TI Designs: TIDA-BIDIR-400-12
Bidirectional DC-DC Converter

Description

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize the system. TI Designs help *you* accelerate your time to market.

Resources

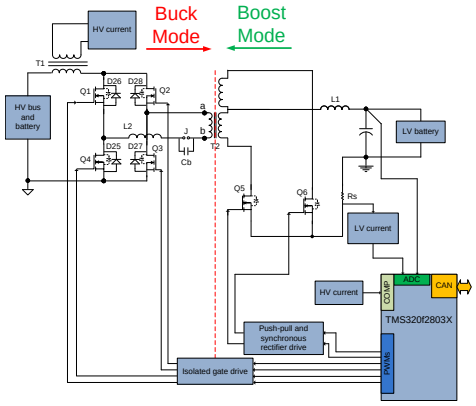
TIDM-BIDIR-400-12	Design Folder
TI E2E™ Community	ASK Our E2E Experts

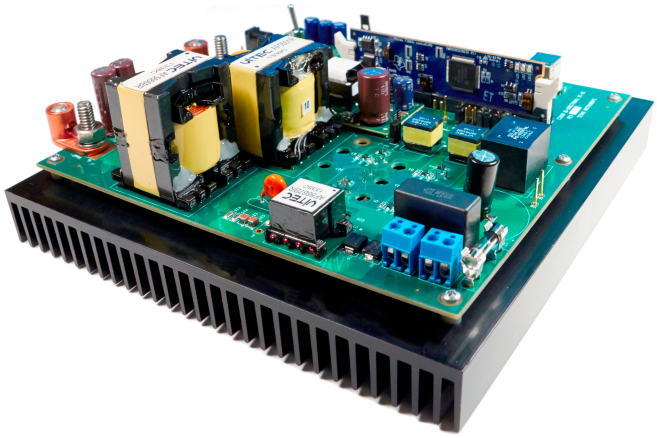
Features

- 200-V DC to 400-V DC HV Bus Voltage Range
- 9-V DC to 13.5-V DC LV Bus Voltage Range
- 300-W Rated Output Operation in Either Direction
- 33-A Rated Current on LV Bus and 1.8-A Rated Current on HV Bus
- Seamless on-the-fly Transitions Between Buck and Boost Modes
- Phase-Shifted Full-Bridge Operation in Buck Mode
- Current-Fed Push-Pull Operation in Boost Mode
- 100-kHz Switching Frequency
- Voltage Mode Control (VMC) and Average Current Mode Control (ACMC) of Output Inductor Current
- Multiple Synchronous Rectification (SR) Switching Schemes
- Fault Protection: Overcurrent, Undervoltage, and Overvoltage

Applications

- Automotive (HEV – Hybrid Electric Vehicles and EV – Electric Vehicles)
- General Digital Power (Industrial Power and Battery Back-Up Systems)





An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 Introduction

Bidirectional DC-DC converters are used in applications where bidirectional power flow may be required. In hybrid electric vehicles (HEVs) and electric vehicles (EVs), these bidirectional converters charge a low-voltage (12 V) battery during normal operation (buck mode) and charge or assist the high-voltage (400 V/600 V) battery or bus in emergency situations like when a high-voltage battery has discharged to a very low energy or capacity level (boost mode). A typical system consists of a full-bridge power stage on the high-voltage (HV) side, which is isolated from a full-bridge or a current-fed push-pull stage on the low-voltage (LV) side.

In this implementation, closed-loop control for both directions of power flow is implemented using TI 32-bit microcontroller TMS320F28035, which is placed on the LV side. Traditionally, microcontrollers have been restricted to performing only supervisory or communications tasks in these systems. With the availability of high-performing microcontroller devices, microcontrollers can close control loops in these systems and handle the traditional microcontroller functions. The transition to digital power control indicates that functions previously implemented in hardware are now implemented in software. In addition to the flexibility, this capability adds to and simplifies the system. These systems can implement advanced control strategies to optimally control the power stage under different conditions and also provide system-level intelligence to make safe and seamless transitions between operation modes and pulse width modulated (PWM) switching patterns.

This document presents the details of this microcontroller-based implementation of an isolated bidirectional DC-DC converter. A phase-shifted full-bridge (PSFB) with synchronous rectification controls power flow from a 400-V bus or battery to the 12-V battery in step-down mode, while a push-pull stage controls the reverse power flow from the low-voltage battery to the high-voltage bus or battery in boost mode. This design is rated for up to 300 W of output power in either mode. The voltage on the high-voltage bus can be in the range from 400 V to 200 V, while that on the low-voltage bus can be in the range from 13.5 V to 9 V. Voltage mode control and average current-mode control are implemented for both modes of operation. Various PWM switching schemes have been implemented with seamless transitions between switching modes and also between the two operation modes.

1.1 Basic Operation

Buck Mode

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 lets all the switching devices to switch with zero-voltage switching (ZVS), resulting in lower switching losses and an efficient converter.

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

A DC-DC converter system can be controlled in various modes like voltage mode control (VMC), average current mode control (ACMC), or peak current mode control (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 circuitry. With a microcontroller-based system, all these modes can be experimented with on the same design with minimal or no additional changes. Figure 1 shows a simplified circuit of a phase-shifted full bridge. MOSFET switches Q1, Q2, Q3, and Q4 form the full bridge on the primary side of the T1 transformer. Q1 and Q4 are switched at 50% duty and 180 degrees out of phase with each other. QC and QD are switched at 50% duty and 180 degrees out of phase with each other. The PWM switching signals for leg Q2–Q3 of the full bridge are phase-shifted with respect to those for leg Q1–Q4. The amount of this phase shift decides the amount of overlap between diagonal switches, which decides the amount of energy transferred. D5 and D6 provide diode rectification on the secondary, while Lo and Co form the output filter. Inductor LR provides assistance to the transformer leakage inductance for resonance operation with MOSFET capacitance and facilitates zero voltage switching (ZVS). Figure 2 provides the switching waveforms for the system in Figure 1.

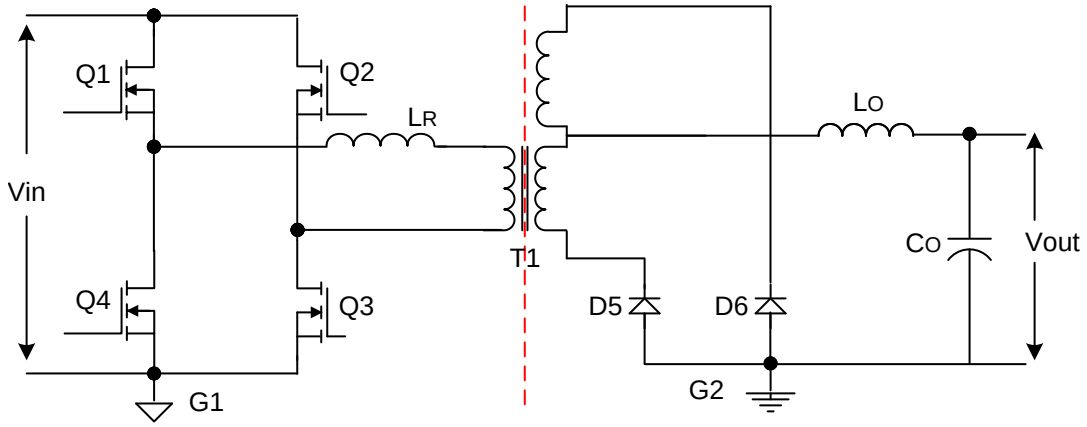


Figure 1. Buck Mode Power Stage

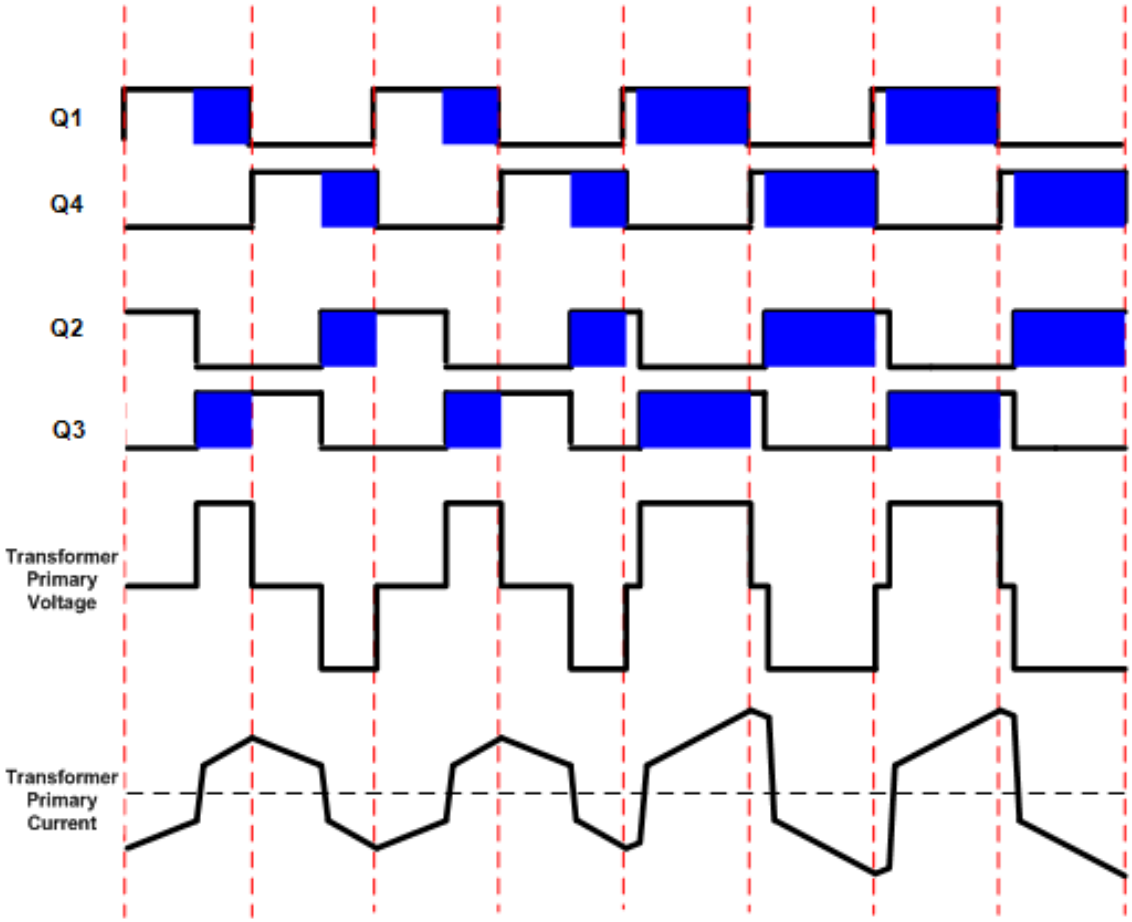


Figure 2. Buck Mode PWM Waveforms

Boost Mode

The synchronous rectifier switches are the push-pull switches in boost mode. The buck mode output inductor acts as a current source in this mode letting this topology work as a current-fed push-pull converter. Full-bridge switches on the HV side may be kept off and their body diodes used for rectification. The full-bridge switches are used for active rectification in the boost mode. The push-pull switches are driven with PWM signals with greater than 50% duty cycles that are 180 degrees out of phase with each other.

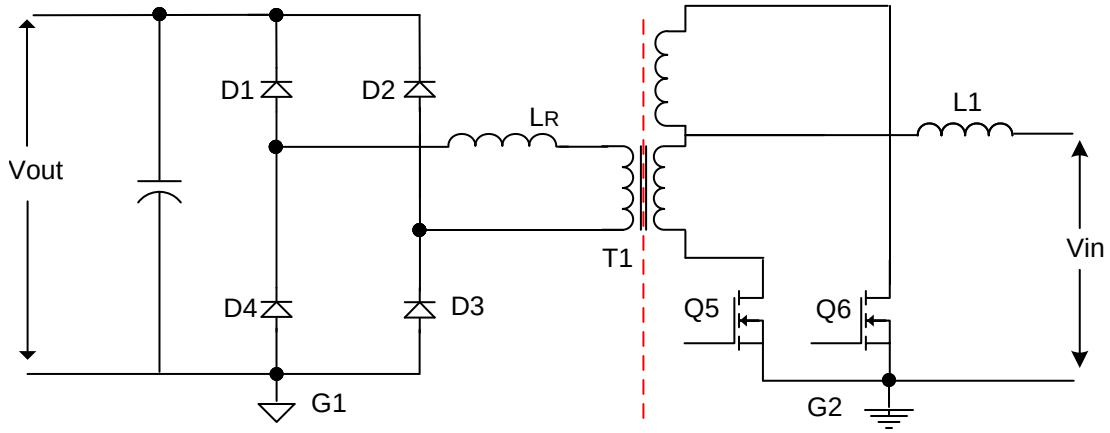


Figure 3. Boost Mode Power Stage

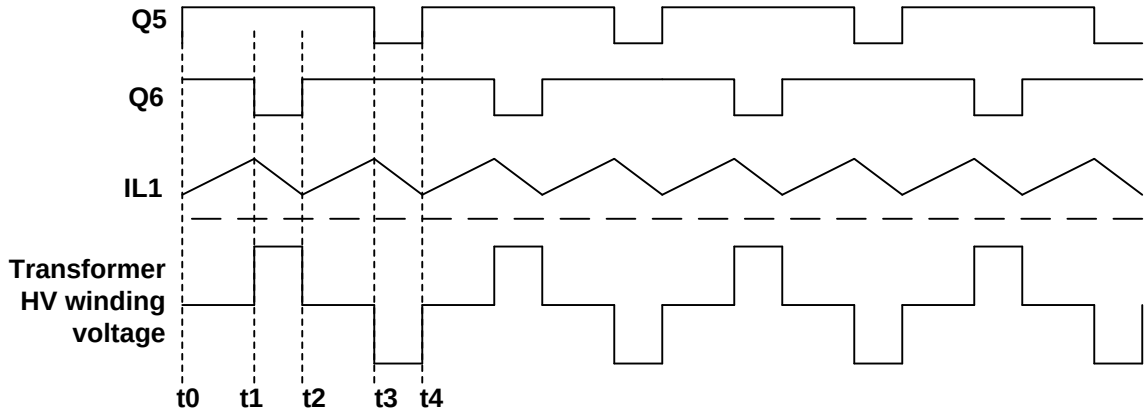


Figure 4. Boost Mode PWM Waveforms

- **t0 – t1:** During this time, Q5 and Q6 are on simultaneously. The inductive energy in the low-voltage winding of the transformer and that in the boost inductor (L1) increases.
- **t1 – t2:** At t1, Q6 is turned off and the stored inductive energy on the LV side is transferred to the HV side through diodes D1 and D3.

The operation during t2–t3 is the same as t0–t1, while that during t3–t4 is similar to t1–t2, except that Q5 is turned off at t3 and diodes D2 and D4 conduct on the HV side.

In this mode of operation, the amount of energy transferred to the HV side is decided by the duty cycle of the signals driving switches Q5 and Q6. Unlike the phase-controlled buck mode, this is a duty-controlled operation.

1.2 Implementation on Bidirectional DC-DC Board

Figure 5 shows a simplified block diagram of the circuit implemented on the bidirectional DC-DC board. In the system shown Figure 5, there are four MOSFET switches (Q1–Q4) that form a full-bridge on the HV side of the isolation transformer and two MOSFET switches (Q5–Q6) on the center-tapped LV side that work as synchronous rectifiers in buck mode and as push-pull switches in boost mode. In boost mode of operation relying solely on full-bridge MOSFET, body diodes (for rectification) significantly deteriorates system efficiency because of slow diode reverse recovery and high circulating currents. Schottky diodes (D25–D28) are connected in anti-parallel configuration to each of the full-bridge switches to overcome this drawback. Full-bridge MOSFETs are switched on and off at appropriate times to provide synchronous rectification in boost mode further improving system efficiency in this mode.

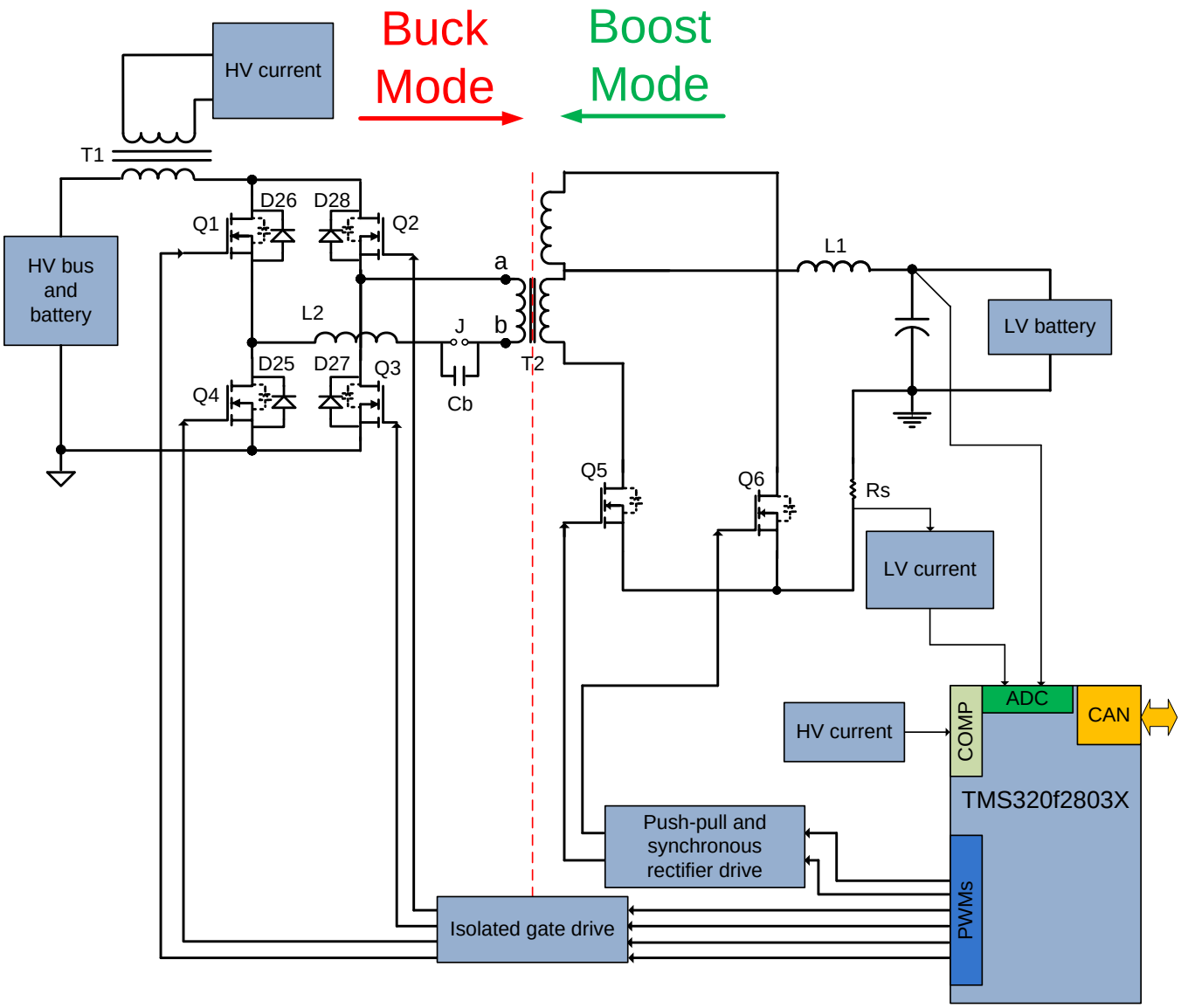


Figure 5. System Block Diagram

The control algorithm is implemented on a C2000™ microcontroller (MCU). The MCU interacts with the power stage by way of feedback signals and PWM outputs. The controller is placed on the LV side on this design. A crucial step when designing an isolated DC-DC system is deciding on the placement of the controller with respect to the location of isolation boundary. Placing the controller on the LV side is beneficial for systems that have multiple rails or handle many signals and control loops on the LV side or communicate with other systems in the application (on the LV side).

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

1.3 System Highlights

Following lists the key features of this implementation:

- 200-V DC to 400-V DC HV bus voltage range
- 9-V DC to 13.5-V DC LV bus voltage range
- 300-W rated output operation in either direction
- 33-A rated current on LV bus and 1.8-A rated current on HV bus
- Seamless on-the-fly transitions between buck and boost modes
- Phase-shifted full-bridge operation in buck mode
- Current-fed push-pull operation in boost mode
- 100-kHz switching frequency
- VMC and ACMC of output inductor current
- Multiple SR switching schemes
- Fault protection: overcurrent, undervoltage, overvoltage

2 Hardware and Resources

Figure 6 shows the key components of the hardware.

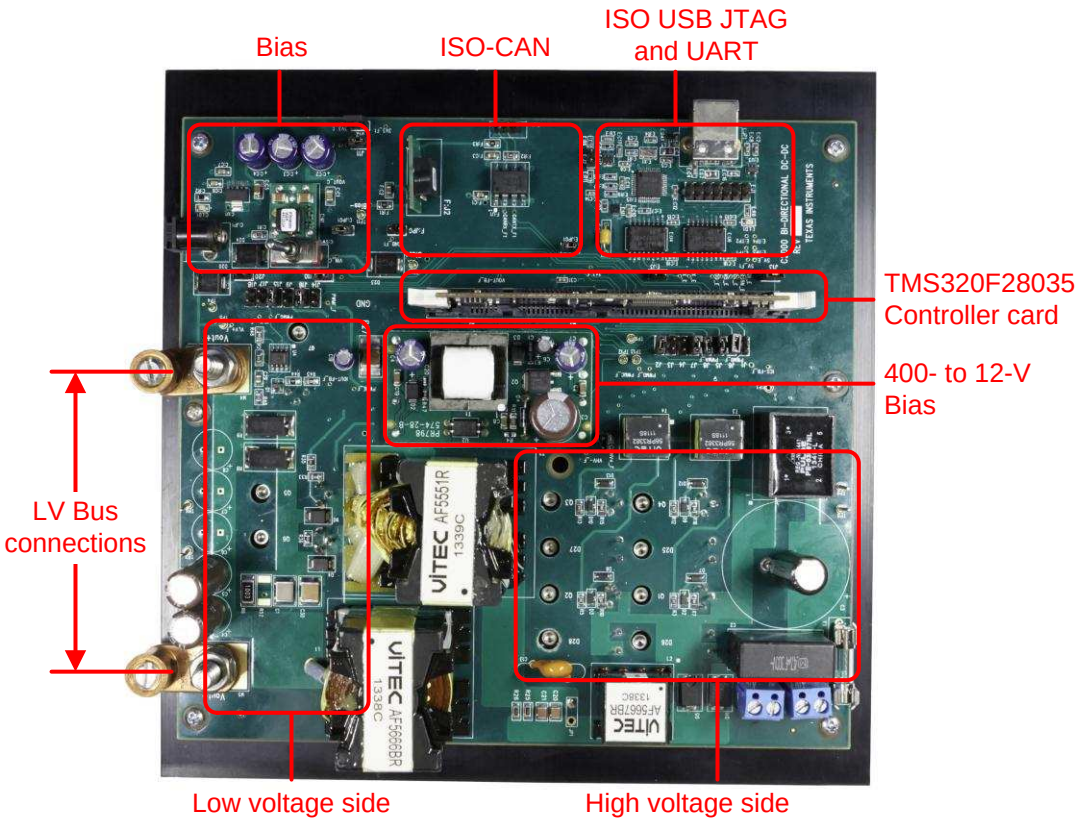


Figure 6. Bidirectional DC-DC Board

Table 1 lists the key signal connections between the F28035 controlCARD and the base board. Figure 7 also provides a relevant portion of the schematic.

Table 1. Key Signal Connections

Signal Name	Description	Connection to controlCARD
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 sync-rectifier/push-pull switch Q6	GPIO-07
ePWM-4B	PWM drive for sync-rectifier/push-pull switch Q5	GPIO-06
VLV-FB	Low-voltage bus – voltage feedback	ADC-A0
ILV-FB	Low-voltage current feedback (jumper J2 populated)	ADC-B3/COMP2A
ILV-FILT	Heavily filtered low-voltage current feedback	ADC-B2
VHV-FB	High-voltage bus – voltage feedback	ADC-B1
IHV-FB	Transformer high-voltage winding current	ADC-A2/COMP1A
IHV-FILT	Heavily filtered transformer high-voltage winding current	ADC-A1

This board provides jumper options for experimentation but some jumpers must be populated to operate the board. The following jumpers must be populated:

- C:JPG1
- E:JPG1
- J2
- J5
- J6
- J7
- J8
- J10
- J11
- J12
- J13
- J15
- J16
- J19

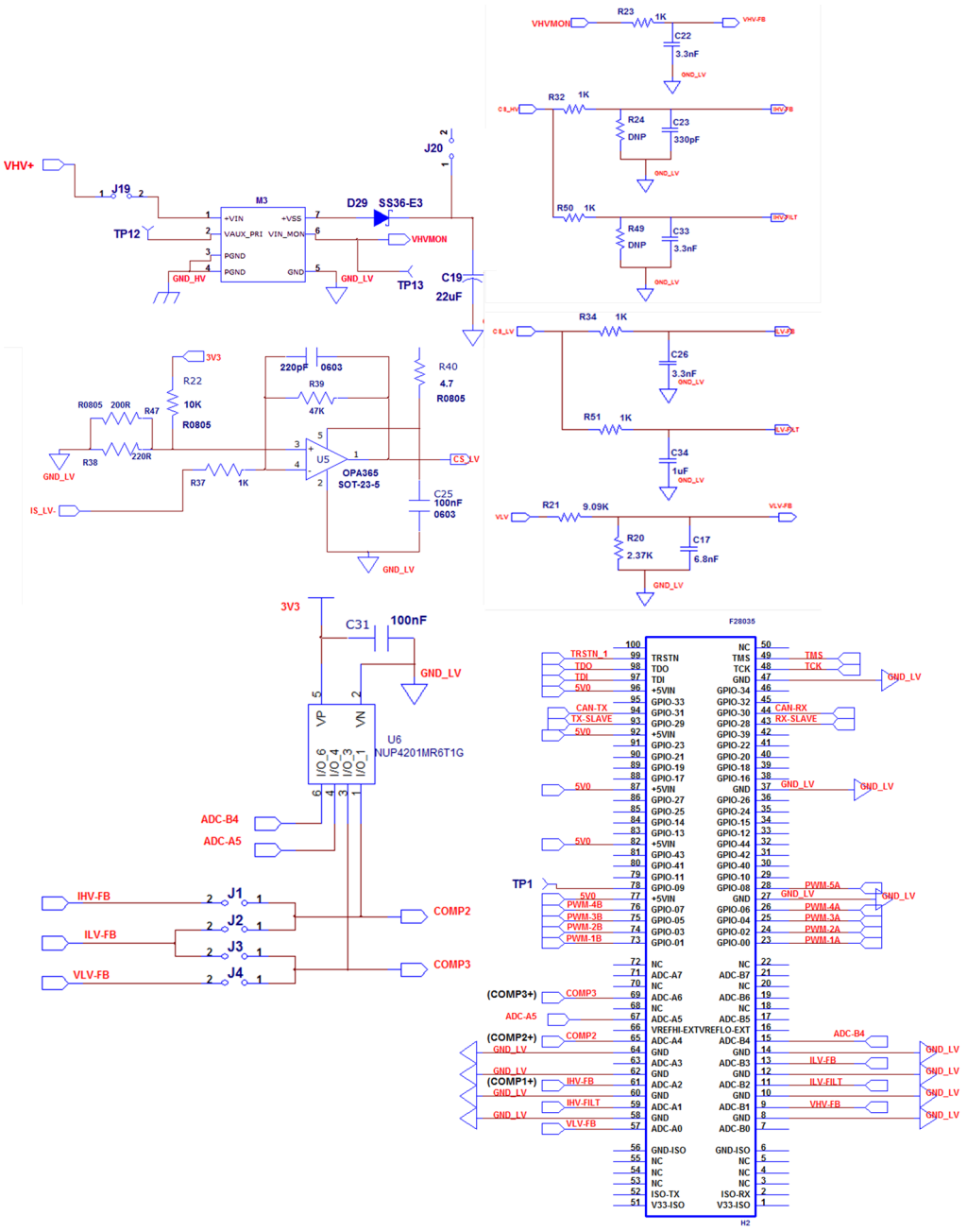


Figure 7. Base Board and controlCARD Signal Interface

3 Test Results

Figure 8 shows the test setup to test this board. DC power supplies were powered the HV and LV buses instead of batteries. A 150-V, 60-A diode (GeneSiC semiconductor, 1N2130A) connected the low-voltage power supply to the LV bus, while a 600-V, 6-A diode (Vishay semiconductor, G1756) connected the high-voltage power supply to the HV bus. An electronic load was used on the LV bus, while a resistive load bank was used on the HV bus.

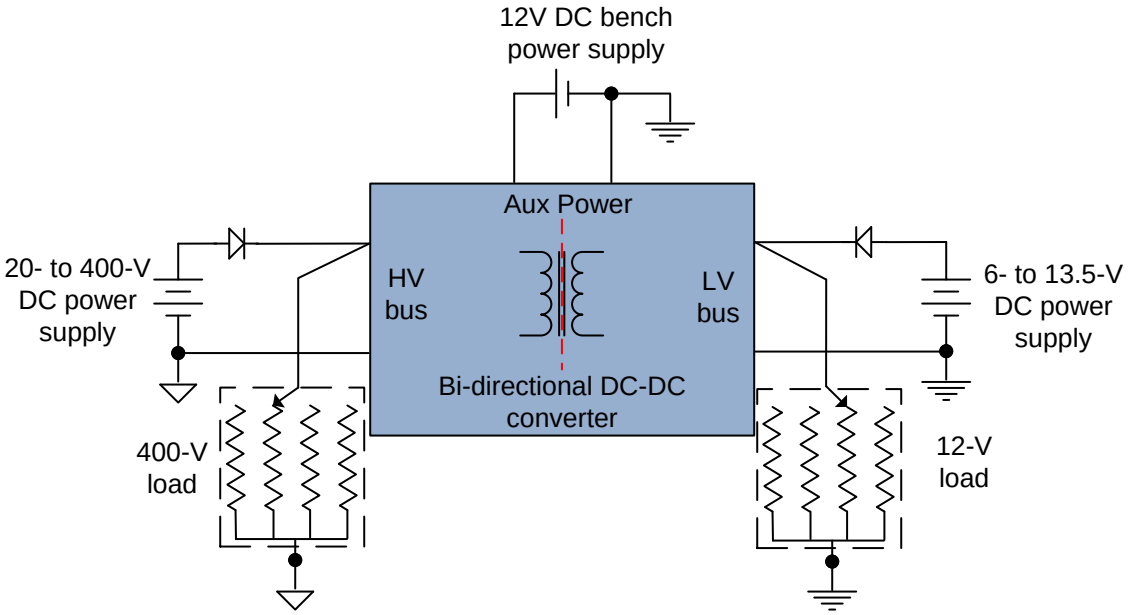


Figure 8. Test Setup

The following images provide some results using this board.

Buck Mode

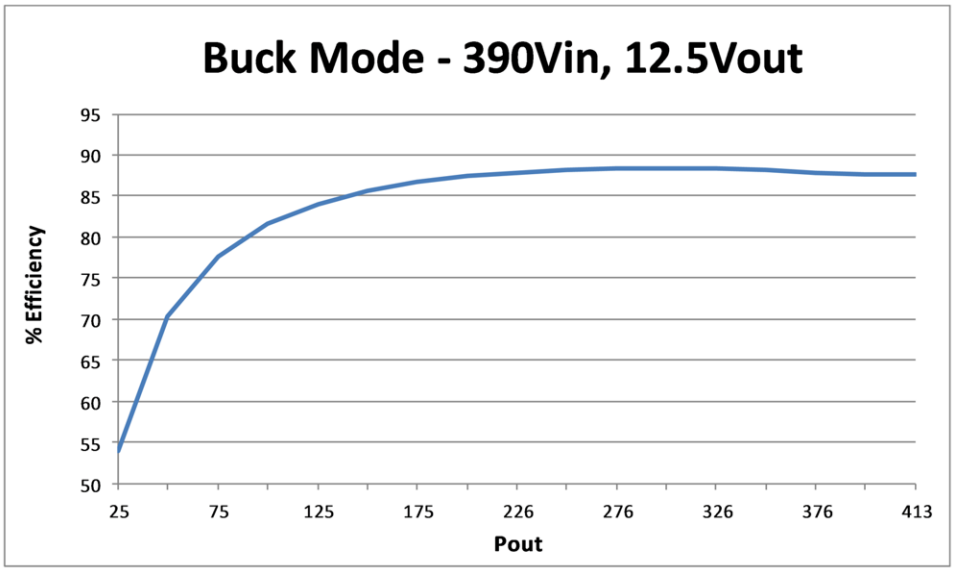


Figure 9. Efficiency vs Load (Buck Mode With 390 Vin)

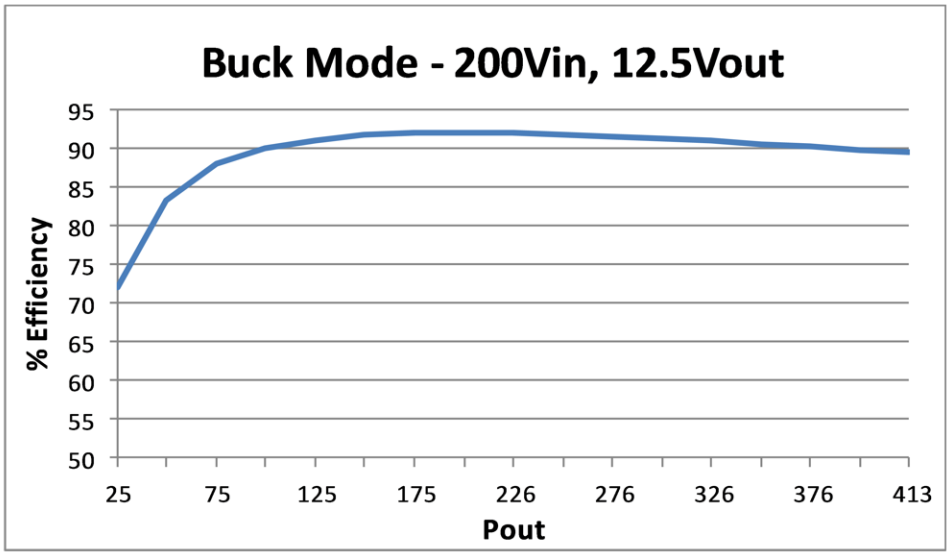


Figure 10. Efficiency vs Load (Buck Mode With 200 Vin)

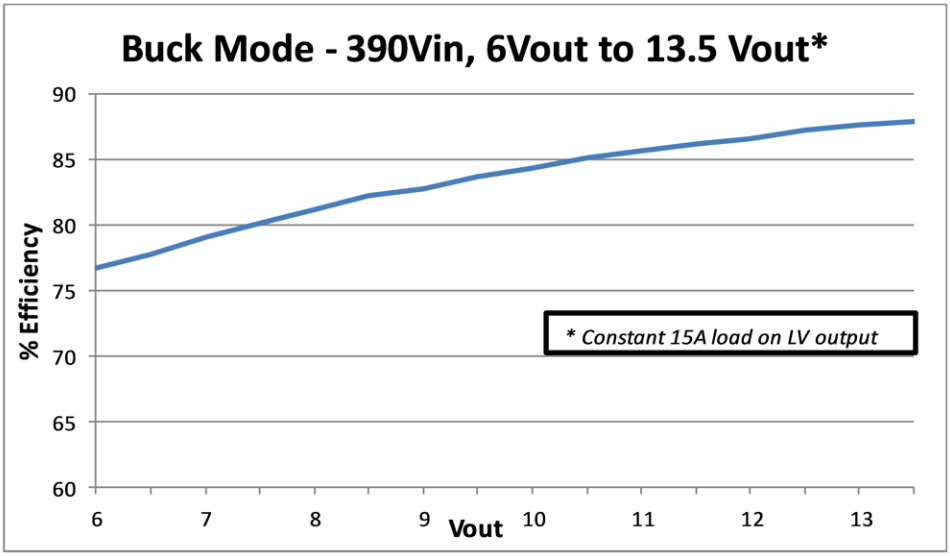


Figure 11. Efficiency vs Output Voltage (Buck Mode With 390 Vin)

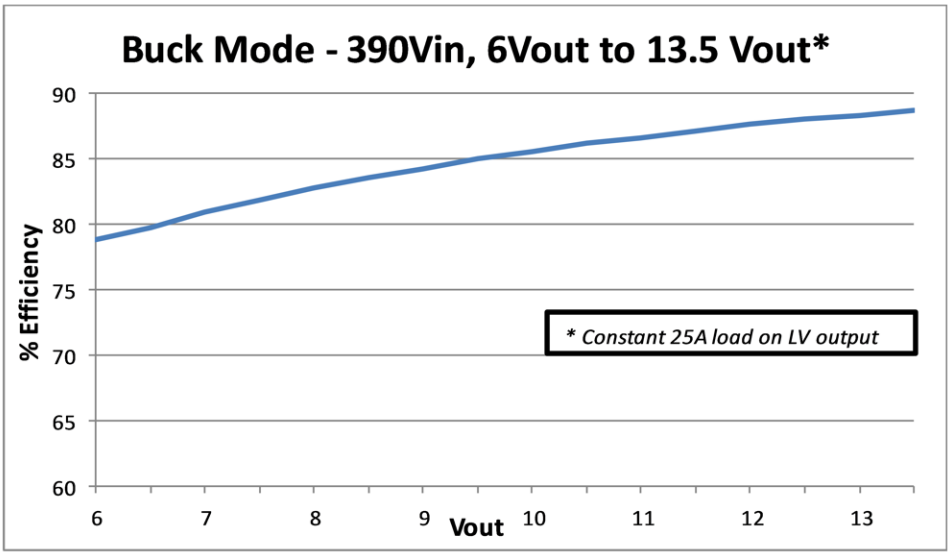


Figure 12. Efficiency vs Output Voltage (Buck Mode With 390 Vin)

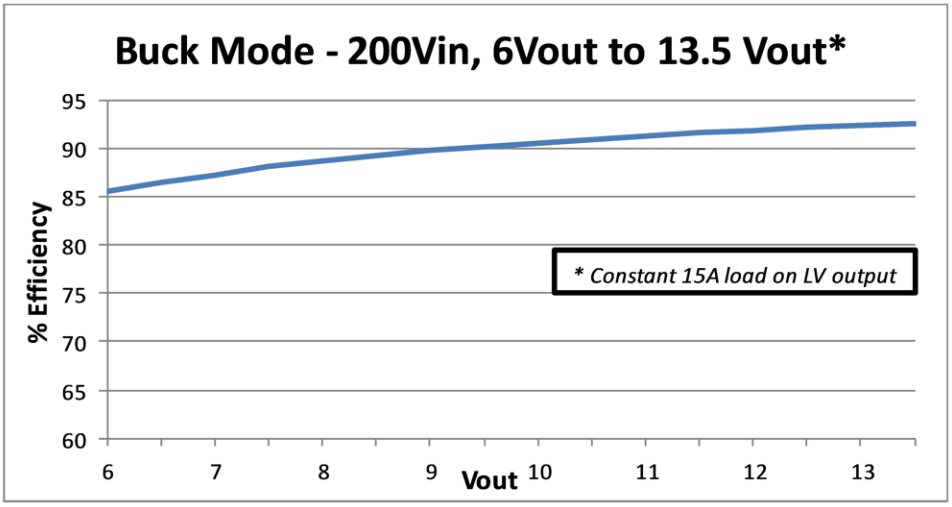


Figure 13. Efficiency vs Output Voltage (Buck Mode With 200 Vin)

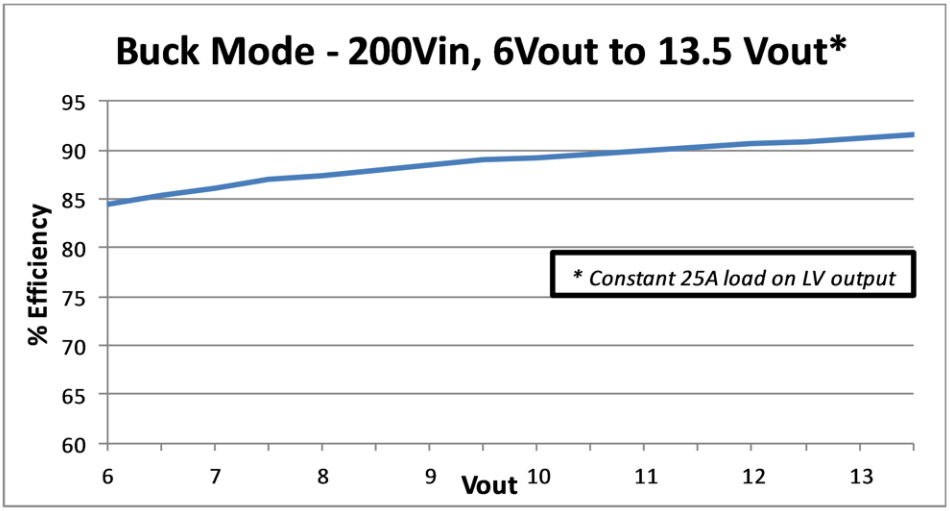


Figure 14. Efficiency vs Output Voltage (Buck Mode with 200 Vin)

Boost Mode

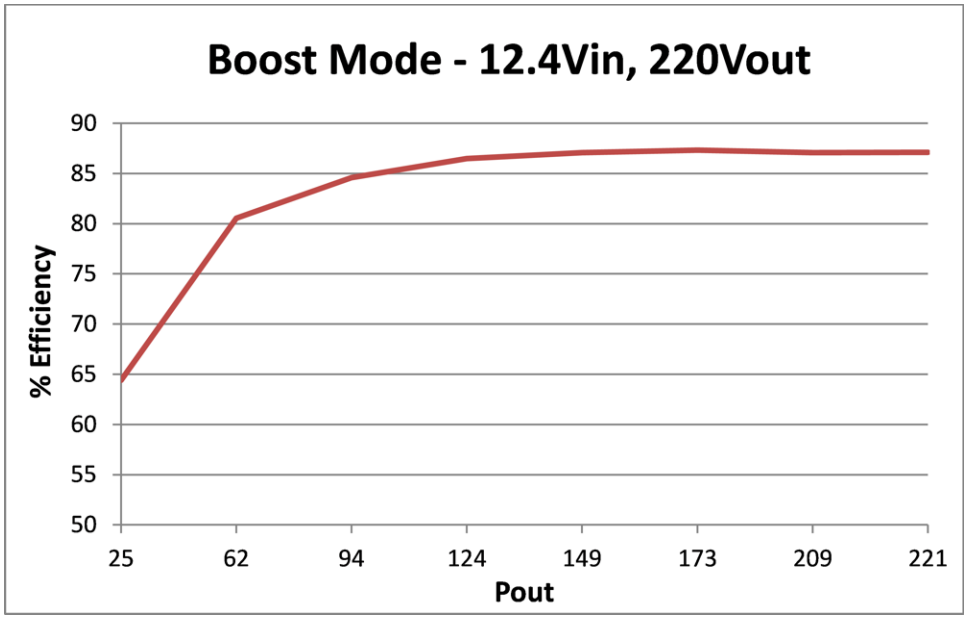


Figure 15. Efficiency vs Load (Boost Mode With 12.4 Vin, 220 Vout)

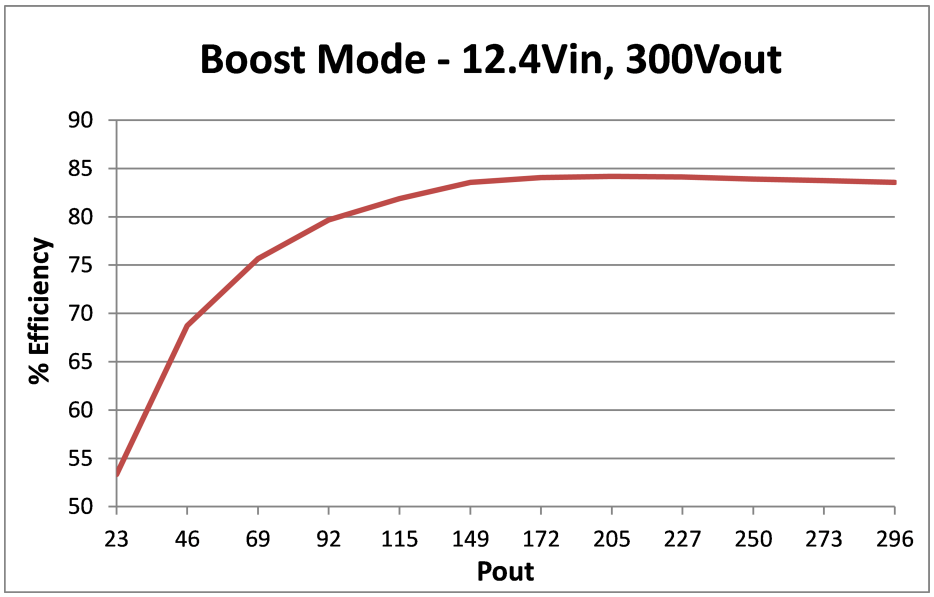


Figure 16. Efficiency vs Load (Boost Mode With 12.4 Vin, 300 Vout)

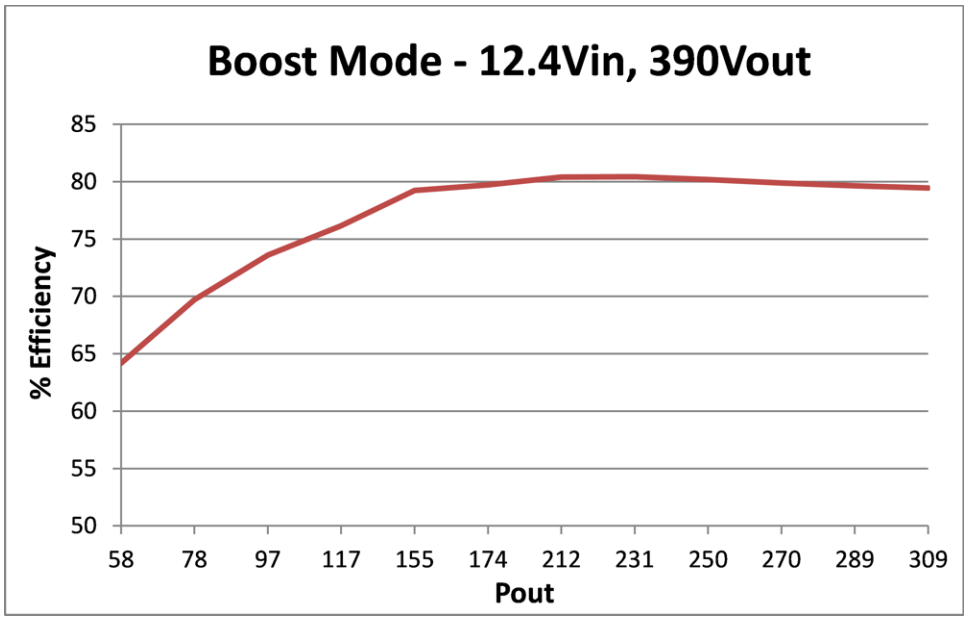


Figure 17. Efficiency vs Load (Boost Mode With 12.4 Vin, 390 Vout)

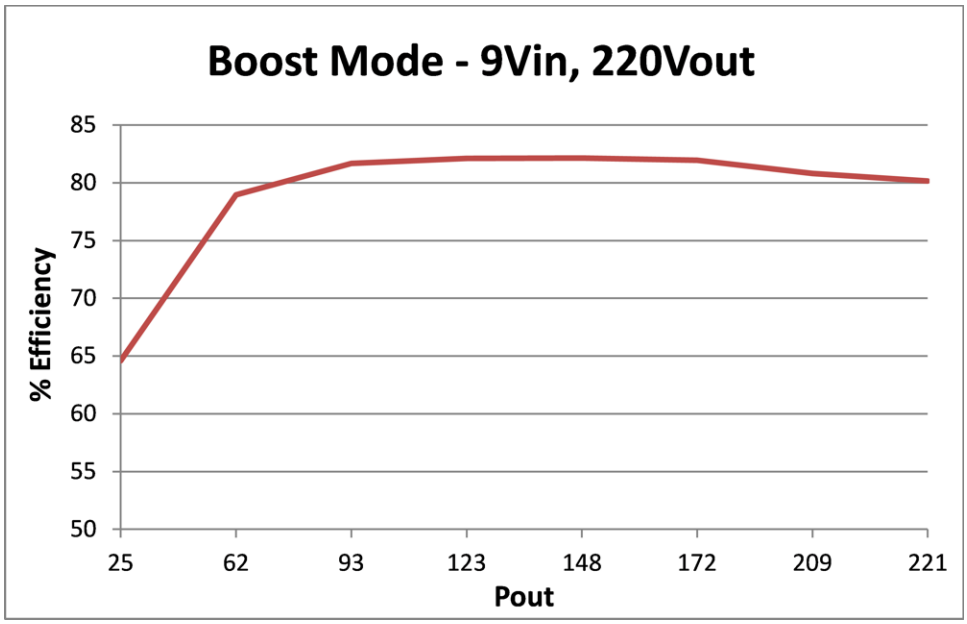


Figure 18. Efficiency vs Load (Boost Mode With 9 Vin, 220 Vout)

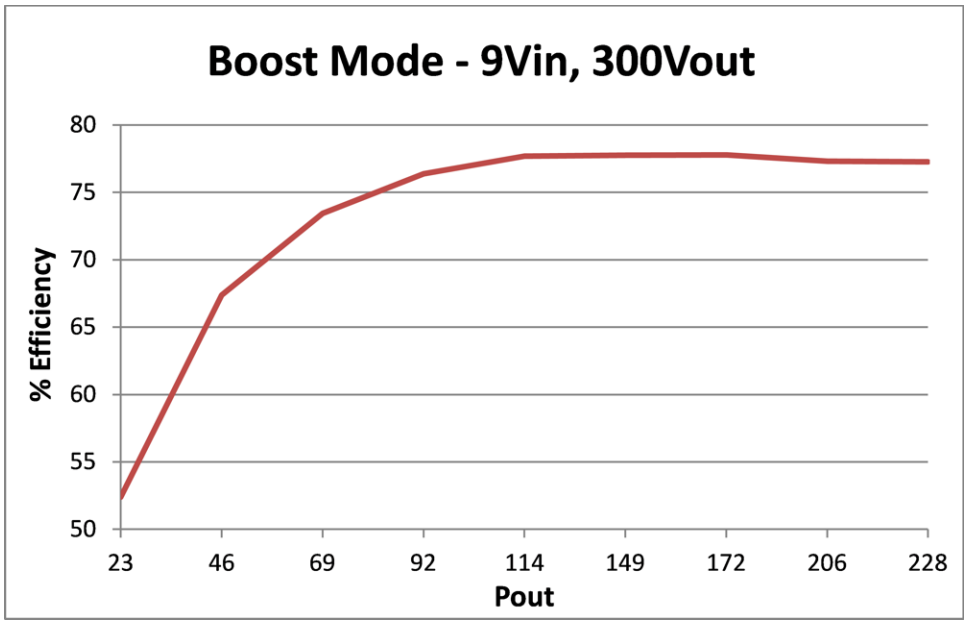


Figure 19. Efficiency vs Load (Boost Mode With 9 Vin, 300 Vout)

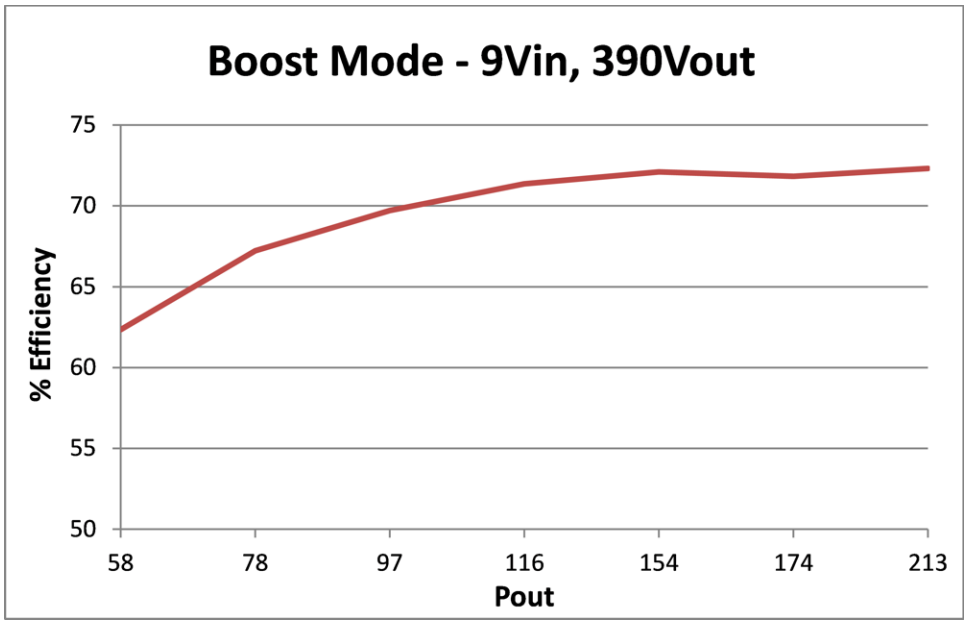


Figure 20. Efficiency vs Load (Boost Mode With 9 Vin, 390 Vout)

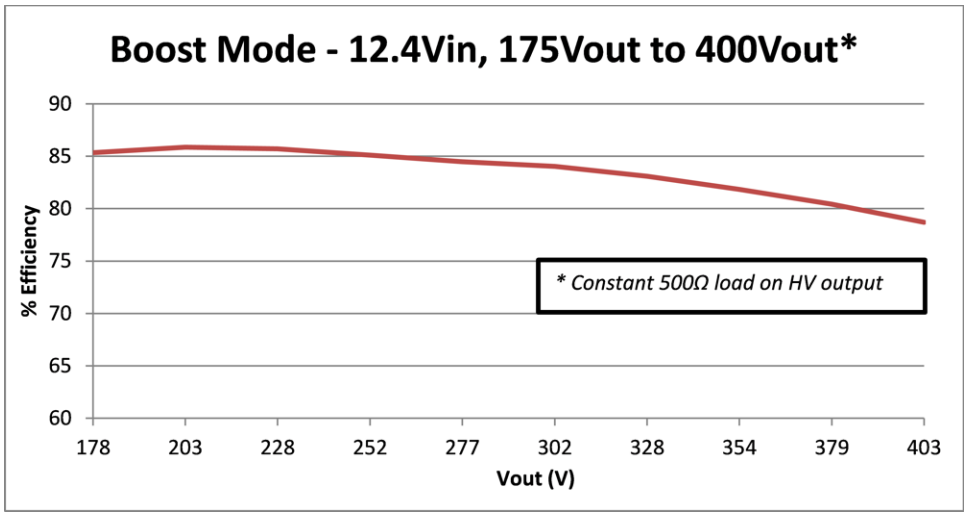


Figure 21. Efficiency vs Output Voltage (Boost Mode With 12.4 Vin)

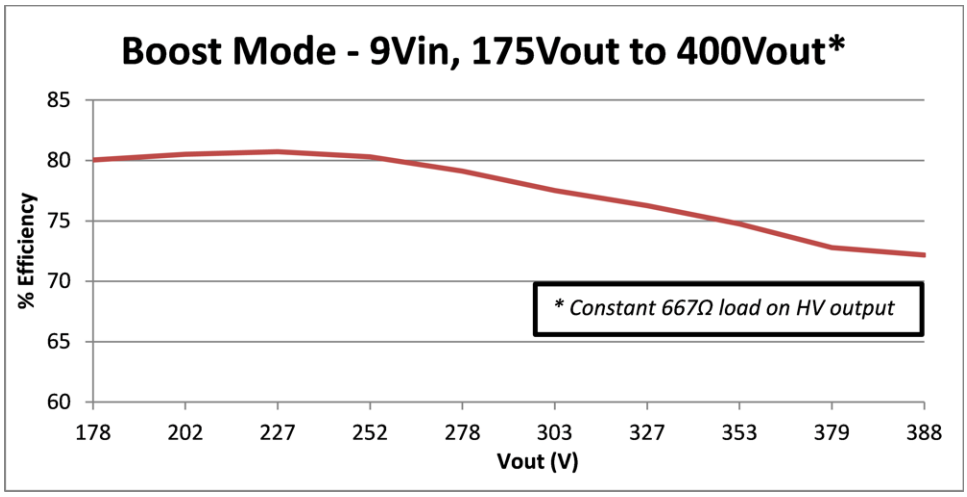


Figure 22. Efficiency vs Output Voltage (Boost Mode With 9 Vin)

Mode Transitions (V_{LV}, V_{HV} Voltages)

The following images provide voltage waveforms observed on the high-voltage bus and the low-voltage bus during mode transitions under various operating conditions. The voltage set point in the new mode (after a mode change command) is set at a point higher than the previous voltage on that bus. For example, when changing from buck mode to boost mode the voltage on the HV bus (V_{HV}) is set 4 V higher than the previous V_{HV} voltage. When changing from boost mode to buck mode, the V_{LV} voltage is set 0.5 V higher than the previous V_{LV} voltage. As a result, an expected drop or lift in voltage is seen on the two buses during mode transitions.

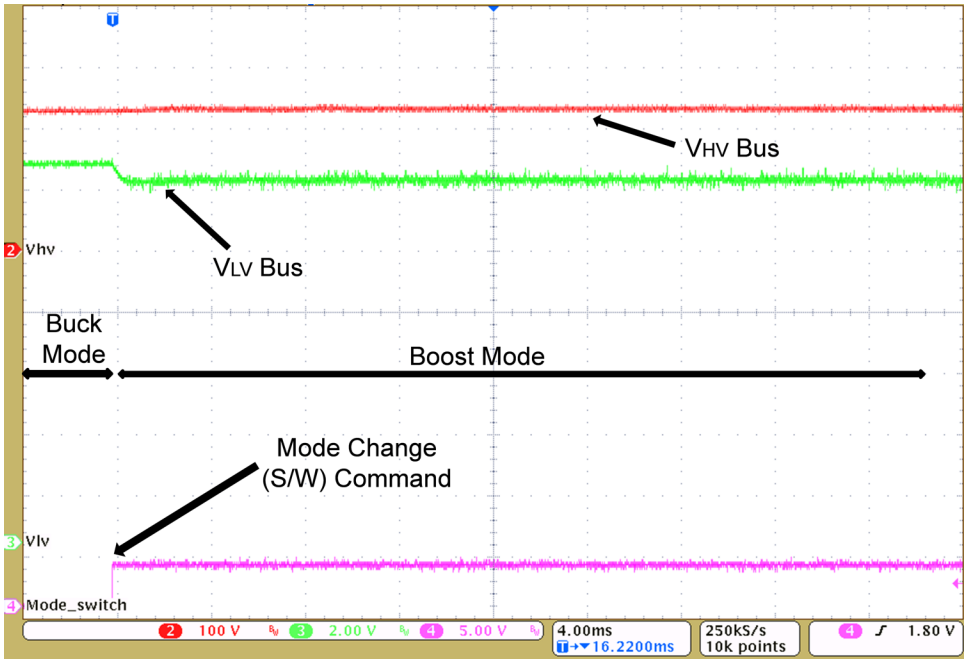


Figure 23. Buck to Boost Mode Transition (V_{HV} = 225 V, 2-kΩ load; V_{LV} = 12.6 V, 3-Ω load)

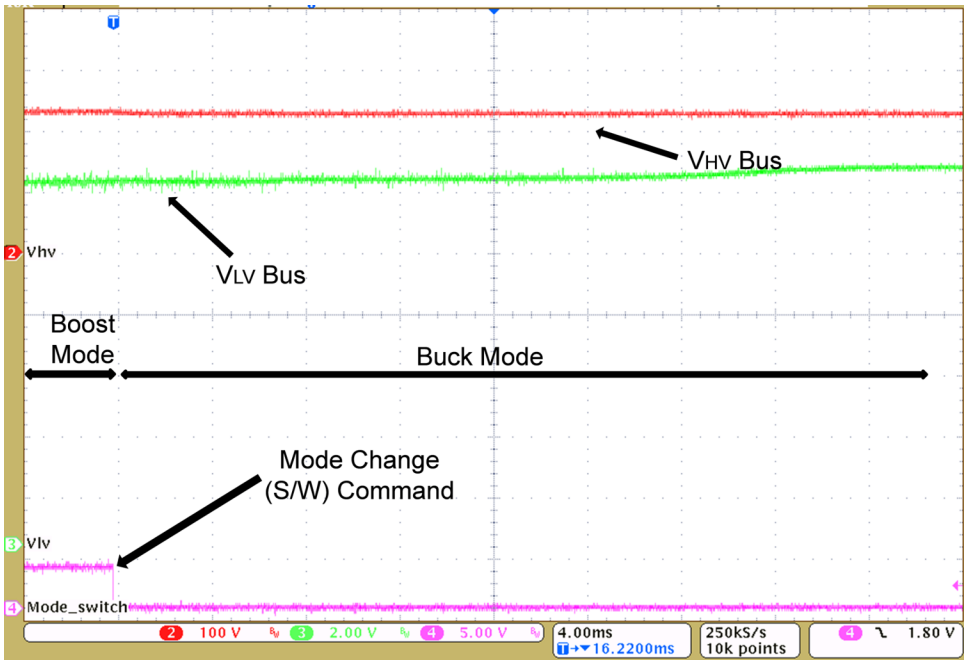


Figure 24. Boost to Buck Mode Transition (V_{HV} = 225V, 2-kΩ load; V_{LV} = 12.6 V, 3-Ω load)

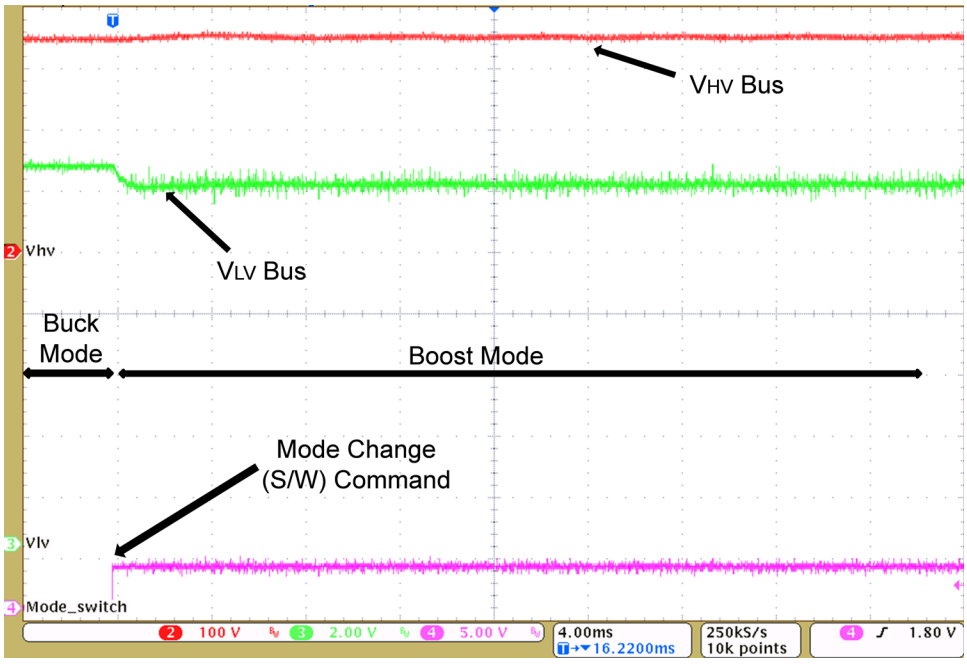


Figure 25. Buck to Boost Mode Transition (V_{HV} = 350 V, 2-kΩ load; V_{LV} = 12.6 V, 10-Ω load)

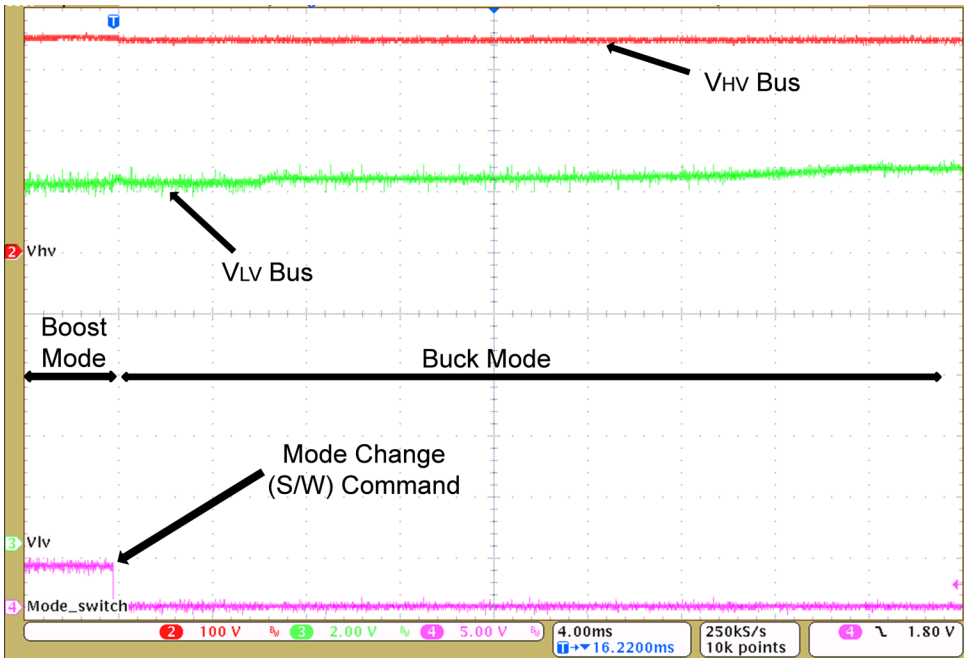


Figure 26. Boost to Buck Mode Transition (V_{HV} = 350 V, 2-kΩ load; V_{LV} = 12.6 V, 10-Ω load)

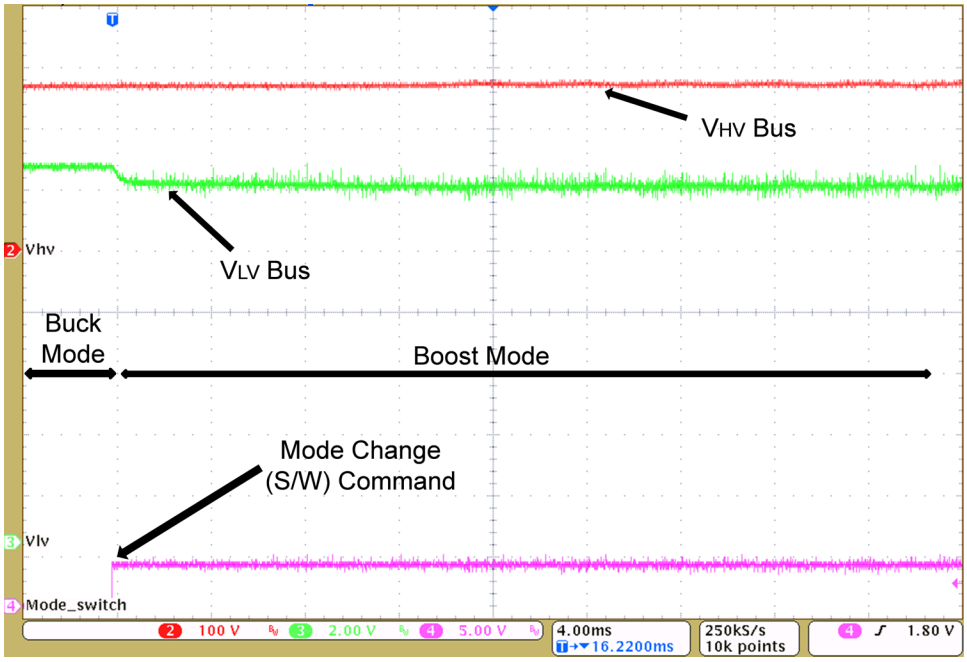


Figure 27. Buck to Boost Mode Transition (V_{HV} = 266 V, 667-Ω load; VLV = 12.6 V, 10-Ω load)

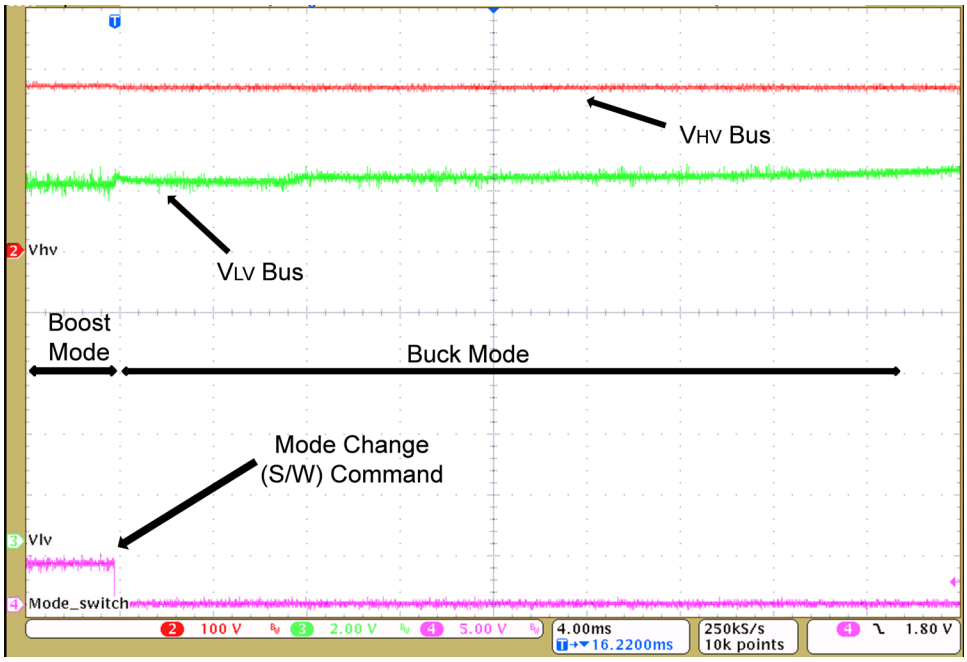


Figure 28. Boost to Buck Mode Transition (V_{HV} = 266 V, 667-Ω load; VLV = 12.6 V, 10-Ω load)

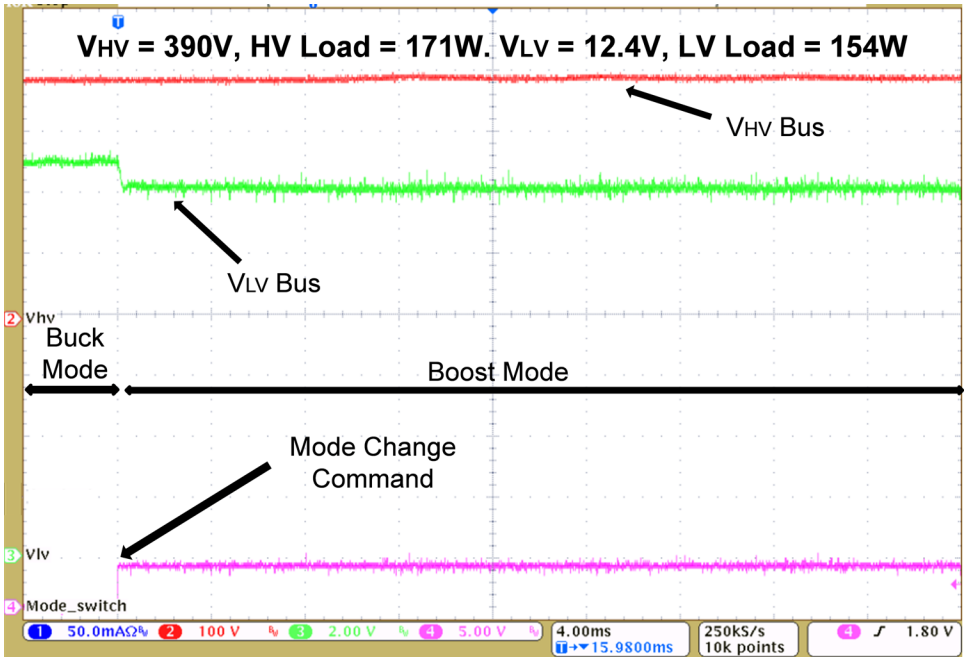


Figure 29. Buck to Boost Mode Transition (V_{HV} = 390 V, 171-W load; V_{LV} = 12.4 V, 154-W load)

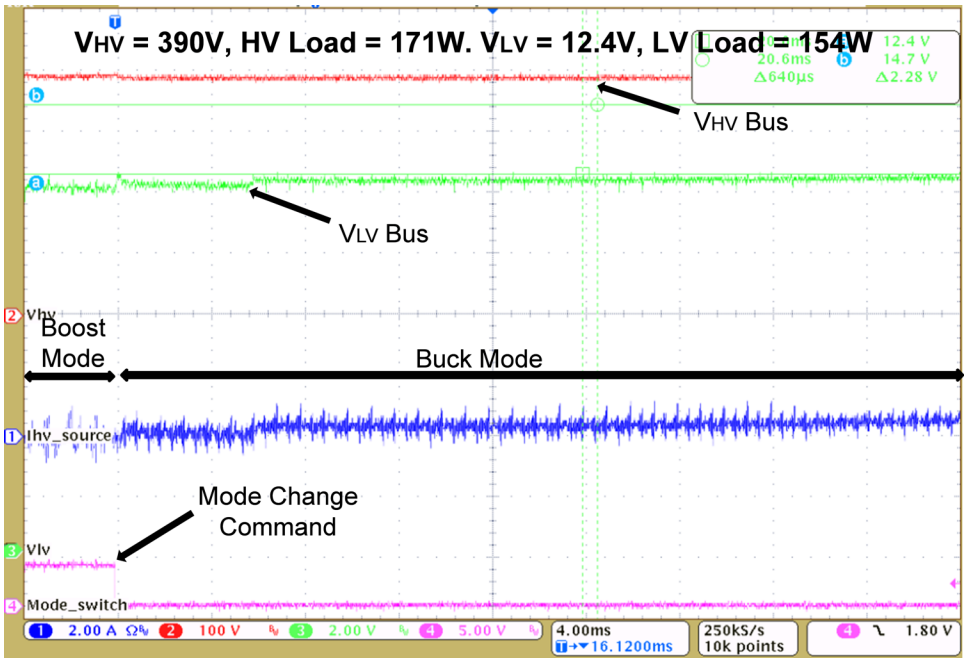


Figure 30. Boost to Buck Mode Transition (V_{HV} = 390 V, 171-W load; V_{LV} = 12.4 V, 154-W load)

Mode Transitions (Close-up)

The following images provide a close-up view of the voltage waveforms observed on the high-voltage winding of main power transformer T2 and PWM drive signals for diagonally opposite full-bridge switches Q2 and Q4 during mode transitions under various operating conditions.

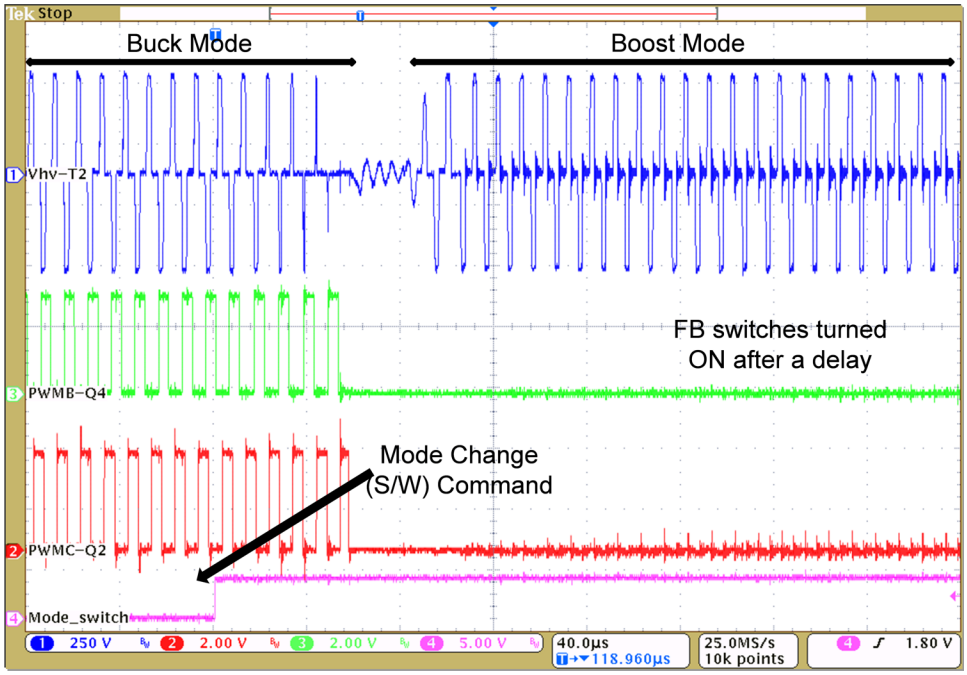


Figure 31. Buck to Boost Mode Transition (V_{HV} = 390 V, 19-W load; V_{LV} = 12.4 V, 307-W load)

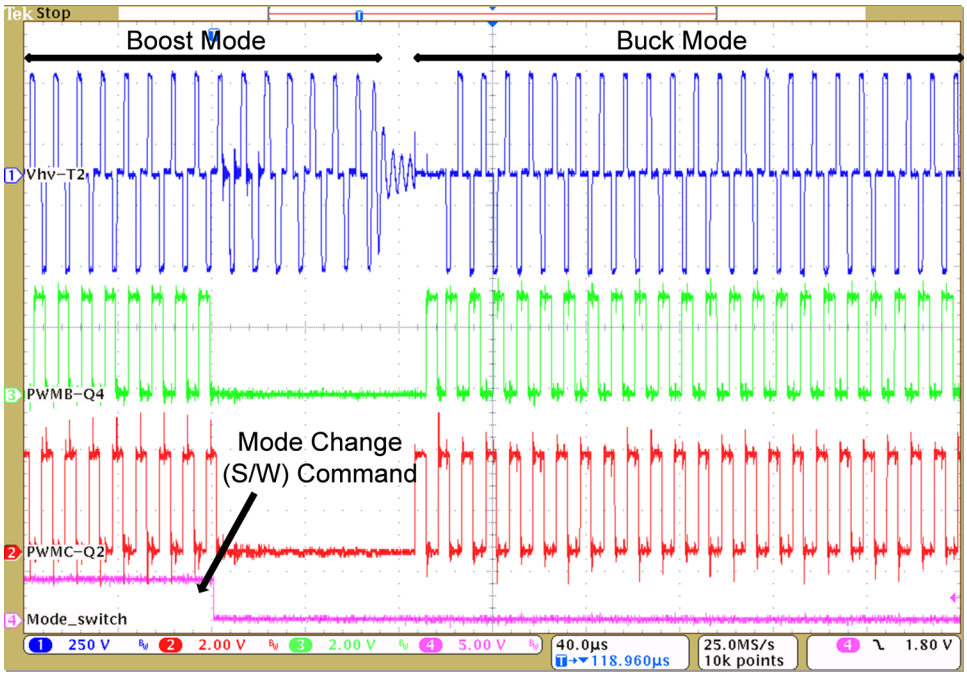


Figure 32. Boost to Buck Mode Transition (V_{HV} = 390 V, 19-W load; V_{LV} = 12.4 V, 307-W load)

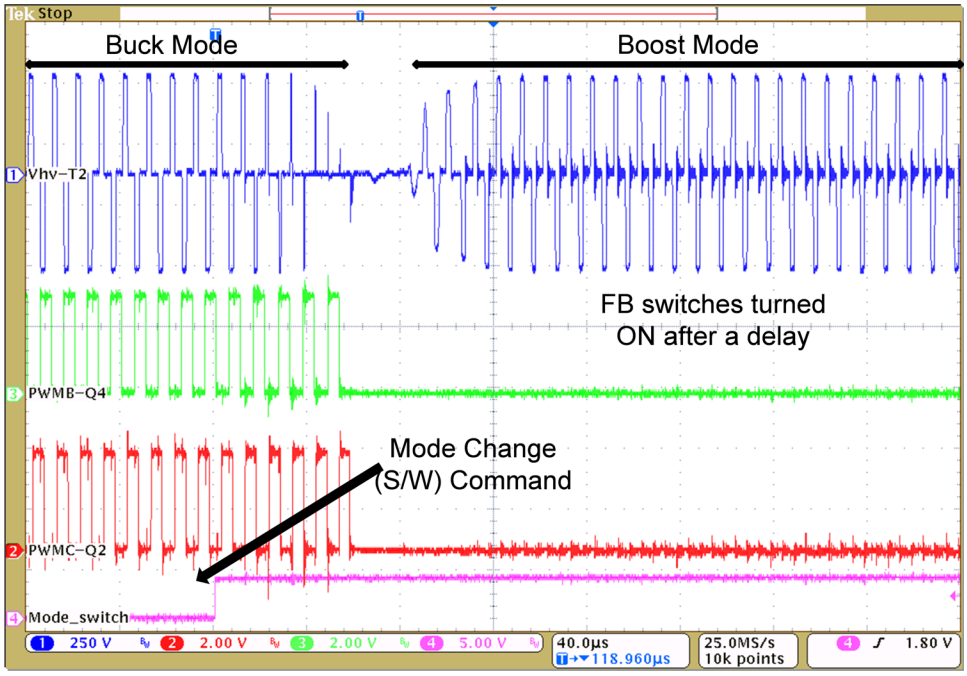


Figure 33. Buck to Boost Mode Transition (V_{HV} = 390 V, 171-W load; V_{LV} = 12.4 V, 154-W load)

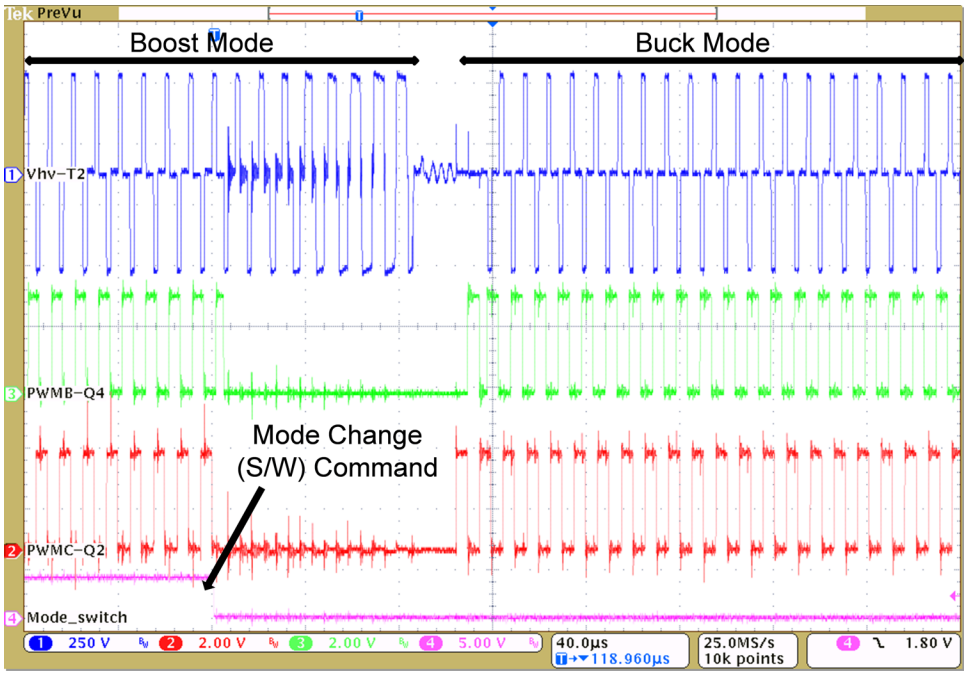


Figure 34. Boost to Buck Mode Transition (V_{HV} = 390 V, 171-W load; V_{LV} = 12.4 V, 154-W load)

4 Buck Mode (Phase-Shifted Full-Bridge) – Functional Description

4.1 Voltage Mode Control

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 35, the controller directly drives and controls the phase shift of PWM signals driving switches in one leg of the bridge with respect to the signals driving switches in the other leg. This phase shift dictates the amount of overlap between diagonally opposite switches, shown in Figure 37. If the overlap between diagonal switches is longer, the time the input voltage imposed across the transformer high-voltage winding is longer and the amount of energy transferred to the low-voltage side is greater. The controller regulates the output by controlling this energy transfer by directly controlling the phase shift between the PWM signals driving the two full-bridge legs. This phase shift is the controlled parameter. The VMC implementation must include a DC blocking capacitor in the transformer high-voltage winding to avoid possible transformer saturation from flux imbalance over time. Jumper JP1 is unpopulated by default. An appropriately rated jumper can be installed at JP1 if PCMC is implemented.

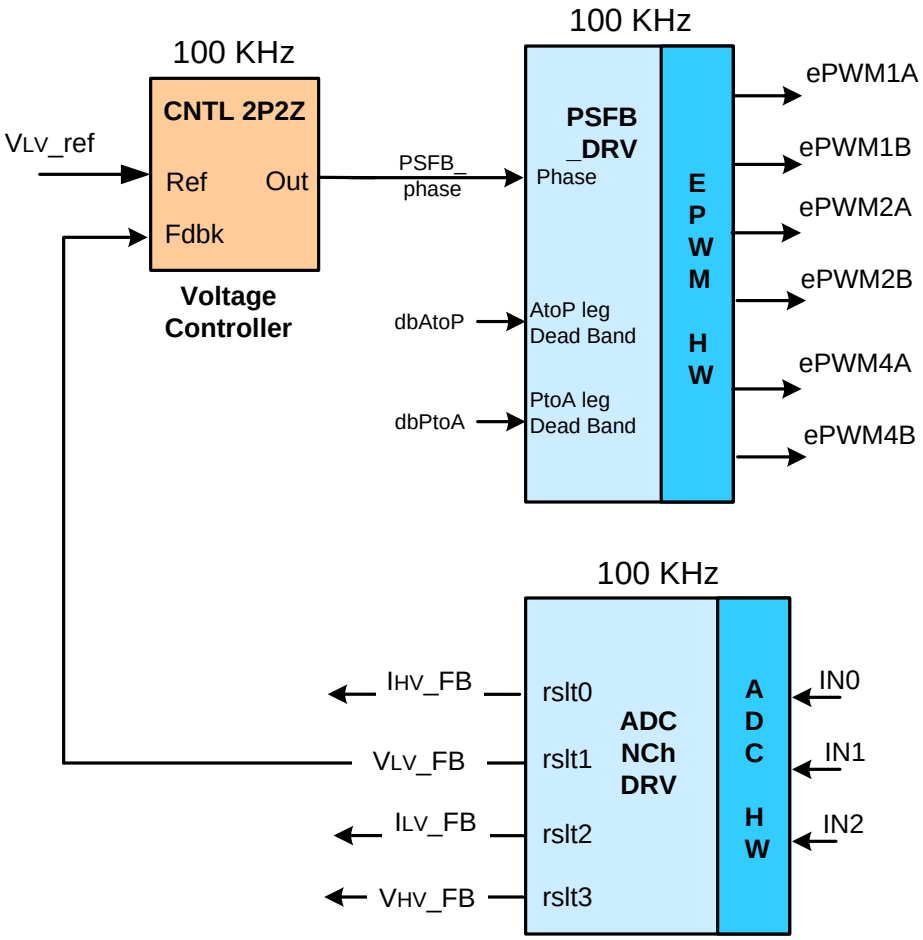


Figure 35. VMC Block Diagram

4.2 Average Current Mode Control of Output Inductor Current

The typical ACMC implementations in power supply applications comprise of an outer voltage control loop and an inner average current loop. The inner average current loop for a PSFB power stage controls the average current in the high-voltage winding. For battery charging applications, use a single average current loop to control the battery charging current in ACMC mode (constant current charging) and to switch to VMC mode for constant voltage charging when required. The ACMC implementation is similar to the VMC implementation in waveform generation and power stage control. As Figure 36 shows, the controller directly drives and controls the phase shift of PWM signals driving switches in one leg of the bridge with respect to the signals driving switches in the other leg. As in VMC mode, the controller regulates the output by controlling energy transfer by directly controlling the phase shift between PWM signals driving the two full-bridge legs. Figure 37 shows various waveforms during buck mode of operation.

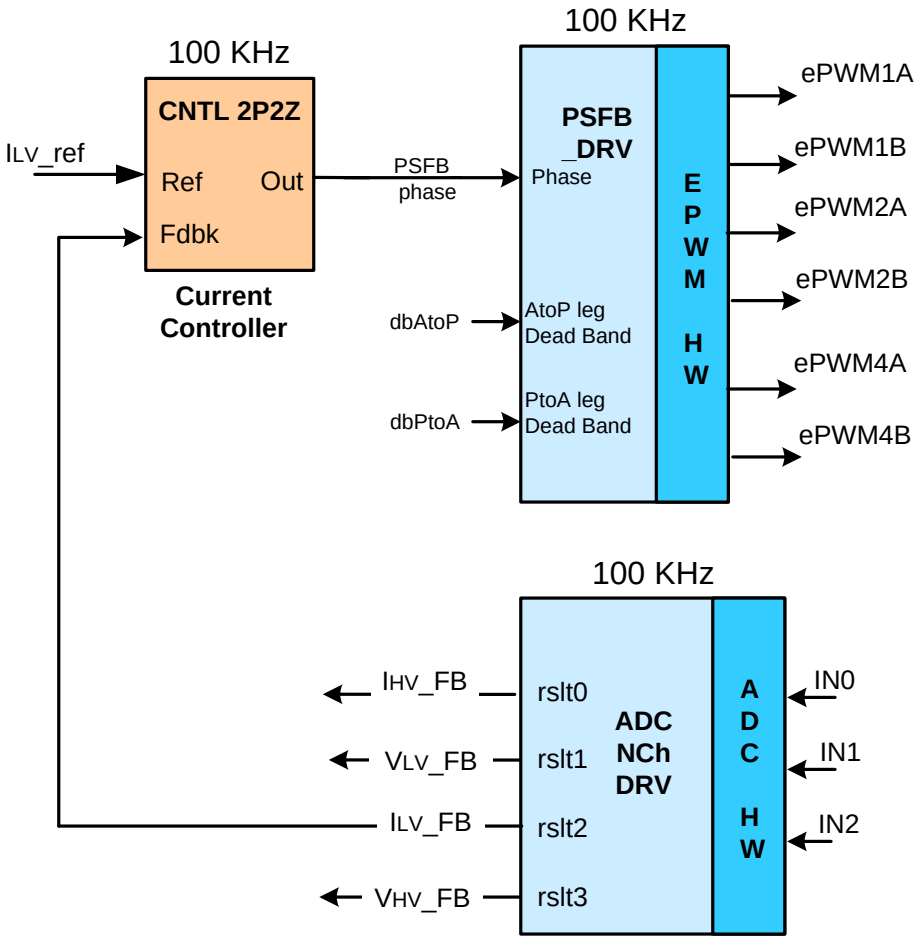


Figure 36. ACMC Block Diagram

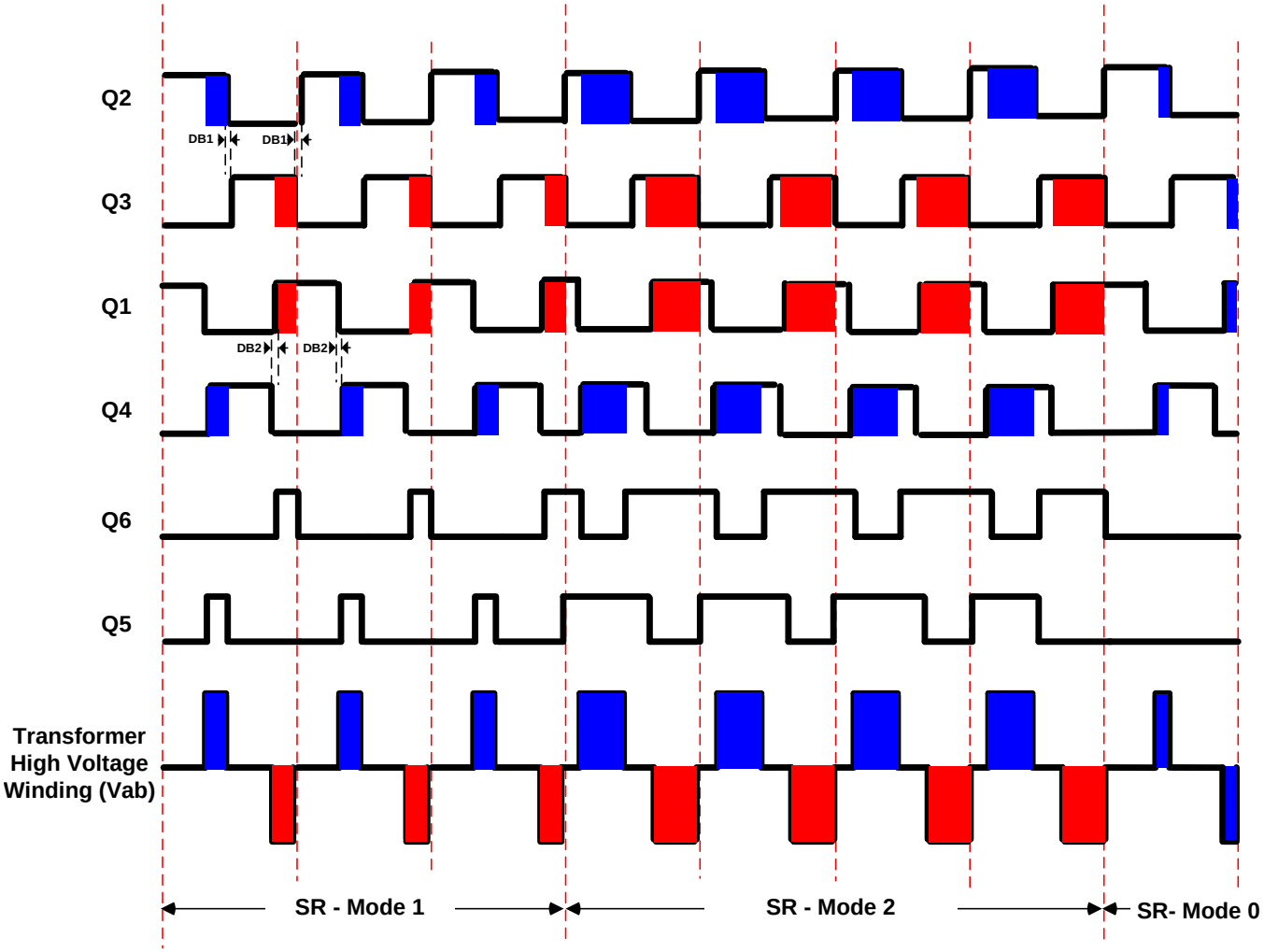


Figure 37. Buck Mode Waveforms

4.3 Zero-Voltage Switching or Low-Voltage Switching

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

Switching transitions for switches in the Q2–Q3 leg end the power transfer interval. This leg is the Active-to-Passive leg. When transitions occur for switches in this leg, current in the high-voltage winding is close to its maximum magnitude for that half of PWM switching cycle. The reflected load current aids the circulating energy in the high-voltage winding circuit during this time, which makes it possible for voltage across switches in this leg to approach zero volts. ZVS for switches can be achieved in this Q2–Q3 leg across a wide load range. As the load decreases, the amount of dead-time must increase to achieve or approach ZVS.

Switching transitions for switches in the Q1–Q4 leg start the power transfer interval. This leg is called the Passive-to-Active leg. During these switching transitions, high-voltage winding current decreases. The high-voltage winding current crosses the zero current value and changes direction. The current going zero and changing direction results in lower available energy for ZVS. For operating 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 to achieve LVS. As the load changes, the time at which the switch must be turned on to achieve LVS changes, requiring dead-time adjustment similar to the Q2–Q3 leg switches.

4.4 Synchronous Rectification

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

- **Mode 0:** This is the classical diode rectification 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 emulate ideal diode behavior. 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 PWM signals driving the corresponding pair of diagonal full-bridge switches overlap.
- **Mode 2:** Useful for all other load conditions. In this mode, synchronous rectifier MOSFETs are OFF only when PWM signals driving the opposite pair of corresponding diagonal full-bridge switches overlap.

Figure 37 shows 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.

5 Buck Mode – Software Overview

5.1 Software Control Flow

The Bi-dir_Buck project uses the C-background/ASM-ISR framework. The C-background/ASM-ISR framework uses C-code as the main supporting program for the application and performs 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 ; typically, this includes ADC reading, control calculations, and PWM and DAC updates. Figure 38 shows the general software flow for this project.

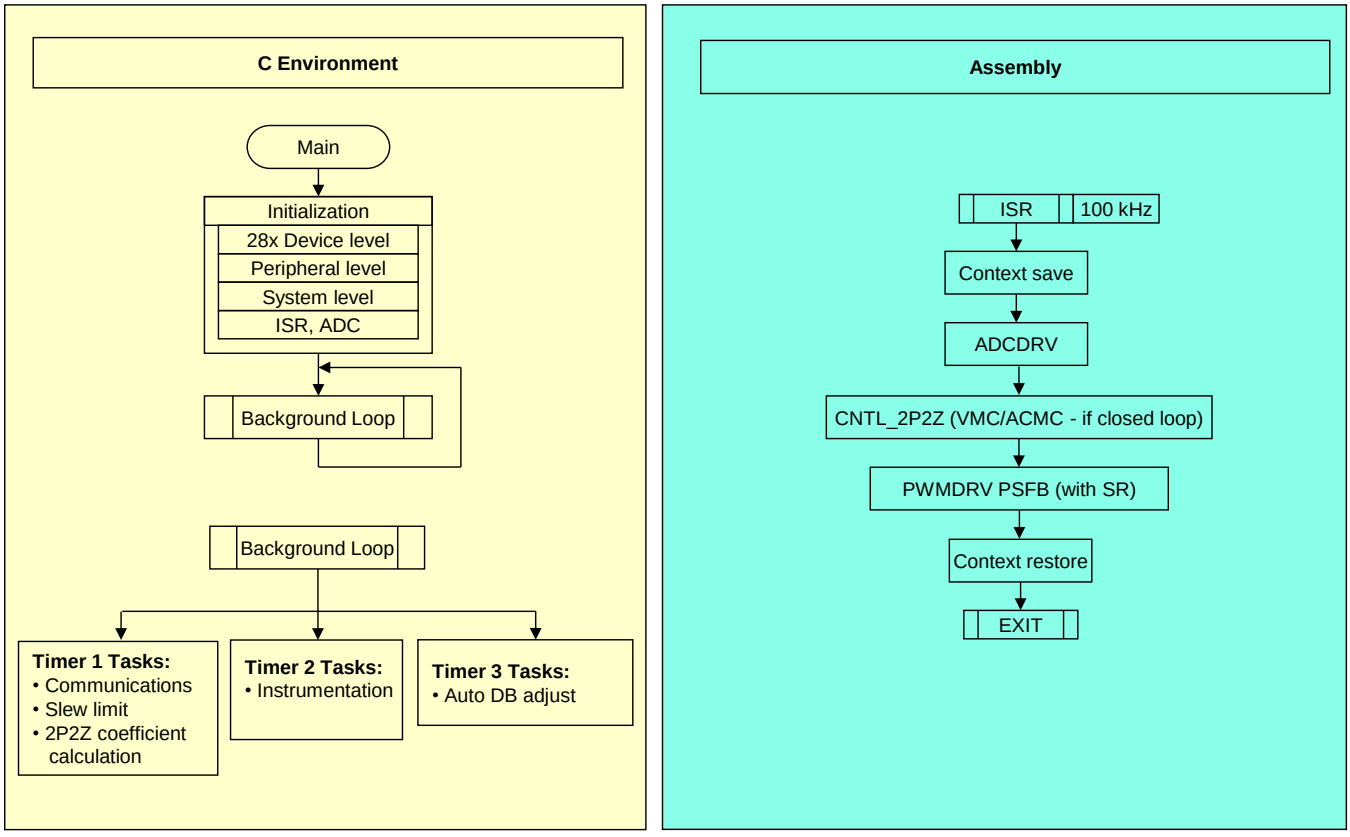


Figure 38. Buck Mode Software Flow

The following key framework C files are used in this project:

- Bi-dir-Main.c – This file initializes, runs, and manages the application.
- Bi-dir-DevInit_F2803x.c – This file manages a one-time initialization and configuration of the micro-controller, and includes functions such as setting up the clocks, PLL, GPIO, and so forth.

The ISR consists of the following file:

- Bi-dir-DPL-ISR.asm – This file contains all time-critical control type code. This file has an initialization section (one-time execute) and a runtime section, which executes at 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. Table 2 lists the following library modules in this project. The C and corresponding assembly module names are the following:

Table 2. Library Modules

C Configure Function	ASM Initialization Macro	ASM Runtime Macro
DAC_Cnf.c		
ADC_SOC_Cnf.c	ADCDRV_4CH_INIT m,n,p,q	ADCDRV_4CH m,n,p,q
PWM_PSFB_VMC_SR_Cnf.c	PWMDRV_PSFB_VMC_SR_INIT m,n,p	PWMDRV_PSFB_VMC_SR m,n,p
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

Figure 39 shows the control blocks.

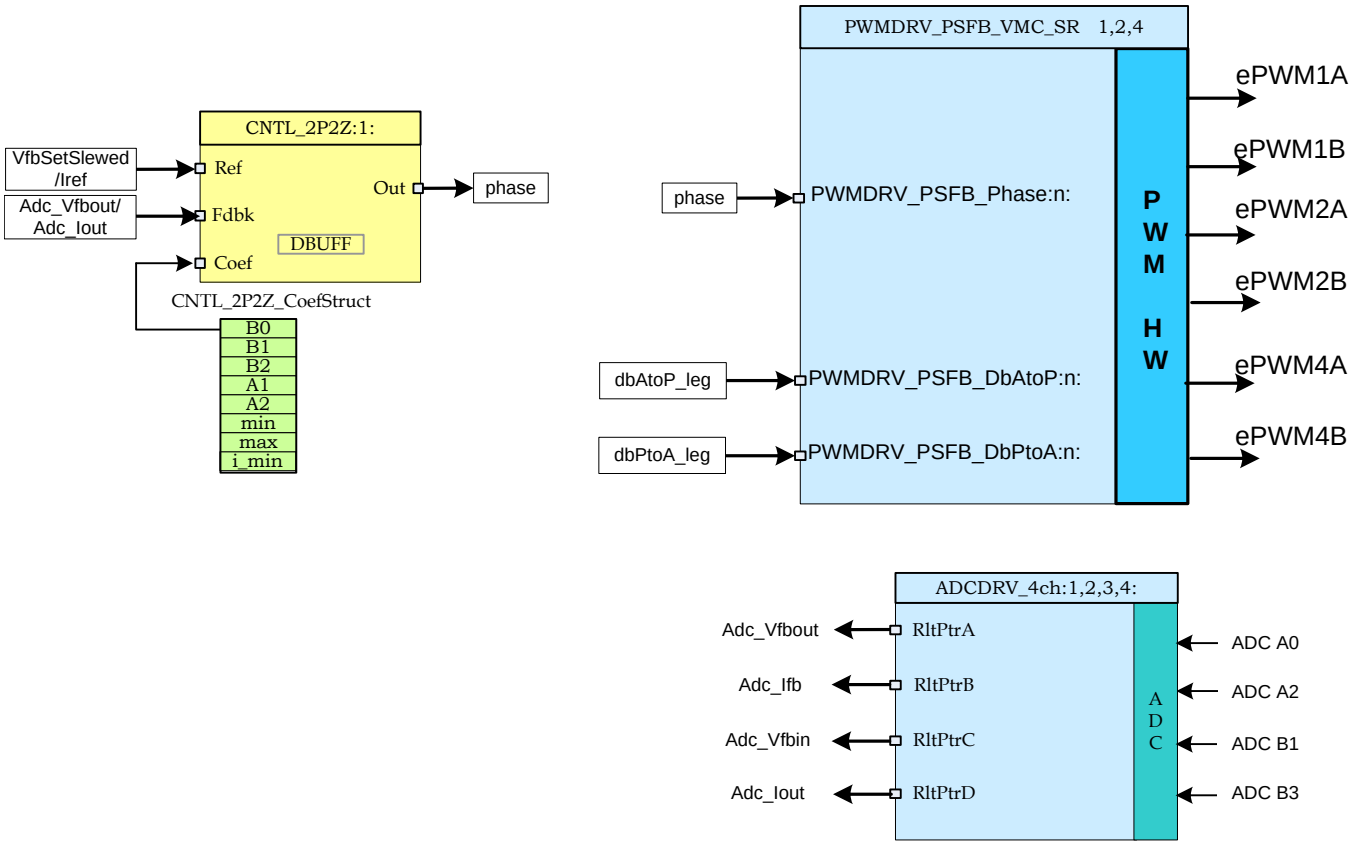


Figure 39. Software Blocks

In Figure 39, blocks in dark blue represent hardware modules on the C2000 MCU. 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 be a PI/PID, a 3-pole 3-zero or any other controller that can be implemented for this application. This type of modular library structure makes it convenient to visualize and understand the complete system software flow, as shown in Figure 40. This structure allows for easy use, addition, and removal of various functionalities. These benefits are demonstrated in this project by implementing an incremental build approach and is discussed in more detail in Section 5.2.

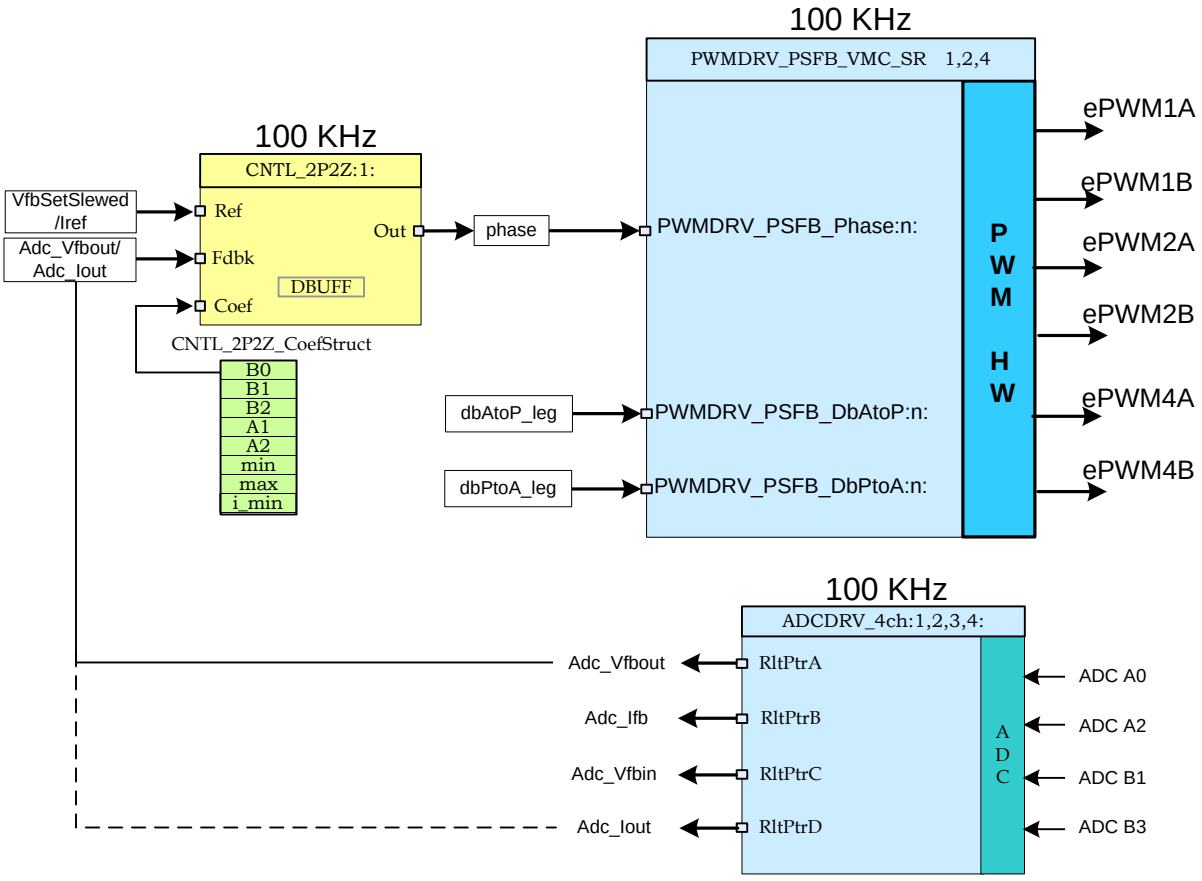


Figure 40. Control Flow

The system is controlled by one voltage/current feedback loop. Figure 40 also gives the rate at which control blocks are executed. For example, the voltage controller is executed at a rate of 100 kHz (the same as the PWM switching frequency). The following explains the control implemented in Figure 40.

The sensed output voltage or current (Adc_Vfbout/Adc_Iout) is compared with slewed version (VfbSetSlewed) of the voltage reference command (Vref) in VMC mode and with the current reference command (Iref) in ACMC mode. The 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 values of dbAtoP_leg and dbPtoA_leg provide dead band values for the Active-to-Passive and Passive-to-Active legs of the full bridge, respectively. These values are used to achieve ZVS/LVS across the load range.

5.2 Incremental Builds

This project is divided into two incremental builds. This approach makes it easier to learn and get familiar with the board and the software. This approach is also good for debugging and testing boards. Table 3 lists the build options. To select a particular build option, set the macro INCR_BUILD in the Bi-dir-Settings.h file, to the corresponding build selection as Table 3 shows. When the build option is selected, compile the complete project by selecting rebuild-all compiler option. Section 6 provides more details to run the build options.

Table 3. Incremental Build Options for Buck Mode

Option	Description
INCR_BUILD = 1	Open-loop PSFB drive with ADC feedback (check PWM drive circuit and sensing circuit)
INCR_BUILD = 2	Closed loop (full PSFB in VMC/ACMC mode)

6 Buck Mode – Procedure for Running the Incremental Builds

The main source files, ISR assembly file, and the project file for C framework to bring up the system are in the following directory (use the latest version of the software package).

- **..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Buck_Mode**

The 6-V to 13.5-V DC power supply and the 400-V load in Figure 7 are not required for buck mode operation.

WARNING

There are high voltages on the board. Only experienced power supply professionals in a lab environment must handle the board. To safely evaluate this board, use an appropriate isolated high-voltage DC source. Before DC power is applied to the board, a voltmeter and an appropriate resistive or electronic load must be attached to the output. Never handle the unit when the power is applied to it.

The following steps show how to build and run the example in the Buck_Mode software.

6.1 Build 1: Open-Loop Check With ADC Feedback

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™ (CCS). Because this system is running open loop, the ADC measured values are used only for instrumentation in this build. This section explores the steps required to build and run a project.

Overview

The software in Build 1 is 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 CCS. 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. Figure 41 shows the blocks in this build. ePWM1A and ePWM1B drive Q2 and Q3 full-bridge switches, respectively, while ePWM2A and ePWM2B drive Q1 and Q4 full-bridge switches, respectively. ePWM4A and ePWM4B drive Q6 and Q5 synchronous rectifier switches, respectively.

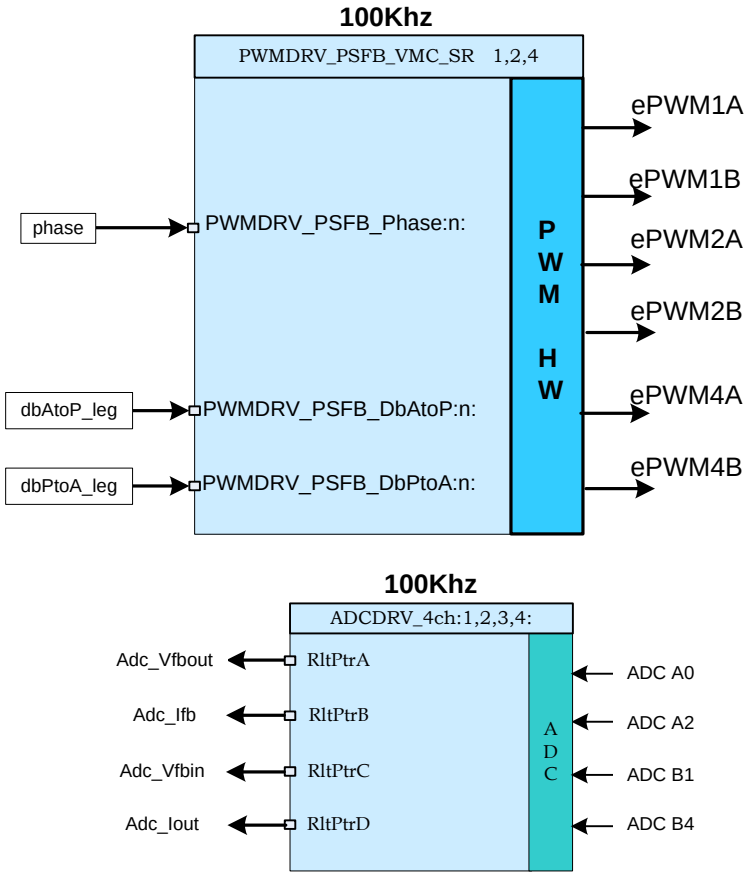


Figure 41. Build 1 Software Blocks

These PWM signals must be generated at a frequency of 100 kHz (that is, 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. That one count of the time base counter of ePWM1, ePWM2, or ePWM4 corresponds to 16.667 ns implies that a PWM period of 10 µs is 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. Outputs of ePWM1A and ePWM1B operate at 50% duty cycle and are complementary. Similarly, ePWM2A and ePWM2B operate at 50% duty cycles and are complementary. Phase for the ePWM2 time base may be changed dynamically with respect to ePWM1 phase. Figure 37 shows these PWM waveforms.

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, the overlap increases, which increases the amount of energy transferred to the secondary. The TBPHS2 value is derived from the input phase command.

Table 4 provides example TBPHS2 values derived for a TBPRD value of 599.

Table 4. Phase Values

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

Each pair of diagonal switches of the full-bridge overlaps once in one PWM period. The most overlap occurs when the phase shift is near 180 degrees. The ISR assembly is triggered on a ZRO (TBCNT1 = 0) event of ePWM1. The ISR is where the control driver macros are executed and the TBPHS2 and TBPHS4 registers are updated.

Regarding where the ADC input is sampled, the integrity of the ADC input signals is important because the analog and digital domains interface at the ADC. Turning a switch on or off in the power stage may result in some noise or disturbance on the signals that are sensed at this time. Even with all the filtering provided on these signals to avoid this noise from showing up at the ADC inputs, sample the ADC inputs at a time to avoid this disturbance.

Sensed output voltage must be sampled in the switching cycle at an appropriate point where the output voltage value is close to its average value. The ADC input signals are sampled at a time so as to get a sample as noise free as possible and to also sample the average output voltage. For the full-bridge, sample at the midpoint of the overlap between two diagonal switches (overlap refers to the time when both switches are on at the same time); that is, as far away from the MOSFET switching as possible. Sampling as far away from MOSFET switching as possible avoids any switching noise to be reflected on the ADC result. The flexibility of ADC and PWM modules on C2000 devices allow 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) triggers five ADC conversions.

Protection

The shutdown mechanism in this project implements overcurrent protection for the transformer high-voltage winding 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, ePWM2, and ePWM4 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, outputs of ePWM1A, ePWM1B, ePWM2A, ePWM2B, ePWM4A, and ePWM4B outputs are driven low immediately to protect the power stage. These outputs are held low until a device reset is executed. Undervoltage and overvoltage protection have not been implemented but have been implemented in the full bidirectional project (Bi_Directional_Full).

6.1.1 Procedure

Start CCS and Open a Project

To quickly execute this build, do the following:

1. Ensure that all jumpers on the board are correctly installed or removed as listed in Section 2.
2. Insert the F28035 controlCARD in the 100-pin DIMM connector.
3. Connect a 12-V DC bench power supply between TP10 and TP11 with correct polarity.
4. Connect an isolated 400-V programmable DC power source to the 400-V input connector and a 12-V load to the 12-V connector (ensure this load does not exceed the board ratings).
5. Connect a USB-B to USB-A cable between the PC and the board.

NOTE: Do not turn on any power supplies at this time.

6. If this is the first time the board is being tested with JTAG connection, run the program_ftdi.bat file in the xds100v2-FT_Prog_v2.2.zip file to program the FTDI chip on the board.
7. Open Code Composer Studio (CCSv5 or later).

8. Maximize Code Composer Studio to fill your screen.
9. Close the welcome screen if it opens.

NOTE: A project contains all the files and build options needed to develop an executable output file (.out), which can be run on the MCU hardware.

10. On the menu bar, click: Project → Import Existing CCS/CCE Eclipse Project.
11. Select the **..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Buck_Mode** directory.
12. Under the Projects tab, ensure Bi-dir_Buck is checked.
13. Click Finish.

NOTE: This project uses all the necessary tools (compiler, assembler, linker) to build the project.

14. In the Project window on the left, click + to the left of Project. See Figure 42.

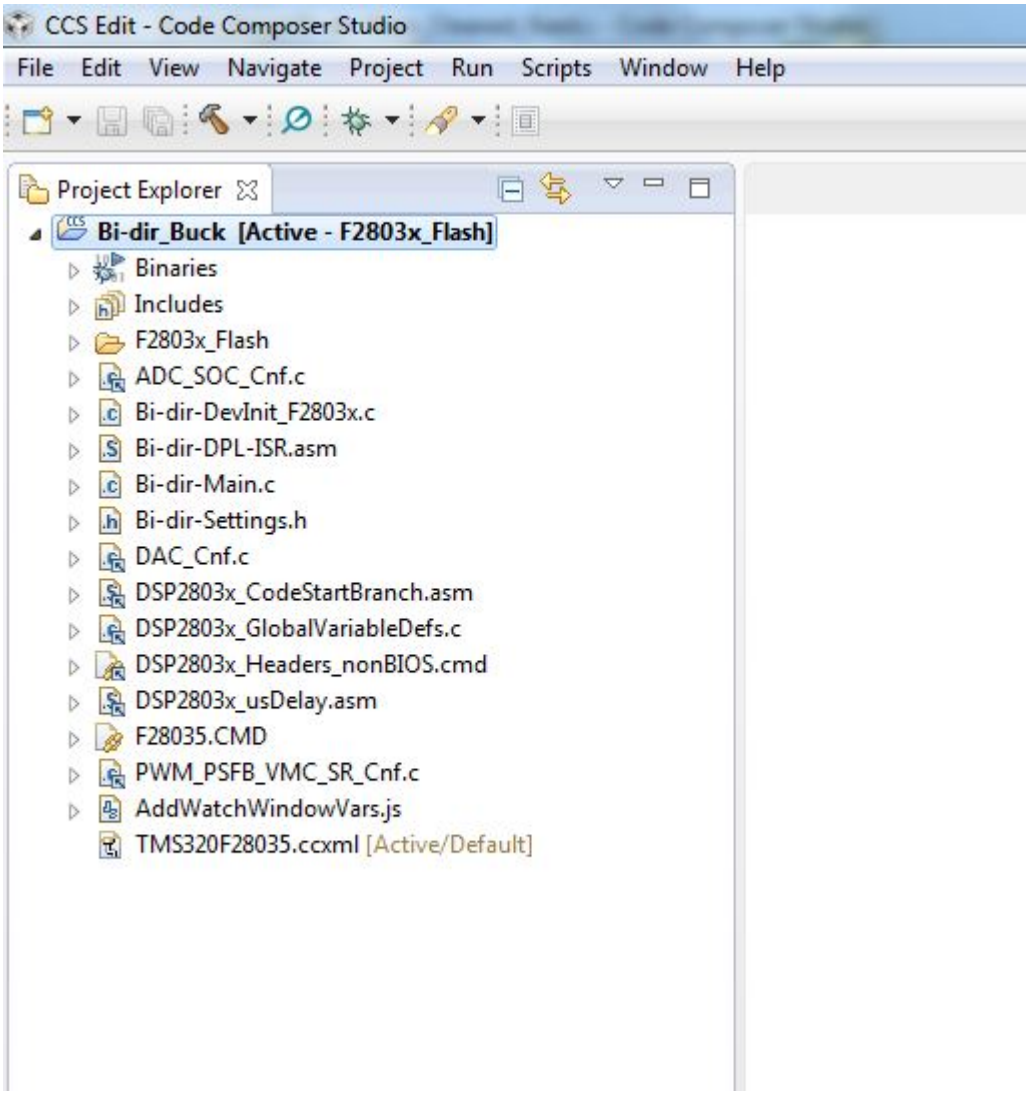


Figure 42. Buck Mode CCS Project Window

15. Locate initialization code for Build 2 in the main file.
16. Inspect the code. (This code is where all the control blocks are configured, initialized, and connected in the control flow.)

Device Initialization, Main, and ISR Files

NOTE: Do not make any changes to the source files.

1. Double-click Bi-dir-DevInit_F2803x.c.
2. Ensure that the system clock, peripheral clock prescale, and peripheral clock enables have been set up.
3. Ensure that the shared GPIO pins have been configured.
4. Double-click Bi-dir-Main.c.
5. View the call to the DeviceInit() function, the code for different incremental build options, the ISR initialization, and the background for(;;) loop.
6. Locate code for Build 1 in the main file.
7. Inspect the code.

NOTE: The code specific to Build 1 is where the PWMDRV_PSFB_VMC_SR block is connected and initialized.

8. Double-click Bi-dir-DPL-ISR.asm.

NOTE: The _DPL_Init and _DPL_ISR sections are where the PWM and ADC driver macro instantiation occurs for initialization and run time, respectively. You can close the inspected files.

Build and Load the Project

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

NOTE: If another option was built previously, right-click on the project name and click Clean Project.

2. Click Project → Build All.
3. Turn on the 12-V DC bench power supply.
4. Click the Debug button.

NOTE: The Build 1 code should compile and load.

To create an F28035 target configuration or copy the configuration from the F28035.ccxml file, do the following:

1. Right-click F28035.ccxml in the Target Configurations window.
2. Right-click on the name of the device in the Debug window to connect the device.
3. Click Run → Load → Load Program.
4. Navigate to Buck_Mode\F2803x_Flash\Bi-dir_Buck.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

5. Stop the program at the start of main().

Debug Environment Windows

Watch local and global variables while debugging code. You can use the memory views and watch views in CCS. CCS can create time and frequency domain plots. You can view waveforms using graph windows.

1. Click View → Scripting console on the menu bar.
2. Use the scripting console Open File ( )command to open the AddWatchWindowVars.js file from the project directory.

This action populates the Expressions window entries. See Figure 43 to see how the Expressions window should look.

NOTE: Some of the variables have not been initialized at this point in the main code and may contain some garbage values

If set, OC_FaultFlgindicates an overcurrent condition that shuts down the PWM outputs. PWM outputs are held in this state until a device reset. The Ipri_trip variable sets the internal 10-bit DAC reference level for the on-chip comparator 1. This is a Q15 number.

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vfbin	int	99.2813 (Q-Value(5))	0x0000041D@Data
(x)= Gui_Vfbout	int	0.0712891 (Q-Value(10))	0x0000041C@Data
(x)= Gui_VfbSet	int	2.0 (Q-Value(10))	0x0000041B@Data
(x)= Gui_Ifb	int	0.0795898 (Q-Value(12))	0x0000041E@Data
(x)= Gui_Iout	int	1.17188 (Q-Value(8))	0x0000041A@Data
(x)= Gui_IoutSet	int	0.0 (Q-Value(8))	0x00000421@Data
(x)= phase	long	0.0 (Q-Value(24))	0x00000426@Data
(x)= OC_FaultFlg	int	0	0x00000422@Data
(x)= Ipri_trip	int	0.899994 (Q-Value(15))	0x00000410@Data
(x)= dbAtoP_leg	int	20	0x00000411@Data
(x)= dbPtoA_leg	int	20	0x0000040B@Data
(x)= auto_DB	int	1	0x0000040A@Data
(x)= SR_mode	int	2	0x0000040C@Data
Add new expression			

Figure 43. Buck Mode Expressions Window: Build 1

Use Real-Time Emulation

Real-time emulation is a special emulation feature that allows the updating of the windows within CCS at a rate up to 10 Hz while the MCU is running. This emulation lets graphs and watch views update and lets you change values in watch or memory windows so that those changes affect the behavior of the MCU. This emulation is useful when tuning control law parameters on-the-fly.

To enable real-time mode, do the following:

1. Hover the mouse on the buttons on the horizontal toolbar.
2. Click 

.

NOTE: If a message box appeasr, select YES to enable debug events. This sets bit 1 (DGBM bit) of status register 1 (ST1) to 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 to update the debugger windows.

When too many windows or variables are open or updating, continuous refresh can cause the refresh frequency to slow down because bandwidth over the emulation link is limited. To change the refresh rate for the Expressions window, do the following:

1. Right-click in the Expressions window.
2. Select Continuous Refresh Interval….
3. Change the continuous refresh interval value. (A rate of 1000 ms is usually enough for these exercises.)
4. Click  for the watch view.

Run the Code

1. Click Run on the toolbar.
2. Set the variable phase to 0.015625 (Q24) in the watch view.

NOTE: This variable denotes the phase shift command to the PWMDRV_PSFB_VMC_SR module. Do not use a value of less than 0.005 for phase.

3. Apply an appropriate resistive load that draws around 3-A to 6-A current at 12-V output to the PSFB system at the DC output.

NOTE: TI recommends using an isolated DC source to supply 400-V DC input to the board.

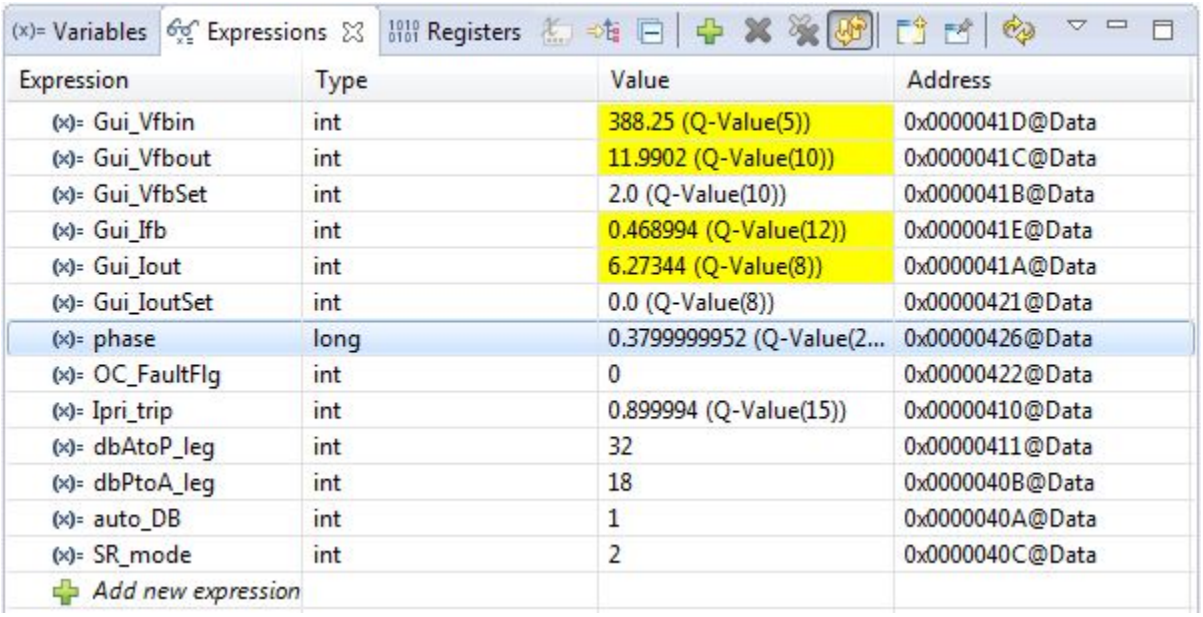
4. Power the input at J1, J2 with 390-V DC.
5. Increase the phase command by setting phase to a higher value (for example, 0.1) in the watch view.
6. Observe the output voltage as it increases.

NOTE: Do not let the voltage exceed the capabilities of the board.

When operating with a certain phase value, reducing the load suddenly can cause the output voltage to suddenly increase. Do not make any sudden changes of load or large increases in phase command when operating in Build 1.

7. Observe the different ADC results in the watch view for different phase values.

Figure 44 is a watch view that corresponds with the operation of the system with a phase command of 0.38 (Q24) with an input voltage of approximately 390 V and a load of approximately 6 A (2 Ω) at 12-V output.



Expression	Type	Value	Address
Gui_Vfbin	int	388.25 (Q-Value(5))	0x0000041D@Data
Gui_Vfbout	int	11.9902 (Q-Value(10))	0x0000041C@Data
Gui_VfbSet	int	2.0 (Q-Value(10))	0x0000041B@Data
Gui_Ifb	int	0.468994 (Q-Value(12))	0x0000041E@Data
Gui_Iout	int	6.27344 (Q-Value(8))	0x0000041A@Data
Gui_IoutSet	int	0.0 (Q-Value(8))	0x00000421@Data
phase	long	0.3799999952 (Q-Value(2...	0x00000426@Data
OC_FaultFlg	int	0	0x00000422@Data
Ipri_trip	int	0.899994 (Q-Value(15))	0x00000410@Data
dbAtoP_leg	int	32	0x00000411@Data
dbPtoA_leg	int	18	0x0000040B@Data
auto_DB	int	1	0x0000040A@Data
SR_mode	int	2	0x0000040C@Data
Add new expression			

Figure 44. Buck Mode Expressions Window: Build 1 (Runtime)

8. Change the SR_mode variable to 0, 1, or 2 from the watch view to change the mode of operation for the synchronous rectifiers. (By default, the synchronous rectifiers are operated in mode 2.)
9. Observe the change in amount of input current being drawn and change in output voltage with different SR modes.
10. Probe the PWM waveforms driving the synchronous rectifier switches.

NOTE: Do not change between different SR modes when operating at very low loads or when the output voltage is very low (less than 6 V). In these cases, use the default SR mode 2.

11. Try different phase values.
12. Observe the corresponding ADC results.

NOTE: Increase phase in small steps. Always observe the output voltage carefully. Do not let the voltage texceed the capabilities of the board.

You can use an oscilliscope to probe waveforms, like the PWM gate drive signals, input voltage, current, and output voltage.

Take appropriate safety precautions and consider appropriate grounding requirements while probing these high voltages and high currents for this isolated DC-DC converter.

To fully halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V DC input and wait a few seconds.
2. Click Halt on the toolbar to halt the processor.
3. Click  to take the MCU out of real-time mode.
4. Reset the MCU.

6.2 Build 2: Closed-Voltage Loop (Full PSFB)

Objective

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

Overview

Figure 45 shows the software blocks in this build. The PWM and ADC driver blocks are used in the same way as in Build 1. A two-pole, two-zero controller is used for the voltage/current loop. Depending on the control loop requirements of the application, some other controller block like a PI, a 3-pole 3-zero, and so forth, may also be used. As seen in Figure 45, 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 does not require a CNF function call

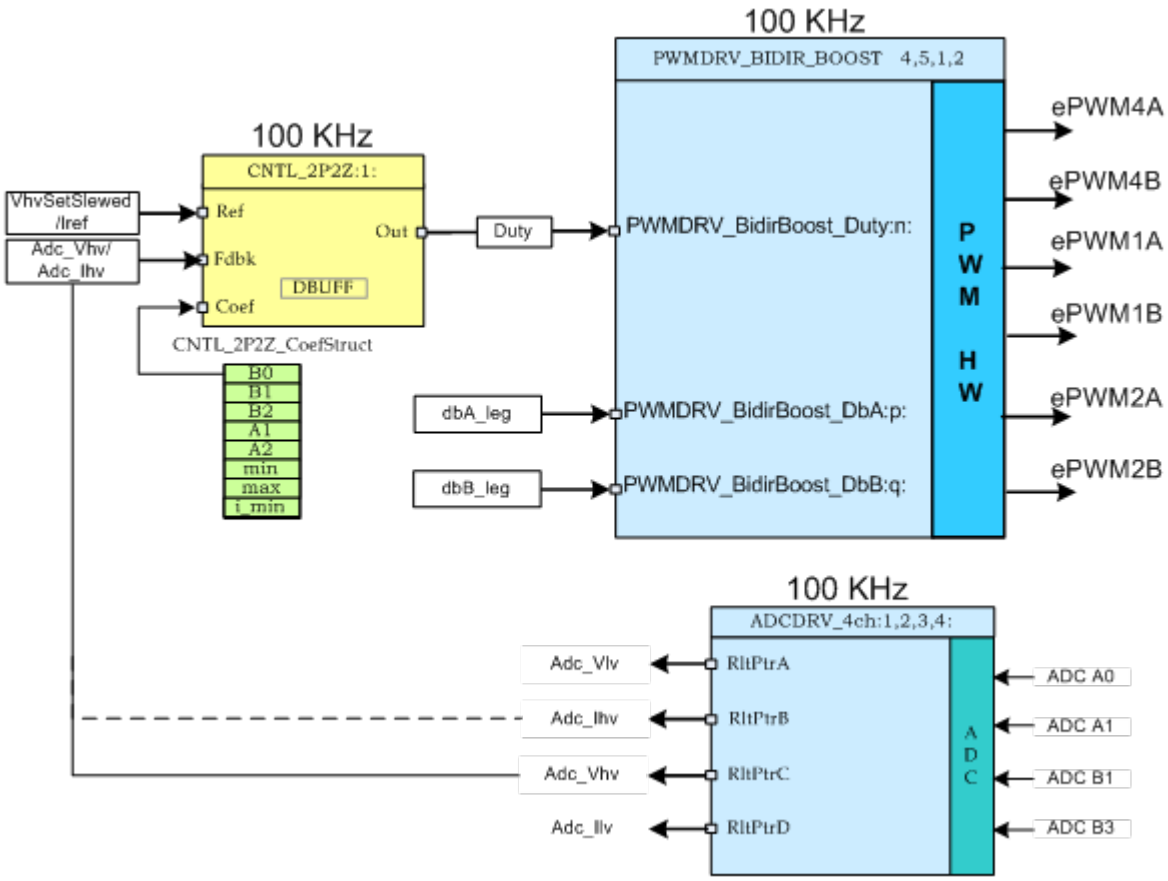


Figure 45. Build 2 Software Blocks

The five coefficients to be modified are stored as elements of the structure CNTL_2P2Z_CoefStruct1 whose other elements clamp 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 and/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 avoid the need for for complex mathematics or analysis tools, TI reduced the coefficient selection problem from five degrees of freedom to three by mapping the more intuitive coefficient gains of P, I, and D to B0, B1, B2, A1, and A2. This reduction independently and gradually adjusts P, I, and D. The following shows these mapping equations.

The compensator block (CNTL_2P2Z) has two poles and two zeros and is based on the general IIR filter structure. The block has a reference input and a feedback input. The feedback is the sensed output voltage (Adc_Vfbout) when VMC mode is selected or the sensed output current (Adc_Iout) when ACMC mode is selected. This selection can be done from the Bi-dir-Settings.h file. The reference input to the controller is a slewed version (VfbSetSlewed) of the output voltage reference command (Vref) or it is the reference output inductor current command (Iref) based on the mode selection. Equation 1 gives the transfer function:

$$\frac{U(z)}{E(z)} = \frac{b_0 + B_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (1)$$

The recursive form of the PID controller is given by Equation 2:

$$u(k) = u(k-1) + b_0 e(k) + b_1 e(k-1) + b_2 e(k-2) \quad (2)$$

Where Equation 3, Equation 4, and Equation 5:

$$b_0 = K_p' + K_i' + K_d' \quad (3)$$

$$b_1 = -K_p' + K_i' - 2K_d' \quad (4)$$

$$b_2 = K_d' \quad (5)$$

And the z-domain transfer function form of this is show in Equation 6:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-1}} = \frac{b_0 z^{-2} + b_1 z + b_2}{z^2 - z} \quad (6)$$

Comparing this with the general form, PID is a special case of CNTL_2P2Z control where Equation 7:

$$a_1 = -1 \text{ and } a_2 = 0 \quad (7)$$

These P, I, and D coefficients are: Pgain, Igain, and Dgain. These P, I, and D coefficients are used in the Q26 format. To simplify tuning from CCS watch views, these three coefficients are further scaled to values from 0 to 999 (Pgain_Gui, Igain_Gui, and Dgain_Gui). The loop parameters can also be changed directly based on tuned values obtained from external mathematical (MATLAB, Mathcad, and so forth) tools. Loop coefficients can be directly changed using variables b2_Gui, b1_Gui, b0_Gui, a2_Gui, and a1_Gui in I5Q10 form, which are then converted to the five Q26 coefficients for the 2P2Z controller.

This project allows easy evaluation of both methods of loop tuning by providing the ability to easily switch between coefficients during execution. This switching 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 CCS. PID-based loop tuning (pid2p2z_GUI = 0) from the GUI environment has been used for this project. Ensure that reliable coefficient values are programmed in b2_Gui, b1_Gui, b0_Gui, a2_Gui, and a1_Gui variables before changing the pid2p2z_GUI to 1.

NOTE: This project does not include valid b2_Gui, b1_Gui, b0_Gui, a2_Gui, and a1_Gui parameter values. TI recommends using the default P-, I-, D-based loop by keeping pid2p2z_GUI = 0.

6.2.1 Procedure

Build and Load the Project

To quickly execute this build using the preconfigured work environment, do the following:

1. Ensure that all jumpers on the board are correctly installed or removed as listed in Section 2.
2. Insert the F28035 controlCARD in the 100-pin DIMM connector.
3. Connect a 12-V DC bench power supply between TP10 and TP11 with correct polarity.
4. Connect an isolated 400-V programmable DC power source to the 400-V input connector and a 12-V load to the 12-V connector (ensure this load does not exceed the board ratings).
5. Connect a USB-B to USB-A cable between the PC and the board.

NOTE: Do not turn on any of the power supplies.

6. If this is the first time the board is being tested with JTAG connection, run the program_ftdi.bat file in the xds100v2-FT_Prog_v2.2.zip file to program the FTDI chip on the board.
7. Open Code Composer Studio (CCSv5 or later).
8. Maximize CCS to fill your screen.
9. Close the welcome screen, if it opens.

NOTE: A project contains all the files and build options needed to develop an executable output file (.out), which can be run on the MCU hardware.

10. On the menu bar, click Project → Import Existing CCS/CCE Eclipse Project.
11. Select the **..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Buck_Mode** directory.
12. Under the Projects tab, ensure Bi-dir_Buck is checked.
13. Click Finish.

NOTE: This project uses all the necessary tools (compiler, assembler, and linker) to build the project.

14. In the Project window on the left, click + to the left of Project.
15. Locate the initialization code for Build 2 in the main file.
16. Inspect the code. (This code is where all the control blocks are configured, initialized, and connected in the control flow.)
17. Select the Incremental build option as 2 in the Bi-dir-Settings.h file.

NOTE: If another build option was built previously, do the following:
1. Right-click the project name.
2. Click Clean Project.
3. Click Project → Build All.
4. Watch the tools run in the build window.

18. Turn on the 12-V DC bench power supply.
19. Click the Debug button.

NOTE: The Build 1 code should compile and load

To create an F28035 target configuration or copy the configuration from the F28035.ccxml file, do the following:

1. Right-click F28035.ccxml in the Target Configurations window.
2. Right-click on the name of the device in the Debug window to connect the device.
3. Click Run → Load → Load Program.
4. Navigate to Buck_Mode\F2803x_Flash\Bi-dir_Buck.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

5. Stop the program at the start of main().

Debug Environment Windows

Watch local and global variables while debugging code. You can use the memory views and watch views in CCS. CCS can create time and frequency domain plots. You can view waveforms using graph windows.

1. Click View → Scripting console on the menu bar.
2. Use the scripting console Open File ( )command to open the AddWatchWindowVars.js file from the project directory.

   This action populates the Expressions window entries. See Figure 46 to see how the Expressions window should look.

NOTE: Some of the variables have not been initialized at this point in the main code and may contain some garbage values

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vfbin	int	99.2813 (Q-Value(5))	0x0000041D@Data
(x)= Gui_Vfbout	int	0.0712891 (Q-Value(10))	0x0000041C@Data
(x)= Gui_VfbSet	int	2.0 (Q-Value(10))	0x0000041B@Data
(x)= Gui_Ifb	int	0.0795898 (Q-Value(12))	0x0000041E@Data
(x)= Gui_Iout	int	1.17188 (Q-Value(8))	0x0000041A@Data
(x)= Gui_IoutSet	int	0.0 (Q-Value(8))	0x00000421@Data
(x)= phase	long	0.0 (Q-Value(24))	0x00000426@Data
(x)= OC_FaultFlg	int	0	0x00000422@Data
(x)= Ipri_trip	int	0.899994 (Q-Value(15))	0x00000410@Data
(x)= dbAtoP_leg	int	20	0x00000411@Data
(x)= dbPtoA_leg	int	20	0x0000040B@Data
(x)= auto_DB	int	1	0x0000040A@Data
(x)= SR_mode	int	2	0x0000040C@Data
Add new expression			

Figure 46. Buck Mode Expressions Window: Build 2

NOTE: See the additional variables in the watch view.

3. Use Gui_VfbSet to set the output voltage command.

Run the Code

1. Enable real-time modes.
2. Enable continuous refresh for watch views and changing the continuous refresh interval.
3. Click Run on the toolbar to run the code.
4. Apply an appropriate resistive load that draws approximately 3-A to 6-A current at 12-V output to the PSFB system at the DC output.

NOTE: TI recommends using an isolated DC source to supply 400-V DC input to the board.

5. Power the 400-V input with 390-V DC.

6. Change the Gui_VfbSet command to 12 V (in Q10).

NOTE: The output voltage starts ramping up to 12 V. This output voltage ramp-up rate can be changed by changing the variable VfbSlewRate. At this point, you can change the Gui_VfbSet command but ensure these changes meet the requirements of the board.

Figure 47 is the watch view that corresponds to the operation of the system with 12 V at the output with an input voltage approximately 390 V and a load of approximately 12 A (1 Ω).

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vfbin	int	387.344 (Q-Value(5))	0x0000041B@Data
(x)= Gui_Vfbout	int	11.9736 (Q-Value(10))	0x0000041D@Data
(x)= Gui_VfbSet	int	12.0 (Q-Value(10))	0x0000041C@Data
(x)= Gui_Ifb	int	0.594727 (Q-Value(12))	0x0000041A@Data
(x)= Gui_Iout	int	11.293 (Q-Value(8))	0x00000422@Data
(x)= Gui_IoutSet	int	0.0 (Q-Value(8))	0x00000421@Data
(x)= phase	long	0.4104562998 (Q-Value(2...	0x00000428@Data
(x)= OC_FaultFlg	int	0	0x00000420@Data
(x)= Ipri_trip	int	0.899994 (Q-Value(15))	0x00000410@Data
(x)= dbAtoP_leg	int	32	0x00000411@Data
(x)= dbPtoA_leg	int	18	0x0000040B@Data
(x)= auto_DB	int	1	0x0000040A@Data
(x)= SR_mode	int	2	0x0000040C@Data
Add new expression			

Figure 47. Buck Mode Expressions Window: Build 2 VMC Mode (Runtime)

7. Change the SR_mode variable to 0, 1, or 2 from the watch view to change the mode of operation for the synchronous rectifiers. (By default, the synchronous rectifiers are operated in mode 2.)
8. Observe the effect of the change in input current being drawn and change in output voltage with different SR modes. (There should be no effect on the output voltage or input voltage.)
9. Probe the PWM waveforms driving the synchronous rectifier switches.

NOTE: Do not change between different SR modes when operating at very low loads or when the output voltage is very low (less than 6 V). In these cases, use the default SR mode 2.

Increase phase in small steps. Always observe the output voltage carefully. Do not let the voltage exceed the capabilities of the board. Ensure that these changes are within the requirements of the board.

You can use an oscilloscope to probe waveforms, like the PWM gate drive signals, input voltage, current, and output voltage.

Take appropriate safety precautions and consider appropriate grounding requirements while probing these high voltages and high currents for this isolated DC-DC converter.

To fully halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V DC input and wait a few seconds.
2. Click Halt on the toolbar to halt the processor.
3. Click 

to take the MCU out of real-time mode.
4. Reset the MCU.

To let the converter be controlled in output inductor control mode (ACMC mode), do the following:

1. Set VMC0_ACMC1 to 1 in the Bi-dir-Settings.h file.

NOTE: If another build option was built previously, do the following:
1. Right-click the project name.
2. Click Clean Project.
3. Click Project → Build All.
4. Watch the tools run in the build window.

2. Right-click F28035.ccxml in the Target Configurations window.
3. Right-click on the name of the device in the Debug window to connect the device.
4. Click Run → Load → Load Program.
5. Navigate to Buck_Mode\F2803x_Flash\Bi-dir_Buck.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

6. Stop the program at the start of main().
7. Enable real-time modes.
8. Enable continuous refresh for watch views and changing the continuous refresh interval.
9. Click Run on the tool bar to run the code.
10. Apply a resistive load of 2 Ω to the PSFB system at the DC output.
11. Power the 400-V input with 390-V DC.
12. Change the Gui_IoutSet command to 4 A (in Q12).

NOTE: The output current should now be approximately 4 A and the output voltage should be approximately 8 V. The output inductor current measurement accuracy typically improves at a higher load. You may now change the Gui_IoutSet command but ensure these changes meet the requirements of the board.

Figure 48 shows the watch view that corresponds to the operation of the system with 6-A current command with a load of 2 Ω at the output and an input voltage of approximately 390 V.

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vfbin	int	387.5 (Q-Value(5))	0x0000041B@Data
(x)= Gui_Vfbout	int	11.3359 (Q-Value(10))	0x0000041D@Data
(x)= Gui_VfbSet	int	2.0 (Q-Value(10))	0x0000041C@Data
(x)= Gui_Ifb	int	0.450684 (Q-Value(12))	0x0000041A@Data
(x)= Gui_Iout	int	6.00391 (Q-Value(8))	0x00000422@Data
(x)= Gui_IoutSet	int	6.0 (Q-Value(8))	0x00000421@Data
(x)= phase	long	0.3539536595 (Q-Value(2...	0x00000428@Data
(x)= OC_FaultFlg	int	0	0x00000420@Data
(x)= Ipri_trip	int	0.899994 (Q-Value(15))	0x00000410@Data
(x)= dbAtoP_leg	int	32	0x00000411@Data
(x)= dbPtoA_leg	int	18	0x0000040B@Data
(x)= auto_DB	int	1	0x0000040A@Data
(x)= SR_mode	int	2	0x0000040C@Data
Add new expression			

Figure 48. Buck Mode Expressions Window: Build 2 ACMC Mode (Runtime)

To fully halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V DC input and wait a few seconds.
2. Click Halt on the toolbar to halt the processor.
3. Click  to take the MCU out of real-time mode.
4. Reset the MCU.
5. Close CCS.

7 Boost Mode (Push-Pull) – Functional Description

7.1 Voltage Mode Control

VMC mode implementation in the boost mode is similar to the VMC implementation in buck mode as shown in Figure 49. The key difference in this mode of operation is that the converter works as a duty-controlled converter, unlike the phase-shift controlled buck mode converter.

The buck mode output inductor acts as a current source in the boost mode, which allows this topology to work as a current-fed push-pull converter. The push-pull switches are driven with PWM signals with greater than 50% duty cycles that are 180 degrees out of phase to each other. This duty cycle dictates the amount of energy transferred to the high-voltage side. The controller regulates the output by controlling this energy transfer by directly controlling the duty cycle of PWM signals driving the two push-pull switches. Full-bridge switches on the HV side may be kept off and their body diodes used for rectification. In this implementation, the full-bridge switches are used for synchronous rectification in the boost mode. Figure 51 shows these waveforms.

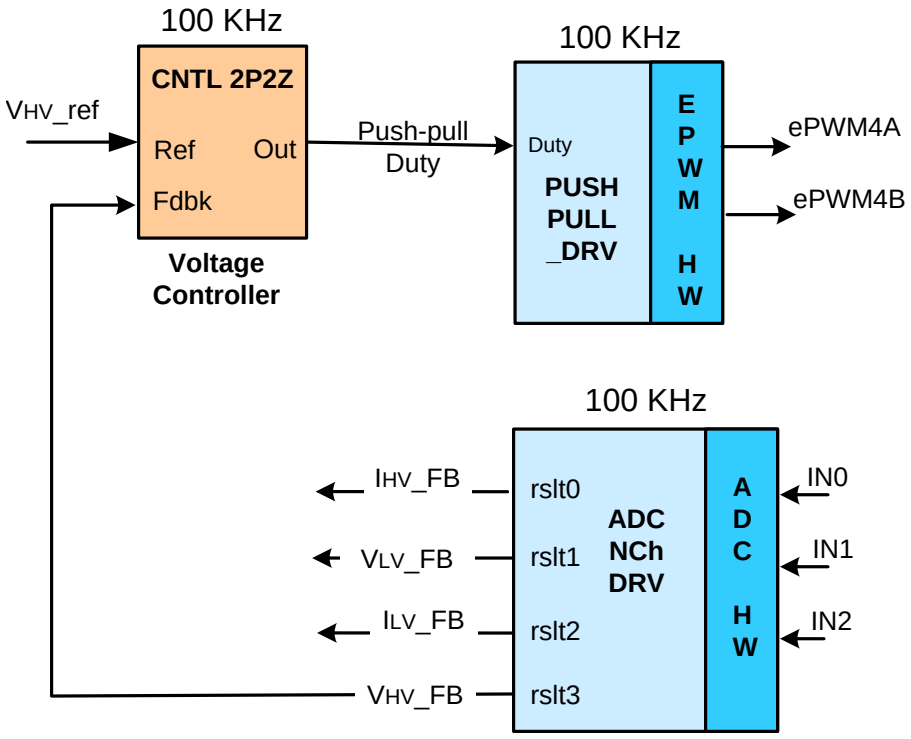


Figure 49. VMC Block Diagram

7.2 Average Current Mode Control (ACMC) of Output Current

Similar to the ACMC implementation in buck mode, this implementation uses a single average current loop to control the high-voltage battery or bus charging current (constant current charging). The ACMC implementation is similar to the VMC implementation in Section 7.1 in terms of waveform generation and power stage control. As shown in Figure 50, the controller directly drives and controls the duty cycle of PWM signals driving the two push-pull switches. As in VMC mode, the controller regulates the output by controlling energy transfer by directly controlling this duty cycle.

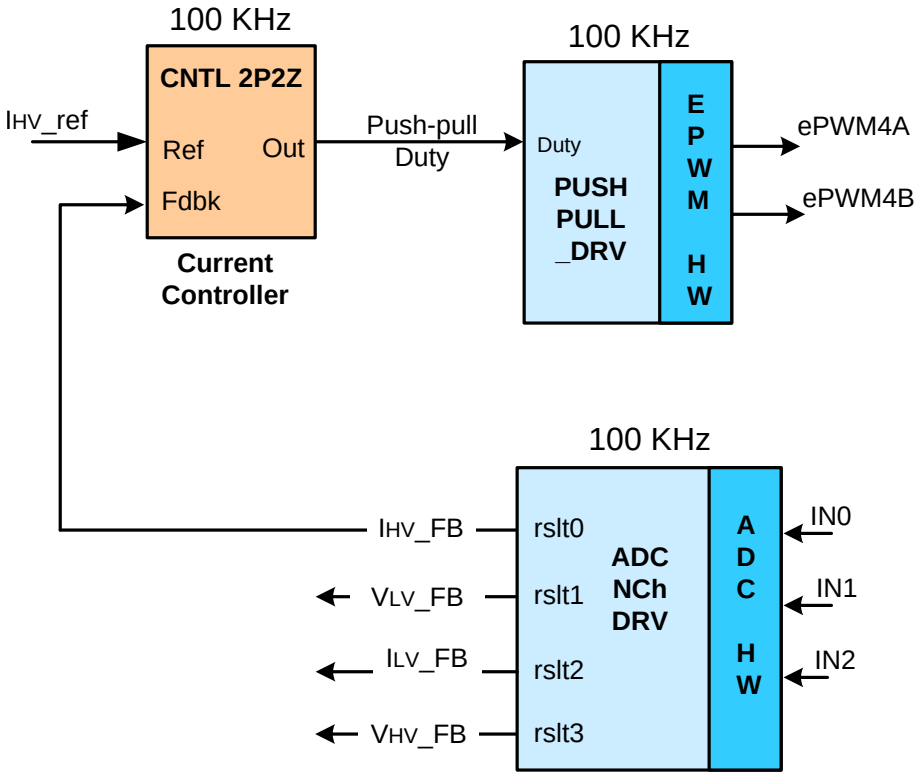


Figure 50. ACMC Block Diagram

Figure 51 shows various waveforms during boost mode of operation.

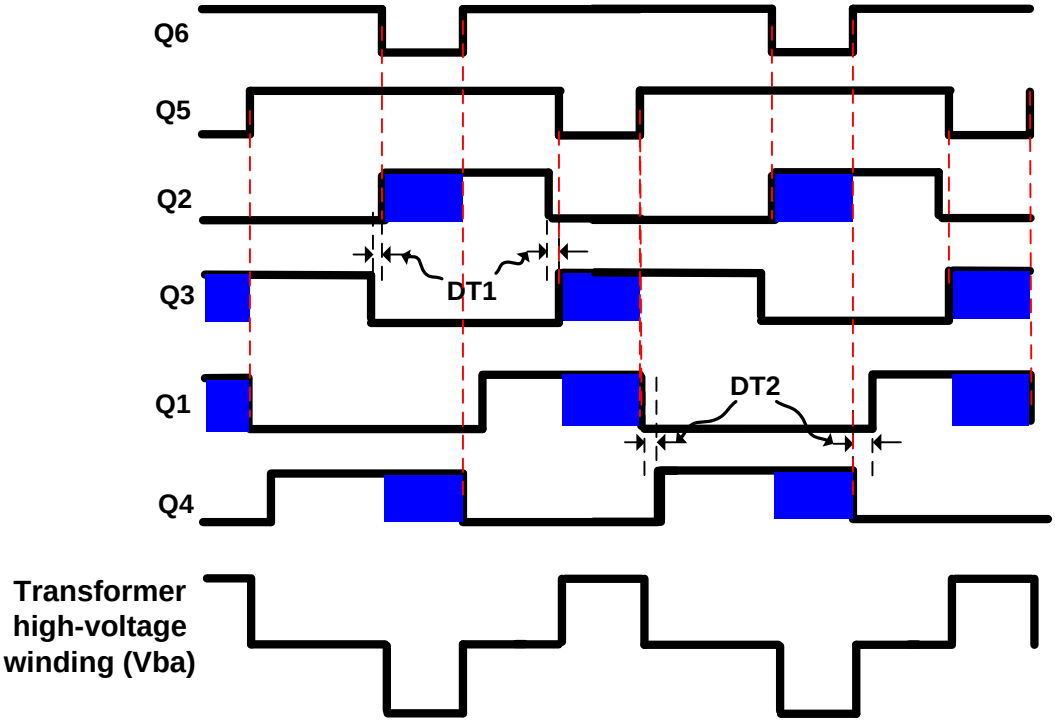


Figure 51. Boost Mode Waveforms

8 Boost Mode – Software Overview

8.1 Software Control Flow

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

Figure 52 shows the general software flow for this project.

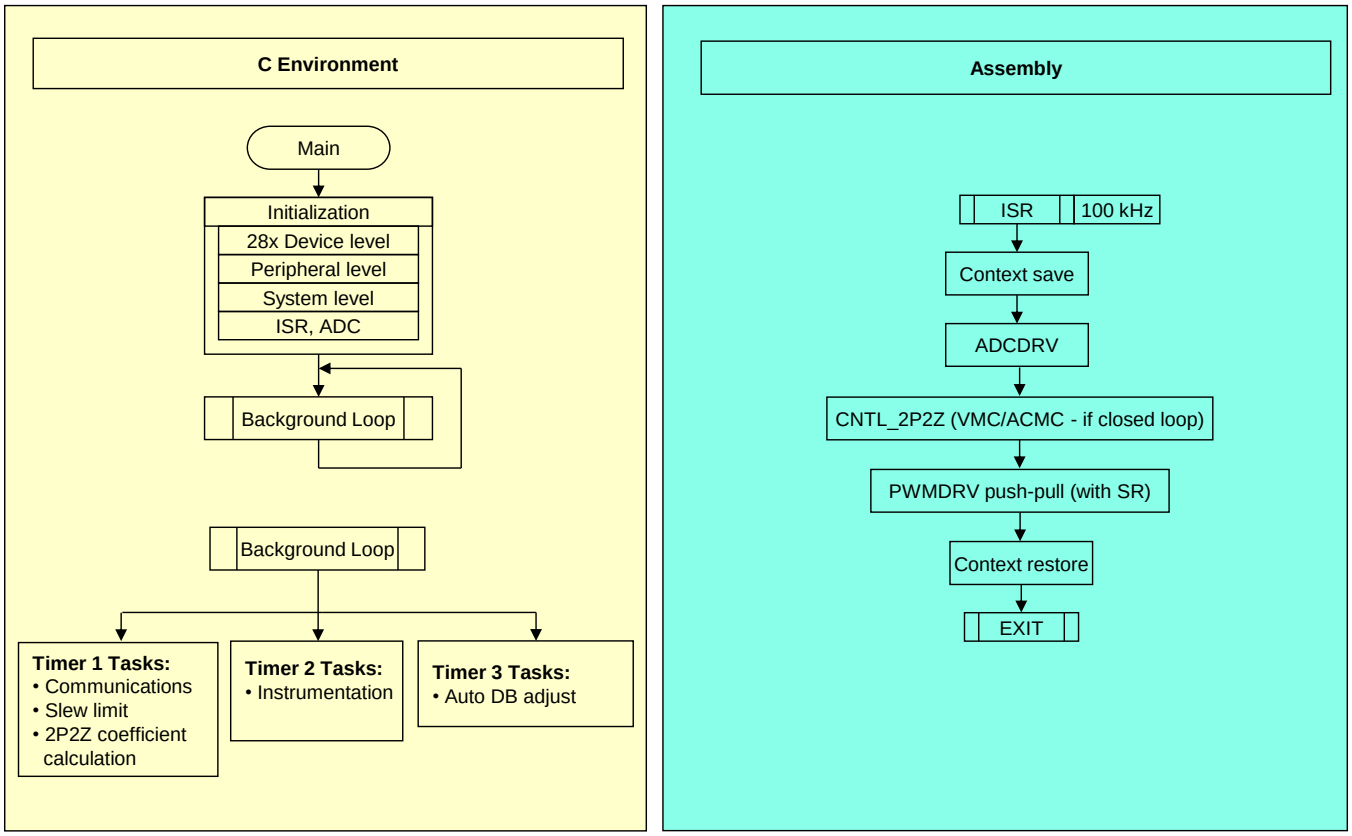


Figure 52. Boost Mode Software Flow

The key framework C files used in this project are the following:

- Bi-dir-Main.c – This file initializes, runs, and manages the application.
- Bi-dir-DevInit_F2803x.c – This file initializes and configures the microcontroller once and includes functions such as setting up the clocks, PLL, GPIO, and so forth.

The ISR consists of the following file:

- Bi-dir-DPL-ISR.asm – This file contains all time-critical control type code. This file has an initialization section (one-time execute) and a runtime section, which executes at 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. Table 5 lists the C and corresponding assembly module names for the library modules used in this project.

Table 5. Library Modules

C Configure Function	ASM Initialization Macro	ASM Runtime Macro
DAC_Cnf.c		
ADC_SOC_Cnf.c	ADCDRV_4CH_INIT m,n,p,q	ADCDRV_4CH m,n,p,q
PWM_PSFB_VMC_SR_Cnf.c	PWMDRV_BIDIR_BOOST_INIT m,n,p,q	PWMDRV_BIDIR_BOOST m,n,p,q
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

Figure 53 shows the control blocks.

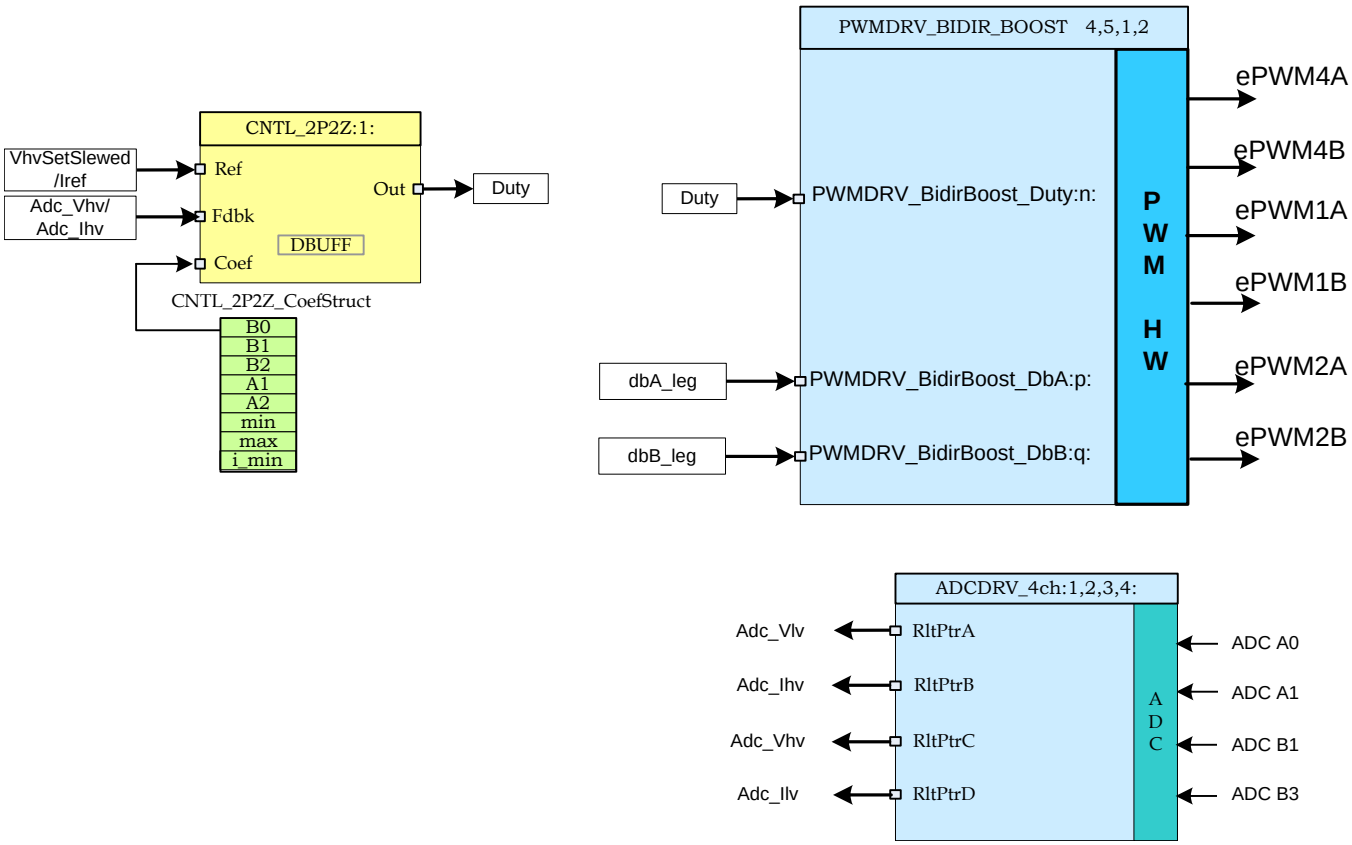


Figure 53. Software Blocks

In Figure 53, dark blue blocks represent hardware modules on the C2000 MCU. Blue blocks are the software drivers for these modules. Yellow blocks are the controller blocks for the control loop. Although a 2-pole 2-zero controller is used here, the controller could be a PI/PID, a 3-pole 3-zero, or any other controller that can be implemented for this application. The modular library structure makes it convenient to visualize and understand the complete system software flow, as shown in Figure 40. This structure also allows for easy use, addition, and removal of various functionalities. This fact is demonstrated in this project by implementing an incremental build approach and is discussed in more detail in the following section. The system is controlled by one voltage or current feedback loop.

Figure 54 also gives the rate at which control blocks are executed. For example, the voltage controller is executed at a rate of 100 kHz (the same as the PWM switching frequency).

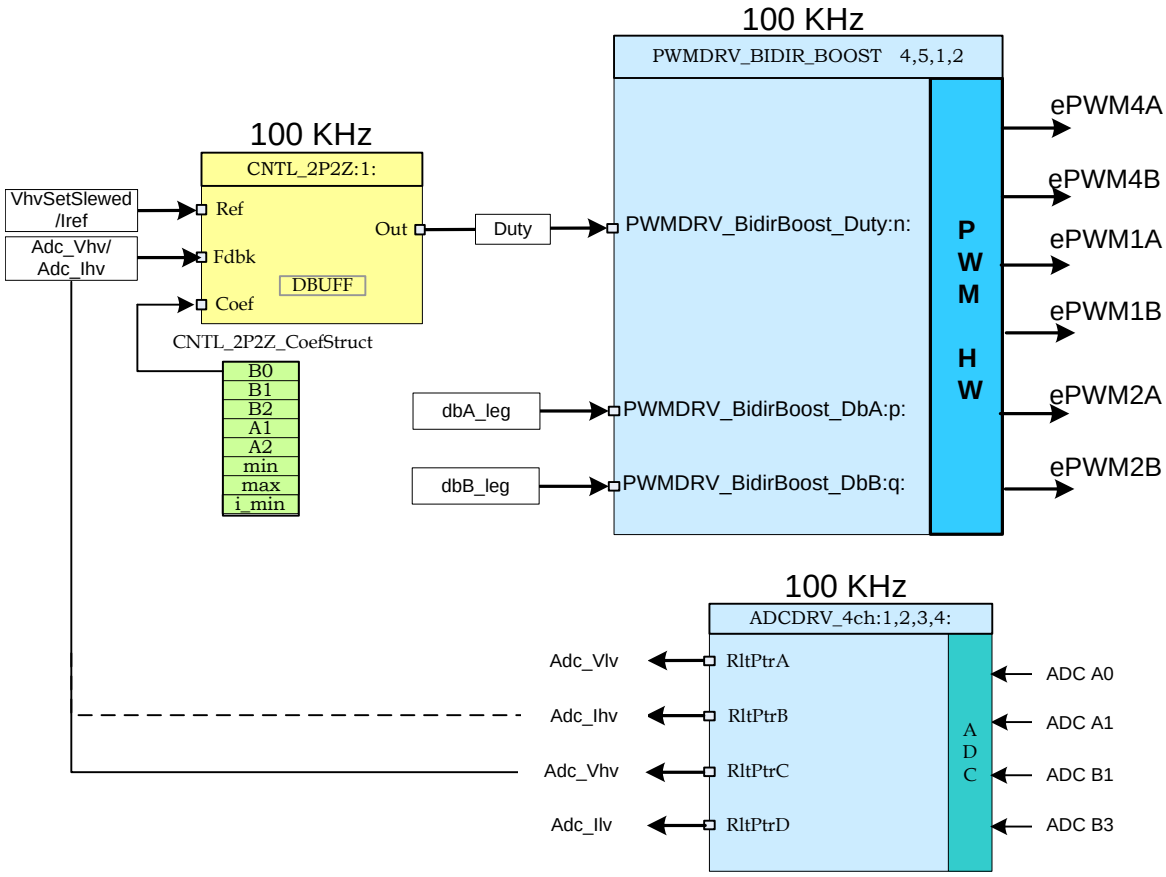


Figure 54. Control Flow

The sensed output voltage/current (Adc_Vhv/Adc_Ihv) is compared with voltage or current reference command (Vref/Iref) in the voltage/current controller. The controller output directly controls the duty cycle of PWM signals driving the two push-pull switches. This duty cycle dictates the amount of phase overlap between the two legs of the full-bridge for synchronous rectification on the high-voltage side. The values of dbA_leg and dbB_leg provide appropriate turnon delay for the full-bridge switches.

8.2 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. Table 6 shows the build options. To select a particular build option, set the macro INCR_BUILD, found in the Bi-dir-Settings.h file, to the corresponding build selection as shown Table 6. When the build option is selected, compile the complete project by selecting rebuild-all compiler option. For more details about running each build option, see Section 9.

Table 6. Incremental Build Options for Buck Mode

Option	Description
INCR_BUILD = 1	Open-loop boost drive with ADC feedback (check PWM drive circuit and sensing circuit)
INCR_BUILD = 2	Closed loop (full boost mode in VMC/ACMC mode)

9 Boost Mode – Procedure for Running Incremental Builds

The main source files, ISR assembly file, and the project file for C framework to bring up the system are in the following directory (use the latest version of the software package).

- **..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Boost_Mode**

Figure 7 shows the hardware setup for boost mode tests.

WARNING

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

The following steps show how to build and run the example in the Boost_Mode software.

9.1 Build 1: Open-Loop Check With ADC Feedback

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 CCS. Because this system is running open-loop, the ADC measured values are used only for instrumentation purposes in this build. This section explores the steps required to build and run a project.

Overview

The software in Build 1 is configured so that you can quickly evaluate the current-fed push-pull PWM driver module by viewing the output waveforms on an oscilloscope and observing the effect of change in duty cycle on the output voltage by interactively adjusting the duty command on CCS. 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 in Section 8. Figure 55 shows the blocks used in this build. ePWM1A and ePWM1B drive Q2 and Q3 full-bridge switches, respectively, while ePWM2A and ePWM2B drive Q1 and Q4 full-bridge switches, respectively. ePWM4A and ePWM4B drive Q6 and Q5 push-pull switches, respectively.

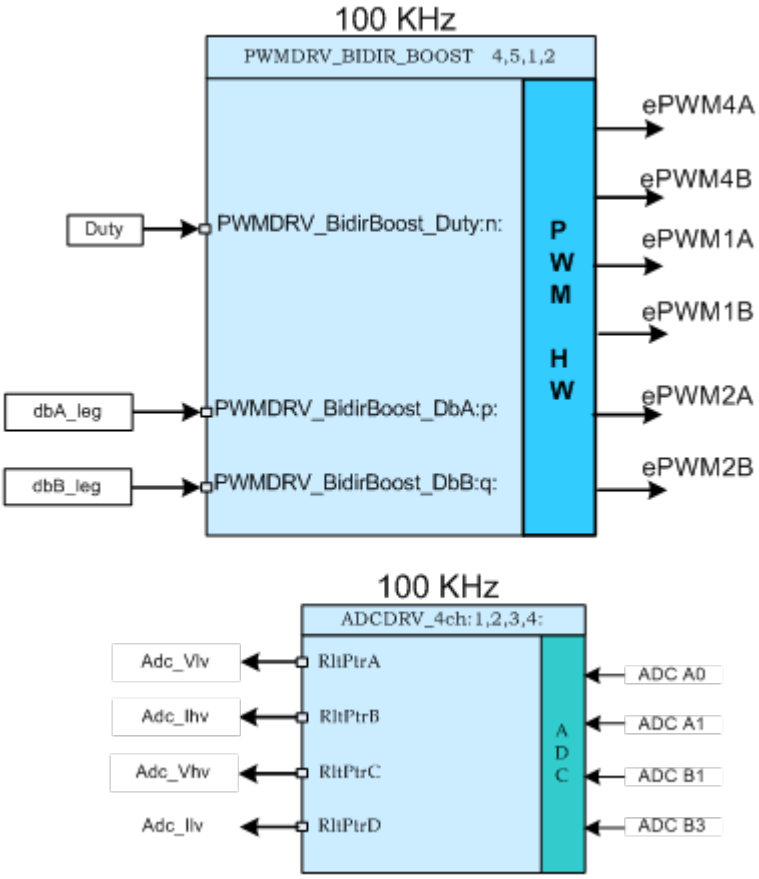


Figure 55. Build 1 Software Blocks

These PWM signals must be generated at a frequency of 100 kHz (that is, 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. That this one count of the time base counter of ePWM1, ePWM2, or ePWM4 corresponds to 16.667 implies that a PWM period of 10 µs is equal to 600 counts of the time base counter (TBCNT1, TBCNT2, and TBCNT4). The ePWM1, ePWM2, and ePWM4 modules are configured to operate in up-down count mode. Outputs of ePWM1A and ePWM1B operate at 50% duty cycle and are complementary. Similarly, ePWM2A and ePWM2B operate at 50% duty cycles and are complementary. ePWM4A and ePWM4B operate with greater than 50% duty cycle and 180 degrees out of phase with respect to each other. Figure 51 shows these PWM waveforms.

When overlap between PWM signals driving the two push-pull switches ends, the corresponding pair of diagonal full-bridge switches are turned on. While Q6 is turned off, PWM signals driving Q2 and Q4 overlap. While Q5 is turned off, PWM signals driving Q1 and Q3 overlap.

The assembly ISR is triggered on a ZRO (TBCNT1 = 0) event of ePWM1. This ISR is where the control driver macros are executed and the push-pull duty command is updated. 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) triggers five ADC conversions.

Protection

The shutdown mechanism used with this project includes overcurrent protection implemented for the transformer high-voltage winding current using on-chip analog comparator 1. Overcurrent protection for LV inductor current is implemented using on-chip analog comparator 2. The reference trip levels are set using the internal 10-bit DACs and fed to the inverting terminals of these comparators. The comparator outputs are configured to generate a one-shot trip action on ePWM1, ePWM2, and ePWM4 whenever the sensed current is greater than the set limit. Overvoltage protections for both LV and HV sides are implemented in the software inside the slower state machine task A1. Whenever an overvoltage condition is detected, a one-shot trip action is initiated on ePWM1, ePWM2, and ePWM4.

The flexibility of the trip mechanism on C2000 devices provides the possibilities for taking different actions on different trip events. In this project, outputs of ePWM1A, ePWM1B, ePWM2A, ePWM2B, ePWM4A, and ePWM4B are driven low immediately to protect the power stage. These outputs are held low until a device reset is executed.

9.1.1 Procedure

Start CCS and Open a Project

To quickly execute this build, do the following:

1. Ensure that all jumpers on the board are correctly installed or removed as listed in Section 2.
2. Insert the F28035 controlCARD in the 100-pin DIMM connector.
3. Connect a 12-V DC bench power supply between TP10 and TP11 with correct polarity.
4. Connect an isolated 400-V programmable DC power source to the 400-V input connector and a 12-V load to the 12-V connector (ensure this load does not exceed the board ratings).
5. Connect a USB-B to USB-A cable between the PC and the board.

NOTE: Do not turn on any power supplies at this time.

6. If this is the first time the board is being tested with JTAG connection, run the program_ftdi.bat file in the xds100v2-FT_Prog_v2.2.zip file to program the FTDI chip on the board.
7. Open Code Composer Studio (CCSv5 or later).
8. Maximize CCS to fill your screen.
9. Close the welcome screen if it opens.

NOTE: A project contains all the files and build options needed to develop an executable output file (.out), which can be run on the MCU hardware.

10. On the menu bar, click: Project → Import Existing CCS/CCE Eclipse Project.
11. Select the **..\controlSUITE\ development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Boost_Mode** directory.
12. Under the Projects tab, ensure Bi-dir_Boost is checked.
13. Click Finish.

NOTE: This project uses all the necessary tools (compiler, assembler, linker) to build the project.

14. In the Project window on the left, click + to the left of Project. See Figure 56.

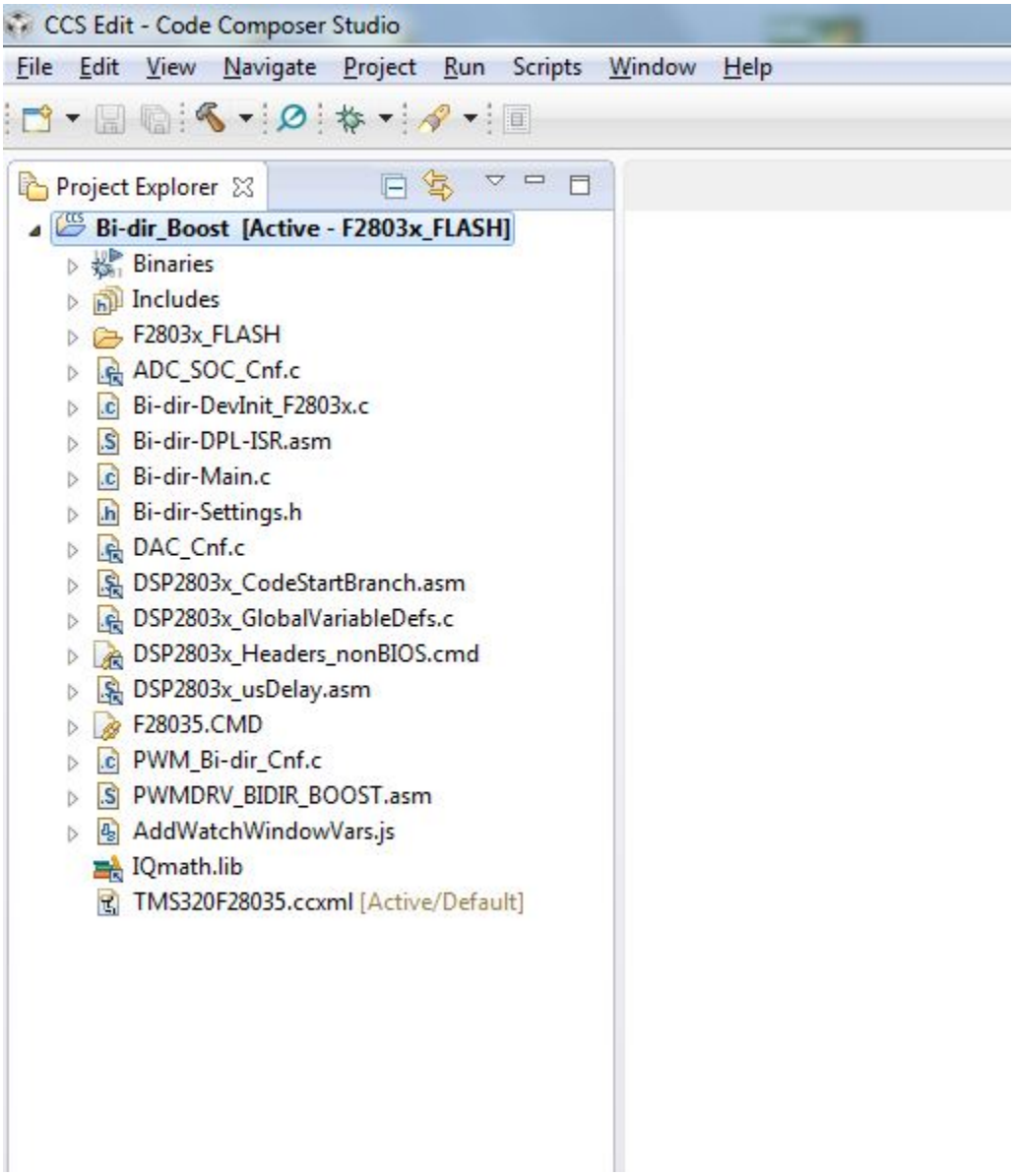


Figure 56. Boost Mode CCS Project Window

Device Initialization, Main, and ISR Files

NOTE: Do not make any changes to the source files.

1. Double-click Bi-dir-DevInit_F2803x.c.
2. Ensure that the system clock, peripheral clock prescale, and peripheral clock enables have been set up.
3. Ensure that the shared GPIO pins have been configured.
4. Double-click Bi-dir-Main.c.
5. View the call to the DeviceInit() function, the code for different incremental build options, the ISR initialization, and the background for(;;) loop.
6. Locate code for build 1 in the main file.

7. Inspect the code.

NOTE: This code is where the ADCDRV_4CH block is connected and initialized in the control flow.

8. Double-click Bi-dir-DPL-ISR.asm.

NOTE: Tthe PWM and ADC driver macro instantiation for initialization and run time occurs in the _DPL_Init and _DPL_ISR sections, respectively. You can close the inspected files.

Build and Load the Project

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

NOTE: If another option was built previously, right-click on the project name and click Clean Project.

2. Click Project → Build All.
3. Turn on the 12-V DC bench power supply.
4. Click the Debug button.

NOTE: The Build 1 code should compile and load.

To create an F28035 target configuration or copy the configuration from the F28035.ccxml file, do the following:

1. Right-click F28035.ccxml in the Target Configurations window.
2. Right-click on the name of the device in the Debug window to connect the device.
3. Click Run → Load → Load Program.
4. Navigate to Boost_Mode\F2803x_Flash\Bi-dir_Boost.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

5. Stop the program at the start of main().

Debug Environment Windows

Watch local and global variables while debugging code. You can use the memory views and watch views in CCS. CCS can create time and frequency domain plots. You can view waveforms using graph windows.

1. Click View → Scripting console on the menu bar.
2. Use the scripting console Open File ( )command to open the AddWatchWindowVars.js file from the project directory.

This action populates the Expressions window entries. See Figure 43 to see how the Expressions window should look.

Some of the variables have not been initialized at this point in the main code and may contain some garbage values

If set, OC_FaultFlg indicates an overcurrent condition that shuts down the PWM outputs. PWM outputs are held in this state until a device reset. The Ihv_trip variable sets the internal 10-bit DAC reference level for the on-chip comparator 1 (HV winding overcurrent threshold), while Ilv_trip variable sets the internal 10-bit DAC reference level for the on-chip comparator 2 (LV inductor overcurrent threshold). These are Q15 numbers. See .

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
Gui_Vhv	int	100.063 (Q-Value(5))	0x00000425@Data
Gui_VhvSet	int	0.0 (Q-Value(5))	0x0000041C@Data
Gui_Vlv	int	0.12793 (Q-Value(10))	0x00000430@Data
Gui_Ihv	int	0.029541 (Q-Value(12))	0x00000428@Data
Gui_IhvSet	int	0.0 (Q-Value(12))	0x0000041D@Data
Gui_Ilv	int	0.3125 (Q-Value(8))	0x0000042F@Data
start_converter	int	0	0x0000042D@Data
Boost_ON	int	0	0x00000429@Data
BoostDuty	long	5.960464478e-07 (Q-Value(24))	0x0000044A@Data
OC_FaultFlg	int	1	0x00000407@Data
HVbus_OV	int	0	0x0000040B@Data
LVbus_OV	int	0	0x00000406@Data
Ihv_trip	int	0.949982 (Q-Value(15))	0x00000418@Data
Ilv_trip	int	0.949982 (Q-Value(15))	0x00000413@Data
dbA1	int	20	0x00000410@Data
dbB2	int	20	0x0000040E@Data
auto_DB	int	1	0x0000040F@Data
Add new expression			

Figure 57. Boost Mode Expressions Window: Build 1

Use Real-Time Emulation

Real-time emulation is a special emulation feature that allows updating of the windows within CCS at a rate up to a 10-Hz rate while the MCU is running. This emulation lets graphs and watch views update and lets you change values in watch or memory windows so that those changes affect the behavior of the MCU. This emulation is useful when tuning control law parameters on-the-fly.

To enable real-time mode, do the following:

1. Hover the mouse on the buttons on the horizontal toolbar.
2. Click Enable Silicon Real-time Mode (service critical interrupts when halted, allow debugger accesses while running).

NOTE: If a message box appears, select yes to enable debug events. This action sets bit 1 (DGBM bit) of status register 1 (ST1) to 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 to update the debugger windows.

When too many windows or variables are open or updating, continuous refresh can cause the refresh frequency to slow down because bandwidth over the emulation link is limited.

To decrease the refresh rate for the expressions window, do the following:

1. Right-click in the Expressions window.
2. Select Continuous Refresh Interval….
3. Change the continuous refresh interval value. (A rate of 1000 ms is usually enough for these exercises.)
4. Click for the watch view.

Run the Code

1. Click Run on the toolbar.
2. Ensure the start_converter variable is set to 0 in the watch view.
3. Ensure the BoostDuty is set to 0.0 (Q24).

NOTE: This variable denotes the duty command to the PWMDRV_BIDIR_BOOST module.

4. Apply a 2-kΩ resistive load to the HV (400-V) DC output.

NOTE: TI recommends using isolated DC supplies to supply 400-V DC and 12-V inputs.

5. Set the 400-V DC supply to output 100-V DC.
6. Turn on this supply.
7. Set start_converter to 1. (The 400-V DC bus voltage increases to approximately 150 V and the 400-V DC supply current reduces to 0.)
8. Set BoostDuty to 0.1 (Q24) in the watch view to increase the duty command.
9. Observe the output voltage as it increases and boost action takes over.

NOTE: Do not let the voltage exceed the capabilities of the board.

When operating with a certain BoostDuty value, reducing the load suddenly can cause the output voltage to suddenly increase. Do not make any sudden changes of load, input voltage, or large increases in BoostDuty command when operating in Build 1.

10. Observe the different ADC results in the watch view for different BoostDuty values.

Figure 58 is a watch view that corresponds with the operation of the system with a BoostDuty command of 0.35 (Q24) to get the HV voltage output of approximately 250 V with a 12-V LV input and a load of 2 kΩ at the HV output. (A load of 100 Ω was also connected at the 12-V connector.)

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	250.188 (Q-Value(5))	0x00000425@Data
(x)= Gui_VhvSet	int	0.0 (Q-Value(5))	0x0000041C@Data
(x)= Gui_Vlv	int	11.9395 (Q-Value(10))	0x00000430@Data
(x)= Gui_Ihv	int	0.39209 (Q-Value(12))	0x00000428@Data
(x)= Gui_IhvSet	int	0.0 (Q-Value(12))	0x0000041D@Data
(x)= Gui_Ilv	int	4.10938 (Q-Value(8))	0x0000042F@Data
(x)= start_converter	int	1	0x0000042D@Data
(x)= Boost_ON	int	1	0x00000429@Data
(x)= BoostDuty	long	0.3499999642 (Q-Value(24))	0x0000044A@Data
(x)= OC_FaultFlg	int	0	0x00000407@Data
(x)= HVbus_OV	int	0	0x0000040B@Data
(x)= LVbus_OV	int	0	0x00000406@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x00000418@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x00000413@Data
(x)= dbA1	int	32	0x00000410@Data
(x)= dbB2	int	18	0x0000040E@Data
(x)= auto_DB	int	1	0x0000040F@Data
Add new expression			

Figure 58. Boost Mode Expressions Window: Build 1 (Runtime)

11. Try several different BoostDuty values.
12. Observe the ADC results.

NOTE: Increase phase in small steps. Always observe the output voltage carefully. Do not let the voltage exceed the capabilities of the board.

You can use an oscilloscope to probe waveforms, like the PWM gate drive signals, input voltage, and current and output voltage.

Take appropriate safety precautions and consider appropriate grounding requirements while probing these high voltages and high currents for this isolated DC-DC converter.

To halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V DC input and wait a few seconds.
2. Click Halt on the toolbar to halt the processor.
3. Click  to take the MCU out of real-time mode.
4. Reset the MCU.

9.2 Build 2: Closed-Voltage Loop

Objective

The objective of this build is to verify the operation of the complete push-pull project from the CCS environment.

Overview

Figure 59 shows the software blocks in this build. The PWM and ADC driver blocks are used in the same way as in Build 1 of this chapter. A two-pole two-zero controller is used for the voltage or current loop. Depending on the control loop requirements of the application, some other controller block like a PI, a 3-pole 3-zero, and so forth may also be used.

As seen Figure 59, 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 does not require a CNF function call.

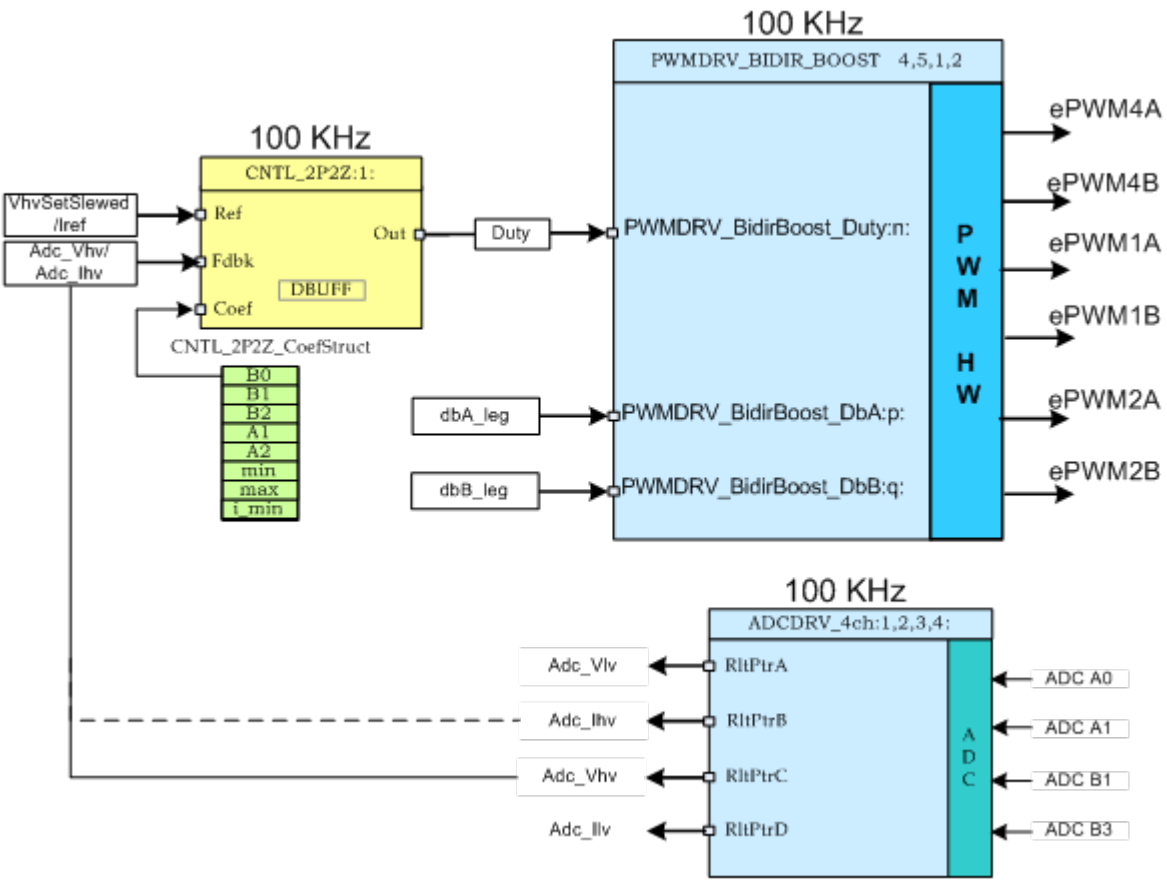


Figure 59. Build 2 Software Blocks

The five coefficients to be modified are stored as elements of the structure CNTL_2P2Z_CoefStruct1, whose other elements clamp the controller output. The CNTL_2P2Z block can be instantiated multiple times if the system requires 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 and/or assistance from tools such as MATLAB, Mathcad, and so forth. These tools offer a Bode plot, root-locus, and other features for determining phase margin, gain margin, and so forth.

To keep loop tuning simple and avoid the need for complex mathematics or analysis tools, TI reduced the coefficient selection problem 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 reduction independently and gradually adjusts P, I, and D. These mapping equations are given by the following equations.

The compensator block (CNTL_2P2Z) has 2 poles and 2 zeros and is based on the general IIR filter structure. This block has a reference input and a feedback input. The feedback is the sensed output voltage (Adc_Vhv) when VMC mode is selected or the sensed output current (Adc_Ihv) when ACMC mode is selected. This selection can be made from the Bi-dir-Settings.h file. The reference input to the controller is a slewed version (VhvSetSlewed) of the output voltage reference command (Vref) or is the reference output current command (Iref) based on the mode selection. Equation 8 gives the transfer function:

$$\frac{U(z)}{E(z)} = \frac{b_0 + B_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{8}$$

The recursive form of the PID controller is given by Equation 9:

$$u(k) = u(k-1) + b_0 e(k) + b_1 e(k-1) + b_2 e(k-2) \tag{9}$$

Where Equation 10, Equation 11, and Equation 12:

$$b_0 = K_p' + K_i' + K_d' \tag{10}$$

$$b_1 = -K_p' + K_i' - 2K_d' \tag{11}$$

$$b_2 = K_d' \tag{12}$$

And the z-domain transfer function form of this is Equation 13:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-1}} = \frac{b_0 z^{-2} + b_1 z + b_2}{z^2 - z} \tag{13}$$

Comparing this with the general form, PID is a special case of CNTL_2P2Z control where Equation 14:

$$a_1 = -1 \text{ and } a_2 = 0 \tag{14}$$

The P, I, and D coefficients are Pgain, Igain, and Dgain. These P, I, and D coefficients are used in the Q26 format. To simplify tuning from CCS watch views, these three coefficients are further scaled to values from 0 to 999 (Pgain_Gui, Igain_Gui, and Dgain_Gui). The loop parameters can also be changed directly based on tuned values obtained from external mathematical (MATLAB, Mathcad, and so forth) tools. Loop coefficients can be directly changed using variables b2_Gui, b1_Gui, b0_Gui, a2_Gui, and a1_Gui in I5Q10 form, which are then converted to the five Q26 coefficients for the 2P2Z controller.

This project allows easy evaluation of both methods of loop tuning by providing the ability to easily switch between coefficients during execution. This switching can be done by 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 CCS. PID-based loop tuning (pid2p2z_GUI = 0) from the GUI environment has been used for this project. Please make sure that reliable coefficient values are programmed in b2_Gui, b1_Gui, b0_Gui, a2_Gui, and a1_Gui variables before changing the pid2p2z_GUI to 1.

NOTE: This project does not include valid b2_Gui, b1_Gui, b0_Gui, a2_Gui, and a1_Gui parameter values. TI recommends using the default P-, I-, D-based loop by keeping pid2p2z_GUI = 0.

9.2.1 Procedure

Build and Load Project

To quickly execute this build using the preconfigured work environment, do the following:

1. Ensure that all jumpers on the board are correctly installed or removed as listed in Section 2.
2. Insert the F28035 controlCARD in the 100-pin DIMM connector.
3. Connect a 12-V DC bench power supply between TP10 and TP11 with correct polarity.
4. Connect an isolated 400-V programmable DC power source to the 400-V input connector and a 12-V load to the 12-V connector (ensure this load does not exceed the board ratings).
5. Connect a USB-B to USB-A cable between the PC and the board.

NOTE: Do not turn on any power supplies at this time.

6. If this is the first time the board is being tested with JTAG connection, run the program_ftdi.bat file in the xds100v2-FT_Prog_v2.2.zip file to program the FTDI chip on the board.
7. Open Code Composer Studio (CCSv5 or later).
8. Maximize CCS to fill your screen.
9. Close the welcome screen if it opens.

NOTE: A project contains all the files and build options needed to develop an executable output file (.out), which can be run on the MCU hardware.

10. On the menu bar, click Project → Import Existing CCS/CCE Eclipse Project.
11. Select the **..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Boost_Mode** directory.
12. Under the Projects tab, ensure Bi-dir_Boost is checked.
13. Click Finish.

NOTE: This project uses all the necessary tools (compiler, assembler, linker) to build the project.

14. In the Project window on the left, click + to the left of Project.
15. Locate the initialization code for Build 2 in the main file.
16. Inspect the code. (This code is where all the control blocks are configured, initialized, and connected in the control flow.)
17. Select the Incremental build option as 2 in the Bi-dir-Settings.h file.

NOTE: If another build option was built previously, do the following:
1. Right-click the project name.
2. Click Clean Project.
3. Click Project → Build All.
4. Watch the tools run in the build window.

18. Turn on the 12-V DC bench power supply.
19. Click the Debug button.

NOTE: The Build 1 code compiles and loads

To create an F28035 target configuration or copy the configuration from the F28035.ccxml file, do the following:

1. Right-click F28035.ccxml in the Target Configurations window.
2. Right-click on the name of the device in the Debug window to connect the device.
3. Click Run → Load → Load Program.
4. Navigate to Boost_Mode\F2803x_Flash\Bi-dir_Boost.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

5. Stop the program at the start of main().

Debug Environment Windows

Watch local and global variables while debugging code. You can use the memory views and watch views in CCS. CCS can create time and frequency domain plots. You can view waveforms using graph windows.

1. Click View → Scripting console on the menu bar.
2. Use the scripting console Open File ( )command to open the AddWatchWindowVars.js file from the project directory.

This action populates the Expressions window entries. See Figure 60 to see how the Expressions window should look.

Some of the variables have not been initialized at this point in the main code and may contain some garbage values

If set, OC_FaultFlgindicates an overcurrent condition that shuts down the PWM outputs. PWM outputs are held in this state until a device reset. The Ipri_trip variable sets the internal 10-bit DAC reference level for the on-chip comparator 1. This is a Q15 number. See Figure 60.

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	100.063 (Q-Value(5))	0x00000425@Data
(x)= Gui_VhvSet	int	0.0 (Q-Value(5))	0x0000041C@Data
(x)= Gui_Vlv	int	0.12793 (Q-Value(10))	0x00000430@Data
(x)= Gui_Ihv	int	0.029541 (Q-Value(12))	0x00000428@Data
(x)= Gui_IhvSet	int	0.0 (Q-Value(12))	0x0000041D@Data
(x)= Gui_Ilv	int	0.3125 (Q-Value(8))	0x0000042F@Data
(x)= start_converter	int	0	0x0000042D@Data
(x)= Boost_ON	int	0	0x00000429@Data
(x)= BoostDuty	long	5.960464478e-07 (Q-Value(24))	0x0000044A@Data
(x)= OC_FaultFlg	int	1	0x00000407@Data
(x)= HVbus_OV	int	0	0x0000040B@Data
(x)= LVbus_OV	int	0	0x00000406@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x00000418@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x00000413@Data
(x)= dbA1	int	20	0x00000410@Data
(x)= dbB2	int	20	0x0000040E@Data
(x)= auto_DB	int	1	0x0000040F@Data
Add new expression			

Figure 60. Boost Mode Expressions Window: Build 2

Run the Code

1. Enable real-time mode.
2. Enable continuous refresh for the watch views and changing the continuous refresh for the watch view.
3. Click Run on the toolbar.
4. Set the start_converter variable to 0.0 (Q24). (This variable denotes the voltage command.)
5. Apply a 2-kΩ resistive load to the PSFB system at the DC output.

NOTE: TI recommends using an isolated DC source to supply 400-V DC input to the board.

6. Set the 400-V DC supply to output 100-V DC.
7. Turn on this supply.
8. Set the output to of 12-V DC supply to 12 V.
9. Turn on this supply.
10. Set start_converter to 1. (The 400-V DC bus voltage increases to approximately 150 V and the 400-V

DC supply reduces to 0.)

11. Increase the voltage command by setting Gui_VhvSet to a value of 200 V in the watch view. (Boost action takes over and the output voltage follows the set command.)
12. Observe the output voltage as it increases.

NOTE: Do not let the output voltage exceed the capabilities of the board.

Do not let the Gui_VhvSet command be less than the 400-V DC power supply voltage setting.

13. Observe the different ADC results in the watch view for different Gui_VhvSet values. (Every time this value is changed the output voltage should ramp-up or ramp-down to this new voltage. Change the output voltage ramp up or ramp down rate by changing the VhvSlewRate.)

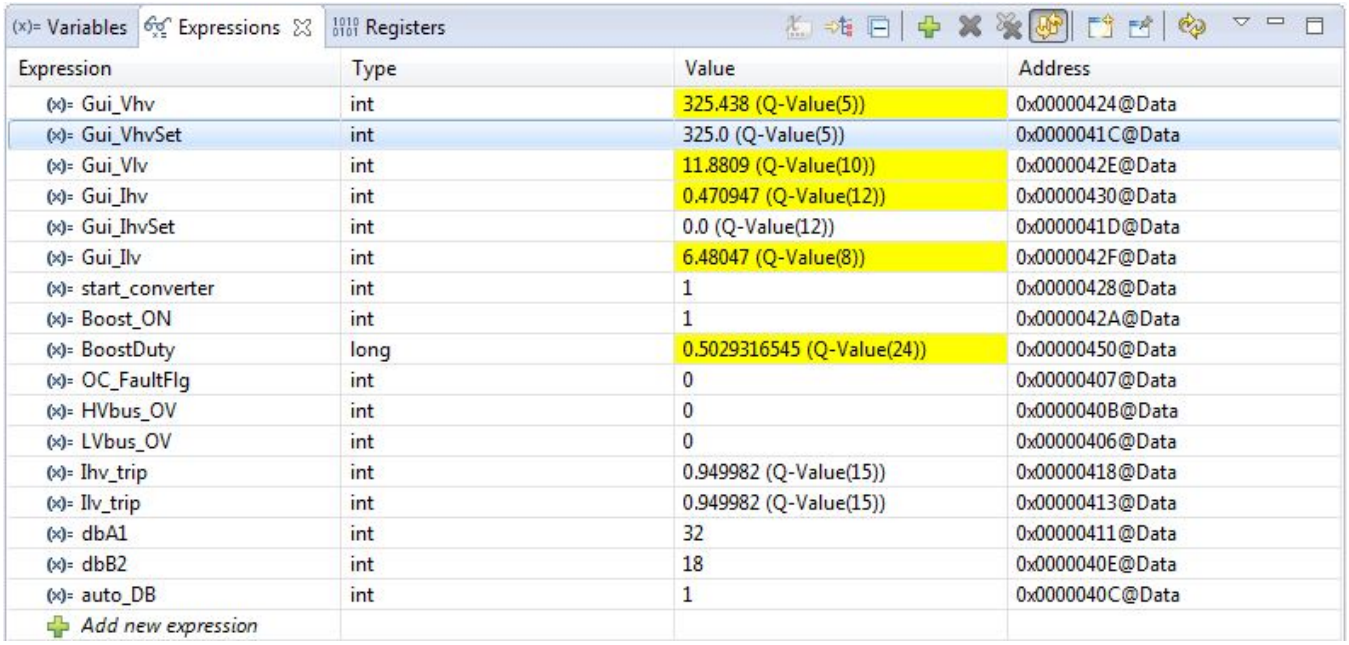
    Figure 61 is a watch view that corresponds with the operation of the system with 225 V with an input voltage of 12 V and a load of approximately 2 Ω.

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	224.719 (Q-Value(5))	0x00000424@Data
(x)= Gui_VhvSet	int	225.0 (Q-Value(5))	0x0000041C@Data
(x)= Gui_Vlv	int	12.0098 (Q-Value(10))	0x0000042E@Data
(x)= Gui_Ihv	int	0.35498 (Q-Value(12))	0x00000430@Data
(x)= Gui_IhvSet	int	0.0 (Q-Value(12))	0x0000041D@Data
(x)= Gui_Ilv	int	3.52344 (Q-Value(8))	0x0000042F@Data
(x)= start_converter	int	1	0x00000428@Data
(x)= Boost_ON	int	1	0x0000042A@Data
(x)= BoostDuty	long	0.30097121 (Q-Value(24))	0x00000450@Data
(x)= OC_FaultFlg	int	0	0x00000407@Data
(x)= HVbus_OV	int	0	0x0000040B@Data
(x)= LVbus_OV	int	0	0x00000406@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x00000418@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x00000413@Data
(x)= dbA1	int	32	0x00000411@Data
(x)= dbB2	int	18	0x0000040E@Data
(x)= auto_DB	int	1	0x0000040C@Data
Add new expression			

Figure 61. Boost Mode Expressions Window: Build 2 VMC Mode (Runtime)

Figure 62 is the watch view that corresponds to the operation of the system with 325 V at the output with an input voltage of 12 V and a load of approximately 2 kΩ. (A load of 100 Ω was also connected at the 12-V connector)



(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	325.438 (Q-Value(5))	0x00000424@Data
(x)= Gui_VhvSet	int	325.0 (Q-Value(5))	0x0000041C@Data
(x)= Gui_Vlv	int	11.8809 (Q-Value(10))	0x0000042E@Data
(x)= Gui_Ihv	int	0.470947 (Q-Value(12))	0x00000430@Data
(x)= Gui_IhvSet	int	0.0 (Q-Value(12))	0x0000041D@Data
(x)= Gui_Ilv	int	6.48047 (Q-Value(8))	0x0000042F@Data
(x)= start_converter	int	1	0x00000428@Data
(x)= Boost_ON	int	1	0x0000042A@Data
(x)= BoostDuty	long	0.5029316545 (Q-Value(24))	0x00000450@Data
(x)= OC_FaultFlg	int	0	0x00000407@Data
(x)= HVbus_OV	int	0	0x0000040B@Data
(x)= LVbus_OV	int	0	0x00000406@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x00000418@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x00000413@Data
(x)= dbA1	int	32	0x00000411@Data
(x)= dbB2	int	18	0x0000040E@Data
(x)= auto_DB	int	1	0x0000040C@Data
Add new expression			

Figure 62. Boost Mode Expressions Window: Build 2, Vout = 325 V

14. Observe the effect of varying load on the output voltage and input current. (The output voltage should not be affected.)

NOTE: Do not let the voltage exceed the capabilities of the board listed in the specifications section of this document.

15. Observe the effect of varying the input voltage (12-V DC power source).

NOTE: Increase phase in small steps. Always observe the output voltage carefully. Do not let the voltage to exceed the capabilities of the board.

You can use an oscilloscope to probe waveforms, like the PWM gate drive signals, input voltage, current voltage, and output voltage.

Take appropriate safety precautions and consider appropriate grounding requirements while probing these high voltages and high currents for this isolated DC-DC converter.

To halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V DC input.
2. Wait a few seconds.
3. Click Halt on the toolbar to halt the processor.
4. Click  to take the MCU out of real-time mode.
5. Reset the MCU.

To let the converter be controlled in output inductor control mode (ACMC mode), do the following:

1. Set VMC0_ACMC1 to 1 in Bi-dir-Settings.h.

NOTE: If another build option was built previously, do the following:
1. Right-click the project name.
2. Click Clean Project.
3. Click Project → Build All.
4. Watch the tools run in the build window.

2. Right-click F28035.ccxml in the Target Configurations window.
3. Right-click on the name of the device in the Debug window to connect the device.
4. Click Run → Load → Load Program.
5. Navigate to Boost_Mode\F2803x_Flash\Bi-dir_Boost.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

6. Stop the program at the start of main().
7. Enable real-time modes.
8. Enable continuous refresh for watch views and changing the continuous refresh interval.
9. Click Run on the toolbar to run the code.
10. Set the start_converter to 0.
11. Set Gui_VhvSet to 0.0 (Q5). (This variable denotes the voltage command.)
12. Apply a 2-kΩ resistive load at the HV (400-V) DC output.
13. Set the 400-V input to output 100-V DC.
14. Turn on this supply.
15. Set the 12-V DC supply to 12 V.
16. Turn on this supply.

17. Set the start_converter to 1. (The 400-V DC bus voltage increases to approximately 150 V and the 400-V DC supply current reduces to 0.)
18. Observe and write down the Gui_Ihv value that appears in the watch window (Q12).
19. Change the Gui_IhvSet command from 0 to a value no more than 0.01 (Q12) of the Gui_Ihv value written in Step 18.

CAUTION

Increase the Gui_IhvSet command in steps less than 0.01 (Q12).

Minor changes in command can create very high voltage changes at the 400-V output depending on the load.

Be careful when experimenting in ACMC mode.

Figure 63 shows the watch view that corresponds to the operation of the system with 0.4-A current command and a load of 2 kΩ with an input voltage of approximately 11.8 V, resulting in a 270-V output. (A load of 100 Ω was also connected at the 12-V connector.)

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	269.281 (Q-Value(5))	0x00000423@Data
(x)= Gui_VhvSet	int	0.0 (Q-Value(5))	0x0000041B@Data
(x)= Gui_Vlv	int	11.7627 (Q-Value(10))	0x0000042D@Data
(x)= Gui_Ihv	int	0.400879 (Q-Value(12))	0x0000042F@Data
(x)= Gui_IhvSet	int	0.399902 (Q-Value(12))	0x0000041C@Data
(x)= Gui_Ilv	int	4.80078 (Q-Value(8))	0x0000042E@Data
(x)= start_converter	int	1	0x00000408@Data
(x)= Boost_ON	int	1	0x00000429@Data
(x)= BoostDuty	long	0.4147862196 (Q-Value(24))	0x0000044E@Data
(x)= OC_FaultFlg	int	0	0x00000407@Data
(x)= HVbus_OV	int	0	0x0000040B@Data
(x)= LVbus_OV	int	0	0x00000406@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x00000427@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x00000413@Data
(x)= dbA1	int	32	0x00000411@Data
(x)= dbB2	int	18	0x0000040E@Data
(x)= auto_DB	int	1	0x0000040C@Data
Add new expression			

Figure 63. Boost Mode Expressions Window: Build 2 ACMC Mode (Runtime)

To halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V supply (set to 100 V) and wait a few seconds.
2. Turn off the 12-V input and wait a few seconds.
3. Click Halt on the toolbar to halt the processor.
4. Click  to take the MCU out of real-time mode.
5. Reset the MCU.
6. Close CCS.

10 Bidirectional DC-DC (Full) – Software Overview

10.1 Software Control Flow

This implementation provides control of power flow in both directions and allows on-the-fly change between buck and boost modes based on software command. VMC mode is used for buck and boost conversions. This project combines build level 2 of buck mode project (see Section 6.2) and build level 2 of boost mode project (see Section 9.2) with the addition of appropriate code to achieve fast and smooth transitions between the two modes.

The Bi-dir_Full project makes use of the C-background/ASM-ISR framework. This framework 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 (typically, this includes ADC reading, control calculations, and PWM and DAC updates). Figure 64 shows the general software flow for this project.

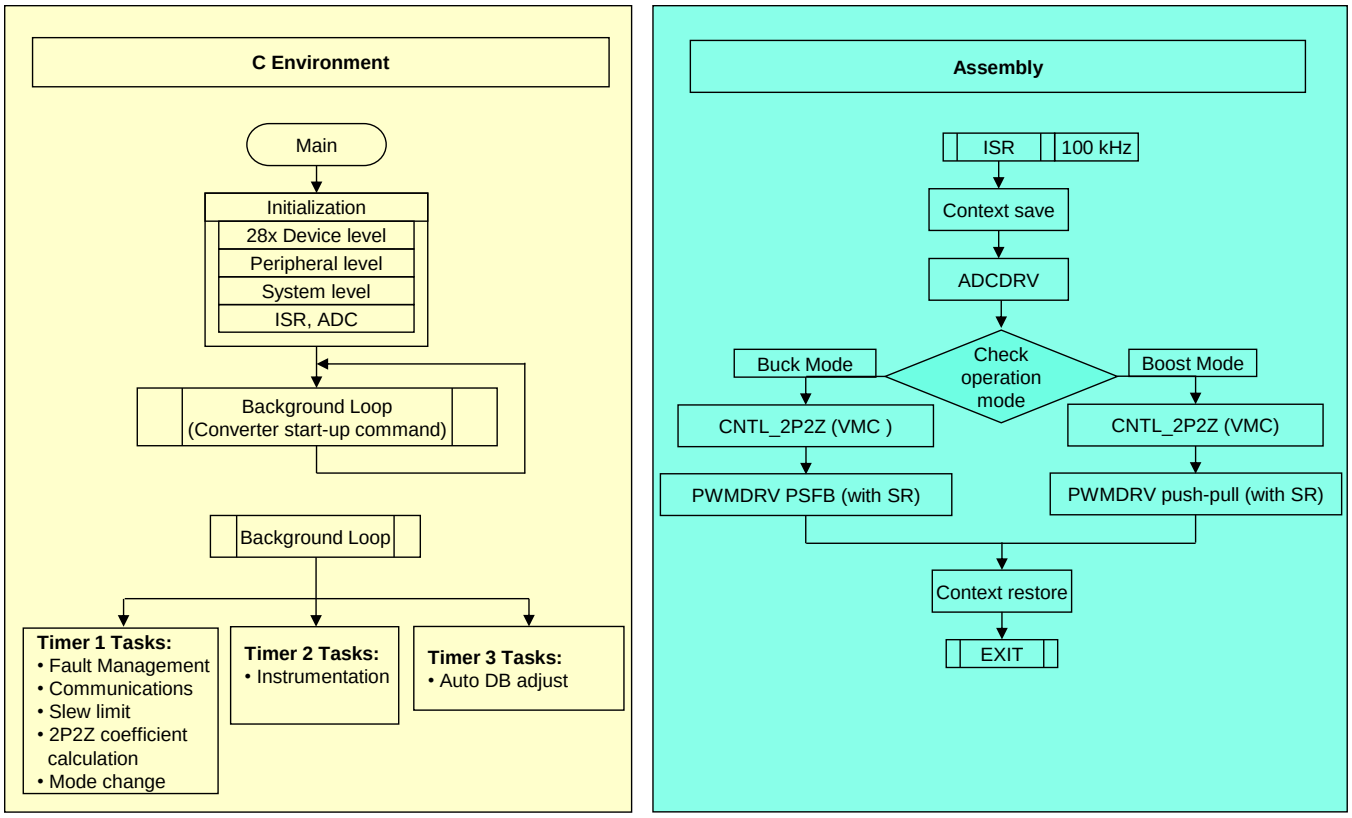


Figure 64. Boost Mode Software Flow

The key framework C files in this project are the following:

- Bi-dir-Main.c – This file initializes, runs, and manages the application.
- Bi-dir-DevInit_F2803x.c – This file initializes and configures the microcontroller once and includes functions such as setting up the clocks, PLL, GPIO, and so forth.

The ISR consists of the following single file:

- Bi-dir-DPL-ISR.asm – This file contains all time-0critical control type code. This file has an initialization section (one-time execute) and a runtime section, which executes at 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. Table 7 lists the C and corresponding assembly module names.

Table 7. Library Modules

C Configure Function	ASM Initialization Macro	ASM Runtime Macro
DAC_Cnf.c		
ADC_SOC_Cnf.c	ADCDRV_4CH_INIT m,n,p,q	ADCDRV_4CH m,n,p,q
PWM_PSFB_VMC_SR_Cnf.c	PWMDRV_PSFB_VMC_SR_INIT m,n,p	PWMDRV_PSFB_VMC_SR m,n,p
PWM_PSFB_VMC_SR_Cnf.c	PWMDRV_BIDIR_BOOST_INIT m,n,p,q	PWMDRV_BIDIR_BOOST m,n,p,q
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

Figure 65 shows the control blocks.

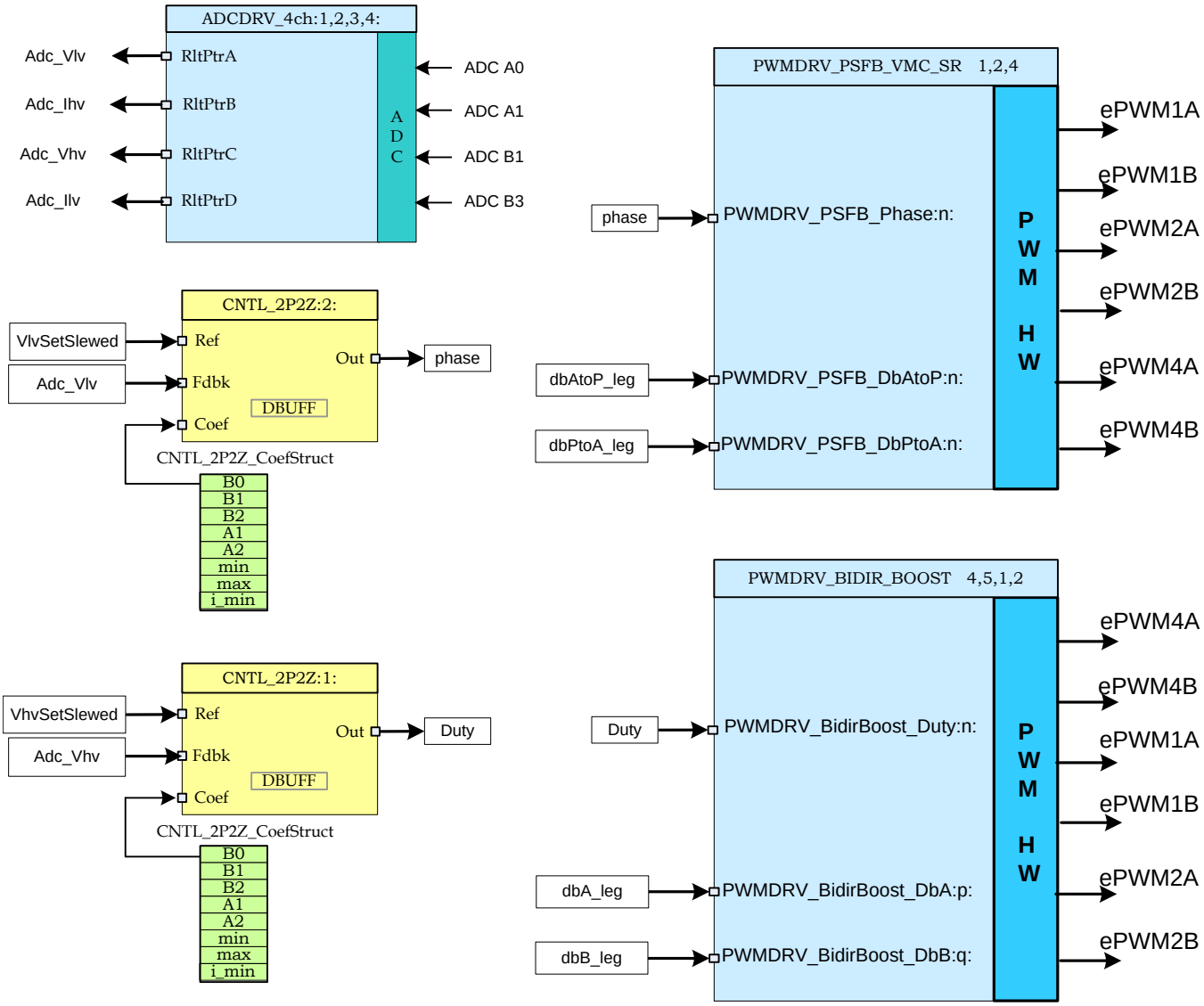


Figure 65. Software Blocks

In Figure 66, dark blue blocks represent hardware modules on the C2000 MCU. Blue blocksare the software drivers for these modules. Yellow blocks are the controller blocks for the control loop. Although a 2-pole 2-zero controller is used here, the controller could be a PI/PID, a 3-pole 3-zero, or any other controller that can be suitably implemented for this application. The modular library structure makes it convenient to visualize and understand the complete system software flow as shown in Figure 66. The structure also allows for easy use, addition, and removal of various functionalities.

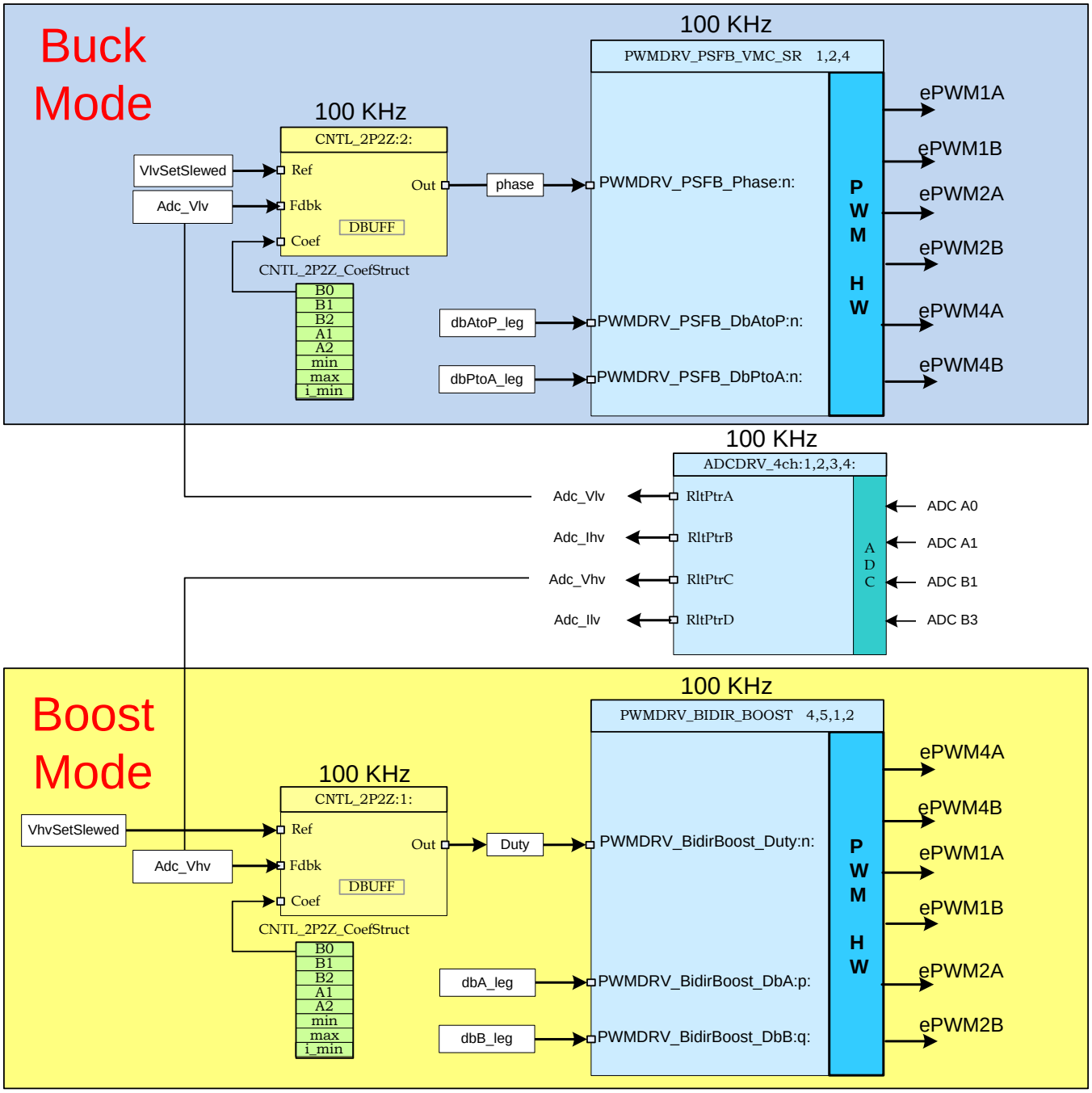


Figure 66. Control Flow

The system is controlled by one voltage feedback loop in both buck and boost modes. When the system is in one mode, only the software blocks necessary for operation in that mode are executed, while blocks needed only for the other mode are not executed. Figure 66 also provides the rate at which control blocks are executed. For example, the voltage controller is executed at a rate of 100 kHz (the same as the PWM switching frequency). The following explains the control implemented in Figure 66.

The sensed output voltage (Adc_Vhv/Adc_Vlv) is compared with a slewed version of the voltage reference command (VhvSetSlewed/VlvSetSlewed) in the voltage controller. The controller output directly controls the duty cycle of PWM signals driving the two push-pull switches in boost mode and phase shift between PWM signals driving the full-bridge switches in buck mode. These PWM driver modules also drive synchronous rectifier switches in both modes.

Mode Transitions

Achieving fast and seamless transitions between the two modes is important. These transitions are achieved by the mode transition code in state-machine task A3. When a buck-to-boost-mode transition is initiated (by setting variable Buck0_Boost1 to 1), the mode transition code executes the buck mode shutdown routine and then brings up the converter in boost mode with the output voltage command (Gui_VhvSet) set to approximately 4 V above the previous voltage on the HV bus (Gui_Vhv). In the same way, when a boost-to-buck-mode transition is initiated (by setting variable Buck0_Boost1 to 0), the mode transition code executes the boost mode shutdown routine and then brings up the converter in buck mode with the output voltage command (Gui_VlvSet) set to approximately 0.5 V above the previous voltage on the LV bus (Gui_Vlv).

Protection

This project has a shutdown mechanism where overcurrent protection is implemented for the transformer high-voltage winding current using on-chip analog comparator 1. Overcurrent protection for LV inductor current is implemented using on-chip analog comparator 2. The reference trip levels are set using the internal 10-bit DACs and fed to the inverting terminals of these comparators. The comparator outputs are configured to generate a one-shot trip action on ePWM1, ePWM2, and ePWM4 whenever the sensed current is greater than the set limit. Overvoltage protection and under-voltage protection for both LV and HV sides are implemented in the software inside the slower state-machine task A1. Whenever an overvoltage or an undervoltage condition is detected, a one-shot trip action is initiated on ePWM1, ePWM2, and ePWM4. The flexibility of the trip mechanism on C2000 devices provides the possibilities for taking different actions on different trip events. In this project, outputs of ePWM1A, ePWM1B, ePWM2A, ePWM2B, ePWM4A, and ePWM4B are driven low immediately to protect the power stage. These outputs are held in this state until a device reset is executed.

11 Bidirectional DC-DC (Full) – Procedure

The main source files, ISR assembly file and the project file for C framework to bring up the system are in the following directory (use the latest version of the software package):

..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Bi_Directional_Full.

Figure 7 shows the hardware setup for these tests.

WARNING

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

The following steps show how to build and run the example in the Bi_Directional_Full software.

Objective

The objective of this project is to evaluate the bidirectional operation of this system and on-the-fly transitions between buck and boost modes of operation. This section explores steps to build and run a project.

Overview

Figure 66 shows the software blocks in this project. The buck and boost mode control blocks are used in the same way as in Section 6.2 and Section 9.2. Two-pole, two-zero controllers are used for the voltage loops. Depending on the control loop requirements of the application, some other controller block like a PI, a 3-pole 3-zero, and so forth may also be used. As seen in the Figure 66, all the software blocks are executed at 100 kHz.

To keep loop tuning simple and without the requirement 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.

These P, I, and D coefficients are Pgainhv, Igainhv, and Dgainhv in boost mode, and Pgainlv, Igainlv, and Dgainlv in buck mode. These P, I, and D coefficients are used in Q26 format. To simplify tuning from CCS watch views, these three coefficients are further scaled to values from 0 to 999 (Pgainhv_Gui, Igainhv_Gui, Dgainhv_Gui, Pgainlv_Gui, Igainlv_Gui, and Dgainlv_Gui). The loop parameters can also be changed directly based on tuned values obtained from external mathematical (MATLAB, Mathcad, and so forth) tools. Loop coefficients can be directly changed using variables b2hv_Gui, b1hv_Gui, b0hv_Gui, a2hv_Gui, a1hv_Gui, b2lv_Gui, b1lv_Gui, b0lv_Gui, a2lv_Gui, and a1lv_Gui in I5Q10 form which are then converted to the five Q26 coefficients for the respective 2P2Z controller.

This project lets you easily evaluate both methods of loop tuning by providing the ability to easily switch between coefficients during execution. Switching between coefficients during execution can be done by changing the pid2p2zhv_GUI and/or pid2p2zlv_GUI variables to 0 or 1 on the watch view from CCS. PID-based loop tuning from the GUI environment has been used for this project. Ensure that reliable coefficient values are programmed in b2hv_Gui, b1hv_Gui, b0hv_Gui, a2hv_Gui, a1hv_Gui, b2lv_Gui, b1lv_Gui, b0lv_Gui, a2lv_Gui, and a1lv_Gui variables before changing the pid2p2zhv_GUI and/or pid2p2zlv_GUI variable to 1.

NOTE: This project does not include valid b2hv_Gui, b1hv_Gui, b0hv_Gui, a2hv_Gui, a1hv_Gui, b2lv_Gui, b1lv_Gui, b0lv_Gui, a2lv_Gui, and a1lv_Gui parameter values. TI recommends using the default P-, I-, D-based loop by keeping pid2p2zhv_GUI = 0 and pid2p2zlv_GUI = 0.

11.1 Procedure

Start CCS and Open a Project

To quickly execute this build, do the following:

1. Ensure that all jumpers on the board are correctly installed or removed as listed in Section 2.
2. Insert the F28035 controlCARD in the 100-pin DIMM connector.
3. Connect a 12-V DC bench power supply between TP10 and TP11 with correct polarity.
4. Connect an isolated 400-V programmable DC power source to the 400-V input connector and a 12-V load to the 12-V connector (ensure this load does not exceed the board ratings).
5. Connect a USB-B to USB-A cable between the PC and the board.

NOTE: Do not turn on any of the power supplies at this time.

6. If this is the first time the board is being tested with JTAG connection, run the program_ftdi.bat file in the xds100v2-FT_Prog_v2.2.zip file to program the FTDI chip on the board.
7. Open Code Composer Studio (CCSv5 or later).
8. Maximize CCS to fill your screen.
9. Close the welcome screen if it opens up.

NOTE: A project contains all the files and build options needed to develop an executable output file (.out), which can be run on the MCU hardware.

10. On the menu bar, click: Project → Import Existing CCS/CCE Eclipse Project.
11. Select the **..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Bi_Directional_Full** directory.
12. Under the Projects tab, ensure Bi-dir_Full is checked.
13. Click Finish.

NOTE: This project uses all the necessary tools (compiler, assembler, and linker) to build the project.

14. In the Project window on the left, click + to the left of Project. See Figure 67.

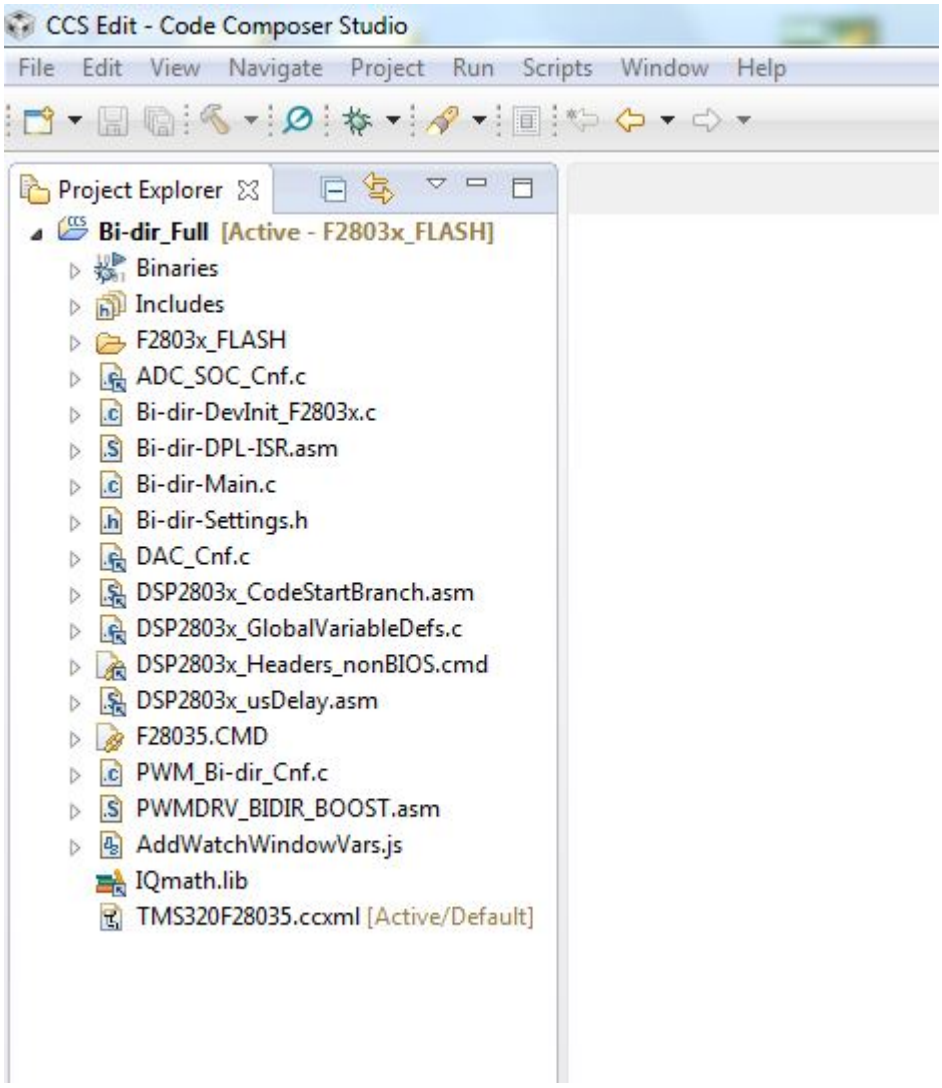


Figure 67. Full Bidirectional Project CCS Project Window

15. Locate initialization code for Build 2 in the main file.
16. Inspect the code. (This code is where all the control blocks are configured, initialized, and connected in the control flow.)

Device Initialization, Main, and ISR Files

NOTE: Do not make any changes to the source files.

1. Double-click Bi-dir-DevInit_F2803x.c.
2. Ensure that the system clock, peripheral clock prescale, and peripheral clock enables have been set up.
3. Ensure that the shared GPIO pins have been configured.
4. Double-click Bi-dir-Main.c.
5. View the call to the DeviceInit() function, the code for different incremental build options, the ISR initialization, and the background for(;;) loop.
6. Locate code in the main file where PWMDRV_PSFB_VMC_SR, PWMDRV_BIDIR_BOOST, CNTL_2P2Z1 (boost), CNTL_2P2Z2 (buck), and ADCDRV_4CH software library blocks are connected

and initialized in the control flow.

7. Open the Bi-dir-DPL-ISR.asm file.
8. Inspect Bi-dir-DPL-ISR.asm for the _DPL_Init and _DPL_ISR sections.

NOTE: The PWM and ADC driver macro instantiation occurs in the _DPL_Init and _DPL_ISR sections for initialization and run time, respectively. You can close the inspected files.

9. Close the files.

Build and Load the Project

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

NOTE: If another option was built previously, right-click on the project name, and click Clean Project.

2. Click Project → Build All.
3. Turn on the 12-V DC bench power supply.
4. Click the Debug button.

NOTE: The Build 1 code should compile and load.

To create an F28035 target configuration or copy the configuration from the F28035.ccxml file, do the following:

1. Right-click F28035.ccxml in the Target Configurations window.
2. Right-click on the name of the device in the Debug window to connect the device.
3. Click Run → Load → Load Program.
4. Navigate to ..\Bi_Directional_Full\F2803x_FLASH\Bi-dir_Full.out to load the code.

NOTE: The CCS Debug icon in the upper right-hand corner must indicate that the program is in the Debug Perspective view.

5. Stop the program at the start of main().

Debug Environment Windows

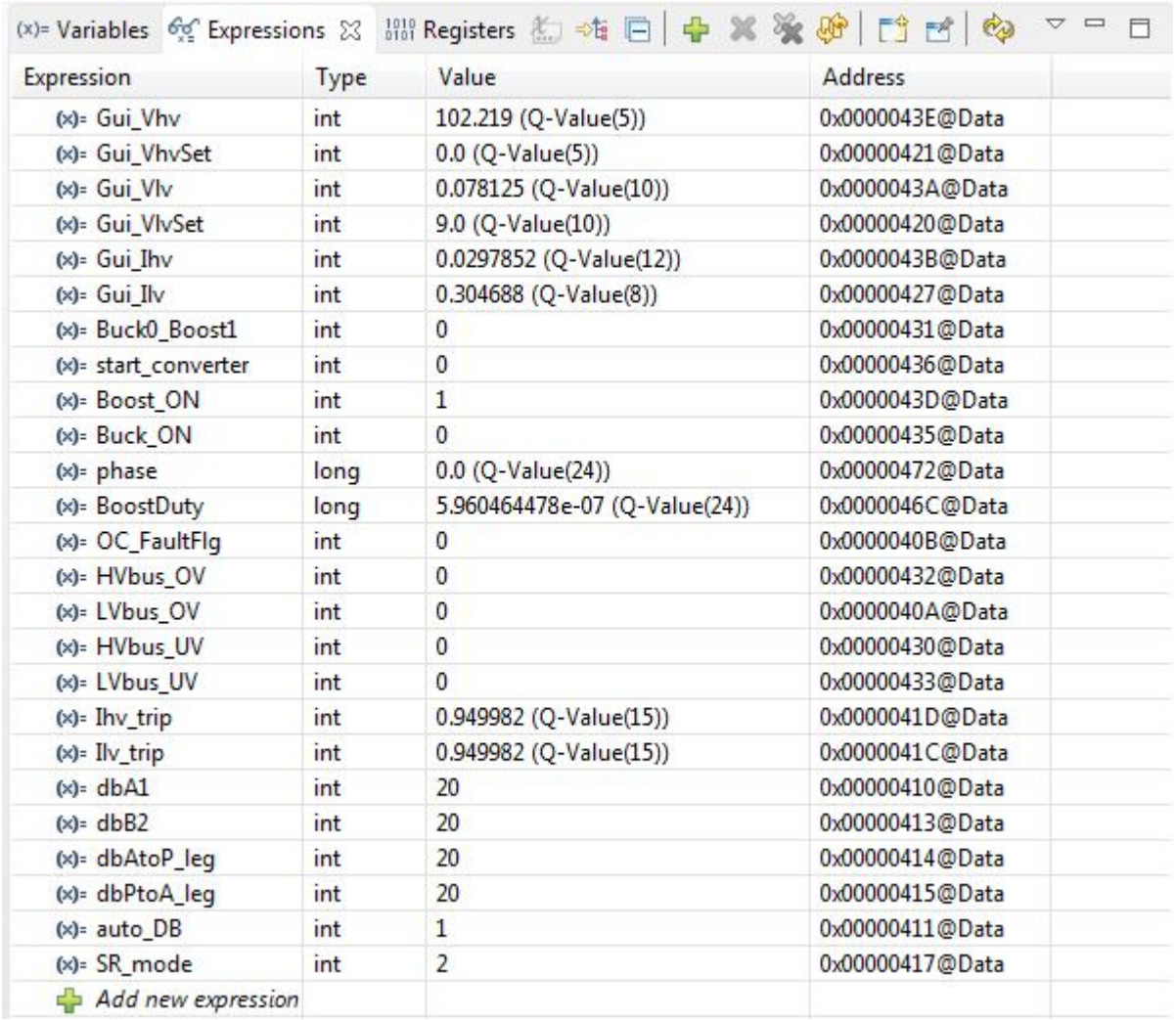
Watch local and global variables while debugging code. You can use the memory views and watch views in CCS. CCS can create time and frequency domain plots. You can view waveforms using graph windows.

1. Click View → Scripting console on the menu bar.
2. Use the scripting console Open File ( )command to open the AddWatchWindowVars.js file from the project directory.

This action populates the Expressions window entries. See Figure 68 to see how the Expressions window should look.

NOTE: Some of the variables have not been initialized at this point in the main code and may contain some garbage values

If set, OC_FaultFlg indicates an overcurrent condition that shuts down the PWM outputs. Similarly, HVbus_OV, LVbus_OV, HVbus_UV, and LVbus_UV indicate overvoltage and undervoltage conditions that shut down the PWM outputs. PWM outputs are held in this state until a device reset. The Ihv_trip variable sets the internal 10-bit DAC reference level for the on-chip comparator 1 (HV winding overcurrent threshold), while Ilv_trip variable sets the internal 10-bit DAC reference level for the on-chip comparator 2 (LV inductor overcurrent threshold). These are Q15 numbers. See Figure 68.



(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	102.219 (Q-Value(5))	0x0000043E@Data
(x)= Gui_VhvSet	int	0.0 (Q-Value(5))	0x00000421@Data
(x)= Gui_Vlv	int	0.078125 (Q-Value(10))	0x0000043A@Data
(x)= Gui_VlvSet	int	9.0 (Q-Value(10))	0x00000420@Data
(x)= Gui_Ihv	int	0.0297852 (Q-Value(12))	0x0000043B@Data
(x)= Gui_Ilv	int	0.304688 (Q-Value(8))	0x00000427@Data
(x)= Buck0_Boost1	int	0	0x00000431@Data
(x)= start_converter	int	0	0x00000436@Data
(x)= Boost_ON	int	1	0x0000043D@Data
(x)= Buck_ON	int	0	0x00000435@Data
(x)= phase	long	0.0 (Q-Value(24))	0x00000472@Data
(x)= BoostDuty	long	5.960464478e-07 (Q-Value(24))	0x0000046C@Data
(x)= OC_FaultFlg	int	0	0x0000040B@Data
(x)= HVbus_OV	int	0	0x00000432@Data
(x)= LVbus_OV	int	0	0x0000040A@Data
(x)= HVbus_UV	int	0	0x00000430@Data
(x)= LVbus_UV	int	0	0x00000433@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x0000041D@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x0000041C@Data
(x)= dbA1	int	20	0x00000410@Data
(x)= dbB2	int	20	0x00000413@Data
(x)= dbAtoP_leg	int	20	0x00000414@Data
(x)= dbPtoA_leg	int	20	0x00000415@Data
(x)= auto_DB	int	1	0x00000411@Data
(x)= SR_mode	int	2	0x00000417@Data
Add new expression			

Figure 68. Full Project Expressions Window

Use Real-Time Emulation

Real-time emulation is a special emulation feature that allows the updating of the windows within CCS at a rate up to a 10 Hz while the MCU is running. This emulation lets graphs and watch views update and lets you change values in watch or memory windows so that those changes affect the behavior of the MCU. This emulation is useful when tuning control law parameters on-the-fly.

To enable real-time mode, do the following:

1. Hover the mouse on the buttons on the horizontal toolbar.
2. Click 

.

NOTE: If a message box appears, select yes to enable debug events. This sets bit 1 (DGBM bit) of status register 1 (ST1) to 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 to update the debugger windows.

When too many windows or variables are open or updating, continuous refresh can cause the refresh frequency to bog down because bandwidth over the emulation link is limited.

To decrease the refresh rate for the expressions window, do the following:

1. Right-click in the Expressions window.
2. Select Continuous Refresh Interval….
3. Change the continuous refresh interval value. (A rate of 1000 ms is usually enough for these exercises.)
4. Click  for the watch view.

Run the Code

1. Click Run on the toolbar.
2. Set the start_converter variable to 0 in the watch view.
3. Set Buck0_Boost1 to 0.
4. Apply an 4-kΩ to 8-kΩ resistive load at the HV (400-V) connector.
5. Apply an appropriate resistive load to the LV (12-V) connector.

NOTE: TI recommends using an isolated DC power supplies to supply 400-V DC and 12-V inputs.

6. Set the 400-V DC supply to output approximately 225 V DC.
7. Turn on this supply.
8. Set the 12-V DC supply to 12-V output.
9. Turn on this supply.
10. Set the start_converter to 1.

NOTE: This action starts DC-DC conversion in buck mode. The Gui_VlvSet command is based on the LV power supply output before setting the start_converter variable to 1 and is set to approximately 0.5 V greater than LV power supply output.

Figure 69 is a watch view that corresponds with the operation of the system in buck mode with the HV voltage of approximately 223 V and a 12-V output with a 10-Ω load. A load of 8 kΩ was also connected at the HV connector.

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= Gui_Vhv	int	223.344 (Q-Value(5))	0x0000043E@Data
(x)= Gui_VhvSet	int	227.625 (Q-Value(5))	0x00000421@Data
(x)= Gui_Vlv	int	12.1172 (Q-Value(10))	0x0000043A@Data
(x)= Gui_VlvSet	int	12.1152 (Q-Value(10))	0x00000420@Data
(x)= Gui_Ihv	int	0.229736 (Q-Value(12))	0x0000043B@Data
(x)= Gui_Ilv	int	0.6875 (Q-Value(8))	0x00000427@Data
(x)= Buck0_Boost1	int	0	0x00000431@Data
(x)= start_converter	int	1	0x00000436@Data
(x)= Boost_ON	int	0	0x0000043D@Data
(x)= Buck_ON	int	1	0x00000435@Data
(x)= phase	long	0.6313103437 (Q-Value(24))	0x00000472@Data
(x)= BoostDuty	long	0.0 (Q-Value(24))	0x0000046C@Data
(x)= OC_FaultFlg	int	0	0x0000040B@Data
(x)= HVbus_OV	int	0	0x00000432@Data
(x)= LVbus_OV	int	0	0x0000040A@Data
(x)= HVbus_UV	int	0	0x00000430@Data
(x)= LVbus_UV	int	0	0x00000433@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x0000041D@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x0000041C@Data
(x)= dbA1	int	32	0x00000410@Data
(x)= dbB2	int	18	0x00000413@Data
(x)= dbAtoP_leg	int	32	0x00000414@Data
(x)= dbPtoA_leg	int	18	0x00000415@Data
(x)= auto_DB	int	1	0x00000411@Data
(x)= SR_mode	int	2	0x00000417@Data
Add new expression			

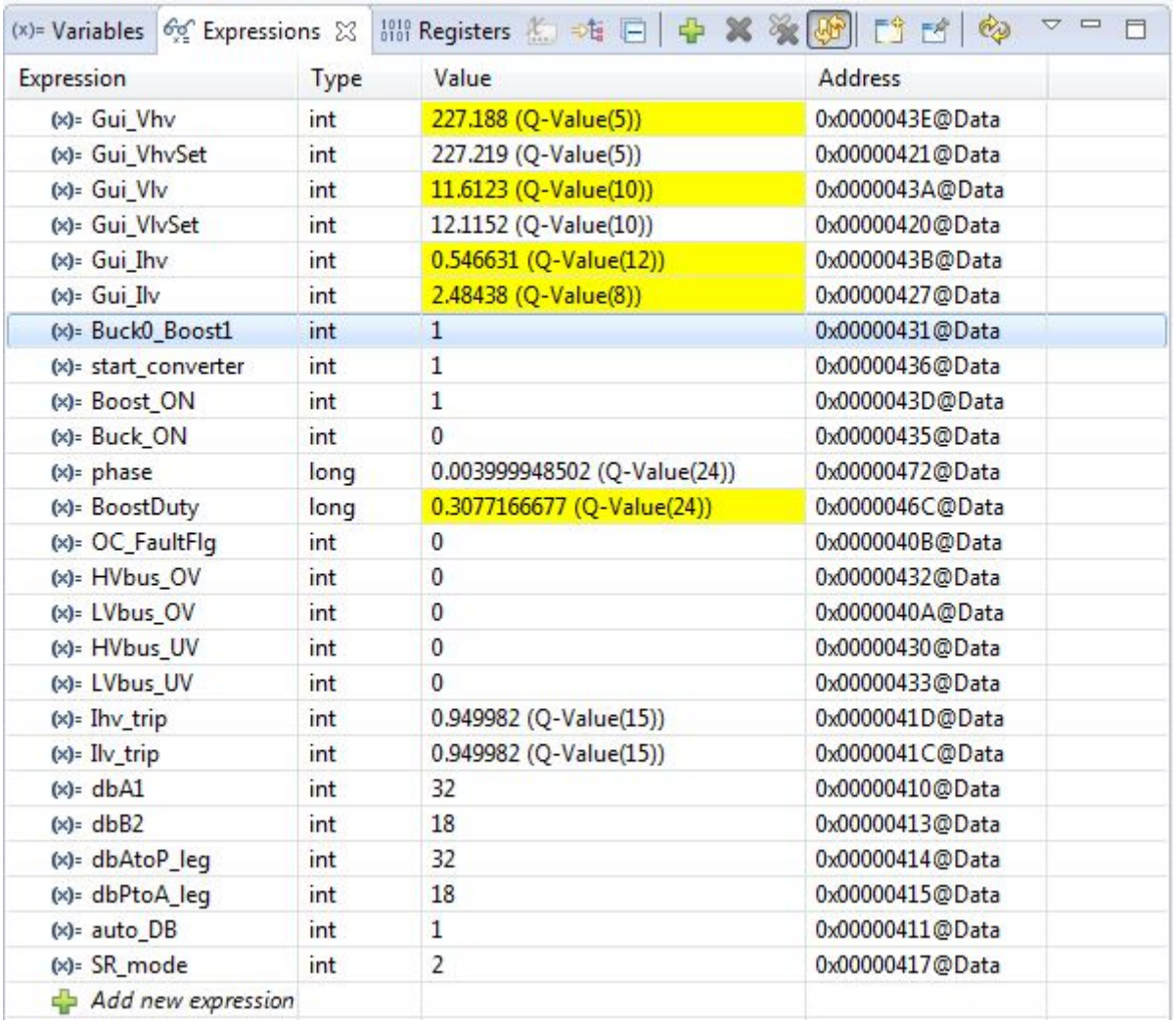
Figure 69. Full Project Expressions Window (Runtime) – Buck Operation

You can change the LV output voltage set command (Gui_VlvSet). Never set this value to less than the LV power supply setting.

NOTE: When operating in buck mode, the Gui_VlvSet command must never be set less than the 12-V DC power supply voltage setting.

11. Set Buck0_Boost1 to 1 to change the operation from buck mode to boost mode. (The Gui_VhvSet command is based on the HV power supply output before the Buck0_Boost1 is set to 1 and is set to approximately 4 V greater than the HV power supply output.)

Figure 70 is a watch view that corresponds to the operation of the system in boost mode with the HV voltage output of approximately 227 V and a 11.6-V input with a 8-kΩ load. A load of 100 Ω was also connected at the LV connector.



Expression	Type	Value	Address
(x)= Gui_Vhv	int	227.188 (Q-Value(5))	0x0000043E@Data
(x)= Gui_VhvSet	int	227.219 (Q-Value(5))	0x00000421@Data
(x)= Gui_Vlv	int	11.6123 (Q-Value(10))	0x0000043A@Data
(x)= Gui_VlvSet	int	12.1152 (Q-Value(10))	0x00000420@Data
(x)= Gui_Ihv	int	0.546631 (Q-Value(12))	0x0000043B@Data
(x)= Gui_Ilv	int	2.48438 (Q-Value(8))	0x00000427@Data
(x)= Buck0_Boost1	int	1	0x00000431@Data
(x)= start_converter	int	1	0x00000436@Data
(x)= Boost_ON	int	1	0x0000043D@Data
(x)= Buck_ON	int	0	0x00000435@Data
(x)= phase	long	0.003999948502 (Q-Value(24))	0x00000472@Data
(x)= BoostDuty	long	0.3077166677 (Q-Value(24))	0x0000046C@Data
(x)= OC_FaultFlg	int	0	0x0000040B@Data
(x)= HVbus_OV	int	0	0x00000432@Data
(x)= LVbus_OV	int	0	0x0000040A@Data
(x)= HVbus_UV	int	0	0x00000430@Data
(x)= LVbus_UV	int	0	0x00000433@Data
(x)= Ihv_trip	int	0.949982 (Q-Value(15))	0x0000041D@Data
(x)= Ilv_trip	int	0.949982 (Q-Value(15))	0x0000041C@Data
(x)= dbA1	int	32	0x00000410@Data
(x)= dbB2	int	18	0x00000413@Data
(x)= dbAtoP_leg	int	32	0x00000414@Data
(x)= dbPtoA_leg	int	18	0x00000415@Data
(x)= auto_DB	int	1	0x00000411@Data
(x)= SR_mode	int	2	0x00000417@Data
Add new expression			

Figure 70. Full Project Expressions Window (Runtime) – Boost Operation

NOTE: You can change the HV output voltage set command (Gui_VhvSet). Never set this value must less than the HV power supply setting.

NOTE: When operating in boost mode, the Gui_VhvSet command must never be set less than the 400-V DC power supply voltage setting.

Always observe the output voltage carefully. Do not let the voltage exceed the capabilities of the board.

You can use an oscilloscope to probe waveforms, like the PWM gate drive signals, input voltage, and current and output voltage.

Take appropriate safety precautions and consider appropriate grounding requirements while probing these high voltages and high currents for this isolated DC-DC converter.

To halt the MCU when in real-time mode, do the following:

1. Turn off the 400-V DC input and wait a few seconds.
2. Click Halt on the toolbar to halt the processor.
3. Click  to take the MCU out of real-time mode.
4. Reset the MCU.

12 References

1. *UCC28950 600-W, Phase-Shifted, Full-Bridge Application Report* (SLUA560B)
2. *600-W, Phase-Shifted, Full-Bridge Converter* (SLUU421A)
3. *Nene, H, "Implementing Advanced Control Strategies for Phase Shifted Full-Bridge DC-DC Converters using Micro-Controllers" PCIM Europe 2011, Nuremberg, Germany.*
4. *Nene, H, "Digital control of a Bi-directional DC-DC Converter for Automotive Applications", in Applied Power Electronics Conference and Exposition, (APEC) 2013, pp. 1360–1365*
5. *HV Phase Shifted Full bridge Developer's Kit* (TMDSHVPSFBKIT)
6. **Bi_dir_Calculations.xls** – a spreadsheet showing key calculations for this project. ..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\Docs
7. **HWdevPkg** – a folder containing various files related to the hardware development of this board:..\controlSUITE\development_kits\BI_DIRECTIONAL_DC_DC_400_12\v1_00_00_00\HWdevPkg

12.1 Trademarks

C2000, Code Composer Studio are trademarks of Texas Instruments.
Mathcad is a registered trademark of PTC Inc.
MATLAB is a registered trademark of The MathWorks, Inc.
All other trademarks are the property of their respective owners.

13 About the Author

HRISHIKESH NENE has been with the C2000 Systems team at TI since 2006. He has worked extensively on design and development of systems and reference solutions for digitally controlled power electronics based systems. For the past several years, his main focus has been on the control of isolated and non-isolated DC-DC converters.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources 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.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265