Two Channel Buck

CCS User Guide

Version 1.1

May 2008

Revised – June 2008

Table of Contents

Introduction

The TwoChannelBuck EVM project provides a straightforward method of learning about digital power with Texas Instruments' F28xxx series digital signal controllers (DSC). In this document you will find lab procedures that explore the EVM's features inside Code Composer Studio. The first lab introduces the software framework and will run the TwoChannelBuck board in an open loop configuration. The next lab shows how the framework can close the loop and teaches how the TI 28xxx DSC can be used to provide sequencing and a soft start-up. Finally, the last lab shows how the control loop is designed and will allow you to tune the transient response via P, I, D parameters.

The TwoChannelBuck EVM project features:

- 10 Amp SyncBuck DC/DC power stages (no heat-sink required) with built-in MOSFET drivers
- Active load for transient response testing (switching timing controlled by ECAP peripheral)
- Voltage input and output measurements via ADC
- Temperature measurement for each channel monitored by ADC
- Current measurement for each channel monitored by ADC
- Over-Current protection and fault flag detection via GPIO
- Closed loop digital control with voltage feedback using F28xxx on-chip ePWM and ADC
- Optional ePWM signals looped back as ADC inputs provides a simple low cost scope to view waveforms in CCS
- 8 x LED indicators are great for diagnostics and fault status
- UART communications header available for Host control
- Host GUI provides a friendly way to control / demo the application, based on open source C# freeware (installed with this package)
- Hardware Developer's package includes Schematics, Bill of materials, Gerber files,…etc (installed with this package)

The TwoChannelBuck project is built within the Texas Instruments' F28xxx System Framework and is configurable to run on any F28xxx target DSC. The project contains C source files which handle the initialization and background tasks, and a time-critical ISR code written in assembly using library modules. Further information on what the System Framework is and how it works can be found in the **SystemFrameworkOverview** guide.

For all information on hardware and software setup, please read **QSG-TwoChannelBuck** noted in the references below.

Caution

During experimentation the power resistors (Load1 – see Hardware overview) may become very HOT if Ch-1 output voltage is left running for extended time (avoid if possible). Please DO NOT TOUCH, allow the resistors to cool down first.

Lab 1: PWM Generation / Open-Loop Control

➢ Objective

The objective of this lab exercise is to demonstrate the topics discussed in this module and control the buck output voltage using simple PWM duty cycle adjustments without feedback. Since this implementation is open-loop and therefore has no requirement for high speed feedback, the ADC will be used to measure various values for instrumentation purposes only. These values will be displayed using CCS. The PWM duty cycle will be adjusted using watch windows. In this lab, you will:

- Control Buck output voltage using simple PWM duty cycle adjustment without feedback
- Use CCS watch window features to conveniently adjust PWM duty cycle

➢ Project Overview

Lab exercises 1, 2, and 3 all use the TwoChannelBuck project. It makes use of the "C-background/ASM-ISR" framework. Futher information on the framework can be found in the **SystemFrameworkOverview** guide. In these lab exercises EPWMs 1 and 2 are used to drive buck stages Channel 1 and Channel 2, respectively.

The key framework files used in this project are:

`TwoChannelBuck-Main.c` – this file is used to initialize, run, and manage the application. This is the "brains" behind the application.

`TwoChannelBuck-DevInit_F28xxx.c` – this file is responsible for a one time initialization and configuration of the device, and includes functions such as setting up the clocks, PLL, GPIO, etc. This file is specific to a family of devices (ex. F280x, F2833x, etc).

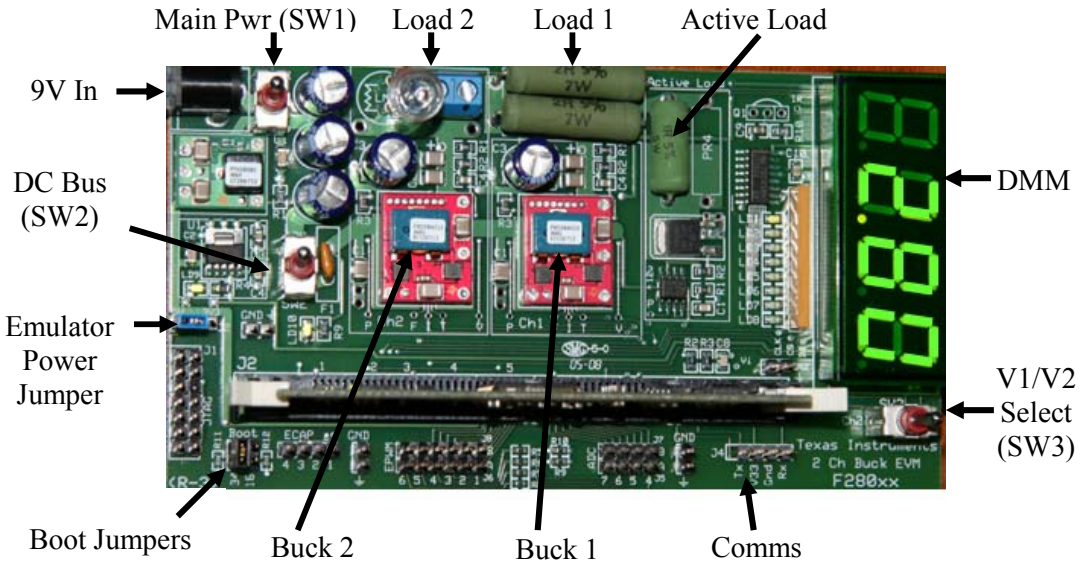
`TwoChannelBuck-ISR.asm` – this file contains all time critical "control type" code. This file has an initialization section that is executed one time by the C-callable assembly subroutine `_ISR_Init`. The file also contains the `_ISR_Run` routine which executes at the same rate as the PWM timebase which is used to trigger it.

`TwoChannelBuck-Settings.h` – this file is used to set global definitions for the project (ie. build options). Note that it is linked to both `TwoChannelBuck-Main.c` and `TwoChannelBuck-ISR.asm`.

The Power Library functions (modules) are "called" from this framework. Library modules may have both a C and an assembly component. In this lab exercise, the following C and corresponding assembly modules are used:

C configuration function	ASM initialization macro	ASM Run time macro
`BuckSingle_CNF()`	`BuckSingle_DRV_INIT n`	`BuckSingle_DRV n`
`ADC_CascSeqCNF()`	*none*	*none*

The TwoChannelBuck EVM consists of two identical buck power stages. The input bus voltage for both stages is 9V. Shown below is a diagram of the Power EVM and some key features.

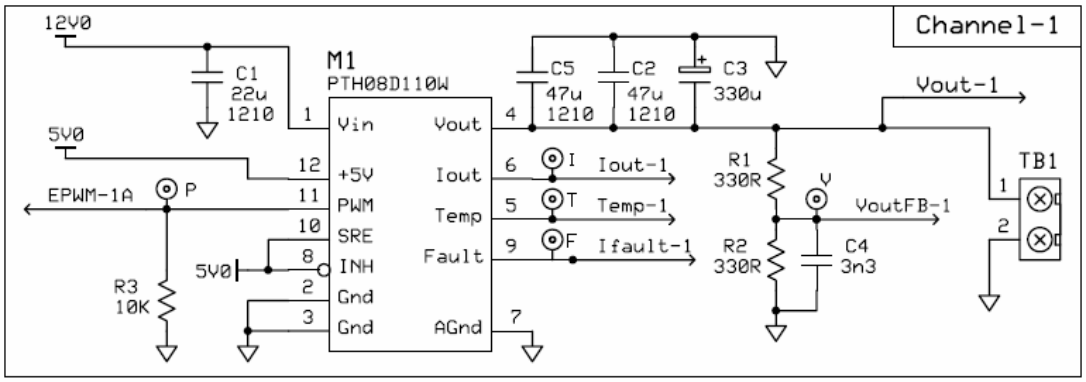


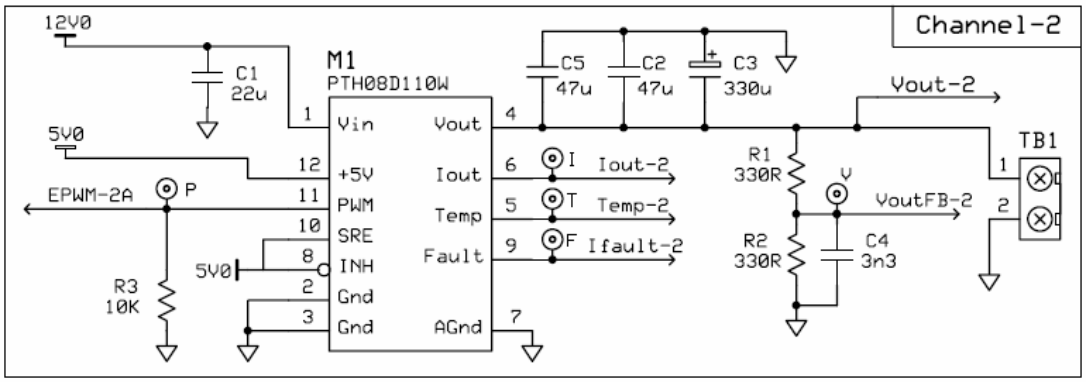
9V In	DC power supply from plug pack (12V supply may be used as well)
Main Pwr	SW1 - Master power switch for entire EVM
DC Bus	SW2 - Power switch for Vin to buck stages only. When off the F2808 DIMM controller card will still operate (next to the DC bus switch is a resettable fuse)
Buck 1, 2	Buck power stage modules with temperature/current measurement and over current protection
Load 1	Buck converter output for Channel 1 is tied to two 2Ω resistors in parallel. If desired, load 1 can be desoldered and a terminal block as in Channel 2 can be reinstalled.

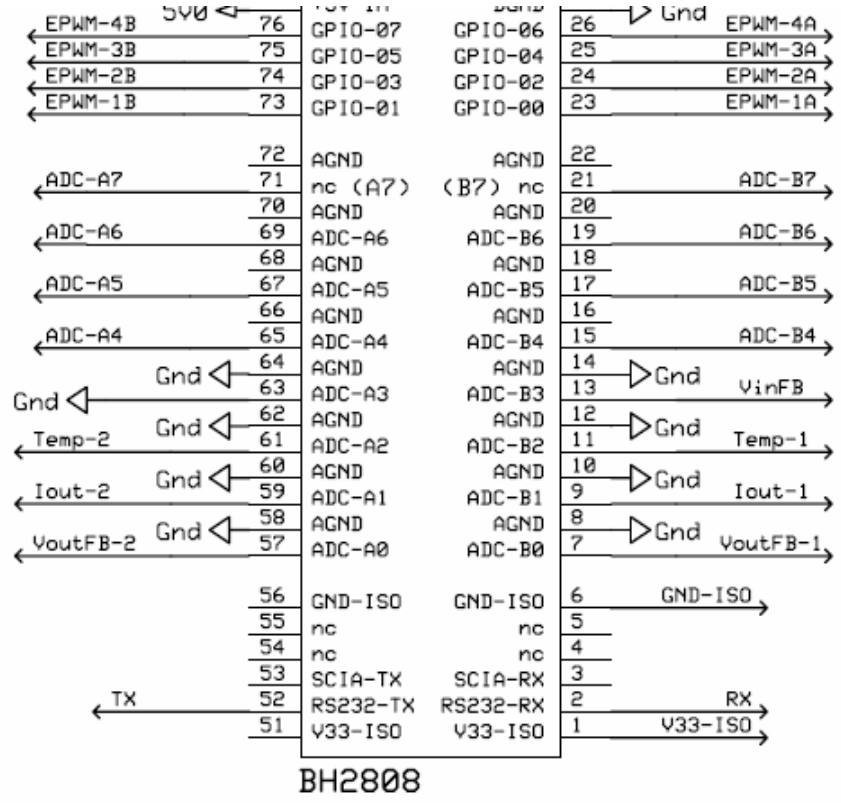
Load 2	Load terminals and/or buck converter output - next to each terminal block is a light bulb or "visual" load (these draw approx 250 mA hot)
Active Load	Software controlled switched load (connected to output of buck 1 only)
DMM	Digital Multi-Meter (has a range of 0~20V, with resolution of 10 mV and is used to measure output voltage of buck conterters)
V1/V2 Select	SW3 - selects the display of the DMM between output voltage of buck 1 and 2
Comms	Serial communications UART (connects to an optional GUI, not used in lab exercises)
Emulator Power Jumper	Sets the voltage sent to power the emulator. No jumper means no power will be sent to an emulator, a jumper at "5V" will power the emulator with 5V, and a jumper at "3V3" will power the emulator with 3.3V.
Boot Jumpers	Controls how the F280x will boot. • If no jumpers are placed the target will boot from flash. • If a jumper is placed at "34", the target will boot from the SCI • If a jumper is placed at "29" and "34", the target will boot from RAM

The key signal connections between the F2808 Digital Signal Controller and the 2 buck stages are listed in the table below. For reference a portion of the schematic is also given.

Signal Name	Description	Connection to target DSC
EPWM-1A	PWM Duty control signal for buck stage 1	GPIO-00
EPWM-2A	PWM Duty control signal for buck stage 2	GPIO-02
VoutFB-1	Voltage feedback for buck stage 1	ADC-B0
VoutFB-2	Voltage feedback for buck stage 2	ADC-A0
Iout-1	Current monitor / measurement buck stage 1	ADC-B1
Iout-2	Current monitor / measurement buck stage 2	ADC-A1
Temp-1	Temperature monitor / measurement buck stage 1	ADC-B2
Temp-2	Temperature monitor / measurement buck stage 2	ADC-A2
Ifault-1	Over-Current flag, digital output from buck stage 1	GPIO-12
Ifault-2	Over-Current flag, digital output from buck stage 2	GPIO-13

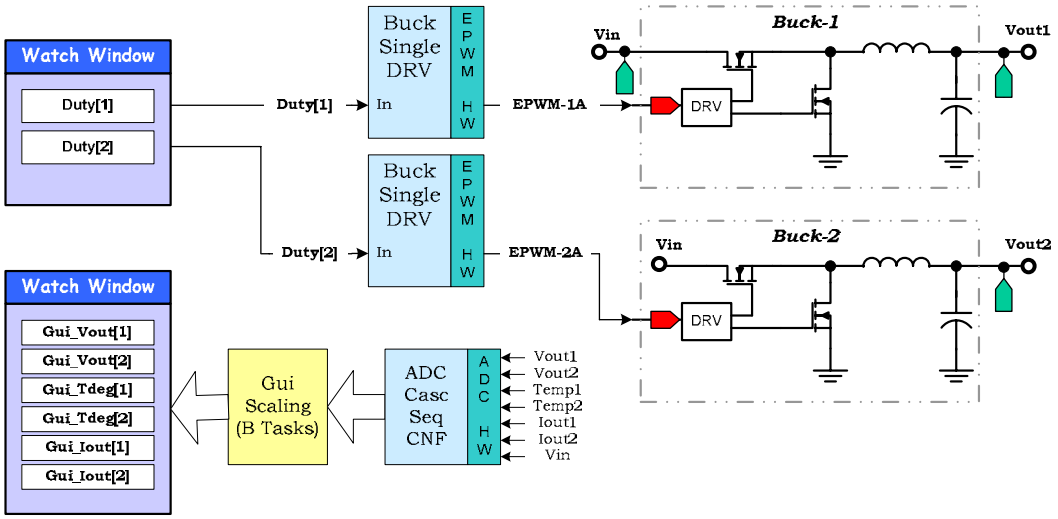

Channel-1
12V0
C1
22u
1210
M1
PTH08D110W
Vin
Vout
+5V
Iout
PWM
Temp
SRE
Fault
INH
Gnd
Gnd
AGnd
C5
47u
1210
C2
47u
1210
C3
330u
Vout-1
5V0
EPWM-1A
P
R3
10K
I
Iout-1
T
Temp-1
F
Ifault-1
R1
330R
V
VoutFB-1
R2
330R
C4
3n3
TB1


Channel-2
12V0
C1
22u
M1
PTH08D110W
Vin
Vout
+5V
Iout
PWM
Temp
SRE
Fault
INH
Gnd
Gnd
AGnd
C5
47u
C2
47u
C3
330u
Vout-2
5V0
EPWM-2A
P
R3
10K
I
Iout-2
T
Temp-2
F
Ifault-2
R1
330R
V
VoutFB-2
R2
330R
C4
3n3
TB1


EPWM-4B
EPWM-3B
EPWM-2B
EPWM-1B
GPIO-07
GPIO-05
GPIO-03
GPIO-01
GPIO-06
GPIO-04
GPIO-02
GPIO-00
EPWM-4A
EPWM-3A
EPWM-2A
EPWM-1A
AGND
ADC-A7
nc (A7)
(B7) nc
ADC-B7
ADC-A6
ADC-A5
ADC-A4
ADC-A3
ADC-A2
ADC-A1
ADC-A0
ADC-B6
ADC-B5
ADC-B4
ADC-B3
ADC-B2
ADC-B1
ADC-B0
Gnd
VinFB
Temp-2
Temp-1
Iout-2
Iout-1
VoutFB-2
VoutFB-1
GND-ISO
nc
SCIA-TX
SCIA-RX
TX
RS232-TX
RS232-RX
RX
V33-ISO
BH2808

➢ Lab Exercise Overview

The software in Lab1 has been configured to independently adjust the duty cycle of EPWM-1A and EPWM-2A. "Net" variable names `Duty[1]` and `Duty[2]` have been declared and "connected" to the inputs of `BuckSingle_DRV` macro. Using the watch window Duty[1] and Duty[2] can be directly adjusted. Below is the system diagram for Lab1.



In the TwoChannelBuck project the assembly ISR `_ISR_Run` routine is triggered by EPWM1. This is where the `BuckSingle_DRV` macros are executed. Therefore, the PWM update rate is equal to the PWM frequency. Since this system is running open-loop, there is not a requirement for high speed feedback. As a result, the ADC function `ADC_CascSeqCNF()` is called in the C background code during initialization, and the ADC measured values are only used for instrumentation purposes. The update rate can be much slower with no need to be synchronized to the PWM or ISR. The ADC values are read directly from the ADC result registers (`AdcMirror.ADCRESULTn`) by the background C code.

A task state-machine has been implemented as part of the background code. Tasks are arranged in groups (A1, A2, A3…, B1, B2, B3…, C1, C2, C3…). Each group is executed according to 3 CPU timers which are configured with periods of 1 ms, 5 ms, and 7.5 ms respectively. Within each group (e.g. "B") each task is run in a "round-robin" manner. For example, group B executes every 5 ms and there are 3 tasks in group B. Therefore, each "B task" will execute once every 15 ms.

In this lab, "A tasks" are used to provide channel enabling, soft start and sequencing for the EVM. In the code the function SerialHostComms() is used to allow an external GUI to edit the project's variables. This GUI is not used for these labs, however more information on its use can be found in the **QSG-TwoChannelBuck-GUI** guide.

System dashboard measurements are conveniently done by group "B tasks" (i.e. B1 – voltage measurement, B2 – temperature measurement, and B3 – current measurement). These tasks convert user inputs/outputs (designated by the Gui_ prefix) from/to a Q format variable and put it into a Q15 format that the framework uses. This is in order for the user to use real units (ie V/ms,

amps) while still saving precision. The details of each conversion are found in the excel spreadsheet TwoChannelBuck-Calculations.

In this project "C tasks" are used for slow-tasks such as coefficient update, active load control, and LED control.

- **Procedure**

Open a CCS Project

1. Turn on the power (SW1) to the 2-channel buck EVM. Open Code Composer Studio and maximize it to fill your screen.

2. Open the project `TwoChannelBuck.pjt` by clicking:

   `Project → Open…`

   and look in `C:\TI_F28xxx_SysSW\TwoChannelBuck\`.

3. This project can be configured to create code for multiple target controlCARDs and can be ran in either flash or RAM. For the labs we will run the F280x in RAM. In the project window, right-click on TwoChannelBuck.pjt and select "Configurations…". In the new window select F280x_RAM, click "Set Active" and then click "Done".

4. Code Composer Studio can automatically load the output file after a successful build. On the menu bar click: `Option → Customize…` and select the `"Program/Project/CIO"` tab, check `"Load Program After Build"`.

   Also, Code Composer Studio can automatically connect to the target when started. Select the `"Debug Properties"` tab, check `"Connect to the target at startup"`, then click `OK`.

Device Initialization, Main, and ISR Files

5. Open and inspect `TwoChannelBuck-DevInit.c` by double clicking on the filename in the project window. Confirm that GPIO00 and GPIO02 are configured to be PWM outputs.

6. Open `TwoChannelBuck-Settings.h`. For this lab we will use incremental build option 1. Please edit the file so that INCR_BUILD is defined as 1

   (i.e. `#define INCR_BUILD 1`)

   then save the file by clicking:

   `File → Save`

Note: This should be the **only** change made to source files for this lab.

7. Open and inspect TwoChannelBuck-Main.c. Notice the code enabled when INCR_BUILD is 1. This section of code is shown here for convenience. Additional comments have been added in *italics.* Note that the run-time macros are executed at the PWM rate of 400 kHz.

```
//---------------------------------------------------------------------
#if (INCR_BUILD == 1)  // Open loop - Channels 1,2
//---------------------------------------------------------------------
#define         prd            250     // Period count = 400 KHz @ 100 MHz
#define         NumActvCh      2       // Number of Active Channels

// "Raw" (R) ADC measurement name defines
#define         VoutR1   AdcMirror.ADCRESULT0  //
#define         VoutR2   AdcMirror.ADCRESULT1  //
#define         IoutR1   AdcMirror.ADCRESULT2  //
#define         IoutR2   AdcMirror.ADCRESULT3  //
#define         TdegR1   AdcMirror.ADCRESULT4  //
#define         TdegR2   AdcMirror.ADCRESULT5  //
#define         VinR     AdcMirror.ADCRESULT6  //
```

The ChSel array is used as input by function ADC_CascSeqCNF. These values will be used by "B" tasks for dashboard calculations, and shown in the Watchwindow.

```
// Channel Selection for Cascaded Sequencer
ChSel[0] = 8;           // B0 - Vout1
ChSel[1] = 0;           // A0 - Vout2
ChSel[2] = 9;           // B1 - Iout1
ChSel[3] = 1;           // A1 - Iout2
ChSel[4] = 10;          // B2 - Temperature-1
ChSel[5] = 2;           // A2 - Temperature-2
ChSel[6] = 11;          // B3 - Vin
```

The 3 configuration functions below are part of the Power Library.

```
BuckSingle_CNF(1, prd, 1, 0);   // ePWM1, Period=prd, Master, Phase=Don't Care
BuckSingle_CNF(2, prd, 0, 0);   // ePWM2, Period=prd, Slave, Phase=0
ADC_CascSeqCNF(ChSel, 2, 7, 1); // ACQPS=2, #Conv=7, Mode=Continuous (1)
EPwm1Regs.CMPB = 2;             // ISR trigger point
ISR_Init();                     // ASM ISR init
```

Duty[1] and Duty[2] variables will directly control the buck duty cycle. The WatchWindow will be used to quickly change these values.

```
Duty[1] = 0x0;
Duty[2] = 0x0;

// Lib Module connection to "nets"
//--------------------------------------
// BuckSingle_DRV connections
Buck_In1 = &Duty[1];
Buck_In2 = &Duty[2];

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

8. Open and inspect `TwoChannelBuck-ISR.asm`. Notice the `_ISR_Init` and `_ISR_Run` sections. This is where the PWM driver macro instantiation is done for initialization and run-time, respectively. The code is shown below for convenience. In Lab1, incremental build option IB1 is used.

```
.if(INCR_BUILD = 1) ; Init time
       BuckSingle_DRV_INIT    1       ; EPWM1A
       BuckSingle_DRV_INIT    2       ; EPWM2A
.endif

.if(INCR_BUILD = 1) ; Run time
       BuckSingle_DRV 1               ; EPWM1A
       BuckSingle_DRV 2               ; EPWM2A
.endif
```

9. Close the inspected files if desired.

Build and Load the Project

10. Click the "`Rebuild All`" button and watch the tools run in the build window. The output file should automatically load.

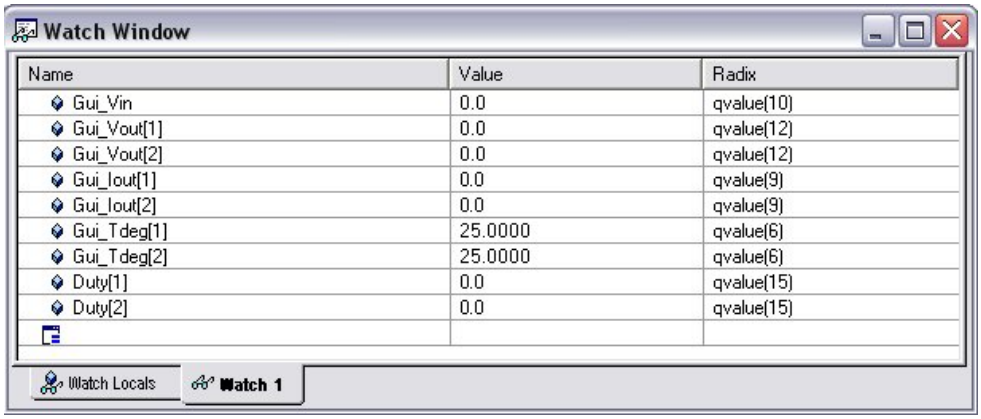
11. Under `Debug` on the menu bar click "`Reset CPU`", "`Restart`", and then "`Go Main`". You should now be at the start of `Main()`.

Setup Watch Window

12. Open the *watch window* to view the variables used in the project.

    Click: `View` → `Watch Window` on the menu bar.

    Click the "Watch 1" tab at the bottom of the watch window. In the empty box in the "Name" column, type the symbol names from the following screen capture and be sure to modify the "Radix" as needed.



The following table gives a description for the variable names:

Variable	Description
Gui_Vin	Voltage input measurement (i.e. DC bus) to each buck power stage
Gui_Vout	Voltage output of each channel, 3 element array, zeroth element not used
Gui_Iout	Current output of each channel, 3 element array, zeroth element not used
Gui_Tdeg	Temperature of each power module, 3 element array, zeroth element not used
Duty[1]	Duty cycle input to BuckSingle_DRV 1
Duty[2]	Duty cycle input to BuckSingle_DRV 2

Save the Workspace

1. Save the current workspace by naming it Lab1.wks and clicking:

   File → Workspace → Save Workspace As…

   and saving in C:\TI_28xxx_SysSW\TwoChannelBuck\

Run the Code – TwoChannelBuck

13. Enable real-time mode by selecting:

    Debug → Real-time Mode

14. A message box *may* appear. If so, select YES to enable debug events. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a "0". The DGBM is the debug enable mask bit. When the DGBM bit is set to "0", memory and register values can be passed to the host processor for updating the debugger windows.

15. Check to see if the windows are set to continuously refresh. Click:

    View → Real-time Refresh Options…

    and check "Global Continuous Refresh".

16. Run the code by using the <F5> key, or using the Run button on the vertical toolbar, or using Debug → Run on the menu bar.

17. Note that in the watch window all values should be ~ zero, except for temperature, which should be approximately equal to room temperature of 25° C.

18. Turn on the 9-volt DC bus (SW2) on the 2-channel buck EVM and observe variable `Gui_Vin` in the watch-window. It should now be approximately 9V.

19. Increase the value of Duty[1] to approximately 0.11. Power stage buck 1 module output voltage should be approximately 1V on the DMM. Be sure that SW3 on the EVM is positioned to select Ch1. With two 2Ω load resistors placed in parallel the equivalent resistance across terminal 1 is 1Ω. The open-loop voltage for Channel 1 is approximately given by:

Voltage	Value of Duty[1]
1V	0.11
2V	0.22
3V	0.33

20. Try the same adjustment on Duty[2]. Be sure SW3 on the EVM is positioned to select Ch2. Note that Channel 2 buck is lightly loaded with a lamp (2~3Ω) and hence a slightly lower Duty[2] value will give the same output voltage as in the Ch1 case. This is expected since we are running the controller in an open loop.

21. Of general interest – during duty/voltage adjustments observe the various watch window variables such as voltage, current and temperature. Gui_Vout should reflect approximately the same value as the DMM display. The current measurement is not very precise as it is designed to measure a range up to 15A. Hence at low current levels accuracy will be quite poor. Temperature should track quite well and the channel supplying the most power will show an observable temperature increase.

22. Turn off the 9-volt DC bus (SW2) on the 2-channel buck EVM. ***(Do not turn off the main power SW1).***

23. Fully halt the DSP in real-time mode. First, halt the processor by using Shift <F5>, or using the `Halt` button on the vertical toolbar, or by using `Debug → Halt`. Then click `Debug → Real-time Mode` and uncheck the `"Real-time mode"` to take the DSP out of real-time mode.

24. ***<u>Do Not Close Code Composer Studio</u> or it will be necessary to setup the project again for the next lab exercise!***

End of Exercise

Lab 2: Closed-Loop Control

➢ Objective

The objective of this lab exercise is to demonstrate the topics discussed in this module and regulate the output voltage of a buck power stage using closed-loop feedback control realized in the form of a software coded loop. Soft-start and shut-down management will be explored using the CCS watch window. ADC management for high-speed feedback and slow instrumentation will be utilized. In this lab you will:

- Regulate buck outputs using Voltage Mode Control (VMC) with closed-loop feedback
- Use soft-start and sequencing fuctions to ensure an "orderly" voltage ramp-up/down
- Adjust the soft-start profile and target voltage using the CCS watch window

➢ Project Overview

The following Power Library modules will be used in this lab exercise. (Note: these are the same library modules used in Lab1 exercise with the addition of more library modules).

C configure function	ASM initialization macro	ASM Run time macro
`BuckSingle_CNF()`	`BuckSingleHR_DRV_INIT n`	`BuckSingleHR_DRV n`
`ADC_DualSeqCNF()`	`ADC_NchDRV_INIT n`	`ADC_NchDRV n`
None	`ControlLaw_2P2Z_INIT n`	`ControlLaw_2P2Z n`
None	`DataLogTST_INIT n`	`DataLogTST n`

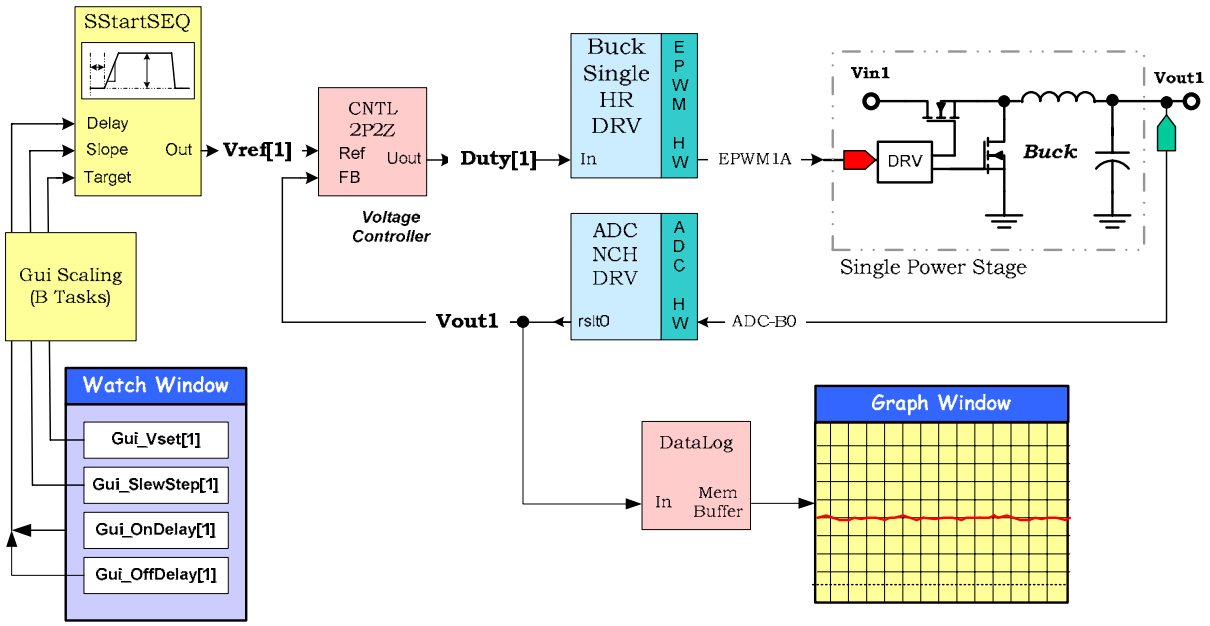
Below is a description and notes for the Power Library modules used in this lab exercise.

`BuckSingleHR_DRV`	This is the high resolution PWM version of BuckSingle used in Lab2. The C configure function (`BuckSingle_CNF`) is applicable for both high-resolution and non-high-resolution versions of macro.
`ADC_NchDRV`	Reads 1st N ADC result registers every PWM cycle and stores to N consecutive memory locations accessible by C. In Lab2, N=1 (i.e. a single voltage is measured as feedback).
`ControlLaw_2P2Z`	This is a 2nd order compensator realized from an IIR filter structure. The 5 coefficients needed for this function are declared in the C background loop as an array of longs. This function is independent of any peripherals and therefore does not require a CNF function call.

`DataLogTST`	Data logging function with time-stamp trigger input. Although not needed in the application itself, it provides a convenient way to visualize the output voltage in a CCS graph window. In Lab3 the data logger will be useful in displaying an output voltage transient.

➢ Lab Exercise Overview

The software in Lab2 has been configured to provide closed-loop voltage control for Channel 1 and Channel 2 of the buck EVM. Additionally, datalogging of the Channel 1 output voltage can be displayed in a CCS graph window. Below is the system diagram for Channel 1 in Lab2.



Note that the system diagram for Channel 2 would look exactly the same except for two key changes:

- The index of all arrayed variables will be 2 instead of 1
- Channel 2 is not configured in software to have a datalogger and therefore it can not display a transient graph

The closed-loop consists of only three modules – `ADC_NchDRV`, `ControlLaw_2P2Z`, and `BuckSingleHR_DRV`. When the code is runing these modules execute as in-line code (no decision making) within the `ISR_Run` routine which is triggered at the PWM rate. To ensure proper operation, Vref is kept at zero until a request is received to enable the output voltage. It is important for a power supply to have a proper start-up and shut-down routine. This is managed by the soft-start and sequencing code which executes in the main background C code

TwoChannelBuck-Main.c. This code ensures that Vref can never have a step change, as direct modification of Vref is not allowed. Vref can only be adjusted indirectly via a target value request. This value will be reached at a given slew-rate. The slew-rate is programmable with delay-on and delay-off time parameters which are useful for staggered sequencing of multiple voltage rails.

In Lab2, the target voltage, slew-rate and delay-on/off parameters are conveniently modified via a watch window. The soft-start and sequencing code is "scaleable" and can manage multiple voltage rails, for example 2, 3…10 or more Vrefs. The interface to this code is via several integer arrays and integer flags. The array index "n" is used to designate the channel number (i.e. n=1 for channel 1, n=2 for channel 2…etc.) Although in C an index of n=0 is valid, it is not used here. Below is a summary of the arrays and their usage.

`Gui_Vset[n]`	Desired output target voltage in Q12 format
`ChannelEnable[n]`	Enables (allows) voltage output to reach target value e.g. `ChannelEnable[1]=1`, turn channel 1 "on" e.g. `ChannelEnable[1]=0`, turn channel 1 "off"
`Gui_SlewStep[n]`	Step size or rate at which the target voltage is ramped to. Displayed in Q13 with units of V/ms
`Gui_OnDelay[n]`	Delay time to turn *on* from the "global" start command (`StartUp=1`) Displayed in Q0 format with units of ms.
`Gui_OffDelay[n]`	Delay time to turn *off* from the "global" stop command (`StartUp=0`) Displayed in Q0 format with units of ms.
`StartUp`	Global turn on/off command. Used to synchronize/sequence all channels e.g. `StartUp=1`, global turn *on* command e.g. `StartUp=0`, global turn *off* command

- **Procedure**

Open a CCS Project

1. Code Composer Studio should still be running from the previous lab exercise. If it is not, then it will be necessary to setup the debug environment and project from the previous lab exercise. This can be done by simply loading the last saved workspace. Click:

   `File → Workspace → Load Workspace`

   In the new window browse open:

   `C:\TI_28xxx_SysSW\TwoChannelBuck\Lab1.wks`

Device Initialization, Main, and ISR Files

2. Open `TwoChannelBuck-Settings.h`. For this lab we will use incremental build option 2. Please edit the file so that INCR_BUILD is defined as 2

   (i.e. `#define INCR_BUILD 2`).

   then save the file by clicking:

   `File → Save`

Note: This should be the **only** change made to source files for this lab.

---

3. Open and inspect `TwoChannelBuck-Main.c` by double clicking on the filename in the project window. Notice the code enabled when INCR_BUILD is 2. This section of code is shown here for convenience. Comments have been added in *italics*.

```
//---------------------------------------------------------------------------
#if (INCR_BUILD == 2) // Closed Loop CH1 & CH2, with SoftStart using separate lib
//  blocks
//---------------------------------------------------------------------------
//#define      prd            500     // Period count = 200 KHz @ 100 MHz
#define        prd            333     // Period count = 300 KHz @ 100 MHz
//#define      prd            250     // Period count = 400 KHz @ 100 MHz
#define        NumActvCh      2       // Number of Active Channels

// "Raw" (R) ADC measurement name defines, ADC result registers are mirrored
// in single cycle read memory for faster access.
#define        VoutR1   AdcMirror.ADCRESULT0
#define        VoutR2   AdcMirror.ADCRESULT1

#define        IoutR1   AdcMirror.ADCRESULT8
#define        IoutR2   AdcMirror.ADCRESULT9
#define        TdegR1   AdcMirror.ADCRESULT10
#define        TdegR2   AdcMirror.ADCRESULT11
#define        VinR     AdcMirror.ADCRESULT12

#define        tPHS2    37             // Phase offset from Master EPWM
```

ADC Sequencer 1 – Vout[1] and Vout[2] used every PWM cycle

```
        // ADC Channel Selection for Sequencer-1
        ChSel[0] = 8;                 // B0 – VoutR1
        ChSel[1] = 0;                 // A0 – VoutR2
```

ADC Sequencer 2 – Instrumentation only, round robin scheme

```
        // ADC Channel Selection for Sequencer-2
        ChSel[8] = 9;                 // B1 – IoutR1
        ChSel[9] = 1;                 // A1 – IoutR2
        ChSel[10] = 10;               // B2 – TdegR1
        ChSel[11] = 2;                // A2 – TdegR2
        ChSel[12] = 11;               // B3 - Vin
```

Soft-Start parameters for channel 1 & 2

```
        Gui_OnDelay[1] = 0;
        Gui_OnDelay[2] = 1000;        // 1 second delay

        Gui_OffDelay[1] = 1000;       // 1 second delay
        Gui_OffDelay[2] = 0;

        Gui_Vset[1] = 7413;           // 1.8 V
        Gui_Vset[2] = 13557;          // 3.3 V
```

```
        Gui_SlewStep[1] = 2048;              // 0.25 V/ms
        Gui_SlewStep[2] = 2048;              // 0.25 V/ms
```

Used for Scope feature via Graph window (Setup for CH1)

```
        DataLogTrigger = 980000;
        ScopeGain = 1;
        ScopeACmode = 0;                     // DC mode initially
```

PWM and ADC configure functions

```
        BuckSingle_CNF(1, prd, 1, 0);        // ePWM1, Period=prd, Master, Phase=0
        BuckSingle_CNF(2, prd, 0, tPHS2);    // ePWM2, Period=prd, Slave, Phase=tPHS2
        EPwm1Regs.CMPB = 193;                // tCMPB1 - ISR trigger point
        // EPwm2Regs.CMPB = 120;             // tCMPB2 - ADC SOS trigger point
        EPwm2Regs.CMPB = 60;                 // tCMPB2 - ADC SOS trigger point
        ADC_DualSeqCNF(ChSel, 1, 2, 1);      // ACQPS=1, Seq1#Conv=2, Seq2#Conv=1
        ISR_Init();                          // ASM ISR init


// Lib Module connection to "nets"
//---------------------------------------
// ADC feedback connections
        ADC_Rslt = &AdcNetBus[1];            // point to 1st element, i.e. AdcCh0

// ------------------------- CHANNEL-1 -------------------------
// CNTL_2P2Z(1) connections
        CNTL_2P2Z_Ref1 = &VrefNetBus[1];     // point to Vref1
        CNTL_2P2Z_Out1 = &UoutNetBus[1];     // point to Uout1
        CNTL_2P2Z_Fdbk1 = &AdcNetBus[1];     // 1st conv result
        CNTL_2P2Z_Coef1 = &Coef2P2Z_1[0];    // point to first coeff of 1st loop
// BUCK_DRV connections
        Buck_In1 = &UoutNetBus[1];           // Ch1 = ePWM1A

// ------------------------- CHANNEL-2 -------------------------
// CNTL_2P2Z(2) connections
        CNTL_2P2Z_Ref2 = &VrefNetBus[2];     // point to Vref2
        CNTL_2P2Z_Out2 = &UoutNetBus[2];     // point to Uout2
        CNTL_2P2Z_Fdbk2 = &AdcNetBus[2];     // 2nd conv result
        CNTL_2P2Z_Coef2 = &Coef2P2Z_2[0];    // point to first coeff of 2nd loop
// BUCK_DRV connections
        Buck_In2 = &UoutNetBus[2];           // Ch2 = ePWM2A
```

Datalogger is an optional feature, not required for loop to run

```
// Data Logger connections, DLTST = DataLogTimeStampTrigger
        DLTST_In1 = &AdcNetBus[1];
        DLTST_TimeBase1 = &ECap1Regs.TSCTR;
        DLTST_TimeStampTrig1 = &DataLogTrigger;
        DLTST_DcOffset1 = 0;
        DLTST_Gain1 = ScopeGain;
```

Compare B event setup to trigger both Sequencer 1 & 2 simultaneously, note: Seq1 has priority

```
// Trigger ADC SOCA & B from EPWM1
//---------------------------------------
        EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTRU_CMPB;  // SOCA on CMPB event
        EPwm1Regs.ETSEL.bit.SOCBSEL = ET_CTRU_CMPB;  // SOCB on CMPB event
        EPwm1Regs.ETSEL.bit.SOCAEN = 1;              // Enable SOC on A group
        EPwm1Regs.ETSEL.bit.SOCBEN = 1;              // Enable SOC on B group
        EPwm1Regs.ETPS.bit.SOCAPRD = ET_1ST;         // Trigger on every event
        EPwm1Regs.ETPS.bit.SOCBPRD = ET_1ST;         // Trigger on every event

#endif // (INCR_BUILD == 2)
```

4. Open and inspect `TwoChannelBuck-ISR.asm`. Notice the `_ISR_Init` and `_ISR_Run` sections. This is where the PWM driver macro instantiation is done for initialization and run-time, respectively. The code is shown below for convenience. In Lab2, incremental build option 2 is used. Note the order for the run time macros:

   1) Measure feedback
   2) Compensate
   3) Update PWM

   Also, the ADC is not run in continuous mode, but rather in SOC trigger mode, and therefore needs to be reset every cycle.

```
;--- Initialization Time
.if(INCR_BUILD = 2)
       ADC_NchDRV_INIT        2       ; 2 Channel, (i.e. N=2)
       ControlLaw_2P2Z_INIT 1
       ControlLaw_2P2Z_INIT 2
       BuckSingleHR_DRV_INIT 1        ; EPWM1A
       BuckSingleHR_DRV_INIT 2        ; EPWM2A
       DataLogTST_INIT        1       ; 1 Channel Data logger
.endif
;---


;--- Run Each Time ISR is Ran
.if(INCR_BUILD = 2)
       ADC_NchDRV     2               ; 1 Channel, (N=2)     (Measure)
       ControlLaw_2P2Z 1                                     (Compensate)
       ControlLaw_2P2Z 2
       BuckSingleHR_DRV 1             ; EPWM1A               (Update)
       BuckSingleHR_DRV 2             ; EPWM2A
       DataLogTST     1               ; 1 Channel Data logger
ADC_Reset:
       MOVW    DP,#ADCTRL2>>6         ; Reset ADC SEQ
       MOV     @ADCTRL2,#0x4101       ; RST_SEQ1=1, SOCA-SEQ1=1, SOCB-SEQ2=1
.endif
;---
```

5. Close the inspected files if desired.

Build and Load the Project

6. Click the "`Rebuild All`" button and watch the tools run in the build window. The output file should automatically load.

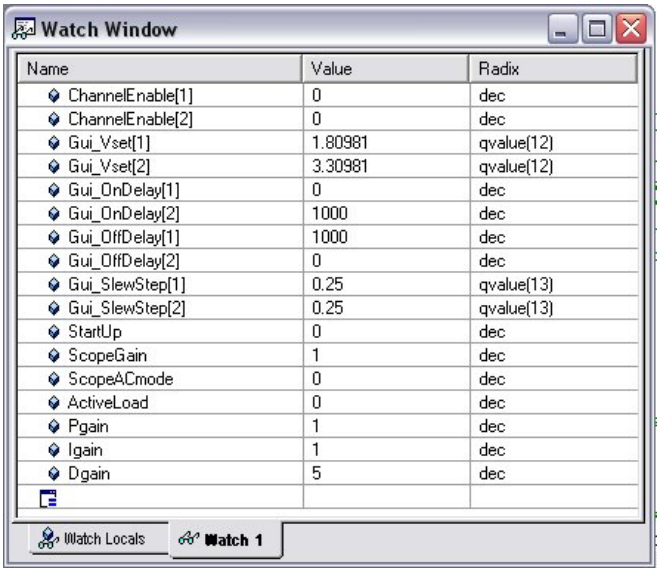
7. Under `Debug` on the menu bar click "`Reset CPU`", "`Restart`", and then "`Go Main`". You should now be at the start of `Main()`.

Setup Watch Window and Graph

8. Another watch window will be opened in addition to the one used in the previous lab exercise. Open the *watch window* to view the variables used in the project.

   Click: `View` → `Watch Window` on the menu bar.

Click the “Watch 1” tab at the bottom of the watch window. In the empty box in the "Name" column, type the symbol names from the following screen capture and be sure to modify the “Radix” as needed.



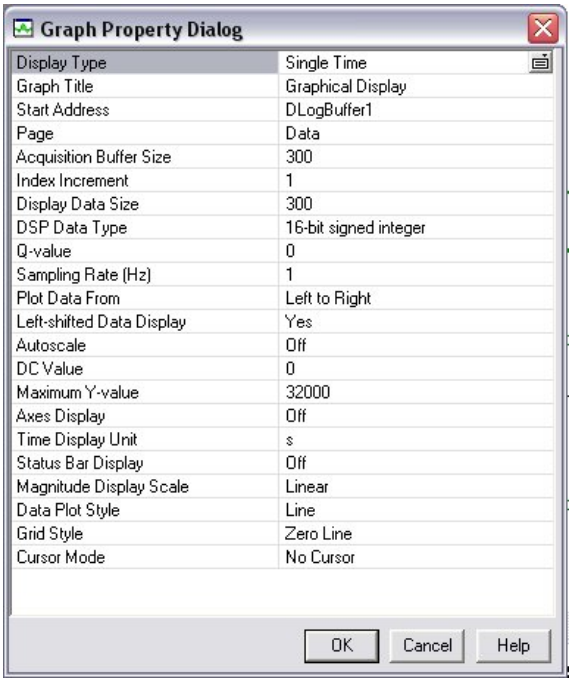
Note: `ScopeGain`, `ScopeACmode`, `ActiveLoad`, `Pgain`, `Igain`, and `Dgain` will be explained and used in the next lab exercise.

The following table gives a description for the variable names:

Variable	Description
`ChannelEnable`	Channel enable array
`Gui_Vset`	Voltage target array
`Gui_OnDelay`	DelayOn array
`Gui_OffDelay`	DelayOff array
`Gui_SlewStep`	Ramp step size array
`StartUp`	Global turn-on/turn-off integer flag

9. Open and setup a time graph windows to plot the data log buffer (ADC result register). This graph will show the output voltage of Channel 1.

Click: `View` → `Graph` → `Time/Frequency`… and set the following values:



Select OK to save the graph options.

Save the Workspace

10. Save the current workspace by naming it `Lab2.wks` and clicking:

    `File → Workspace → Save Workspace As…`

    and saving in `C:\TI_28xxx_SysSW\TwoChannelBuck\`.

Run the Code – TwoChannelBuck

11. Enable real-time mode by selecting:

    `Debug → Real-time Mode`

12. A message box *may* appear. If so, select YES to enable debug events. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a "0". The DGBM is the debug enable mask bit. When the DGBM bit is set to "0", memory and register values can be passed to the host processor for updating the debugger windows.

13. Check to see if the windows are set to continuously refresh. Click:

    `View → Real-time Refresh Options…`

    and check `"Global Continuous Refresh"`.

14. Run the code by using the <F5> key, or using the `Run` button on the vertical toolbar, or using `Debug` → `Run` on the menu bar.

15. Turn on the 9-volt DC bus (SW2) on the 2-channel buck EVM and observe variable `Gui_Vin` in the watch-window. It should now be approximately 9V. Also the Graph window should show a trace hovering at "0V".

16. In the watch window turn on Channel 1 output by setting `ChannelEnable[1]=1`. Power stage buck 1 module output voltage should ramp quickly to ~1.8V (be sure that SW3 on the EVM is positioned to select Ch1). Note that directly turning-on (enabling) an individual channel ignores the `Gui_OnDelay` and `Gui_OffDelay` parameters since synchronization to a global trigger is not utilized.

17. Enable Channel 2 output by setting `ChannelEnable[2]=1`. Power stage buck 2 module output voltage should new ramp quickly to ~3.3V (this can be verified by setting SW2 to select Ch2 and checking the value on the display). Also note that LMP1, the load for Channel 2, is now lit.

18. Decrease the value of `Gui_Vset[1]` and `Gui_Vset[2]` to approximately 1.0. The voltage output of both power stage buck 1 and power stage buck 2 should now be updated to 1.0V. In the watch windows notice that `Gui_Iout[2]` is less than `Gui_Iout[1]`. This is because the LMP1, the load of Channel 2, has more resistance than the 1Ω resistance found across Channel 1. When you are done turn off Channel 1 and Channel 2 by setting `ChannelEnable[1]=0` and `ChannelEnable[2]=0`.

19. Both channels can also be enabled or disabled by using the global turn-on flag – `StartUp`. This flag allows for time sequencing of Channel 1 and 2 and uses `Gui_OnDelay` and `Gui_OffDelay` to give the delay before turning on/off each channel. These values may be modified in the watch window and are set by default to:

```
Gui_OnDelay[1] = 0;
Gui_OnDelay[2] = 1000;      // 1 second delay

Gui_OffDelay[1] = 1000;     // 1 second delay
Gui_OffDelay[2] = 0;
```

20. For example, modify these values as follows:

```
Gui_OnDelay[1]=1000
Gui_OnDelay[2]=5000

Gui_OffDelay[1]=7500
Gui_OffDelay[2]=2500

StartUp=1
```

    This will trigger the global turn flag. Then Channel 1 will start ramping up after 1000 ms and Channel 2 will start ramping after 5000 ms. Setting `StartUp=0` will trigger a global turn off which will cause Channel 1 to ramp down after 7500 ms and Channel 2 to begin ramping down after 2500 ms.

21. The ramp-up and ramp-down rates can also be modified for either Channel 1 or Channel 2. In this lab code, up and down ramp rates are the same. The ramp rate for Channel 1 is set by the parameter `Gui_SlewStep[1]`, and the ramp rate of Channel 2 is set by `Gui_SlewStep[2]` The default value for both is 0.25 V/ms. Change both to 0.005 V/ms in the watch window and view the result. Follow these steps:

    ```
    Gui_SlewStep[1]=0.005
    Gui_SlewStep[2]=0.005

    ChannelEnable[1]=1      Outputs should now ramp up at slow rate
    ChannelEnable[2]=1

    ChannelEnable[1]=0      Outputs should also ramp down at a slow rate
    ChannelEnable[2]=0
    ```

22. Turn off the 9-volt DC bus (SW2) on the 2-channel buck EVM. ***(Do not turn off the main power SW1).***

23. Fully halt the DSP in real-time mode. First, halt the processor by using Shift <F5>, or using the `Halt` button on the vertical toolbar, or by using `Debug → Halt`. Then click `Debug → Real-time Mode` and uncheck the `"Real-time mode"` to take the DSP out of real-time mode.

24. ***<u>Do Not Close Code Composer Studio</u> or it will be necessary to setup the debug environment windows again for the next lab exercise!***

End of Exercise

Lab 3: Tuning the Loop

➢ Objective

The objective of this lab exercise is to demonstrate the topics discussed in this module and tune the closed-loop buck power stage for improved transient performance using visual "trial and error" methods rather than a mathematical approach. The transient response will be modified by interactively adjusting the system proportional (P), integral (I), and derivative (D) gains using the watch window. An active load circuit enabled by software will provide a repetitive step change in load. The CCS graph window feature will be used to view the transient response in real-time. The Digital Power software framework, associated files, and library modules will be used. In this lab you will:

- Tune closed-loop Buck power stage for improved transient performance using visual "trial and error" approach (as opposed to a mathematical approach)
- Use an active load circuit which, when enable by software, provides a repetive step change in load
- Use the CCS graph window feature to view and assist you in achieving a desired transient response

➢ Project Overview

The software code used in Lab3 is exactly the same code as used in Lab2. All of the files and build options are identical. The five coefficients to be modified are stored in the arrays `Coef2P2Z` and`_1 Coef2P2Z_2[n]`. Directly manipulating these five coefficients independently by trial and error is almost impossible, and requires mathematical analysis and/or assistance from tools such as MATLAB, MathCad, etc. These tools offer Bode plot, root locus and other features for determining phase margin, gain margin, etc.

To keep loop tuning simple and without the need for complex mathematics or analysis tools, the coefficient selection problem has been reduced from five degrees of freedom to three, by conveniently mapping the more intuitive coefficient gains of P, I and D to B0, B1, B2, A1, and A2. This allows P, I, and D to be adjusted independently and gradually. This method requires a periodic transient or disturbance to be present, and a means to observe it while interactively making adjustments. The data-logging feature introduced in Lab2 provides a convenient way to observe the output transient while the built-in active load (controlled by the ECAP peripheral) on the EVM can provide the periodic disturbance.

The compensator block (macro) used is `ControlLaw_2P2Z`. This block has 2 poles and 2 zeros and is based on the general IIR filter structure. The transfer function is given by:

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

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

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

where:

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

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

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

Comparing this with the general form, we can see that PID is nothing but a special case of a 2pole/2zero controller where:

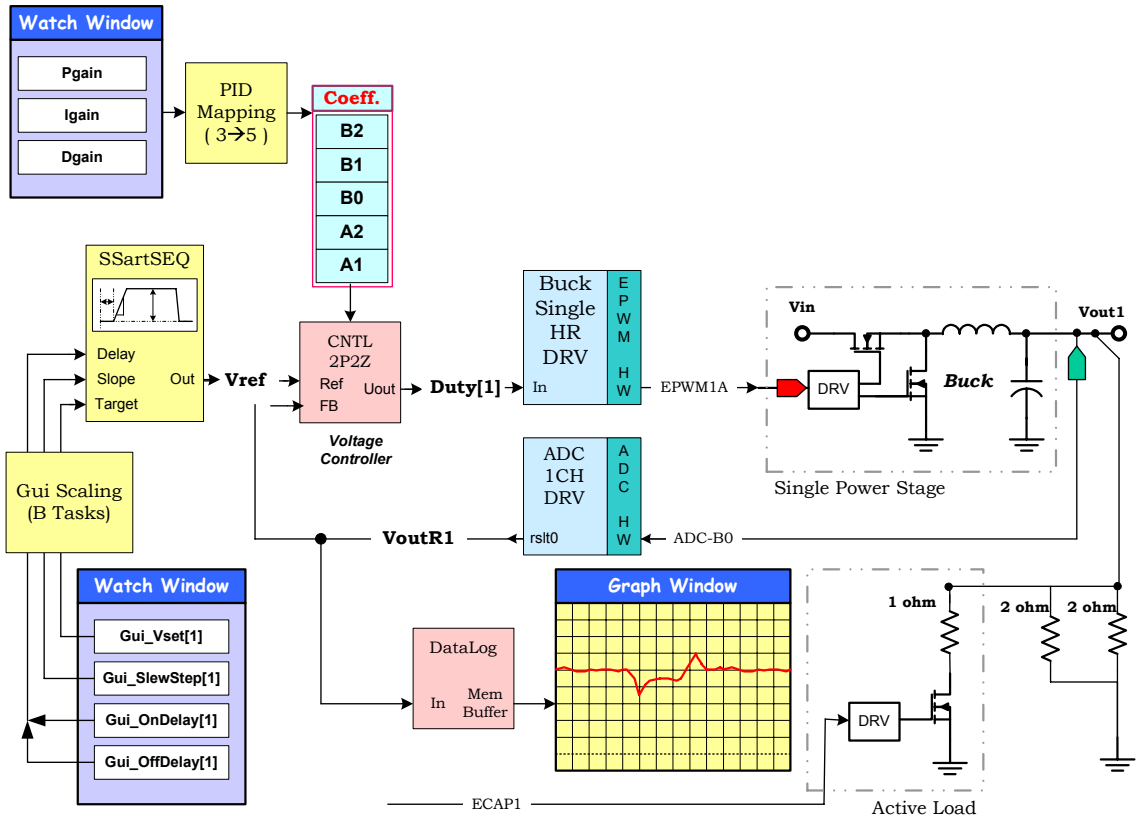
$$a_1 = -1 \text{ and } a_2 = 0$$

In the lab exercise, you will inspect the C code in which these coefficients are initialized.

➢ Lab Exercise Overview

In Lab2 the software was configured to provide closed-loop voltage control for Ch1 of the buck EVM and datalogging of the output which is displayed in a CCS graph window. Lab3 will additionally allow modification of the five coefficients associated with the 2nd order `ControlLaw_2P2Z` compensator block. This modification will be done "on the fly" by editing `Pgain`, `Igain` and `Dgain` in the watch window while the buck output is put under transient stresses. These stresses are caused by an active load which is switched periodically by the ECAP peripheral. Note that because the active load is connected to Channel 1, Channel 2 will not be used in this lab.

The following figure is the system diagram for Lab3.



The default coefficient settings chosen for Lab2 provide very poor performance (low gains). Initially Lab3 will use the same settings. The control loop will be soft-stared to the target Vout value, the same way it was done in Lab2. At this point, the active load will be enabled and a load resistor of equal value to the static load will be switched in and out periodically.

In addition to the watch window variables discussed in Lab3, a few others will be used in the loop tuning process. These include the P, I and D gains, active load enable, and CCS graph window (scope control). The table below summarises these new variables.

`Pgain`	Proportional gain; value adjustment : 0 ~ 1000
`Igain`	Integral gain; value adjustment : 0 ~ 1000
`Dgain`	Derivative gain; value adjustment : 0 ~ 1000
`ActiveLoad`	Enable (value=1) / Disable (value=0) flag for the active load circuit
`ScopeACmode`	Sets the CCS Scope (graph window) to operate in AC mode (i.e. removing the DC component) and is useful for zooming into the transient only
`ScopeGain`	Vertical gain adjustment for the CCS scope (much like a real oscilloscope)

➢ **Procedure**

Open a CCS Project

1. Code Composer Studio should still be running from the previous lab exercise. If not, then it will be necessary to setup the project and debug environment from the previous lab exercises. This can be done by simply loading the last saved workspace. Click:

   `File → Workspace → Load Workspace`

   In the new window browse open:

   `C:\TI_28xxx_SysSW\TwoChannelBuck\Lab2.wks`

Device Initialization, Main, and ISR Files

2. Open `TwoChannelBuck-Settings.h`. For this lab we will use incremental build option 2. Please verify that INCR_BUILD is defined as 2 (i.e. `#define INCR_BUILD 2`).

   If the incremental build is not 2, please edit the file then save it by clicking:

   `File → Save`

Note: This should be the **only** change made to source files for this lab.

---

3. Open and inspect `TwoChannelBuck-Main.c` by double clicking on the filename in the project window. Notice the coefficient initialization and P, I, and D mapping equation. A section of code is shown here for convenience.

```
Pgain = 1;     Igain = 1;     Dgain = 5;            // very "loose" initially

// Coefficient init for Single Loop
       Coef2P2Z_1[0] = Dgain * 67108;                          // B2
       Coef2P2Z_1[1] = (Igain - Pgain - Dgain - Dgain)*67108;  // B1
       Coef2P2Z_1[2] = (Pgain + Igain + Dgain)*67108;          // B0
       Coef2P2Z_1[3] = 0;                                      // A2 = 0
       Coef2P2Z_1[4] = 67108864;                               // A1 = 1 in Q26
       Coef2P2Z_1[5] = Dmax[1] * 67108;            // Clamp Hi limit (Q26)
       Coef2P2Z_1[6] = 0x00000000;                 // Clamp Lo
```

   (Note: 67108 = ~ 0.001 in Q26 format)

4. Close the inspected files if desired.

Build and Load the Project

5. Click the "`Rebuild All`" button and watch the tools run in the build window. The output file should automatically load.

6. Under Debug on the menu bar click "Reset CPU", "Restart", and then "Go Main". You should now be at the start of Main().

Save the Workspace

7. The watch windows and graph window were setup in the previous lab exercise. A workspace needs to be saved to include the project for this lab exercise. Save the current workspace by naming it Lab3.wks and clicking:

   File → Workspace → Save Workspace As…

   and saving in C:\TI_F28xxx_SysSW\TwoChannelBuck\.

Run the Code – TwoChannelBuck

8. Enable real-time mode by selecting:

   Debug → Real-time Mode

9. A message box *may* appear. If so, select YES to enable debug events. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a "0". The DGBM is the debug enable mask bit. When the DGBM bit is set to "0", memory and register values can be passed to the host processor for updating the debugger windows.

10. Check to see if the windows are set to continuously refresh. Click:

    View → Real-time Refresh Options…

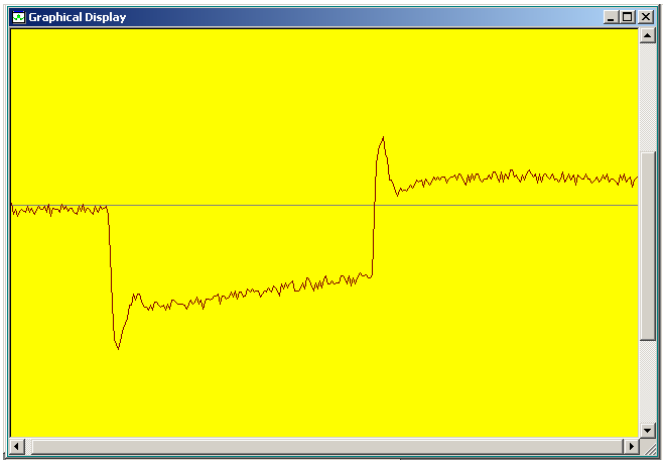
    and check "Global Continuous Refresh".

11. Run the code by using the <F5> key, or using the Run button on the vertical toolbar, or using Debug → Run on the menu bar.

12. Turn on the 9-volt DC bus (SW2) on the TwoChannelBuck EVM and observe variable Gui_Vin in the watch-window. It should now be approximately 9V. Also the Graph window should show a trace hovering at "0V".

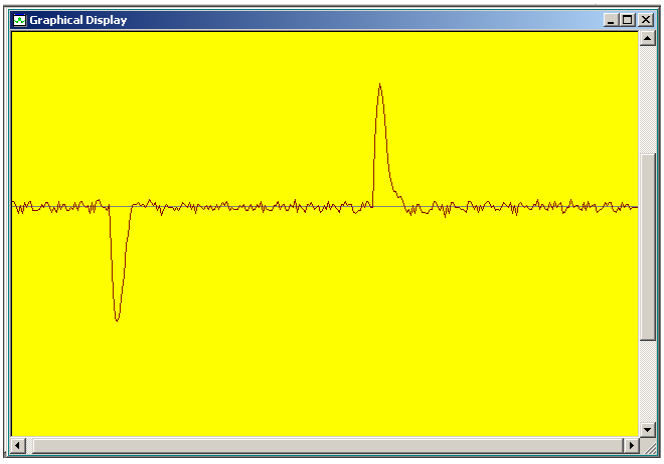
13. In the watch window turn on Channel 1 output by setting ChannelEnable[1]=1. Power stage buck 1 module output voltage should ramp quickly to ~1.8V (be sure that SW3 on the EVM is positioned to select Ch1).

14. Enable the active load circuit by setting variable ActiveLoad = 1 in the watch window. To better view only the transient or AC component of the output voltage, set the graph to AC mode by changing variable ScopeACmode = 1. This should put the output waveform at the graph "zero" line. Optionally, set the scope gain higher by changing variable ScopeGain = 4. If lab is working correctly, then the graph window should look something like the following:

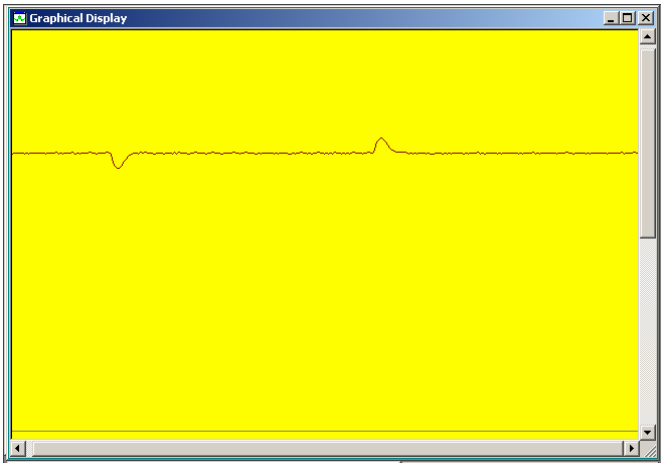


The negative going transient is when the extra load is switched in and the positive going overshoot is when the extra load is removed.

15. While observing the transient in real time, gradually adjust Pgain, Igain, and Dgain in the Watch Window to get the best transient response (i.e. least pertubation from the zero line). Gradual adjustment is recommended as a large change may cause the system to go unstable. A suggested procedure is given below:
    - Gradually increase Igain until the negative going transient flattens out near the zero line
    - Increase Pgain until some oscillation (2~3 cycles) occurs
    - Increase Dgain to remove some of the oscillation
    - Increase Igain again
    - keep iterating very gradually until an acceptable transient response is achieved – this may look something like the graph shown below:



16. Reduce the scope vertical gain back to 1 (`ScopeGain = 1`) and set the graph back in DC mode (`ScopeACmode = 0`). The voltage output response should look something like this now:



17. With the active load still enabled, try shutting down Channel 1 output and then bringing it back up under "soft" shut-down and start-up conditions. The closed loop should be stable during the ramping even during the switching transients.

18. Turn off the 9-volt DC bus (SW2) on the 2-channel buck EVM. ***(Do not turn off the main power SW1).***

19. Fully halt the DSP in real-time mode. First, halt the processor by using Shift <F5>, or using the `Halt` button on the vertical toolbar, or by using `Debug → Halt`. Then click `Debug → Real-time Mode` and uncheck the `"Real-time mode"` to take the DSP out of real-time mode.

20. Close Code Composer Studio and turn off the power (SW1) to the 2-channel buck EVM.

End of Exercise

References

- **QSG-TwoChannelBuck** – provides all the information on software and hardware setup necessary to quickly get started with the TwoChannelBuck project.

  C:\TI_28xxx_SysSW\TwoChannelBuck\~Docs\QSG-TwoChannelBuck.pdf

- **System Framework Overview** – presents more information on the system framework found in all F28xxx EVM projects.

  C:\TI_28xxx_SysSW\~Docs\SystemFrameworkOverview.pdf

- **QSG-TwoChannelBuck-GUI** – gives an overview on how to quickly demo the TwoChannelProject using an intuitive GUI interface.

  C:\TI_28xxx_SysSW\TwoChannelBuck\~Docs\QSG-TwoChannelBuck-GUI.pdf

- **TwoChannelBuck-Calculations** – a spreadsheet showing a few of the key calculations made within the TwoChannelBuck project.

  C:\TI_28xxx_SysSW\TwoChannelBuck\TwoChannelBuck-Calculations.xls

- **TwoChBuck-HWdevPkg** – a folder that contains various files related to the hardware on the EVM board (schematics, bill of materials, gerber files, pcb layout, etc). All schematics and pcb files created with the freeware ExpressPCB package.

  C:\TI_28xxx_SysHW\TwoChBuck-HWdevPkg

- **F28xxx User's Guides** -

  http://www.ti.com/f28xuserguides