data	int	Natural
auto_DB	1	0x00008310@Data	int	Natural
SR_mode	2	0x00008312@Data	int	Natural
Gui_Pout	0.0	0x00008321@Data	int	Q-Value(6)
FaultFlg	-30140	0x0000833A@Data	int	Natural
Iref	0	0x0000835C@Data	long	Natural
Gui_Vfbin	-846.656	0x00008331@Data	int	Q-Value(5)
<new>				

4.2.5 Using Real-Time Emulation

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

1. Enable real-time mode by hovering your mouse on the buttons on the horizontal toolbar and clicking



the *Enable Silicon Real-Time Mode* (service critical interrupts when halted, allow debugger accesses while running) button.

2. Select YES to enable debug events, if a message box appears. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a “0”. The DGBM is the debug enable mask bit. When the DGBM bit is set to “0”, memory and register values can be passed to the host processor for updating the debugger windows.
3. Now click the *Enable Polite Real-Time Mode* button on the same horizontal toolbar.
4. When a large number of windows are open, as bandwidth over the emulation link is limited, updating too many windows and variables in continuous refresh can cause the refresh frequency to bog down.



Right click on the button in the watch view and select “*Customize Continuous Refresh Interval..*”. You can slow down the refresh rate for the watch view variables by changing the *Continuous refresh interval (milliseconds)* value. A rate of 4000 ms is usually enough for these exercises.

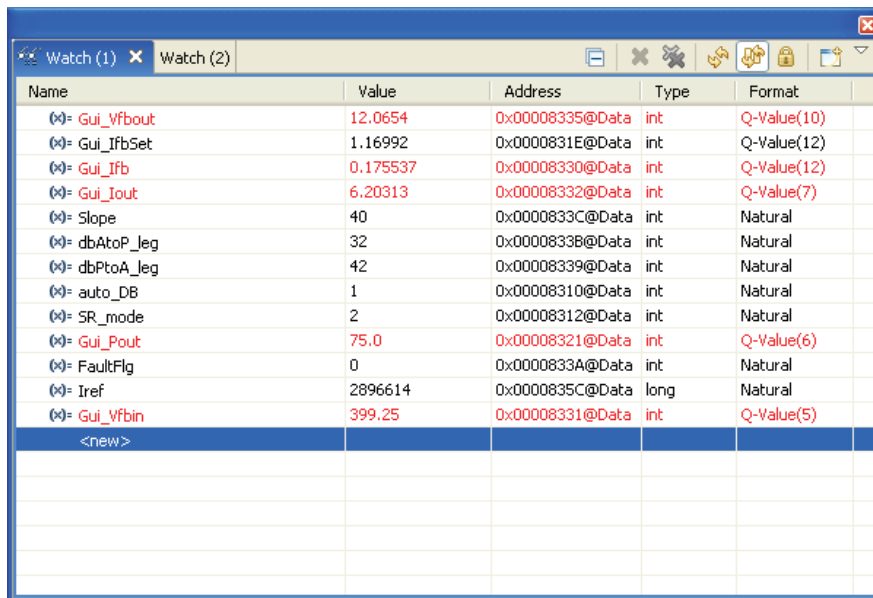
5. Click on the Continuous Refresh button  for the watch view.

4.2.6 Run the Code

1. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.
2. In the watch view, the variable *Gui_IfbSet* should be set to 0.15 (Q12). This variable denotes the peak current reference command in Amperes and drives *Iref* to the *DACDRV_RAMP* module. Do not use a value of less than 0.15 for *Gui_IfbSet*.
3. Apply an appropriate resistive load to the PSFB system at the DC output. A load that draws around 3A – 6A current at 12 V output is a good starting point.

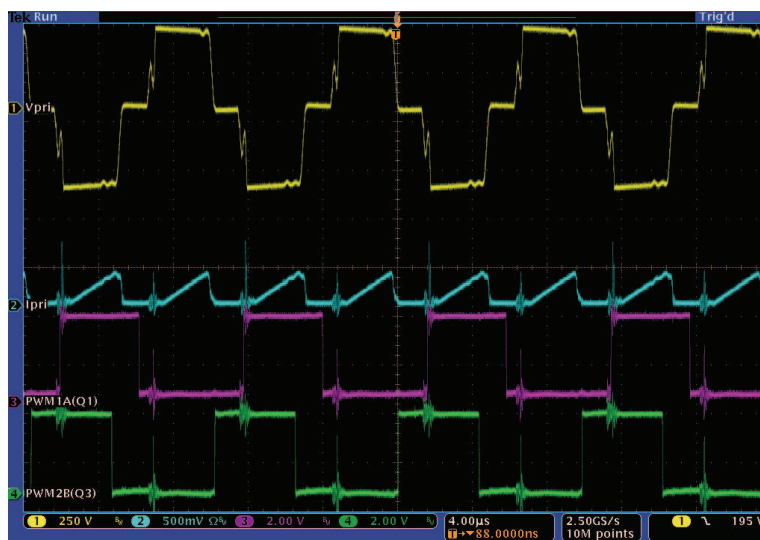
NOTE: For safety reasons, it is recommended to use an isolated DC source to supply 400 V DC input to the board.



4. Power the input at J1, J2 with 400 V DC.
5. Increase the peak current reference command by setting *Gui_IfbSet* to a higher value 0.25 (say) in the watch view. The output voltage should increase. Observe the output voltage carefully, this should not be allowed to exceed the capabilities of the board. Keep in mind that when operating with a certain *Gui_IfbSet* value, if the load is suddenly reduced, the output voltage will go up. Therefore, do not make any sudden changes of load or big increases in *Gui_IfbSet* command when operating in build 1.
6. Observe the different ADC results in the watch view for different *Gui_IfbSet* values. Here is the watch view that corresponds to the operation of the system with a *Gui_IfbSet* command of 1.17A with an input voltage of around 400 V and a load of around 6A at 12 V output.



Name	Value	Address	Type	Format
Gui_Vfbout	12.0654	0x00008335@Data	int	Q-Value(10)
Gui_IfbSet	1.16992	0x0000831E@Data	int	Q-Value(12)
Gui_Ifb	0.175537	0x00008330@Data	int	Q-Value(12)
Gui_Iout	6.20313	0x00008332@Data	int	Q-Value(7)
Slope	40	0x0000833C@Data	int	Natural
dbAtoP_leg	32	0x0000833B@Data	int	Natural
dbPtoA_leg	42	0x00008339@Data	int	Natural
auto_DB	1	0x00008310@Data	int	Natural
SR_mode	2	0x00008312@Data	int	Natural
Gui_Pout	75.0	0x00008321@Data	int	Q-Value(6)
FaultFlg	0	0x0000833A@Data	int	Natural
Iref	2896614	0x0000835C@Data	long	Natural
Gui_VfbIn	399.25	0x00008331@Data	int	Q-Value(5)
<new>				

7. The following oscilloscope capture shows transformer primary voltage, primary sensed current and PWM waveforms driving two diagonally opposite switches (Q1 and Q3) seen under conditions described above.



8. By default, the synchronous rectifiers are operated in mode 2. You can change their mode of operation by changing *SR_mode* variable to 0, 1 or 2 from the watch view. Observe the change in amount of input current being drawn and change in output voltage with different SR modes. You can also probe the PWM waveforms driving the synchronous rectifier switches. Do not change between different SR modes when operating at very low loads or when the output voltage is very low (less than 6V). In these cases use the default SR mode 2.
9. Try different *Gui_lfbSet* values and observe the corresponding ADC results. Increase *Gui_lfbSet* in small steps. Always observe the output voltage carefully, this should not be allowed to exceed the capabilities of the board. Different waveforms, like the PWM gate drive signals, input voltage and current and output voltage may also be probed using an oscilloscope. Appropriate safety precautions should be taken and appropriate grounding requirements should be considered while probing these high voltages and high currents for this isolated DC-DC converter.
10. Fully halting the MCU when in real-time mode is a two-step process. With the 400 V DC input turned off, wait a few seconds. First, halt the processor by using the Halt button on the toolbar, or by using  Target → Halt. Then click the  button again to take the MCU out of real-time mode and then reset the MCU.
11. You can choose to leave Code Composer Studio running for the next exercise or optionally close it.

4.3 Build 2: Full HVPSFB

4.3.1 Objective

The objective of this build is to verify the operation of the complete PCMC-based HVPSFB project from the Code Composer Studio environment.

4.3.2 Overview

Figure 21 shows the software blocks used in this build. A 2-pole 2-zero controller is used for the voltage loop. Depending on the application's control loop requirements some other controller block like a PI, a 3-pole 3-zero, and so forth may also be used. As seen in Figure 21, the voltage loop block is executed at 100 KHz. CNTL2P2Z is a second order compensator realized from an IIR filter structure. This function is independent of any peripherals and, therefore, does not require a CNF function call.

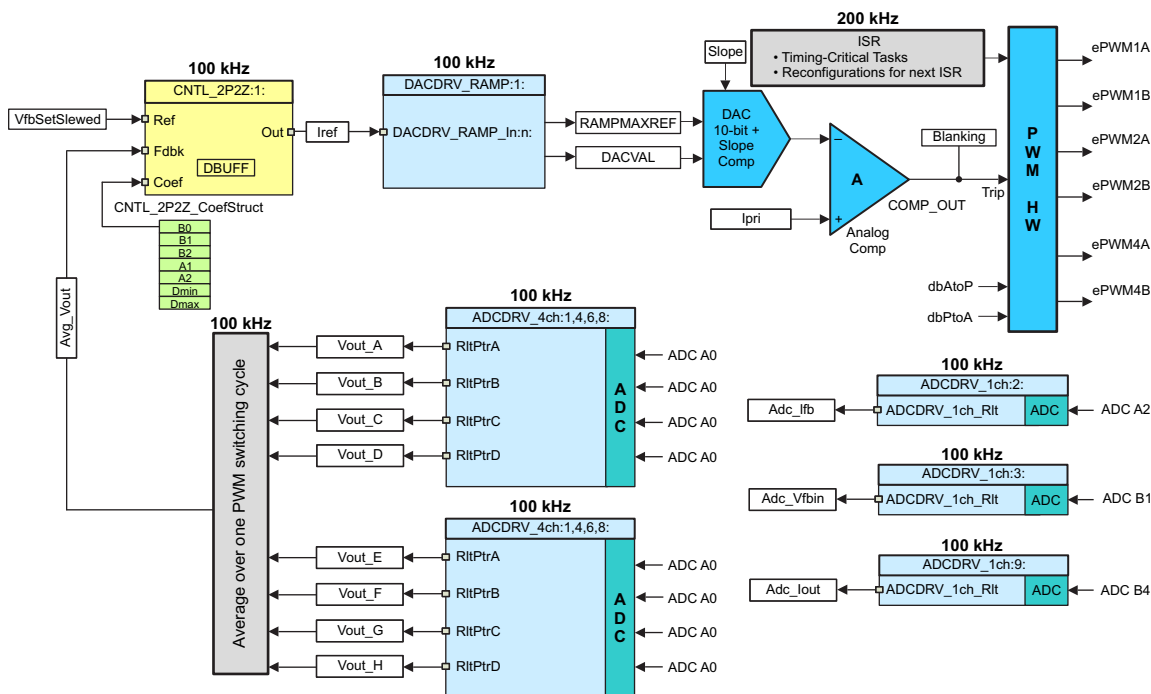


Figure 21. Build 2 Software Blocks

The five coefficients to be modified are stored as elements of the structure CNTL_2P2Z_CoefStruct1 whose other elements are used for clamping the controller output. The CNTL_2P2Z block can be instantiated multiple times if the system needs multiple loops. Each instance can have a separate set of coefficients. Directly manipulating the five coefficients independently by trial and error is almost impossible, and requires mathematical analysis or assistance from tools such as matlab, mathcad, and so forth. These tools offer bode plot, root-locus and other features for determining phase margin, gain margin, and so forth.

To keep loop tuning simple and without the need for complex mathematics or analysis tools, the coefficient selection problem has been reduced from five degrees of freedom to three, by conveniently mapping the more intuitive coefficient gains of P, I and D to B0, B1, B2, A1, and A2. This allows P, I and D to be adjusted independently and gradually. These mapping equations are given below.

The compensator block (CNTL_2P2Z) has two poles and two zeros and is based on the general IIR filter structure. It has a reference input and a feedback input. For the voltage loop the feedback is the average output voltage calculated over one PWM cycle (*Avg_Vout*), while the reference input to the controller is a slewed version (*VfbSetSlewed*) of the output voltage reference command (*Vref*). The transfer function is given by:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}}$$

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

$$u(k) = u(k-1) + b_0e(k) + b_1e(k-1) + b_2e(k-2)$$

where:

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

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

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 - z^{-1}} = \frac{b_0z^2 + b_1z + b_2}{z^2 - z}$$

Comparing this with the general form, you can see that PID is nothing but a special case of CNTL_2P2Z control where:

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

For the voltage loop, these P, I and D coefficients are: Pgain, Igain and Dgain. These P, I and D coefficients are used in Q26 format. To simplify tuning from the GUI environment (or from Code Composer Studio watch views) these three coefficients are further scaled to values from 0 to 999 (Pgain_Gui, Igain_Gui and Dgain_Gui).

From the GUI environment the voltage loop can also be tuned using two poles (*fp1*, *fp2*), two zeroes (*fz1*, *fz2*) and gain (*Kdc*). These parameters provide *b2_Gui*, *b1_Gui*, *b0_Gui*, *a2_Gui* and *a1_Gui* coefficients in I5Q10 form that are then converted to the five Q26 coefficients for the 2P2Z controller. Although not recommended, *b2_Gui*, *b1_Gui*, *b0_Gui*, *a2_Gui* and *a1_Gui* values may also be directly changed from the Code Composer Studio environment using watch views. For details on these calculations, see the HVPSFB-Calculations.xls file located at www.ti.com/controlsuite. The equations for deriving coefficient values based on poles, zeroes, gain and switching frequency are also clear from the GUI source file.

This project allows easy evaluation of both methods of loop tuning by providing the ability to easily switch between coefficients during execution. This can be done by simply clicking on the *2P2Z(On)/PID(Off)* button on the GUI or changing the *pid2p2z_GUI* variable to 0 or 1 on the watch view from Code Composer Studio. PID-based loop tuning (*pid2p2z_GUI = 0*) from the GUI environment was used as a starting point. Poles, zeroes and gain corresponding to these PID tuned coefficients were then used as a starting point for further loop tuning based on the second method (*pid2p2z_GUI = 1*). Much better results were then achieved by changing poles, zeroes and gain from the GUI environment to tune for optimum dynamic performance. By default coefficients based on these tuned poles, zeroes and gain values (*pid2p2z_GUI = 1* – default) are used.

NOTE: When tuning the system using the 2-pole 2-zero controller coefficient adjustments, if a choice of poles, zeroes and *Kdc* values is made in the GUI such that it results in the magnitude of any of the coefficients (*b2*, *b1*, *b0*, *a2*, *a1*) to be greater than or equal to 32, then these coefficient values are not sent to the controller by the GUI.

4.4 Constant Current (CC) and Constant Power (CP)

Simple implementation for experimenting with constant current and constant power functionality has been included in this project. By default this functionality is disabled. Constant current function is implemented by changing the clamping value of the voltage loop controller. Constant power function is implemented by adjusting the voltage loop reference command based on load to maintain constant power at the output. [Figure 22](#) provides the complete software flowchart for these functions.

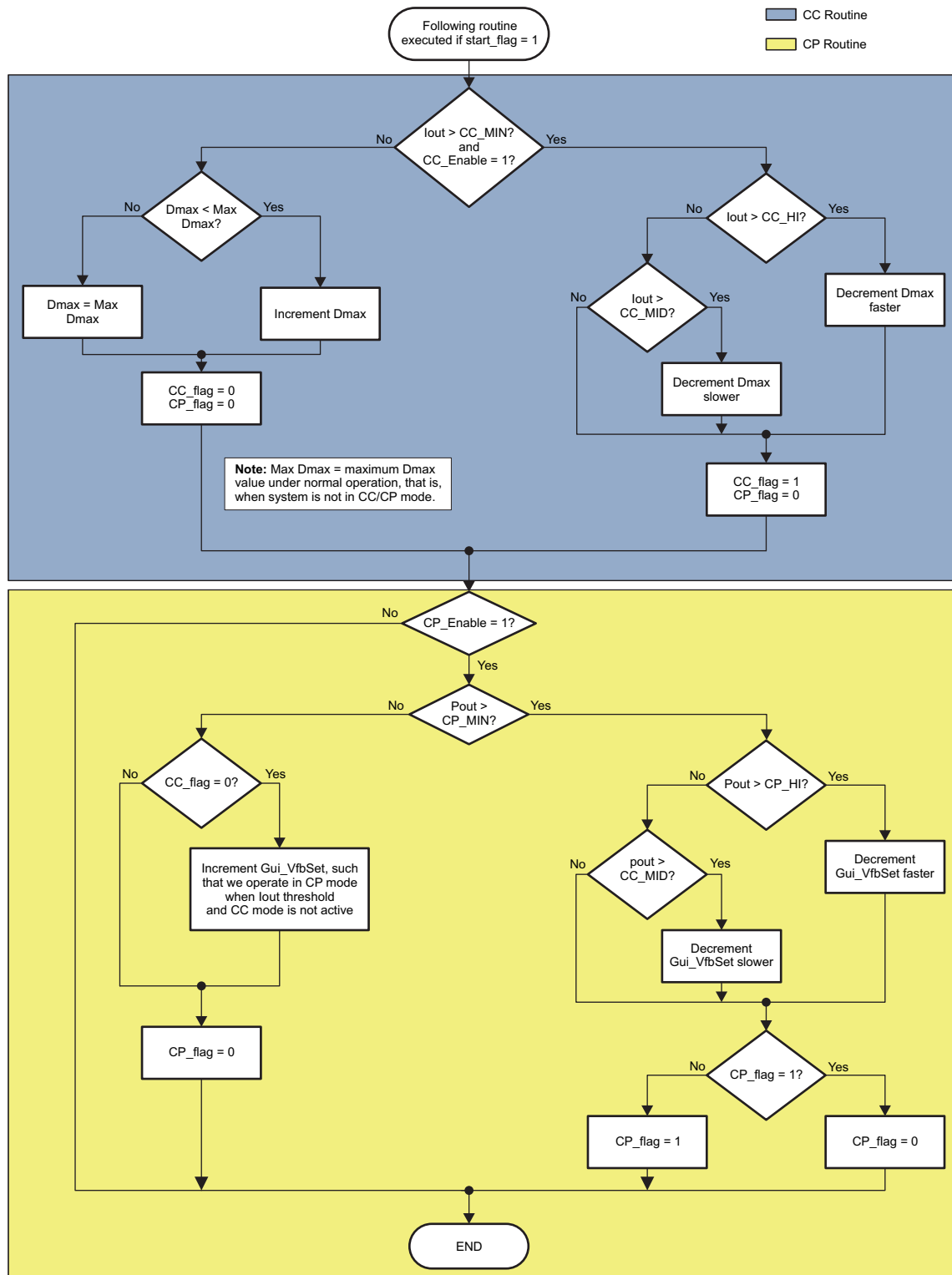


Figure 22. Constant Current and Constant Power Software Flowchart

4.5 Procedure

Build and Load Project

Use the following steps to quickly execute this build using the pre-configured work environment:

1. Follow Steps 1 to 2 exactly as in build 1. If you were working on build 1 the last time Code Composer Studio was used, the same workspace should open up with the project.
2. If this is not the case, you can open the workspace used for build1 by clicking File → Switch Workspace and then navigating to the correct workspace. If a workspace was not saved or got deleted, please follow Steps 3 and 4 exactly as in build 1.
3. Locate and inspect the initialization code specific to build 2 in the main file. This is where all the control blocks are configured, initialized and connected in the control flow.
4. Select the Incremental build option as 2 in the *HVPSFB-Settings.h*.


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

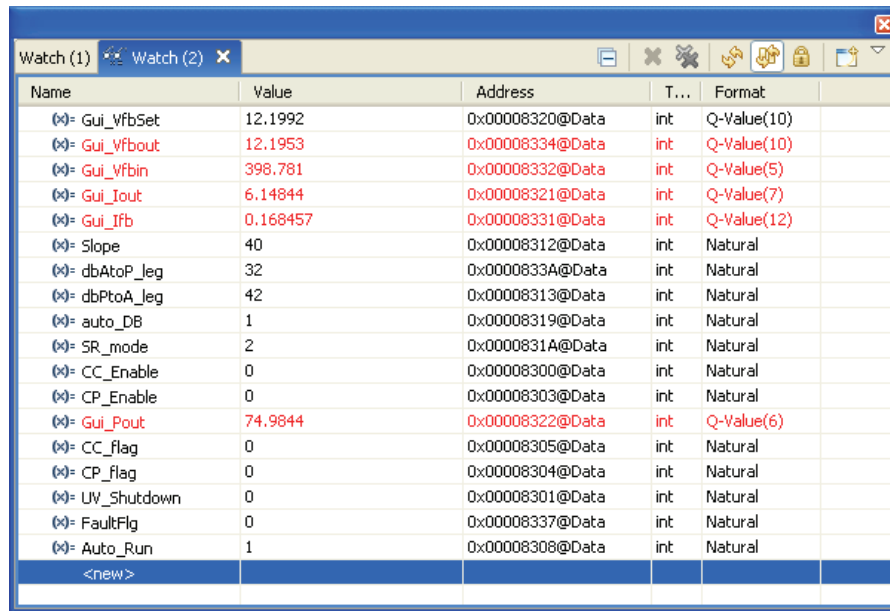
5. Click Project → "Rebuild All" button and watch the tools run in the build window.
6. Click Target → "Debug Active Project". The program will be loaded into the FLASH or RAM memory depending on the project configuration. This project only comes in F2802x_FLASH configuration. You should now be at the start of Main()..

4.5.1 Debug Environment Windows

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

If a watch view did not open when the debug environment was launched, open a new watch view and add various parameters to it by following the procedure given below.

1. Click: View → Watch on the menu bar.
2. Click the "Watch (1)" tab.
3. Click on the *New Watch View*  button if a watch view is already open from the debug environment saved for build 1. A "Watch (2)" tab will open and you can then drag it to be viewed in a window of your choice. You can add any variables to this watch view tab.
4. Type the symbol name of the variable you want to watch in the empty box in the "Name" column and press enter on keyboard. Be sure to modify the "Format" as needed. The watch view should look something like the following. Please note that some of the variables have not been initialized at this point in the main code and may contain some garbage values.



Name	Value	Address	T...	Format
(x)= Gui_VfbSet	12.1992	0x00008320@Data	int	Q-Value(10)
(x)= Gui_Vfbout	12.1953	0x00008334@Data	int	Q-Value(10)
(x)= Gui_VfbIn	398.781	0x00008332@Data	int	Q-Value(5)
(x)= Gui_Iout	6.14844	0x00008321@Data	int	Q-Value(7)
(x)= Gui_Ifb	0.168457	0x00008331@Data	int	Q-Value(12)
(x)= Slope	40	0x00008312@Data	int	Natural
(x)= dbAtoP_leg	32	0x0000833A@Data	int	Natural
(x)= dbPtoA_leg	42	0x00008313@Data	int	Natural
(x)= auto_DB	1	0x00008319@Data	int	Natural
(x)= SR_mode	2	0x0000831A@Data	int	Natural
(x)= CC_Enable	0	0x00008300@Data	int	Natural
(x)= CP_Enable	0	0x00008303@Data	int	Natural
(x)= Gui_Pout	74.9844	0x00008322@Data	int	Q-Value(6)
(x)= CC_flag	0	0x00008305@Data	int	Natural
(x)= CP_flag	0	0x00008304@Data	int	Natural
(x)= UV_Shutdown	0	0x00008301@Data	int	Natural
(x)= FaultFlg	0	0x00008337@Data	int	Natural
(x)= Auto_Run	1	0x00008308@Data	int	Natural
<new>				

Note that there are additional variables in the watch view.

5. Gui_VfbSet is used to set the output voltage command.

4.5.2 Run the Code

1. Follow Steps 1 to 2 of the build 1 procedure to enable real-time mode and continuous refresh for the watch views and also for changing the continuous refresh interval for the watch view if needed.
2. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.
3. Apply an appropriate resistive load to the PSFB system at the DC output. A load that draws around 3A – 6A current at 12 V output is a good starting point.

NOTE: For safety reasons, it is recommended to use an isolated DC source to supply 400 V DC input to the board.

4. Power the input at J1, J2 with 400 V DC.
5. By default, *Auto_Run* is set to 1. If it is not, make it 1 from the watch view. The output voltage should now start ramping up to 12 V. This output voltage ramp up rate can be changed by changing the variable *VfbSlewRate*.

The following graphic shows a watch view that corresponds to the operation of the system with 12.2 V at the output with an input voltage of around 400 V and a load of around 6A output.

Name	Value	Address	T...	Format
(x)= Gui_vfbSet	12.1992	0x00008320@Data	int	Q-Value(10)
(x)= Gui_vfbout	12.1953	0x00008334@Data	int	Q-Value(10)
(x)= Gui_vfbIn	398.781	0x00008332@Data	int	Q-Value(5)
(x)= Gui_Iout	6.14844	0x00008321@Data	int	Q-Value(7)
(x)= Gui_Ifb	0.168457	0x00008331@Data	int	Q-Value(12)
(x)= Slope	40	0x00008312@Data	int	Natural
(x)= dbAtoP_leg	32	0x0000833A@Data	int	Natural
(x)= dbPtoA_leg	42	0x00008313@Data	int	Natural
(x)= auto_DB	1	0x00008319@Data	int	Natural
(x)= SR_mode	2	0x0000831A@Data	int	Natural
(x)= CC_Enable	0	0x00008300@Data	int	Natural
(x)= CP_Enable	0	0x00008303@Data	int	Natural
(x)= Gui_Pout	74.9844	0x00008322@Data	int	Q-Value(6)
(x)= CC_flag	0	0x00008305@Data	int	Natural
(x)= CP_flag	0	0x00008304@Data	int	Natural
(x)= UV_Shutdown	0	0x00008301@Data	int	Natural
(x)= FaultFlg	0	0x00008337@Data	int	Natural
(x)= Auto_Run	1	0x00008308@Data	int	Natural
<new>				

6. Figure 23 shows the transformer primary voltage, the primary sensed current, and PWM waveforms driving two diagonally opposite switches (Q1 and Q3) seen under conditions described above. ZVS switching of switch Q3 and LVS switching of switch Q4 are also shown under these conditions.

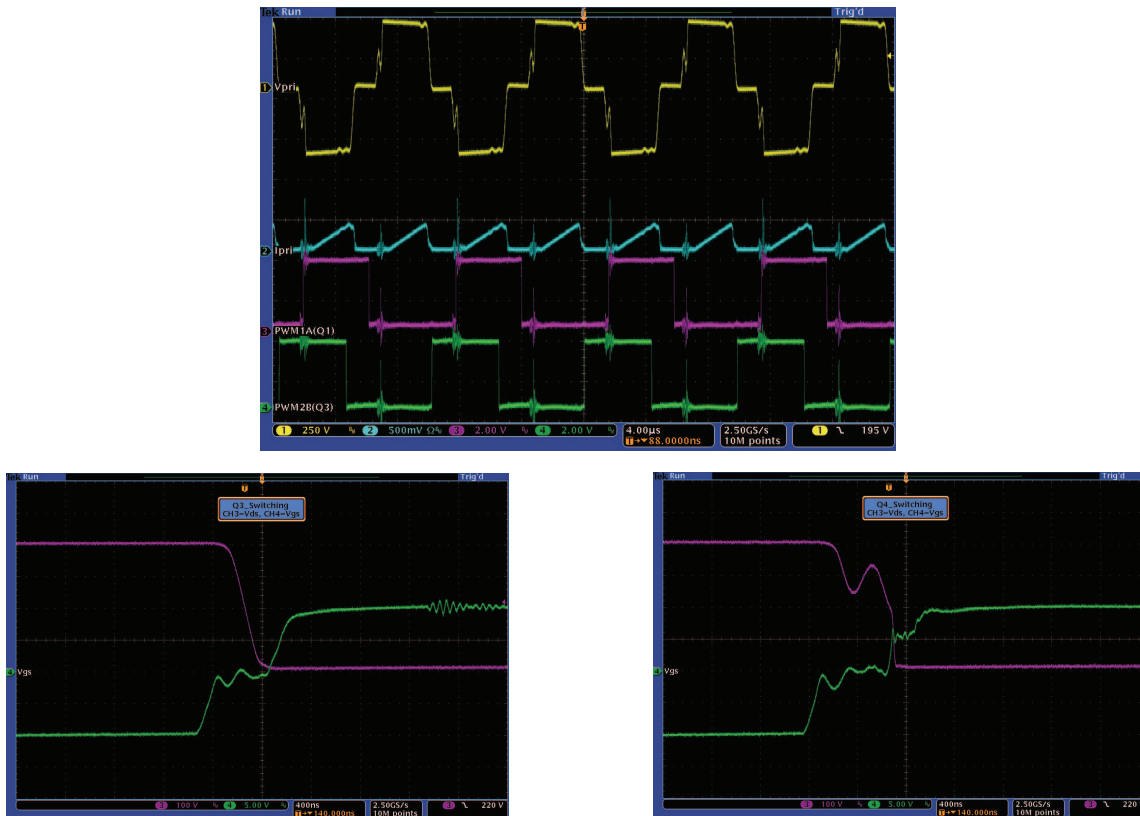


Figure 23. Transformer Primary Voltage, Primary Sensed Current and PWM Waveforms Driving Two Diagonally Opposite Switches (Q1 and Q3)


7. By default, the synchronous rectifiers are operated in mode 2. You can change their mode of operation by changing the SR_mode variable to 0, 1 or 2 from the watch view. Observe the change in the amount of input current being drawn and the change in the output voltage with the different SR modes.

You can also probe the PWM waveforms driving the synchronous rectifier switches. Do not change between different SR modes when operating at very low loads (below 3A). In these cases, use the default SR mode 2.

8. Observe the effect of varying the load on the output voltage and input current. There should be virtually no effect on the output voltage. Similarly, observe the effect of varying the input voltage. Again, there should be virtually no effect on the output voltage.

NOTE: Make sure that these changes are made within the abilities of the board as listed in the specifications section of this document.

9. Different waveforms, like the PWM gate drive signals, input voltage and current and output voltage can also be probed using an oscilloscope. Appropriate safety precautions should be taken and appropriate grounding requirements should be considered while probing these high voltages and high currents for this isolated DC-DC converter.
10. Simple 'Constant Current (CC)' and 'Constant Power (CP)' functions are also implemented in this project and can be experimented with by enabling and disabling their corresponding flags (*CC_Enable* and *CP_Enable*).
11. Fully halting the MCU, when in real-time mode, is a two-step process. Wait a few seconds with the 400 V DC input turned off. First, halt the processor by using the Halt button on the toolbar, or by using

Target → Halt. Then click the  button again to take the MCU out of real-time mode and then reset the MCU.

12. Close Code Composer Studio.

5 Software overview - VMC

5.1 Software Control Flow

The HVPSFB_VMC project makes use of the “C-background/ASM-ISR” framework. It uses C-code as the main supporting program for the application, and is responsible for all system management tasks, decision making, intelligence, and host interaction. The assembly code is strictly limited to the ISR, which runs all the critical control code and typically includes ADC reading, control calculations, and PWM and DAC updates.

Figure 24 depicts the general software flow for this project.

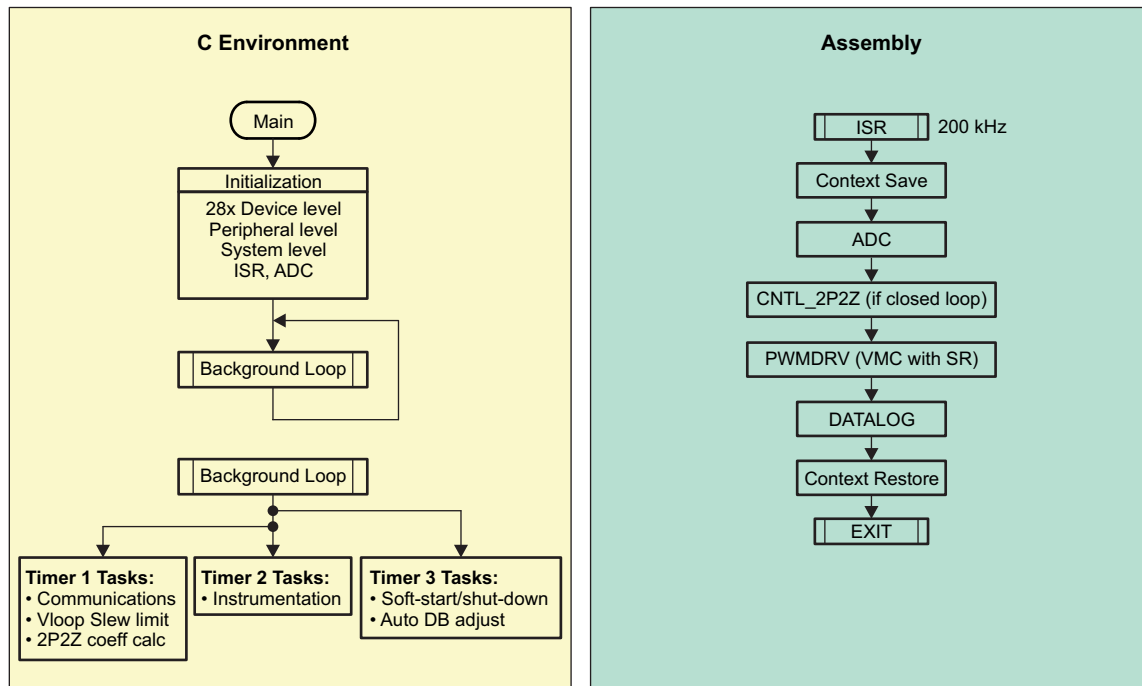


Figure 24. VMC Software Flow

The key framework C files used in this project are:

- HVPSFB-Main.c – this file is used to initialize, run, and manage the application. This is the “brains” behind the application.
- HVPSFB-DevInit.c – this file is responsible for a one time initialization and configuration of the F280x device, and includes functions such as setting up the clocks, PLL, GPIO, and so forth.

The ISR consists of a single file:

- HVPSFB-DPL-ISR.asm – this file contains all time critical “control type” code. This file has an initialization section (one time execute) and a run-time section, which executes (typically) at the same rate as the PWM timebase used to trigger it.

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

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

Table 5. Library Modules

C configure function	ASM initialization macro	ASM run-time macro
PWMDRV_PSFB_VMC_SR_CNF	PWMDRV_PSFB_VMC_SR_INIT n,m,p	PWMDRV_PSFB_VMC_SR n,m,p
ADC_SOC_Cnf.c	ADCDRV_4CH_INIT m,n,p,q	ADCDRV_4CH m,n,p,q
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

The control blocks can also be represented graphically as shown in [Figure 25](#).

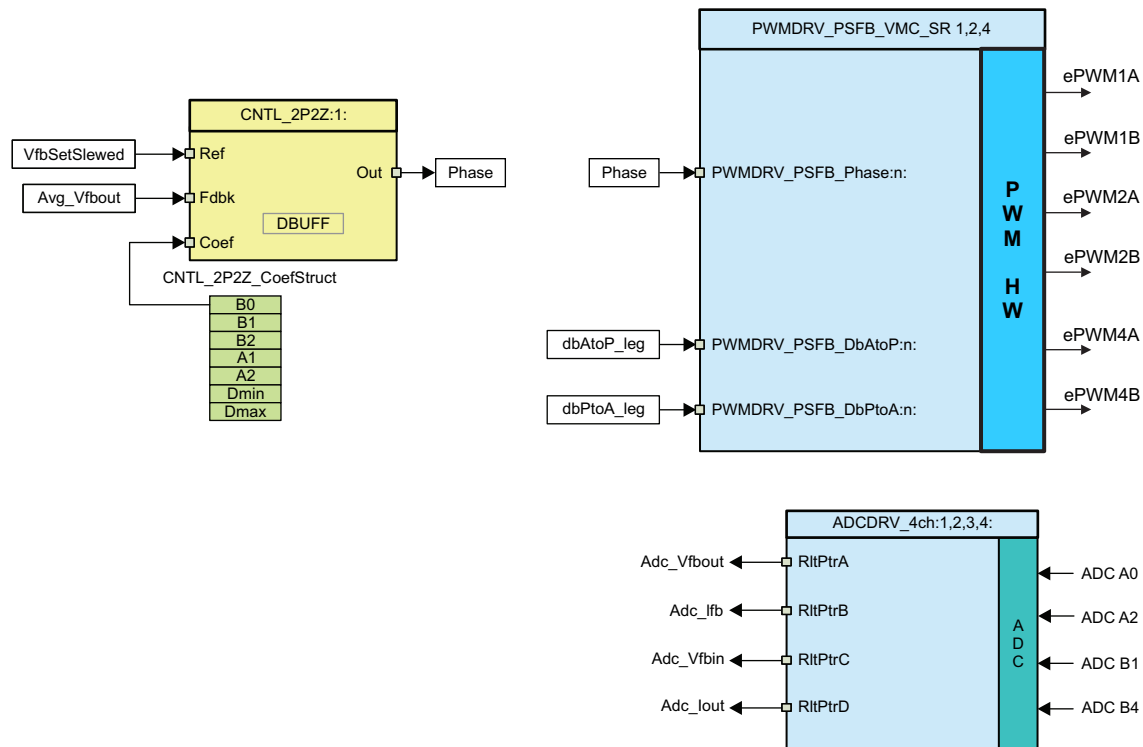


Figure 25. VMC Software Blocks

Notice the color coding for the modules in [Figure 25](#). Blocks in 'dark blue' represent hardware modules on the C2000 micro-controller. 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 and PID, a 3-pole 3-zero or any other controller that can be suitably implemented for this application. Such a modular library structure makes it convenient to visualize and understand the complete system software flow as shown in [Figure 26](#). It also allows for easy use and additions and 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.

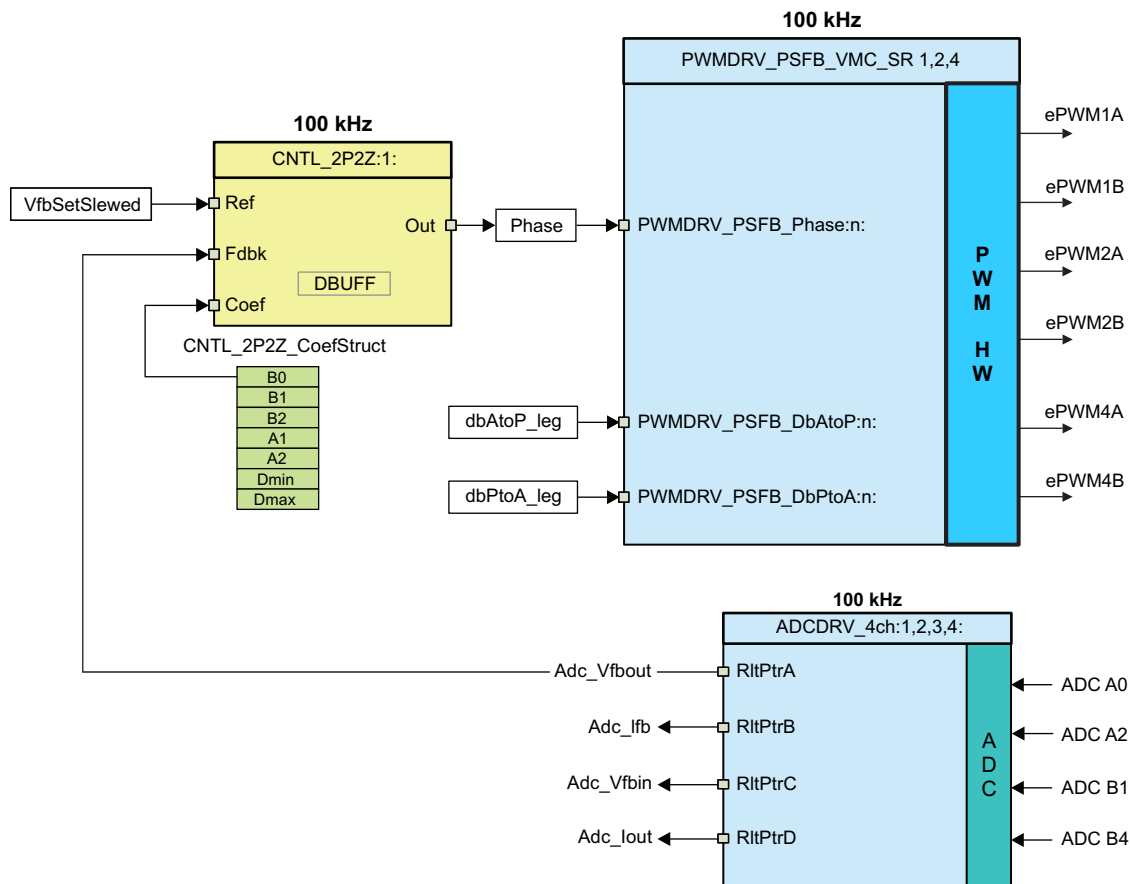


Figure 26. VMC Control Flow

The system is controlled by one voltage feedback loop. Figure 26 also gives the rate at which control blocks are executed. For example, the voltage controller is executed at a rate of 100 kHz (same as the PWM switching frequency). The following explains the control implemented above.

The sensed output voltage (*Adc_Vfbout*) is compared with slewed version (*VfbSetSlewed*) of the voltage reference command (*Vref*) in the voltage controller. The voltage controller output directly controls phase shift between PWM signals driving the two legs of the full bridge. This dictates the amount of phase overlap between the two legs of the full bridge to regulate the output voltage. The *dbAtoP_leg* and *dbPtoA_leg* values provide dead band values for the ‘Active to Passive’ and ‘Passive to Active’ legs of the full bridge, respectively. These are used to achieve ZVS and LVS across the load range.

5.2 Incremental Builds

This project is divided into two incremental builds. This makes it easier to learn and get familiar with the board and the software. This approach is also good for debugging and testing the boards. The build options are shown below. To select a particular build option,

1. Set the macro INCR_BUILD, found in the HVPSFB-Settings.h file, to the corresponding build selection as shown below.
2. Compile the complete project by selecting the rebuild-all compiler option, once the build option is selected.

Section 6 provides more details to run each of the build options.

Table 6. Incremental Build Options for VMC

Incremental Build Options	
INCR_BUILD = 1	Open loop PSFB drive with ADC feedback (check PWM drive circuit and sensing circuit)
INCR_BUILD = 2	Closed voltage loop (full PSFB in VMC mode)

6 Procedure for Running the Incremental Builds - VMC

The main source files, ISR assembly file and the project file for C framework to bring up the PSFB system are located in the following directory (please use the latest version of the software package. Version 1.1 is the latest software version as of March 2012) `..\controlSUITE\development_kits\HVPSFB_v1.1\HVPSFB_VMC`. The projects included with this software are targeted for Code Composer Studio v4.

WARNING

There are high voltages present on the board. It should only be handled by experienced power supply professionals in a lab environment. To safely evaluate this board, an appropriate isolated high voltage DC source should be used. Before DC power is applied to the board, a voltmeter and an appropriate resistive or electronic load must be attached to the output. The unit should never be handled when the power is applied to it.

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

6.1 Build 1: Open Loop Check With ADC Feedback

6.1.1 Objective

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

6.1.2 Overview

The software in Build1 has been configured so that you can quickly evaluate the phase shifted full bridge PWM driver module by viewing the output waveforms on an oscilloscope and observing the effect of change in phase on the output voltage by interactively adjusting the phase on Code Composer Studio. Additionally, you can evaluate the ADC driver module by viewing the ADC sampled data in the watch view.

The PWM and ADC driver macro instantiations are executed inside the `_DPL_ISR` as mentioned in [Section 5](#). [Figure 27](#) shows the blocks used in this build. `ePWM1A` and `ePWM1B` drive Q2 and Q3 full-bridge switches, while `ePWM2A` and `ePWM2B` drive Q1 and Q4 full-bridge switches. `ePWM4A` and `ePWM4B` drive Q6 and Q5 synchronous rectifier switches.

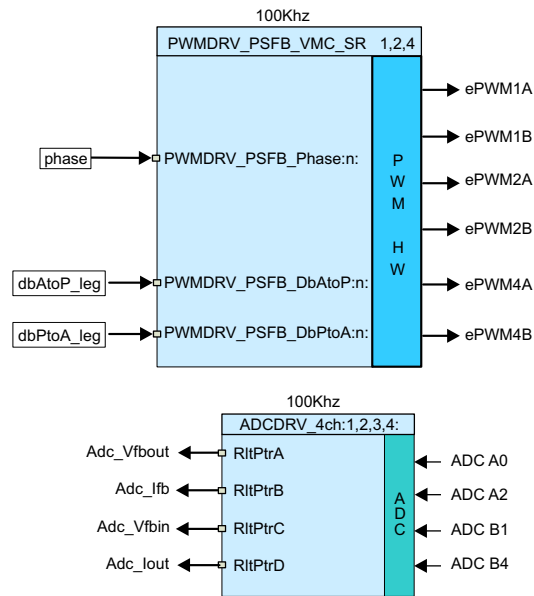


Figure 27. Build 1 Software Blocks

These PWM signals need to be generated at a frequency of 100 kHz (a period of 10 μs). With the MCU operating at 60 MHz, one count of the time base counter of ePWM1, ePWM2 or ePWM4 corresponds to 16.667 ns. This implies that a PWM period of 10 μs will be equivalent to 600 counts of the time base counter (TBCNT1, TBCNT2 and TBCNT4). The ePWM1 and ePWM2 modules are configured to operate in up-count mode, while ePWM4 operates in up-down count mode. ePWM1A and ePWM1B outputs operate at 50% duty cycle and are complementary to each other. Similarly ePWM2A and ePWM2B operate at 50% duty cycles and are complementary to each other. Phase for ePWM2 time base may be changed dynamically with respect to ePWM1 phase. These PWM waveforms are shown in Figure 14.

The *phase* input to the PWM driver module decides the amount of phase shift between PWM1 and PWM2 time bases. This *phase* value controls the amount of overlap between PWM signals driving diagonally opposite switch pairs of the full bridge. As *phase* increases, amount of overlap increases, which increases the amount of energy transferred to the secondary. The TBPHS2 value is derived from the input *phase* command.

Table 7 gives example TBPHS2 values derived for a TBPRD value of 599.

Table 7. Phase Values for Reference

Phase (Q24)	TBPHS2 = (phase*TBPRD/2^25)	Phase Shift in Degrees
2097152d	37	22.5
8388608d	149	90
16776704d	299	180

Note that each pair of diagonal switches of the full bridge overlaps once in one PWM period. This means the most overlap will occur when the phase shift is close to 180°.

The assembly ISR is triggered on a ZRO (TBCNT1 = 0) event of ePWM1. This is where the control driver macros are executed and the Time Base Phase Registers (TBPHS2 and TBPHS4) are updated.

There is an important consideration regarding where the ADC input is sampled. The integrity of the ADC input signals is of high importance since this is where signals from the analog and digital domains interface. Turning a switch ON or OFF in the power stage can result in some noise or disturbance on the signals that are to be sensed around this point in time. Even with all the filtering provided on these signals to avoid this noise from showing up at the ADC inputs, it is prudent to sample the ADC inputs at a time such that this disturbance is avoided.

Moreover, as discussed in [Section 2.5](#), sensed output voltage should preferably be sampled at an appropriate point in the switching cycle where the output voltage value is close to its average value. To achieve this, the ADC input signals are sampled at a time so as to get as noise free a sample as possible and to also sample the average output voltage. For the full bridge this is achieved by sampling at the midpoint of the overlap between two diagonal switches (here overlap refers to the period when both switches are ON at the same time - as far away from the MOSFET switching as possible). This avoids any switching noise to be reflected on the ADC result. The flexibility of ADC and PWM modules on C2000 devices allow such precise and flexible triggering of ADC conversions. The ADC driver modules are used to read 12-bit ADC results and convert them to Q24 values. In every PWM cycle PWM2 SOCA (start of conversion A) is used to trigger five ADC conversions.

6.1.3 Protection

At this stage, it is appropriate to introduce the shutdown mechanism used with this project. Here overcurrent protection is implemented for the transformer primary current using on-chip analog comparator 1. The reference trip level is set using the internal 10-bit DAC and fed to the inverting terminal of this comparator. The comparator outputs are configured to generate a one-shot trip action on ePWM1 and ePWM2 whenever the sensed current is greater than the set limit. The flexibility of the trip mechanism on C2000 devices provides the possibilities for taking different actions on different trip events. In this project ePWM1A, ePWM1B, ePWM2A, and ePWM2B outputs are driven low immediately to protect the power stage. These outputs are held in this state until a device reset is executed. For details on these calculations, see the HVPSFB-Calculations.xls file located at www.ti.com/controlsuite.

Input under voltage and over voltage lockout is also implemented in the software.

6.1.4 Resource Mapping

The key signal connections between the C2000 micro-controller and the HVPSFB stage are summarized in [Table 8](#). Note that PWM mapping is different between VMC and PCMC projects. Jumper (J2, J3 – PCMC and VMC PWM drive jumper enables) on the controller card need to be correctly configured for VMC mode (default jumper positions are set for PCMC operation). Here are the jumper configurations for the two modes:

- PCMC: J2(1) → J3(1), J2(2) → J3(2), J2(3) → J3(3), J2(4) → J3(4), J2(7) → J3(7), J2(8) → J3(8),
- VMC: J2(1) → J3(3), J2(2) → J3(4), J2(3) → J3(1), J2(4) → J3(2), J2(7) → J3(8), J2(8) → J3(7)

Table 8. HVPSFB Signal Interface Reference - VMC

Signal Name	Description	Connection to C2000 Controller
ePWM-1A	PWM drive for full-bridge switch Q2	GPIO-00
ePWM-1B	PWM drive for full-bridge switch Q3	GPIO-01
ePWM-2A	PWM drive for full-bridge switch Q1	GPIO-02
ePWM-2B	PWM drive for full-bridge switch Q4	GPIO-03
ePWM-4A	PWM drive for synchronous rectifier switch Q6	GPIO-06
ePWM-4B	PWM drive for synchronous rectifier switch Q5	GPIO-07
Vout	PSFB output voltage	ADC-A0
lfb	Transformer primary current	ADC-A2 and COMP1A
lfb	Transformer primary current	ADC-A4 and COMP2A ⁽¹⁾
Vfbin	PSFB input voltage	ADC-B1
lout1	PSFB output current	ADC-B3
lout2	PSFB output current (heavily filtered)	ADC-B4

⁽¹⁾ Default jumper J4 configuration on controller card. Input is configurable by jumper J4 selection.

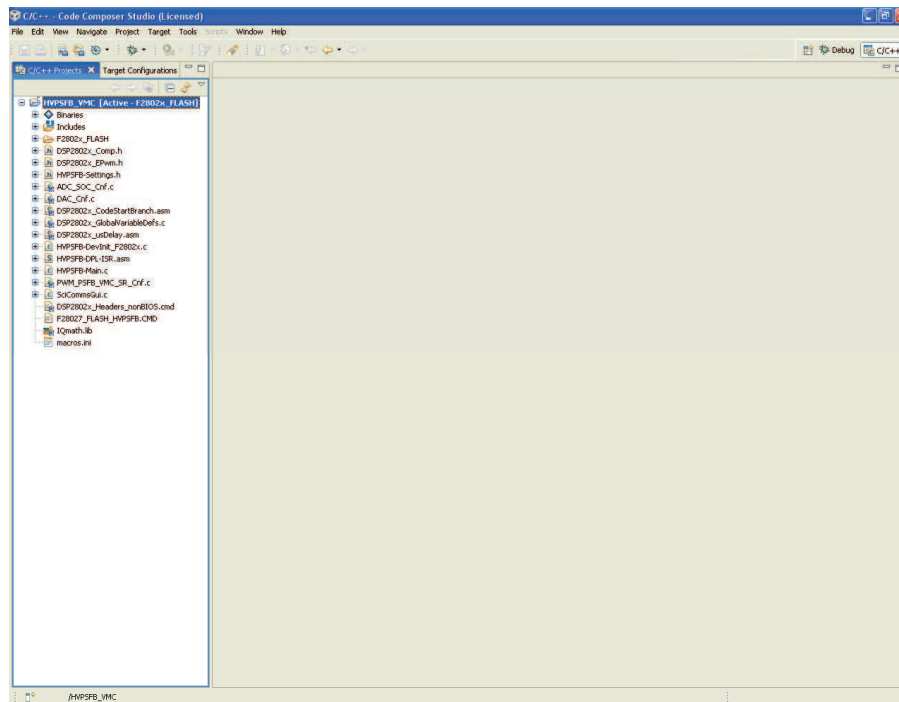
NOTE: The HVPSFB_PCMC and HVPSFB_VMC projects use DSP2802x_Comp.h and DSP2802x_EPWM.h header files that are located in their own project directories instead of the ones located in the device_support directory. The two files in device_support directory will be updated at a later update of *ControlSuite*, at which time these updated files can be used with the two projects.

6.2 Procedure

6.2.1 Start Code Composer Studio and Open a Project

Use the following steps to quickly execute this build:

1. Make sure that jumper J6 on the baseboard is not populated, while jumper J8 is populated.
2. By default, resistor R6 on Piccolo Macro of the controller card and jumper J1 are removed to enable – boot from FLASH. Re-populate these two to run and program RAM or program FLASH.
3. Connect USB connector to the Piccolo controller card for emulation. Use of an appropriate isolated DC power supply set to output around 400 V DC is recommended. The DC power supply needs to remain off before it is connected to J1 and J2 on the main board.
4. Use a 20AWG 600 V wire to connect the Power Source to J1 and J2. Make sure that the polarity of this connection is correct. Apply an appropriate resistive or DC electronic load to the phase shifted full bridge system at the DC output at J3 and J4.
5. Do not turn on 400 V DC power at this time.
6. Power up the bias supply between TP1 and TP2 with around 11V DC (this voltage must be less than 12 V).
7. Double click on the Code Composer Studio icon on the desktop.
8. Maximize Code Composer Studio to fill your screen.
9. Close the welcome screen if it opens up.
10. A project contains all the files and build options needed to develop an executable output file (.out) that can run on the MCU hardware. On the menu bar, click → Project Import Existing CCS/CCE Eclipse Project and under *Select root directory* navigate to and select the `..\controlSUITE\development_kits\HVPSFB_v1.1\HVPSFB_VMC` directory. Make sure that the Projects tab HVPSFB_VMC is checked.
11. Click Finish. This project will invoke all the necessary tools (compiler, assembler, linker) to build the project.
12. Click the plus sign (+) to the left of Project (in the project window on the left). Your project window will look like the following graphic.



6.2.2 Device Initialization, Main, and ISR Files

NOTE: DO NOT make any changes to the source files – ONLY INSPECT.

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

```
PWMDRV_PSFb_VMC_SR_CNF(1, PWM_PRD, 1, 1); // ePWM1 and ePWM2, Period=PWM_PRD,
                                           // SR_Enable=1 (ePWM4), Compl_Prot=1
                                           // Connect the PWMDRV_PSFb_VMC_SR driver block
PWMDRV_PSFb_Phase1 = &phase                // Point to the phase net
PWMDRV_PSFb_DbAtop1 = &dbAtop_leg;         // Point to the left leg dead band adjust
PWMDRV_PSFb_DbPtoA1 = &dbPtoA_leg;        // Point to the right leg dead band adjust
```

4. Locate and inspect the following code in the main file under initialization code specific for build 1. This is where the ADCDRV_4CH block is configured, initialized and connected in the control flow.

```
#define Vfb_outR      AdcResult.ADCRESULT1 //
#define IfbR         AdcResult.ADCRESULT2 //
#define Vfb_inR      AdcResult.ADCRESULT //

#define IoutR        AdcResult.ADCRESULT4 //

// Channel Selection for Cascaded Sequencer
```

```

ChSel[0] = 0;    // A0 - O/P Voltage - Dummy
ChSel[1] = 0;    // A0 - O/P Voltage
ChSel[2] = 2;    // A2 - Transformer Primary Current
ChSel[3] = 9;    // B1 - I/P Voltage

ChSel[4] = 12;   // B4 - Iout2

TrigSel[0] = 7;  // O/P Voltage sampling triggered by EPWM2 SOCA - Dummy
TrigSel[1] = 7;  // O/P Voltage sampling triggered by EPWM2 SOCA
TrigSel[2] = 7;  // Transformer Primary Current sampling triggered by EPWM2 SOCA
TrigSel[3] = 7;  // I/P Voltage sampling triggered by EPWM2 SOCA

TrigSel[4] = 7;  // Iout sampling triggered by EPWM2 SOCA

EALLOW;
AdcRegs.SOCPRCTL.bit.SOCPRIORITY = 4; // SOC0-3 are high priority
EDIS;
ADC_SOC_CNF(ChSel,TrigSel,ACQPS, 16, 0); // ACQPS=8, No ADC channel triggers an interrupt
IntChSel > 15,
                                     // Mode= Start/Stop (0)
                                     // ADC feedback connections

    ADCDRV_4ch_RltPtrA = &Adc_Vfbout;

```

5. Open and inspect HVPSFB-DPL-ISR.asm. Notice the `_DPL_Init` and `_DPL_ISR` sections. This is where the PWM and ADC driver macro instantiation is done for initialization and runtime, respectively. Optionally, you can close the inspected files.

6.2.3 Build and Load the Project

1. Select the Incremental build option as 1 in the HVPSFB-Settings.h file.

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

2. Click Project → "Rebuild All" button and watch the tools run in the build window.
3. Click Target → "Debug Active Project". Code Composer Studio 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 go to . In the New target Configuration Window type in the name of the .ccxml file for the target you will be working with (Example: xds100-F28027.ccxml). Check "Use shared location" and click Finish.
4. In the .ccxml file that opens up select Connection as "Texas Instruments XDS100v2 USB Emulator" and under the device, scroll down and select "TMS320F28027". Click Save.
5. Click Target → "Debug Active Project". The program will be loaded into the FLASH or RAM memory depending on the project configuration selected. This project comes in only F2802x_FLASH configuration. You should now be at the start of Main().

6.2.4 Debug Environment Windows

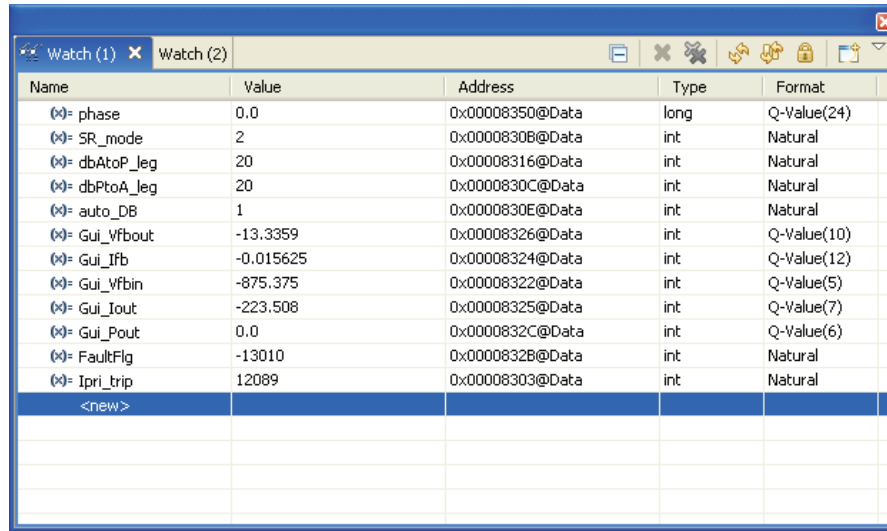
It is standard debug practice to watch local and global variables while debugging code. There are various methods for doing this in Code Composer Studio, such as memory views and watch views. Additionally, Code Composer Studio has the ability to make time (and frequency) domain plots. This allows you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

If a watch view did not open when the debug environment was launched, open a new watch view and add various parameters to it by following the procedure given below.

1. Click View → Watch on the menu bar.
2. Click the "Watch (1)" tab. You can add any variables to the watch view.

3. Type the symbol name of the variable you want to watch in the empty box in the "Name" column and press enter on keyboard. Be sure to modify the "Format" as needed. The watch view should look something like the following. Please note that some of the variables have not been initialized at this point in the main code and may contain some garbage values.

FaultFlg, if set, indicates an over current condition (discussed above), which shuts down the PWM outputs. PWM outputs are held in this state until a device reset (follow proper procedure in). The *Ipri_trip* variable sets the internal 10-bit DAC reference level for the on-chip comparator 1. Please note that this is a Q15 number.



Name	Value	Address	Type	Format
phase	0.0	0x00008350@Data	long	Q-Value(24)
SR_mode	2	0x0000830B@Data	int	Natural
dbAtoP_leg	20	0x00008316@Data	int	Natural
dbPtoA_leg	20	0x0000830C@Data	int	Natural
auto_DB	1	0x0000830E@Data	int	Natural
Gui_Vfbout	-13.3359	0x00008326@Data	int	Q-Value(10)
Gui_Ifb	-0.015625	0x00008324@Data	int	Q-Value(12)
Gui_Vfbin	-875.375	0x00008322@Data	int	Q-Value(5)
Gui_Iout	-223.508	0x00008325@Data	int	Q-Value(7)
Gui_Pout	0.0	0x0000832C@Data	int	Q-Value(6)
FaultFlg	-13010	0x0000832B@Data	int	Natural
Ipri_trip	12089	0x00008303@Data	int	Natural
<new>				

6.2.5 Using Real-Time Emulation

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

1. Enable real-time mode by hovering your mouse on the buttons on the horizontal toolbar and clicking on




the *Enable Silicon Real-Time Mode* (service critical interrupts when halted, allow debugger accesses when running) button.

2. Select YES to enable debug events, if a message box appears. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a "0". The DGBM is the debug enable mask bit. When the DGBM bit is set to "0", memory and register values can be passed to the host processor for updating the debugger windows.

3. Click the  *Enable Polite Real-Time Mode* button on the same horizontal toolbar.

4. When a large number of windows are open, as bandwidth over the emulation link is limited, updating too many windows and variables in continuous refresh can cause the refresh frequency to bog down.

Right click on the  button in the watch view and select "*Customize Continuous Refresh Interval..*". You can slow down the refresh rate for the watch view variables by changing the *Continuous refresh interval (milliseconds)* value. A rate of 4000 ms is usually enough for these exercises.

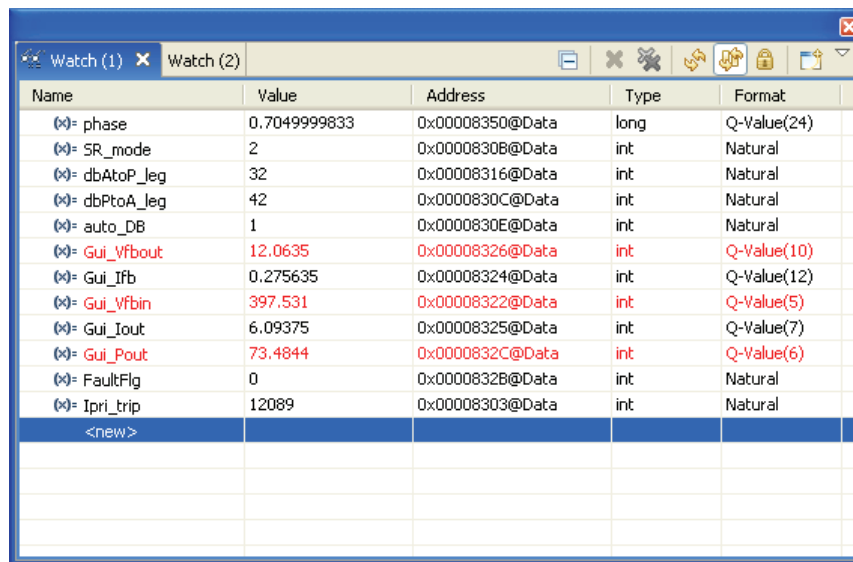
5. Click on the *Continuous Refresh* button  for the watch view.

6.2.6 Run the Code

1. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.
2. Set the variable *phase* to 0.015625 (Q24), in the watch view. This variable denotes the phase shift command to the *PWMDRV_PSFBS_VMC_SR* module. Do not use a value of less than 0.005 for *phase*.
3. Apply an appropriate resistive load to the PSFB system at the DC output. A load that draws around 3A – 6A current at 12 V output is a good starting point.

NOTE: For safety reasons, it is recommended to use an isolated DC source to supply 400 V DC input to the board.



4. Power the input at J1, J2 with 400 V DC.
5. Increase the phase command by setting *phase* to a higher value 0.1(say) in the watch view. The output voltage should increase. Observe the output voltage carefully, this should not be allowed to exceed the capabilities of the board. Keep in mind that when operating with a certain *phase* value if the load is suddenly reduced, the output voltage will go up. Therefore, do not make any sudden changes of load or big increases in phase command when operating in build 1.
6. Observe the different ADC results in the watch view for different *phase* values.
7. Here is the watch view that corresponds to the operation of the system with a *phase* command of 0.705 (Q24) with an input voltage of around 400 V and a load of around 6A at 12 V output.



Name	Value	Address	Type	Format
phase	0.7049999833	0x00008350@Data	long	Q-Value(24)
SR_mode	2	0x0000830B@Data	int	Natural
dbAtoP_leg	32	0x00008316@Data	int	Natural
dbPtoA_leg	42	0x0000830C@Data	int	Natural
auto_DB	1	0x0000830E@Data	int	Natural
Gui_Vfbout	12.0635	0x00008326@Data	int	Q-Value(10)
Gui_Ifb	0.275635	0x00008324@Data	int	Q-Value(12)
Gui_VfbIn	397.531	0x00008322@Data	int	Q-Value(5)
Gui_Iout	6.09375	0x00008325@Data	int	Q-Value(7)
Gui_Pout	73.4844	0x0000832C@Data	int	Q-Value(6)
FaultFlg	0	0x0000832B@Data	int	Natural
Ipri_trip	12089	0x00008303@Data	int	Natural
<new>				

- The following oscilloscope capture shows the transformer primary voltage and the sensed primary current seen under conditions described above.



- By default, the synchronous rectifiers are operated in mode 2. You can change their mode of operation by changing SR_mode variable to 0, 1 or 2 from the watch view. Observe the change in amount of input current being drawn and change in output voltage with different SR modes. You can also probe the PWM waveforms driving the synchronous rectifier switches. Do not change between different SR modes when operating at very low loads or when the output voltage is very low (less than 6V). In these cases use the default SR mode 2.
- Try different phase values and observe the corresponding ADC results. Increase the phase 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 can also be probed using an oscilloscope. The appropriate safety precautions should be taken and appropriate grounding requirements should be considered while probing these high voltages and high currents for this isolated DC-DC converter.
- Fully halting the MCU when in real-time mode is a two-step process. Wait a few seconds, with the 400 V DC input turned off. First, halt the processor by using the Halt button on the toolbar, or by using  Target → Halt. Then click the  button again to take the MCU out of real-time mode and then reset the MCU.
- Leave Code Composer Studio running for the next exercise or optionally close it.

6.3 Build 2: Closed Voltage Loop (Full HVPSFB in VMC mode)

6.3.1 Objective

The objective of this build is to verify the operation of the complete VMC based HVPSFB project from the Code Composer Studio environment.

6.3.2 Overview

Figure 28 shows the software blocks used in this build. The PWM and ADC driver blocks are used in the same way as in the previous build. A 2-pole 2-zero controller is used for the voltage loop. Depending on the application's control loop requirements some other controller block like a PI, a 3-pole 3-zero, and so forth can also be used. As seen in Figure 28, the voltage loop block is executed at 100 KHz. CNTL2P2Z is a second order compensator realized from an IIR filter structure. This function is independent of any peripherals and, therefore, does not require a CNF function call.

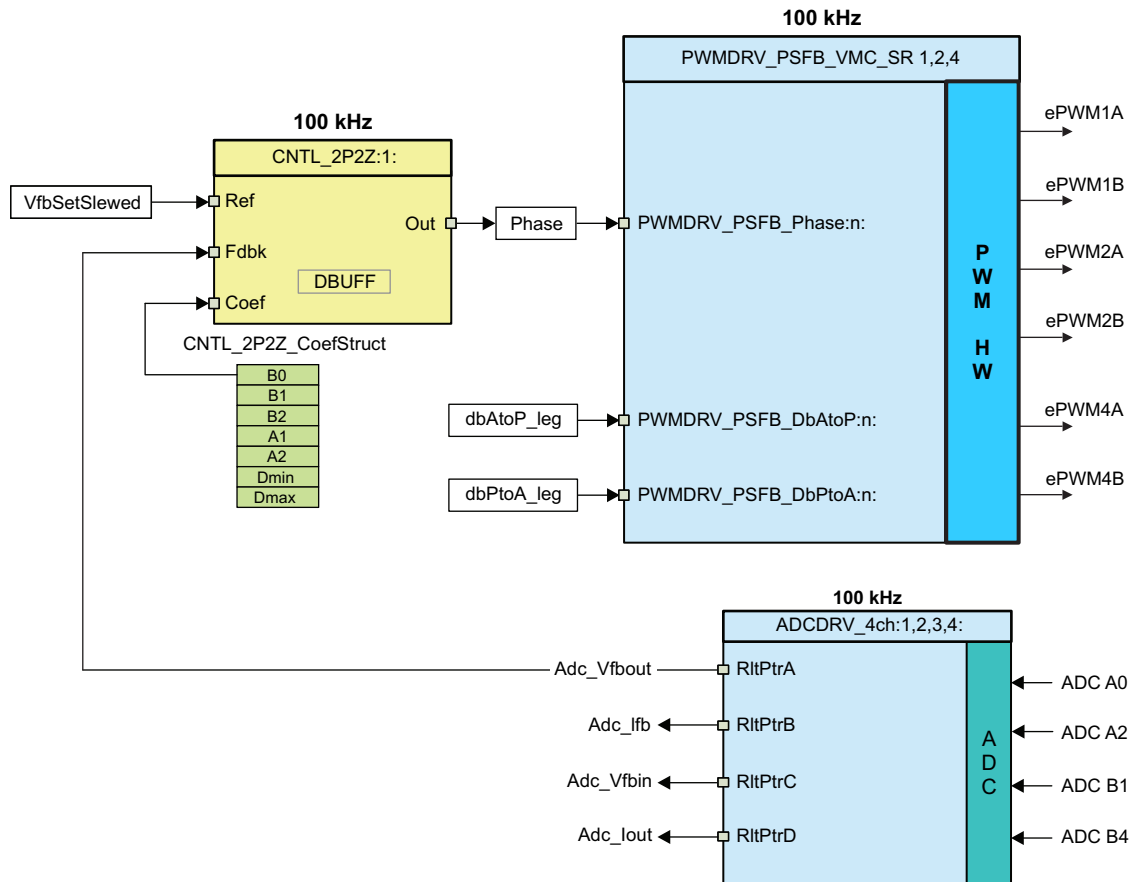


Figure 28. Build 2 Software Blocks

The five coefficients to be modified are stored as elements of the structure CNTL_2P2Z_CoefStruct1 whose other elements are used for clamping the controller output. The CNTL_2P2Z block can be instantiated multiple times if the system needs multiple loops. Each instance can have separate set of coefficients. Directly manipulating the five coefficients independently by trial and error is almost impossible, and requires mathematical analysis or assistance from tools such as matlab, mathcad, and so forth. These tools offer bode plot, root-locus and other features for determining phase margin, gain margin, and so forth.

To keep loop tuning simple and without the need for complex mathematics or analysis tools, the coefficient selection problem has been reduced from five degrees of freedom to three, by conveniently mapping the more intuitive coefficient gains of P, I and D to B0, B1, B2, A1, and A2. This allows P, I and D to be adjusted independently and gradually. These mapping equations are given below.

The compensator block (CNTL_2P2Z) has 2 poles and 2 zeros and is based on the general IIR filter structure. It has a reference input and a feedback input. For the voltage loop, the feedback is the sensed output voltage (*Adc_Vfbout*), while the reference input to the controller is a slewed version (*VfbSetSlewed*) of the output voltage reference command (*Vref*). The transfer function is given by:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}}$$

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

$$u(k) = u(k-1) + b_0e(k) + b_1e(k-1) + b_2e(k-2)$$

where:

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

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

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 - z^{-1}} = \frac{b_0z^2 + b_1z + b_2}{z^2 - z}$$

Comparing this with the general form, you can see that PID is nothing but a special case of CNTL_2P2Z control where:

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

For the voltage loop, these P, I and D coefficients are: Pgain, Igain and Dgain. These P, I and D coefficients are used in Q26 format. To simplify tuning from the GUI environment (or from Code Composer Studio watch views) these three coefficients are further scaled to values from 0 to 999 (Pgain_Gui, Igain_Gui and Dgain_Gui).

From the GUI environment, the voltage loop can also be tuned using two poles (fp1, fp2), two zeroes (fz1, fz2) and gain (Kdc). These parameters provide b2_Gui, b1_Gui, b0_Gui, a2_Gui and a1_Gui coefficients in I5Q10 form that are then converted to the five Q26 coefficients for the 2P2Z controller. Although not recommended, b2_Gui, b1_Gui, b0_Gui, a2_Gui and a1_Gui values can also be directly changed from the Code Composer Studio environment using watch views. For details on these calculations, see the HVPSFB-Calculations.xls file located at www.ti.com/controlsuite. The equations for deriving coefficient values based on poles, zeroes, gain and switching frequency are also clear from the GUI source file.

This project allows easy evaluation of both methods of loop tuning by providing the ability to easily switch between coefficients during execution. This can be done by simply clicking on the 2P2Z(On) and PID(Off) button on the GUI or changing the pid2p2z_GUI variable to 0 or 1 on the watch view from Code Composer Studio. PID-based loop tuning (pid2p2z_GUI = 0) from the GUI environment was used as a starting point. Poles, zeroes and gain corresponding to these PID tuned coefficients were then used as a starting point for further loop tuning based on the second method (pid2p2z_GUI = 1). Much better results were then achieved by changing poles, zeroes and gain from the GUI environment to tune for optimum dynamic performance. By default, coefficients based on these tuned poles, zeroes and gain values (pid2p2z_GUI = 1 – default) are used.

NOTE: When tuning the system using the 2-pole 2-zero controller coefficient adjustments, if a choice of poles, zeroes and Kdc values is made in the GUI such that it result in the magnitude of any of the coefficients (b2, b1, b0, a2, a1) to be greater than or equal to 32, then these coefficient values are not sent to the controller by the GUI.

6.4 Procedure

6.4.1 Build and Load Project

Use the following steps to quickly execute this build using the pre-configured work environment:

1. Follow Steps 1 to 2 exactly as in build 1. If you were working on build 1 the last time Code Composer Studio was used, the same workspace should open up with the project.
2. If this is not the case, you can open the workspace used for build1 by clicking File → Switch Workspace and then navigating to the correct workspace. If a workspace was not saved or got deleted, follow Steps 3 and 4 exactly as in build1.
3. Locate and inspect the initialization code specific to build 2 in the main file. This is where all the control blocks are configured, initialized and connected in the control flow.
4. Select the Incremental build option as 2 in the *HVPSFB-Settings.h*.


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

5. Click Project → "Rebuild All" button and watch the tools run in the build window.
6. Click Target → "Debug Active Project". The program will be loaded into the FLASH or RAM memory depending on the project configuration. This project comes in only F2802x_FLASH configuration. You should now be at the start of Main().

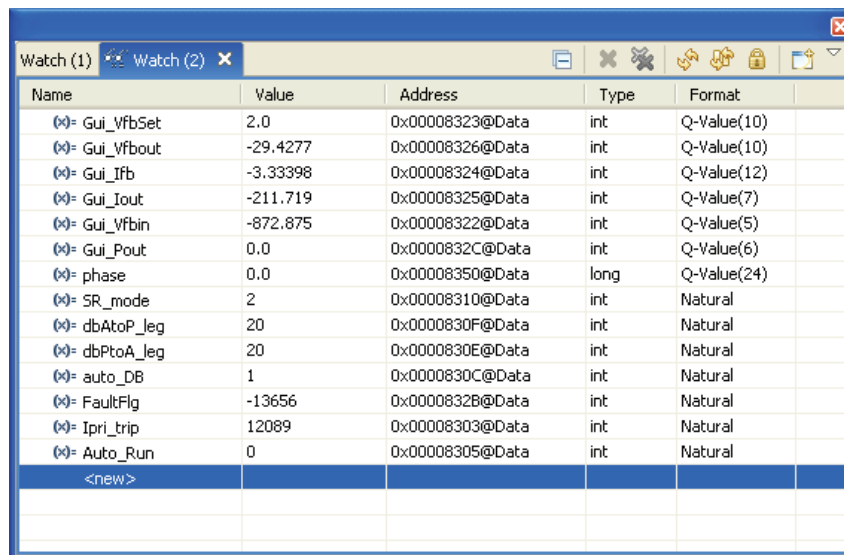
6.4.2 Debug Environment Windows

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

If a watch view did not open when the debug environment was launched, open a new watch view and add various parameters to it by following the procedure given below.

1. Click View → Watch on the menu bar.
2. Click the "Watch (1)" tab.
3. Click on the  button if a watch view is already open from the debug environment saved for build 1. A "Watch (2)" tab will open and you can then drag it to be viewed in a window of your choice. You can add any variables to this watch view tab.
4. Type the symbol name of the variable you want to watch in the empty box of the "Name" column and press enter on keyboard. Be sure to modify the "Format" as needed. The watch view should look something like the following graphic.

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



Name	Value	Address	Type	Format
(x)= Gui_VfbSet	2.0	0x00008323@Data	int	Q-Value(10)
(x)= Gui_Vfbout	-29.4277	0x00008326@Data	int	Q-Value(10)
(x)= Gui_Ifb	-3.33398	0x00008324@Data	int	Q-Value(12)
(x)= Gui_Iout	-211.719	0x00008325@Data	int	Q-Value(7)
(x)= Gui_VfbIn	-872.875	0x00008322@Data	int	Q-Value(5)
(x)= Gui_Pout	0.0	0x0000832C@Data	int	Q-Value(6)
(x)= phase	0.0	0x00008350@Data	long	Q-Value(24)
(x)= SR_mode	2	0x00008310@Data	int	Natural
(x)= dbAtoP_leg	20	0x0000830F@Data	int	Natural
(x)= dbPtoA_leg	20	0x0000830E@Data	int	Natural
(x)= auto_DB	1	0x0000830C@Data	int	Natural
(x)= FaultFlg	-13656	0x0000832B@Data	int	Natural
(x)= Ipri_trip	12089	0x00008303@Data	int	Natural
(x)= Auto_Run	0	0x00008305@Data	int	Natural
<new>				

Note that there are additional variables in the watch view.

5. *Gui_VfbSet* is used to set the output voltage command.

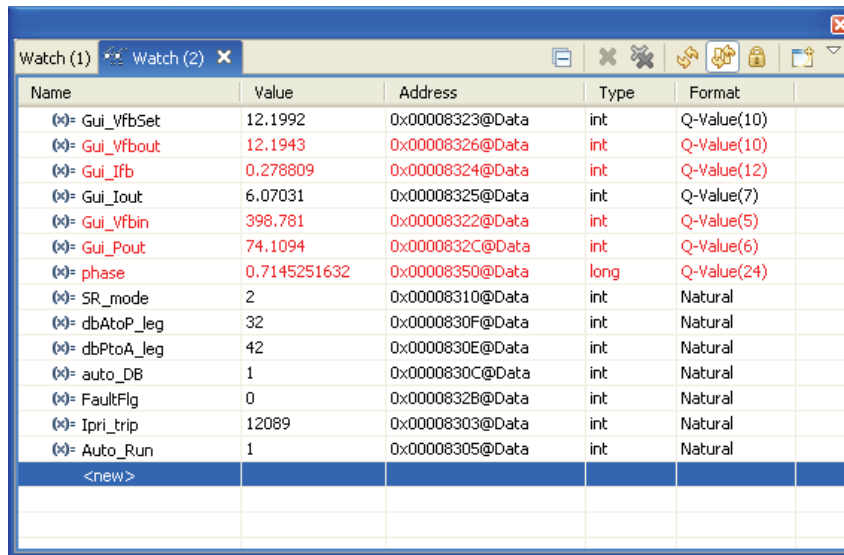
6.4.3 Run the Code

1. Follow steps 1 to 2 of build 1 procedure to enable real-time mode and continuous refresh for the watch views and also for changing the continuous refresh interval for the watch view if needed.
2. Run the code by using the <F8> key, or using the Run button on the toolbar, or using Target → Run on the menu bar.

3. Apply an appropriate resistive load to the PSFB system at the DC output. A load that draws around 3A – 6A current at 12 V output is a good starting point.

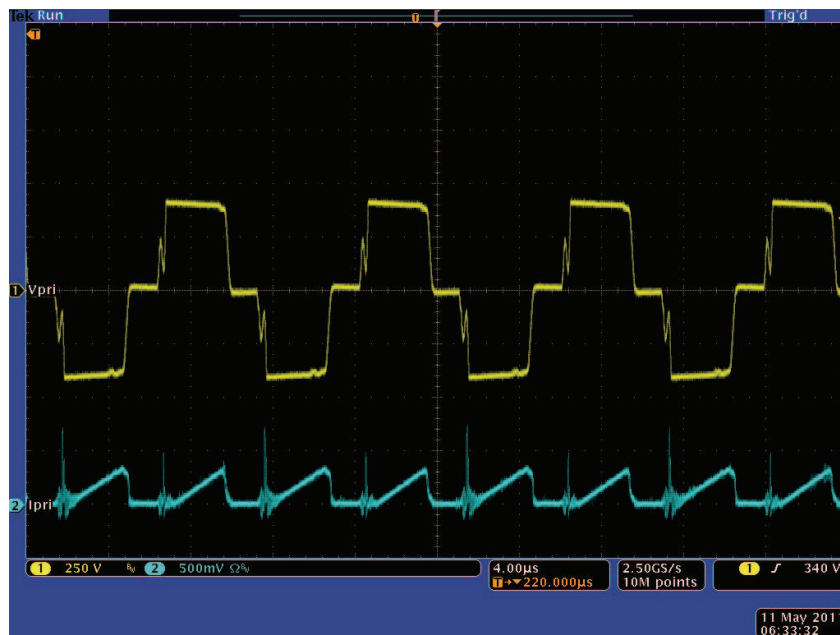
NOTE: For safety reasons, it is recommended to use an isolated DC source to supply 400 V DC input to the board.

4. Power the input at J1, J2 with 400 V DC.
5. By default, *Auto_Run* is set to 1. If it is not, make it 1 from the watch view. The output voltage should now start ramping up to 12 V. This output voltage ramp up rate can be changed by changing the variable *VfbSlewRate*.
6. Here is the watch view that corresponds to the operation of the system with 12 V at the output with an input voltage of around 400 V and a load of around 6A output.



Name	Value	Address	Type	Format
Gui_VfbSet	12.1992	0x00008323@Data	int	Q-Value(10)
Gui_Vfbout	12.1943	0x00008326@Data	int	Q-Value(10)
Gui_Ifb	0.278809	0x00008324@Data	int	Q-Value(12)
Gui_Iout	6.07031	0x00008325@Data	int	Q-Value(7)
Gui_VfbIn	398.781	0x00008322@Data	int	Q-Value(5)
Gui_Pout	74.1094	0x0000832C@Data	int	Q-Value(6)
phase	0.7145251632	0x00008350@Data	long	Q-Value(24)
SR_mode	2	0x00008310@Data	int	Natural
dbAtoP_leg	32	0x0000830F@Data	int	Natural
dbPtoA_leg	42	0x0000830E@Data	int	Natural
auto_DB	1	0x0000830C@Data	int	Natural
FaultFlg	0	0x0000832B@Data	int	Natural
Ipri_trip	12089	0x00008303@Data	int	Natural
Auto_Run	1	0x00008305@Data	int	Natural
<new>				

7. The following graphic shows waveforms captured under this condition.



8. By default, the synchronous rectifiers are operated in mode 2. You can change their mode of operation by changing the *SR_mode* variable to 0, 1 or 2 from the watch view. Observe the change in the amount of input current being drawn and the change in output voltage with the different *SR* modes.


You can also probe the PWM waveforms driving the synchronous rectifier switches. Do not change between different SR modes when operating at very low loads (below 3A). In these cases, use the default SR mode 2.

9. Observe the effect of varying load on the output voltage and input current. There should be virtually no effect on the output voltage. Similarly, observe the effect of varying the input voltage. Again, there should be virtually no effect on the output voltage.

NOTE: Make sure that these changes are made within the abilities of the board as listed in the specifications section of this document.

10. Different waveforms, like the PWM gate drive signals, input voltage and current and output voltage may also be probed using an oscilloscope. Appropriate safety precautions should be taken and appropriate grounding requirements should be considered while probing these high voltages and high currents for this isolated DC-DC converter.
11. Fully halting the MCU when in real-time mode is a two-step process. Wait a few seconds with the 400 V DC input turned off. First, halt the processor by using the Halt button on the toolbar, or by using



Target → Halt. Then, click the  button again to take the MCU out of real-time mode and then reset the MCU.

12. Close Code Composer Studio.

7 References

1. *UCC28950 600-W, Phase-Shifted, Full-Bridge Application Report* ([SLUA560](#))
2. *Using the UCC28950EVM-442 User's Guide* ([SLUU421](#))
3. H. Nene, "Implementing Advanced Control Strategies for Phase Shifted Full-Bridge DC-DC Converters using Micro-Controllers" PCIM Europe 2011, Nuremberg, Germany.
4. *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide* ([SPRUGE9](#))
5. HVPSFB-Calculations.xls – a spreadsheet showing key calculations for the HVPSFB project
www.ti.com/controlsuite:...\controlSUITE\development_kits\HVPSFB_v1.1\~Docs
6. QSG-HVPSB-Rev1.1.pdf– gives an overview on how to quickly run and experiment with the HVPSFB project using an intuitive GUI interface.
www.ti.com/controlsuite:...\controlSUITE\development_kits\HVPSFB_v1.1\~Docs
7. HVPSFB_RevB-HWdevPkg – a folder containing various files related to the baseboard and Piccolo-A controller card.
www.ti.com/controlsuite:...\controlSUITE\development_kits\HVPSFB_v1.1\~HVPSFB_HWdevPkg\RevB

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com