Important Notice

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI’s publication of information regarding any third party’s products or services does not constitute TI’s approval, warranty or endorsement thereof.

Copyright © 2008 Texas Instruments Incorporated

Revision History

January 2008 – Revision 1.0
May 2008 – Revision 1.1

Mailing Address

Texas Instruments
Training Technical Organization
7839 Churchill Way
M/S 3984
Dallas, Texas 75251-1903
Workshop Topics

Workshop Outline ...............................................................................................................................................4

1 – Introduction to Digital Power Supply Design .........................................................................................5
   What is a Digital Power Supply? ..................................................................................................................5
   Why use Digital Control Techniques? .........................................................................................................6
   Peripherals used for Digital Power Supply Design .......................................................................................10
   Development Tools and Software ..............................................................................................................12

Lab1: Exploring the Development Environment ............................................................................................16

2 – Driving the Power Stage with PWM Waveforms ....................................................................................24
   Open-Loop System Block Diagram .............................................................................................................24
   Generating PWM using the ePWM Module ..................................................................................................25
   Power Stage Topologies and Software Library Support ...............................................................................28

Lab2: PWM Generation / Open-Loop Control ...............................................................................................32

3 – Controlling the Power Stage with Feedback ..........................................................................................42
   Closed-Loop System Block Diagram ...........................................................................................................42
   ADC Module Block Diagram ......................................................................................................................43
   Digital Control of Power Converter ...........................................................................................................43
   High-Resolution PWM Benefits ................................................................................................................46
   Soft Start – Starting the Loop .......................................................................................................................47

Lab3: Closed-Loop Control ..............................................................................................................................48

4 – Tuning the Loop for Good Transient Response ......................................................................................57
   Digital Power Supply Control Theory ..........................................................................................................57
   Intuitive Loop Tuning – “Visually without Math” .........................................................................................59
   Active Load Feature of the Power EVM .........................................................................................................63

Lab4: Tuning the Loop ......................................................................................................................................64
   Multi-Loop Control ....................................................................................................................................71

5 – Summary and Conclusion .........................................................................................................................73
   Review of Workshop Topics and Exercises ..................................................................................................73
   TI Digital Power Products ............................................................................................................................74
   C2000 Digital Signal Controller Family .....................................................................................................75
   UCD9xxx Digital Power Controller Family ..................................................................................................80
   Where to Find More Information ...............................................................................................................81
Workshop Outline

<table>
<thead>
<tr>
<th>Workshop Outline</th>
</tr>
</thead>
</table>
| 1. Introduction to Digital Power Supply Design  
  • *Lab: Exploring the Development Environment* |
| 2. Driving the Power Stage with PWM Waveforms  
  • *Lab: PWM Generation / Open-Loop Control* |
| 3. Controlling the Power Stage with Feedback  
  • *Lab: Closed-Loop Control* |
| 4. Tuning the Loop for Good Transient Response  
  • *Lab: Tuning the Loop* |
| 5. Summary and Conclusion |
1 – Introduction to Digital Power Supply Design

What is a Digital Power Supply?

The controller block is what differentiates between a digital power system and a conventional analog power system

What is Digital Power?

Generic Power System Block Diagram

The controller block is what differentiates between a digital power system and a conventional analog power system
Why use Digital Control Techniques?

Why Digital Control Techniques?

<table>
<thead>
<tr>
<th>Analog Controller</th>
<th>Digital Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ High bandwidth</td>
<td>+ Insensitive to environment (temp, drift,...)</td>
</tr>
<tr>
<td>+ High resolution</td>
<td>+ SW programmable / flexible solution</td>
</tr>
<tr>
<td>+ Easy to understand / use</td>
<td>+ Precise / predictable behavior</td>
</tr>
<tr>
<td>+ Historically lower cost</td>
<td>+ Advanced control possible (non-linear, multi-variable)</td>
</tr>
<tr>
<td>Component drift and aging / unstable</td>
<td>Bandwidth limitations (sampling loop)</td>
</tr>
<tr>
<td>Component tolerances</td>
<td>PWM frequency and resolution limits</td>
</tr>
<tr>
<td>Hardwired / not flexible</td>
<td>Numerical problems (quantization, rounding,...)</td>
</tr>
<tr>
<td>Limited to classical control theory only</td>
<td>AD / DA boundary (resolution, speed, cost)</td>
</tr>
<tr>
<td>Large parts count for complex systems</td>
<td>CPU performance limitations</td>
</tr>
<tr>
<td>Bias supplies, interface requirements</td>
<td>S/w programmable / flexible solution</td>
</tr>
<tr>
<td>Precise / predictable behavior</td>
<td>Can perform multiple loops and “other” functions</td>
</tr>
<tr>
<td>Hardwired / not flexible</td>
<td>High bandwidth</td>
</tr>
<tr>
<td>Limited parts count for complex systems</td>
<td>High resolution</td>
</tr>
<tr>
<td>Easy to understand / use</td>
<td>Easy to understand / use</td>
</tr>
<tr>
<td>Historically lower cost</td>
<td>Historically lower cost</td>
</tr>
</tbody>
</table>

Benefits of Digital Control

Traditional Analog Power Supply
- Multiple chips for control
- Micro-controller for supervisory
- Dedicated design

Digital controller enables multi-threaded applications

Eliminate Components
Reduce Manufacturing Cost
Better Performance Across Corners
One Design, Multiple Supplies
Failure Prediction
One Device, Multiple DC Outputs
Variable DC Output
### Analog Control System

**R** + \[\sum \] **e** → **C** (controller) → **P** (plant) → **Y**

\[ C(s) = \frac{R_2}{R_1} \left( \frac{1 + R_1 C_1}{1 + R_2 C_2} \right) \]

Differential equations
1st, 2nd, 3rd,… order

Need to find:
\[ R_1, R_2, C_1, C_2 \]

Laplace Transform

### Digital Control System

**R** + \[\sum \] **E** → **C_d** (controller) → **D-A** → **A-D** → **P** (plant) → **Y**

Difference equation
\[ U(n) = a_1 U(n-1) + a_2 U(n-2) + b_0 + b_1 E(n-1) + b_2 E(n) \]

where… \[ E(n) = R(n) - Y(n) \]

Need to find:
\[ a_1, a_2, b_0, b_1, b_2 \]

Differential equations
1st, 2nd, 3rd,… order

Laplace Transform

OR

Z Transform
1 – Introduction to Digital Power Supply Design

Time Sampled Systems

Digital Processor

Control

Law

A-D

Σ

Ref

+ -

D-A

Continuous

t

Discrete
t

t

Sample

Period

Processor Bandwidth

Sample Freq (=PWM)  Sample Period

(kHz)  (ns)

100  10000
300  3333
500  2000
700  1429
1000  1000
1500  667
2000  500
**Time Division Multiplexing (TDM)**

\[ y(n) \quad T_{\text{SAMPLE}} \quad t \]

Processor 1: Control Code (C1)  
Processor 2: Control Code (C2)  
Processor 3: Control Code (C3)  
Single CPU: C1, C2, C3

**Digitally Controlled Power Supply**

DSC

DAC (PWM)

ADC

"Plant"

0110101100  
1011011101  
0010100111

"High fidelity"  
Translation boundary
1 – Introduction to Digital Power Supply Design

System Mapping

Peripherals used for Digital Power Supply Design

TMS320F280x

Efficient 32-bit Processor Capability

- Single-cycle 32-bit multiplier makes computationally intensive control algorithms more efficient
- Three 32-bit timers support multiple control loops / time bases
- Single cycle read-modified-write in any memory location and 32-bit registers improve control algorithm efficiency
- Real-time JTAG debug shortens development cycle
- Fast & flexible interrupt management significantly reduce interrupt latency

### MIPS vs PWM

<table>
<thead>
<tr>
<th>PWM freq. (kHz)</th>
<th>PWM per. (μs)</th>
<th>Processor MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>20.0</td>
<td>2000</td>
</tr>
<tr>
<td>100</td>
<td>10.0</td>
<td>1000</td>
</tr>
<tr>
<td>200</td>
<td>5.0</td>
<td>500</td>
</tr>
<tr>
<td>250</td>
<td>4.0</td>
<td>400</td>
</tr>
<tr>
<td>300</td>
<td>3.3</td>
<td>333</td>
</tr>
<tr>
<td>500</td>
<td>2.0</td>
<td>200</td>
</tr>
<tr>
<td>750</td>
<td>1.3</td>
<td>133</td>
</tr>
<tr>
<td>1000</td>
<td>1.0</td>
<td>100</td>
</tr>
</tbody>
</table>

MIPS = Million Instruction Per Second

### ePWM “DAC” Capability

- Number of channels scalable and resources allocated per channel
- Two independent PWM outputs per module
- Dedicated time-base timer
- Two independent compare registers
- Multi-event driven waveform
- Trip zones and event interrupts
- F2808 offers 6 modules
- Provides ePWM DAC capability for DPS
- Switching can be programmed as Asymmetric or Symmetric PWM
- High-Resolution PWM mode

### PWM effective resolution (CPU=100MHz)

<table>
<thead>
<tr>
<th>PWM (kHz)</th>
<th>Standard PWM</th>
<th>HR-PWM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bits</td>
<td>%</td>
</tr>
<tr>
<td>50</td>
<td>11.0</td>
<td>0.05</td>
</tr>
<tr>
<td>100</td>
<td>10.0</td>
<td>0.10</td>
</tr>
<tr>
<td>150</td>
<td>9.4</td>
<td>0.15</td>
</tr>
<tr>
<td>250</td>
<td>8.6</td>
<td>0.25</td>
</tr>
<tr>
<td>500</td>
<td>7.6</td>
<td>0.50</td>
</tr>
<tr>
<td>750</td>
<td>7.1</td>
<td>0.75</td>
</tr>
<tr>
<td>1000</td>
<td>6.6</td>
<td>1.00</td>
</tr>
</tbody>
</table>
12-bit ADC Capability

Control Peripherals

- Fast & Flexible 12-bit 16-Channel ADC
  - 12.5 MSPS throughput
  - Dual sample/hold enable simultaneous sampling or sequencing sampling modes
  - Analog input: 0V to 3V
  - 16 channel, multiplexed inputs
  - Auto Sequencer supports up to 16 conversions without CPU intervention
  - Sequencer can be operated as two independent 8-state sequencers or as one large 16-state sequencer
  - Sixteen result registers (individually addressable) to store conversion values

ADC Utilization: # Channels (“Loops”) vs. PWM frequency

<table>
<thead>
<tr>
<th>PWM (kHz)</th>
<th># Channels</th>
<th>PWM (kHz)</th>
<th># Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>24</td>
<td>125</td>
<td>50</td>
</tr>
<tr>
<td>250</td>
<td>12</td>
<td>250</td>
<td>25</td>
</tr>
<tr>
<td>500</td>
<td>6</td>
<td>500</td>
<td>13</td>
</tr>
<tr>
<td>750</td>
<td>4</td>
<td>750</td>
<td>8</td>
</tr>
<tr>
<td>1000</td>
<td>3</td>
<td>1000</td>
<td>6</td>
</tr>
</tbody>
</table>

Development Tools and Software

Code Composer Studio

- Project Manager:
  - Source & object files
  - File dependencies
  - Compiler, Assembler & Linker build options

- Full C/C++ & Assembly Debugging:
  - C & ASM Source
  - Mixed mode
  - Disassembly (patch)
  - Set Break Points
  - Set Probe Points

- Editor:
  - Structure Expansion

- Menus or Icons
- Help
- CPU Window
- Status Window
- Watch Window
- Graph Window
- Memory Window
Software Library Approach

Modular Software Architecture

“Signal Net” based module connectivity

Initialization time

Run time - ISR

// pointer & Net declarations
Int *In1A, *In1B, *Out1, *In2A...

// “connect” the modules
In1A=In1; In1B=In1B; In2A=In2A; In3A=In3A;

// inputs
Out4=Out4; Out5=Out5; // outputs
Out1=Out1; In4A=In4A; // Net1
Out2=Out2; In4B=In4B; // Net2
Out3=Out3; In4C=In4C; // Net3
Out3=Out3; In5A=In5A; // Net4
Out2=Out2; In5B=In5B; // Net5
Out1=Out1; In5C=In5C; // Net6
Out4=Out4; In5D=In5D; // Net7
Out5=Out5; In5E=In5E; // Net8
Out6=Out6; In5F=In5F; // Net9
Out7=Out7; In5G=In5G; // Net10
Out8=Out8; In5H=In5H; // Net11
Out9=Out9; In5I=In5I; // Net12
Out10=Out10; In5J=In5J; // Net13
Out11=Out11; In5K=In5K; // Net14
Out12=Out12; In5L=In5L; // Net15
Out13=Out13; In5M=In5M; // Net16
Out14=Out14; In5N=In5N; // Net17
Out15=Out15; In5O=In5O; // Net18
Out16=Out16; In5P=In5P; // Net19
Out17=Out17; In5Q=In5Q; // Net20
Out18=Out18; In5R=In5R; // Net21
Out19=Out19; In5S=In5S; // Net22
Out20=Out20; In5T=In5T; // Net23
Out21=Out21; In5U=In5U; // Net24
Out22=Out22; In5V=In5V; // Net25
Out23=Out23; In5W=In5W; // Net26
Out24=Out24; In5X=In5X; // Net27
Out25=Out25; In5Y=In5Y; // Net28
Out26=Out26; In5Z=In5Z; // Net29
Out27=Out27; In5AA=In5AA; // Net30
Out28=Out28; In5AB=In5AB; // Net31
Out29=Out29; In5AC=In5AC; // Net32
Out30=Out30; In5AD=In5AD; // Net33
Out31=Out31; In5AE=In5AE; // Net34
Out32=Out32; In5AF=In5AF; // Net35
Out33=Out33; In5AG=In5AG; // Net36
Out34=Out34; In5AH=In5AH; // Net37
Out35=Out35; In5AI=In5AI; // Net38
Out36=Out36; In5AJ=In5AJ; // Net39
Out37=Out37; In5AK=In5AK; // Net40
Out38=Out38; In5AL=In5AL; // Net41
Out39=Out39; In5AM=In5AM; // Net42
Out40=Out40; In5AN=In5AN; // Net43
Out41=Out41; In5AO=In5AO; // Net44
Out42=Out42; In5AP=In5AP; // Net45
Out43=Out43; In5AQ=In5AQ; // Net46
Out44=Out44; In5AR=In5AR; // Net47
Out45=Out45; In5AS=In5AS; // Net48
Out46=Out46; In5AT=In5AT; // Net49
Out47=Out47; In5AU=In5AU; // Net50
Out48=Out48; In5AV=In5AV; // Net51
Out49=Out49; In5AW=In5AW; // Net52
Out50=Out50; In5AX=In5AX; // Net53
Out51=Out51; In5AY=In5AY; // Net54
Out52=Out52; In5AZ=In5AZ; // Net55
Out53=Out53; In5BA=In5BA; // Net56
Out54=Out54; In5BB=In5BB; // Net57
Peripheral Drivers

CPU dependency only:
- Math / algorithms
- Per-Unit math (0-100%)
- Independent of Hardware

Depends on:
- PWM frequency
- System clock frequency

// pointer & Net declarations
int *CNTL_Ref1, *CNTL_Fdbk1, *CNTL_Out1;
int *BUCK_In1, *ADC_Rslt1;
int Vref, Duty, Vout;

// “connect” the modules
CNTL_Ref1 = &Vref;
CNTL_Out1 = &Duty;
BUCK_In1 = &Duty;
CNTL_Fdbk1 = &Vout;
ADC_Rslt1 = &Vout;

Dual Buck Example
Software Block Execution

BG

SStartSeq
Comms
Other....

ISR
(400 kHz)

Context Save

Loop-1
ADC_DRV(1)
CNTL_2P2Z(1)
BUCK_DRV(1)

Loop-2
ADC_DRV(2)
CNTL_2P2Z(2)
BUCK_DRV(2)

Context Restore
Lab1: Exploring the Development Environment

Objective

The objective of this lab exercise is to demonstrate the topics discussed in this module and become familiar with the operation of Code Composer Studio (CCS). Steps required to build and run a project will be explored. The project will generate various PWM waveforms which will be viewed using the CCS graphical capabilities. The slider feature in CCS will be used to adjust the duty, phase, and dead-band values of the waveforms. Additionally, the Digital Power software framework, associated files, and library modules will be used.

Lab1: Exploring the Development Environment

- Navigate CCS features
- Understand DPS library structure
- Generate and visualize PWM waveforms

Project Overview

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

The key framework C files used in this project are:

PWMexplore-Main.c – this file is used to initialize, run, and manage the application. This is the “brains” behind the application.

PWMexplore-DevInit.c – this file is responsible for a one time initialization and configuration of the F280x device, and includes functions such as setting up the clocks, PLL, GPIO, etc.
Lab1: Exploring the Development Environment

The ISR consists of a single file:

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

The Power Library functions (modules) are “called” from this framework. Library modules may have both a C and an assembly component. In this lab exercise, six library modules (all PWM waveform generators or drivers) are used. The C and corresponding assembly module names are:

<table>
<thead>
<tr>
<th>C configure function</th>
<th>ASM initialization macro</th>
<th>ASM run-time macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuckSingle_CNF()</td>
<td>BuckSingle_DRV_INIT n</td>
<td>BuckSingle_DRV n</td>
</tr>
<tr>
<td>BuckDual_CNF()</td>
<td>BuckDual_DRV_INIT n</td>
<td>BuckDual_DRV n</td>
</tr>
<tr>
<td>MPhIL_CNF()</td>
<td>MPhIL_DRV_INIT n, N</td>
<td>MPhIL_DRV n, N</td>
</tr>
<tr>
<td>FullBridgePS_CNF()</td>
<td>FullBridgePS_DRV_INIT n</td>
<td>FullBridgePS_DRV n</td>
</tr>
<tr>
<td>FullBridgeIBM_CNF()</td>
<td>FullBridgeIBM_DRV_INIT n</td>
<td>FullBridgeIBM_DRV n</td>
</tr>
<tr>
<td>PFC2PhIL_CNF()</td>
<td>PFC2PhIL_DRV_INIT n</td>
<td>PFC2PhIL_DRV_INIT n</td>
</tr>
</tbody>
</table>

These blocks can also be represented graphically. This helps visualize the system software flow and function input/output. The PWM driver modules used in Lab1 are:

![PWM Driver Modules Diagram](image-url)
Lab Exercise Overview

The software in Lab1 has been configured so the user can quickly evaluate the 6 PWM driver modules by viewing the output waveforms and interactively adjusting the duty, phase, and deadband values. The graphing feature of CCS is used to visualize the waveform. The ADC peripheral is configured to provide a “scope” capture function. The PWM outputs on the buck EVM are directly connected to ADC inputs via zero ohm resistors. Collected data samples are stored in four separate memory buffers, hence a simple 4-channel scope is realized. CCS can link each memory buffer to a graph window and display the captured data. With the real-time feature enabled, this data can be captured at high speed and streamed back via JTAG (at a slower rate) to update the graph windows periodically (~200 ms update rate).

Since the PWM waveforms being sampled are essentially “square waves” (high speed edges) they have been scaled down in frequency to approximately 10 kHz. This allows the ADC sampling to better capture and display the edge transitions in the graph window during datalogging. As Lab1 is more for visual demonstration purposes, the high speed ISR code subroutine _ISR_Run has been allocated to datalogging. The PWM driver macros are running at a much slower update rate from subroutine _ISR_Pseudo, which is conveniently called directly from C. The PWM driver macro instantiation convention however is still the same as in the more typical case where _ISR_Run is used to execute all PWM updates and loop control. This will be the convention used in Labs 2, 3 and 4 where an actual 2-channel buck stage will be controlled with high speed PWM outputs.

The following diagram shows an example of how the Full Bridge Phase Shifted PWM module is evaluated in this lab. This setup is essentially the same for all 6 cases, except the PWM driver macro module is swapped and the appropriate sliders used to adjust the relevant timing are selected.
Procedure

Start CCS and Open a Project

1. Move the switch SW1 to the “on” position to power the 2-channel buck EVM board.

2. Double click on the Code Composer Studio icon on the desktop. Maximize Code Composer Studio to fill your screen. Code Composer Studio has a Connect/Disconnect feature which allows the target to be dynamically connected and disconnected. This will reset the JTAG link and also enable “hot swapping” a target board. Connect to the target.

   Click: Debug → Connect

   The menu bar (at the top) lists File ... Help. Note the horizontal tool bar below the menu bar and the vertical tool bar on the left-hand side. The window on the left is the project window and the large right hand window is your workspace.

3. A project contains all the files and build options needed to develop an executable output file (.out) which can be run on the DSP hardware. A project named Lab1.pjt has been created for this lab exercise. Open the project by clicking:

   Project → Open...

   and look in C:\C28x_DPS\LABS\LAB1. This project (.pjt file) will invoke all the necessary tools (compiler, assembler, linker) to build the project. It will also create a folder that will hold immediate output files.

4. In the project window on the left, click the plus sign (+) to the left of Project. Now, click on the plus sign next to Lab1.pjt. Click on the plus sign next to Source to see the current source file list.

5. A GEL file can be used to create a custom GEL menu and automate steps in CCS. A GEL file which will setup sliders has been created for this lab exercise. The slider will be used to adjust the duty, phase, and dead-band values of the waveforms. Load the PWMexplore.gel file by clicking:

   File → Load Gel...

   and look in C:\C28x_DPS\LABS\LAB1.

Device Initialization, Main, and ISR Files

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

6. Open and inspect PWMexplore-DevInit.c by double clicking on the filename in the project window. Notice that system clock, peripheral clock prescale, and peripheral clock enables have been setup. Next, notice that the shared GPIO pins have been configured.
7. Open and inspect PWMexplore-Main.c. Notice the background for(;;) loop and the case statements. This is where each of the PWM configuration functions are called for the 6 cases previously described. The case statement provides a convenient way to showcase each PWM driver quickly and interactively for demonstration purposes.

8. Open and inspect PWMexplore-ISR.asm. Notice the _ISR_Init and _ISR_Pseudo sections. This is where the PWM driver macro instantiation is done for initialization and runtime, respectively. Optionally, you can close the inspected files.

**Build and Load the Project**

9. The top four buttons on the horizontal toolbar control code generation. Hover your mouse over each button as you read the following descriptions:

<table>
<thead>
<tr>
<th>Button</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compile File</td>
<td>Compile, assemble the current open file</td>
</tr>
<tr>
<td>2</td>
<td>Incremental Build</td>
<td>Compile, assemble only changed files, then link</td>
</tr>
<tr>
<td>3</td>
<td>Rebuild All</td>
<td>Compile, assemble all files, then link</td>
</tr>
<tr>
<td>4</td>
<td>Stop Build</td>
<td>Stop code generation</td>
</tr>
</tbody>
</table>

10. 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.

11. Click the “Rebuild All” button and watch the tools run in the build window. The output file should automatically load.

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

**Debug Environment Windows**

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

13. 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 name "ConfigOption" and press enter on keyboard. Next add the following other symbol names: “Duty1”, “Duty2”. 
“Phase”, “DbLeft”, and “DbRight”. The watch window should look something like:

14. Open and setup two dual time graph windows to plot the four data log buffers A, B, C and D (ADC result registers). Click: View → Graph → Time/Frequency... and set the following values:

Select OK to save the graph options.

**Saving the Workspace Environment**

The workspace contains all of the elements that make up the current Code Composer Studio working environment. These elements include the project, project settings, configuration settings, and windows such as watch window and graphs. A workspace can be saved in a workspace file (*.wks) and reloaded at a later time. This is very useful for a subsequent Code Composer Studio session, or if a problem occurs and the tools need to be reset.

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

File → Workspace → Save Workspace As...
and saving in C:\C28x_DPS\LABS\LAB1.

When needed, a workspace can be loaded by clicking:

File → Workspace → Load Workspace...

and looking in the saved location.

**Using Real-time Emulation**

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

16. Enable real-time mode by selecting:

   Debug → Real-time Mode

17. 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.

18. The graph windows should be open. In real-time mode, we would like to have our window continuously refresh. Click:

   View → Real-time Refresh Options...

   and check “Global Continuous Refresh”. Use the default refresh rate of 100 ms and select OK. Alternately, we could have right clicked on each window individually and selected “Continuous Refresh”.

   Note: “Global Continuous Refresh” causes all open windows to refresh at the refresh rate. This can be problematic when a large number of windows are open, as bandwidth over the emulation link is limited. Updating too many windows can cause the refresh frequency to bog down. In that case, either close some windows, or disable global refresh and selectively enable “Continuous Refresh” for individual windows of interest instead.

**Run the Code – PWMexplore**

19. 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. The top graph window should display two PWM waveforms generated by the two BuckSingle macros.

20. In the watch window, the variable **ConfigOption** should be set to 1. This option or case selects the BuckSingle macro (actually there are two of them) as the active waveform generator. Change the option to 2, and examine the waveforms for the BuckDual.
Next, try the other options. Below is a list of the active PWM driver macro for each selected ConfigOption:

<table>
<thead>
<tr>
<th></th>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BuckSingle</td>
<td>(uses 2 single buck modules)</td>
</tr>
<tr>
<td>2</td>
<td>BuckDual</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MPhIL</td>
<td>(Multi-Phase Interleaved)</td>
</tr>
<tr>
<td>4</td>
<td>FullBridgePS</td>
<td>(Phase-shifted full bridge)</td>
</tr>
<tr>
<td>5</td>
<td>FullBridgeIBM</td>
<td>(IBM method Full bridge)</td>
</tr>
<tr>
<td>6</td>
<td>PFC2PhiIL</td>
<td>(2 phase Interleaved PFC)</td>
</tr>
</tbody>
</table>

21. Select the BuckSingle again (ConfigOption = 1). Open sliders D1Slider, D2Slider and TrigSlider by using GEL → PWM explore Sliders → and move the sliders into the workspace area. The D1Slider and D2slider control the duty cycle of each BuckSingle. The TrigSlider works by moving a trigger point similar to a trigger on an oscilloscope, and permits the waveform to be viewed more conveniently. Note, when adjusting the sliders the actual value in the watch window also changes. The value can be changed by directly editing the watch window, but the slider position will not be updated.

22. Next, select FullBridgePS (ConfigOption = 4). Open sliders PhaseSlider, DbLSlider, and DbRSlider (GEL → PWM explore sliders→). The PhaseSlider controls the phase relationship between the left and right legs of the full bridge. The DbLslider and DbRSlider control the deadband of left leg and right leg, respectively.

23. Fully halting the DSP when in real-time mode is a two-step process. 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. If time permits, evaluate the other PWM macro drivers. The D1Slider is used to adjust duty in ConfigOptions 3, 5, and 6. The LLdelSlider (left-leg delay) and LRdelSlider (right-leg delay) is used to adjust the delay between bottom falling edge to top rising edge for left and right full bridge legs, respectively in ConfigOption 5.

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

End of Exercise
Driving the Power Stage with PWM Waveforms

- Open-Loop System Block Diagram
- Generating PWM using the ePWM Module
- Power Stage Topologies and Software Library Support

Open-Loop System Block Diagram

Simple Open-Loop Diagram
Generating PWM using the ePWM Module

Scaleable PWM Peripherals
Each peripheral module has the same structure

- Resources allocated on a per channel basis
- Each channel (module) supports 2 independent PWM outputs (A&B)
- # Channels easily scaleable – software reuse
- Time-base synch feature for all channels
- 6 modules (12 PWM outputs) on F2808
- Key features:
  - Phase & edge control
  - New counting modes
  - Independent deadband
  - Flexible trip-zones
  - High frequency chopper mode

**ePWM Module Block Diagram**
Module Sync and Phase Control

Key Features
- Multi-event driven waveform generator
- Events drive outputs A and B independently.
- Full control on waveform polarity
- Full transparency on waveform construction
- S/W forcing events supported
- All events can generate interrupts & ADC SOC

Action Qualifier Module (AQ)

Events | Actions
--- | ---
Zero (ZRO) | Clear Lo Set Hi Toggle
TBCTR (Up) equals: CMPA (CAu) | Set Hi
Period (PRD) | Clear Lo
TBCTR (Down) equals: CMPA (CAD) | Toggle
S/W force | Clear Lo

TBCTR

---

EPWMA

EPWMB

TBCTR = Period
TBCTR = Zero
TBCTR = Compare A
TBCTR = Compare B
S/W force

TBCTR Direction

---

Period CMPB CMPA CBu CBd CAu CAD ZRO

---

C28x Digital Power Supply Workshop
**Simple Waveform Construction**

*Diagram showing PWM waveforms and related components.*

**Fault Management Support**

*Diagram illustrating fault management system with trip zones and action on fault.*

**Trip Zones:**
- 6 independent zones (TZ1~TZ6)
- Force High, Low or HiZ on trip
- One-time trip → catastrophic failure
- Cycle-by-cycle → current limit mode
- TZ1~TZ6 can trigger interrupt
Power Stage Topologies and Software Library Support

Multi-Phase Interleaved (MPI)

Switching Requirements – MPI

- Asymmetrical PWM case
- Complementary output generated by dead-band unit
- CMPB triggers ADC SOC

INIT-time
- Period (1,2,3)
- CAu Action (1,2,3)
- PRD Action (1,2,3)
- Phase (2,3)
- PRD Interrupt (1)
- CBu ADC SOC (1,2,3)
- Dead-band

RUN-time
- CMPA (1,2,3)
- CMPB (1,2,3)
Half H-Bridge (HBB)

- Master
  - Phase Reg
  - Sync In
  - Sync Out
  - CNT = Zero
  - CNT = CMPB
  - Φ = 0°
  - Phase Reg
  - Master
  - 1
  - Ext Sync In (optional)

EPWM1A
EPWM1B

V_{DC_{bus}}

V_{OUT}

Switching Requirements – HBB

- Up/Down Count
- Asymmetrical PWM
- dead-band on A only
- 50% max Modulation (controlled by CMPA)

**INIT-time**
- ZRO Action (A,B)
- CAu Action
- CBd ADC trigger
- CBd ADC trigger
- DBRED

**RUN-time**
- CMPA
- CMPB (optional)

Compare A modulation range:
0 < CMPA < (PRD – ½ x DBRED)
Phase Shifted Full Bridge (PSFB)

Switching Requirements – PSFB

- Asymmetrical PWM
- Using dead-band module
- Phase (F) is the control variable
- Duty fixed at ~ 50%
- RED / FED control ZVS trans. i.e. via resonance
- CMPB can trigger ADC SOC

**INIT-time**

- Period (1,2)
- CMPA (1,2) ~ 50%
- CAu action (1,2)
- ZRO action (1,2)
- CBu trigger for ADC SOC

**RUN-time**

- Phase (2) – every cycle
- FED / RED (1,2) – slow loop
Lab2: 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 without a requirement for high speed feedback, the ADC will be used to measure various values for instrumentation purposes and will be displayed using CCS. The PWM duty cycle will be adjusted using watch windows or sliders. The Digital Power software framework, associated files, and library modules will be used.

**Project Overview**

Lab exercises 2, 3, and 4 use the TwoChannel project. It makes use of the “C-background/ASM-ISR” framework. In Lab1 various PWM waveforms were generated using the EPWM modules 3, 4, and 5. The PWM outputs on the workshop EVM were not connected to power stages, but were looped back as inputs to the ADC. In lab exercises 2, 3, and 4 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:

TwoChannel-Main.c – this file is used to initialize, run, and manage the application. This is the “brains” behind the application.

TwoChannel-DevInit.c – this file is responsible for a one time initialization and configuration of the F280x device, and includes functions such as setting up the clocks, PLL, GPIO, etc.
TwoChannel_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 _ISR_Run routine executes at the same rate as the PWM timebase which is used to trigger it.

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

<table>
<thead>
<tr>
<th>C configure function</th>
<th>ASM initialization macro</th>
<th>ASM Run time macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuckSingle_CNFire</td>
<td>BuckSingle_DRV_INIT n</td>
<td>BuckSingle_DRV n</td>
</tr>
<tr>
<td>ADC_CascSeqCNF()</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

The workshop Power EVM consists of two identical buck power stages. The input bus voltage for both stages is 12V. Shown below is a diagram of the Power EVM and some key features.

12V In          DC power supply from plug pack
Main Pwr (SW1)  SW1 - Master power switch for entire EVM
DC Bus (SW2)    SW2 - Power switch for Vin to buck stages only and when off F2808 DIMM controller card still operates (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, 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)

V1/V2 Select (SW3)
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 converters)
V1/V2 Select        SW3 - selects between output voltage of buck 1 and 2
Comms               Serial communications UART (optional for user, not used in lab exercises)

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.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Description</th>
<th>Connection to F2808</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPWM-1A</td>
<td>PWM Duty control signal for buck stage 1</td>
<td>GPIO-00</td>
</tr>
<tr>
<td>EPWM-2A</td>
<td>PWM Duty control signal for buck stage 2</td>
<td>GPIO-02</td>
</tr>
<tr>
<td>VoutFB-1</td>
<td>Voltage feedback for buck stage 1</td>
<td>ADC-B0</td>
</tr>
<tr>
<td>VoutFB-2</td>
<td>Voltage feedback for buck stage 2</td>
<td>ADC-A0</td>
</tr>
<tr>
<td>Iout-1</td>
<td>Current monitor / measurement buck stage 1</td>
<td>ADC-B1</td>
</tr>
<tr>
<td>Iout-2</td>
<td>Current monitor / measurement buck stage 2</td>
<td>ADC-A1</td>
</tr>
<tr>
<td>Temp-1</td>
<td>Temperature monitor / measurement buck stage 1</td>
<td>ADC-B2</td>
</tr>
<tr>
<td>Temp-2</td>
<td>Temperature monitor / measurement buck stage 2</td>
<td>ADC-A3</td>
</tr>
<tr>
<td>Ifault-1</td>
<td>Over-Current flag, digital output from buck stage 1</td>
<td>GPIO-01</td>
</tr>
<tr>
<td>Ifault-2</td>
<td>Over-Current flag, digital output from buck stage 2</td>
<td>GPIO-03</td>
</tr>
</tbody>
</table>
Lab Exercise Overview

The software in Lab2 has been configured to independently adjust the duty cycle of EPWM-1A and EPWM-2A. “Net” variable names Duty1 and Duty2 have been declared and “connected” to the inputs of BuckSingle_DRV macro. Using either the watch window or the appropriate slider, Duty1 and Duty2 can be directly adjusted. Below is the system diagram for Lab2.

In Lab2 (as well as lab exercises 3 and 4) 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, 4 ms, and 8 ms respectively. Within each group (e.g. “B”) each task is run in a “round-robin” manner. For example, group B executes every 4 ms, and there are 3 tasks in group B. Therefore, B1, B2, and B3 execute once every 12 ms. System dashboard measurements are conveniently done by group 3 tasks (i.e. B1 – voltage measurement, B2 – current measurement, and B3 – temperature measurement).
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. A project named Lab2.pjt has been created for this lab exercise. Open the project by clicking:

   Project → Open...

   and look in C:\C28x_DPS\LABS\LAB2.

3. Load the TwoChannel.gel file by clicking:

   File → Load Gel...

   and look in C:\C28x_DPS\LABS\LAB2.

Device Initialization, Main, and ISR Files

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

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

5. Open and inspect TwoChannel-Main.c. Notice the incremental build option 1 (i.e. IB1). A section of code is shown here for convenience. Comments have been added in italics. Note that the run-time macros are executed at the PWM rate of 300 kHz.

```c
//=============================================================
#elif (IB1)  // Open loop - Channels 1,2
//=============================================================
#define  prd  333 // Period count = 300 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  TempR1  AdcMirror.ADCRESULT4 //
#define  TempR2  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
EPwm1Regs.CMPB = 2; // ISR trigger point
ISR_Init(); // ASM ISR init

Duty1 and Duty2 variables will directly control the buck duty cycle. Sliders will be used to quickly change these values.
Duty1 = 0x0;
Duty2 = 0x0;

// Module connection to "nets" done here
//----------------------------------------
// BUCK_DRV connections
Buck_In1 = &Duty1;
Buck_In2 = &Duty2;
#endif // (IB1)

6. Open and inspect TwoChannel-ISRa.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 IB1 is used.

.if(IB1) ; Init time
   BuckSingle_DRV_INIT 1 ; EPWM1A
   BuckSingle_DRV_INIT 2 ; EPWM2A
.endif

.if(IB1) ; Run time
   BuckSingle_DRV 1 ; EPWM1A
   BuckSingle_DRV 2 ; EPWM2A
.endif

7. Optionally, you can close the inspected files.

**Build and Load the Project**

8. Click the “Rebuild All” button and watch the tools run in the build window. The output file should automatically load.

9. 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**

10. 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 “Type” and “Radix” as needed.
The following table gives a description for the variable names:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VinMeas</td>
<td>Voltage input measurement (i.e. DC bus) to each buck power stage</td>
</tr>
<tr>
<td>Vmeas[1]</td>
<td>Voltage output of each channel, 3 element array, zeroth element not used</td>
</tr>
<tr>
<td>Imeas[1]</td>
<td>Current output of each channel, 3 element array, zeroth element not used</td>
</tr>
<tr>
<td>TdegC[1]</td>
<td>Temperature of each power module, 3 element array, zeroth element not used</td>
</tr>
<tr>
<td>Duty1</td>
<td>Q15 value (0~7FFFh) for duty input to BuckSingle_DRV1</td>
</tr>
<tr>
<td>Duty2</td>
<td>Q15 value (0~7FFFh) for duty input to BuckSingle_DRV2</td>
</tr>
</tbody>
</table>

Note 7FFFh = 32,768 = 100% duty

Save the Workspace

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

File ➔ Workspace ➔ Save Workspace As...

and saving in C:\C28x_DPS\LABS\LAB2.

Run the Code – TwoChannel

12. Enable real-time mode by selecting:

Debug ➔ Real-time Mode

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

View → Real-time Refresh Options...

and check “Global Continuous Refresh”.

15. 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.

16. 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.

17. Turn on the 12-volt DC bus (SW2) on the 2-channel buck EVM and observe variable VinMeas in the watch-window. It should now be approximately 12V.

18. Open sliders D1Slider and D2Slider by using GEL → 2-Channel Sliders → and move the sliders into the workspace area. D1Slider and D2Slider are used to change variables Duty1 and Duty2, respectively. Increase the value of Duty1 to approximately 2800 (decimal). 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 the load resistor (1Ω) connected to terminal 1, the open-loop voltage for Channel 1 is approximately given by:

<table>
<thead>
<tr>
<th>Voltage (V)</th>
<th>Duty (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2800</td>
</tr>
<tr>
<td>2</td>
<td>5600</td>
</tr>
<tr>
<td>3</td>
<td>8400</td>
</tr>
</tbody>
</table>

19. Try the same adjustment on Duty2. Be sure SW3 on the EVM is positioned to select Ch2. Note that Channel 2 buck is only lightly loaded with a lamp (2~3Ω) and hence a slightly lower Duty2 value will give the same output voltage as in the Ch1 case.

20. Of general interest – during duty/voltage adjustments observe the various watch window variables such as voltage, current and temperature. Vmeas 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.

21. Turn off the 12-volt DC bus (SW2) on the 2-channel buck EVM. (Do not turn off the main power SW1).

22. 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.

23. In the project window right click on Lab2.pjt and select Close.
24. Do Not Close Code Composer Studio or it will be necessary to setup the debug environment windows again for the next lab exercise!

End of Exercise
3 – Controlling the Power Stage with Feedback

Controlling the Power Stage with Feedback

- Closed-Loop System Block Diagram
- Analog-to-Digital Converter Module
- Digital Control of Power Converter
- High Resolution PWM Benefits
- Soft Start – Starting the Loop

Closed-Loop System Block Diagram

The “Closed-Loop”
ADC Module Block Diagram

Digital Control of Power Converter

\[ G_c(z) = \frac{U(z)}{E(z)} \frac{B_z + B_z z^{-1} + B_z z^{-2}}{1 - A_z z^{-1} - A_z z^{-2}} \]

\[ U(n) = B_z E(n) + B_z E(n-1) + B_z E(n-2) + A_z U(n-1) + A_z U(n-2) \]
### Digital Control of Power Converter

**Steady State Limit Cycle**

- **Vref**
- **ΔVc**
- **ΔVs**
- **Vo levels (DPWM duty ratio steps)**

#### Understanding the Diagram:
- **Error bins** are indicated for ADC levels.
- The steady state output is shown with and without limit cycles.

### High Frequency PWM

**PWM resolution** is given by

$$ PWM\ resolution = \log_2 \left( \frac{T_{PWM}}{T_{SysClk}} \right) $$

#### PWM Resolution Table (F2808 – SysClk = 100 MHz)

<table>
<thead>
<tr>
<th>PWM Freq (kHz)</th>
<th>Regular resolution (bits) (%)</th>
<th>High resolution (bits) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>16.0</td>
<td>16.0</td>
</tr>
<tr>
<td>150</td>
<td>15.4</td>
<td>15.4</td>
</tr>
<tr>
<td>250</td>
<td>14.7</td>
<td>14.7</td>
</tr>
<tr>
<td>500</td>
<td>13.7</td>
<td>13.7</td>
</tr>
<tr>
<td>750</td>
<td>13.1</td>
<td>13.1</td>
</tr>
<tr>
<td>1000</td>
<td>12.7</td>
<td>12.7</td>
</tr>
<tr>
<td>1500</td>
<td>12.1</td>
<td>12.1</td>
</tr>
<tr>
<td>2000</td>
<td>11.7</td>
<td>11.7</td>
</tr>
</tbody>
</table>
High Resolution PWM (HRPWM)

- Significantly increases the resolution of conventionally derived digital PWM
- Uses 8-bit extensions to Compare registers (CMPxHR) and Phase register (TBPHSHR) for edge positioning control
- Typically used when PWM resolution falls below ~9-10 bits which occurs at frequencies greater than ~200 kHz (with system clock of 100 MHz)
- Not all ePWM outputs support HRPWM feature (see device data manual)

Resolution Loss – Low Duty Utilization

<table>
<thead>
<tr>
<th>Vin</th>
<th>0.8</th>
<th>1</th>
<th>1.2</th>
<th>1.8</th>
<th>2.5</th>
<th>3.3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>94%</td>
<td>93%</td>
<td>91%</td>
<td>87%</td>
<td>82%</td>
<td>76%</td>
<td>64%</td>
</tr>
<tr>
<td>12</td>
<td>93%</td>
<td>92%</td>
<td>90%</td>
<td>85%</td>
<td>79%</td>
<td>73%</td>
<td>58%</td>
</tr>
<tr>
<td>10</td>
<td>92%</td>
<td>90%</td>
<td>88%</td>
<td>82%</td>
<td>75%</td>
<td>67%</td>
<td>50%</td>
</tr>
<tr>
<td>9</td>
<td>91%</td>
<td>89%</td>
<td>87%</td>
<td>80%</td>
<td>72%</td>
<td>63%</td>
<td>44%</td>
</tr>
<tr>
<td>8</td>
<td>90%</td>
<td>88%</td>
<td>85%</td>
<td>78%</td>
<td>69%</td>
<td>59%</td>
<td>38%</td>
</tr>
<tr>
<td>7</td>
<td>89%</td>
<td>86%</td>
<td>83%</td>
<td>74%</td>
<td>64%</td>
<td>53%</td>
<td>29%</td>
</tr>
<tr>
<td>6</td>
<td>87%</td>
<td>83%</td>
<td>80%</td>
<td>70%</td>
<td>58%</td>
<td>45%</td>
<td>17%</td>
</tr>
</tbody>
</table>
High-Resolution PWM Benefits

Benefit of High Resolution PWM

HiRes PWM (150ps) vs Regular PWM (10ns)

No Limit cycle
Edge control is precise

Limit cycle problem
Edge jumps around

Managing the “Closed-Loop”

Fault Trip Dead Band

Control “2P2Z”
Ref FB Uout

Duty Clamp

Open/Closed Loop

Feedback
Simple User Interface Control

Supervisory - BG

Fault Trip
Dead Band
Open/Closed Loop
Duty Clamp
Vref

Control Engine(s)

Ref
Vout

Control
"2P2Z"

Vset
Duty
Voltage
Feedback

Coefficients

Coeff - B2
Coeff - B1
Coeff - B0
Coeff - A2
Coeff - A1

Coeff set 1
Coeff set 2
Coeff set 3

Vout Monitor
Duty Monitor

Soft Start – Starting the Loop

Soft-Start and Sequencing Multi \( V_{\text{out}} \)
Lab3: 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 and sliders. ADC management for high-speed feedback and slow instrumentation will be utilized. The Digital Power software framework, associated files, and library modules will be used.

**Lab3: Closed-Loop Control**

- Regulate the Buck output by using Voltage Mode Control (VMC) with closed-loop feedback
- Soft-start and sequencing function used to ensure an “orderly” voltage ramp-up/down
- Soft-start profile and target voltage is conveniently adjusted by using the CCS watch window and slider buttons feature

**Project Overview**

The following Power Library modules will be used in this lab exercise. (Note: these are the same library modules used in Lab2 exercise with the addition of other library modules).

<table>
<thead>
<tr>
<th>C configure function</th>
<th>ASM initialization macro</th>
<th>ASM Run time macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuckSingle_CNF()</td>
<td>BuckSingleHR_DRV_INIT n</td>
<td>BuckSingleHR_DRV n</td>
</tr>
<tr>
<td>ADC_DualSeqCNF()</td>
<td>ADC_NchDRV_INIT n</td>
<td>ADC_NchDRV n</td>
</tr>
<tr>
<td>none</td>
<td>ControlLaw_2P2Z_INIT n</td>
<td>ControlLaw_2P2Z n</td>
</tr>
<tr>
<td>none</td>
<td>DataLogTST_INIT n</td>
<td>DataLogTST n</td>
</tr>
</tbody>
</table>
Below is a description and notes for the Power Library modules used in this lab exercise.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BuckSingleHR_DRV</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ADC_NchDRV</strong></td>
<td>Reads $1^\text{st}$ N ADC result registers every PWM cycle and stores to N consecutive memory locations accessible by C. In Lab3, N=1 (i.e. a single voltage is measured as feedback).</td>
</tr>
<tr>
<td><strong>ControlLaw_2P2Z</strong></td>
<td>This is a 2$^\text{nd}$ 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.</td>
</tr>
<tr>
<td><strong>DataLogTST</strong></td>
<td>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 Lab4 the data logger will be useful in displaying an output voltage transient.</td>
</tr>
</tbody>
</table>

### Lab Exercise Overview

The software in Lab3 has been configured to provide closed-loop voltage control for Channel 1 of the buck EVM. Additionally, datalogging of the output can be displayed in a CCS graph window. Below is the system diagram for Lab3.
The closed-loop consists of only three modules – ADC_1ChDRV, CNTL_2P2Z, and BuckSingleHR_DRV. When the code is running 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 TwoChannel-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 Lab3, the target voltage, slew-rate and delay-on/off parameters are conveniently modified via a watch window. A slider can also be used to adjust the output target voltage by “connecting” it to the Vsoft variable. 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.

<table>
<thead>
<tr>
<th>Vsoft[n]</th>
<th>Desired output target voltage in Q15 format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. Vsoft[2]=4000, set channel 2 output voltage to 4000 “units”</td>
</tr>
<tr>
<td>ChannelEnable[n]</td>
<td>Enable (allow) voltage output to reach target value</td>
</tr>
<tr>
<td></td>
<td>e.g. ChannelEnable[3]=1, turn channel 3 “on”</td>
</tr>
<tr>
<td></td>
<td>e.g. ChannelEnable[2]=0, turn channel 2 “off”</td>
</tr>
<tr>
<td>SlewStep[n]</td>
<td>Step size or rate at which the target voltage is ramped to</td>
</tr>
<tr>
<td></td>
<td>e.g. SlewStep[2]=15, increment or decrement by 15 units at every call</td>
</tr>
<tr>
<td>OnDelay[n]</td>
<td>Delay time to turn on from the “global” start command (StartUp=1)</td>
</tr>
<tr>
<td></td>
<td>e.g. OnDelay[3]=2000, start channel 3 after delay of 2000 “time units”</td>
</tr>
<tr>
<td>OffDelay[n]</td>
<td>Delay time to turn off from the “global” stop command (StartUp=0)</td>
</tr>
<tr>
<td></td>
<td>e.g. OffDelay[3]=1000, stop channel 3 after delay of 1000 “time units”</td>
</tr>
<tr>
<td>StartUp</td>
<td>Global turn on/off command. Used to synchronize/sequence all channels</td>
</tr>
<tr>
<td></td>
<td>e.g. StartUp=1, global turn on command</td>
</tr>
<tr>
<td></td>
<td>e.g. StartUp=0, global turn off command</td>
</tr>
</tbody>
</table>
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 debug environment from the previous lab exercise.

2. A project named Lab3.pjt has been created for this lab exercise. Open the project by clicking:

   `Project → Open…`

   and look in `C:\C28x_DPS\LABS\LAB3`.

   The TwoChannel.gel file and watch window should still be loaded from the previous lab.

Device Initialization, Main, and ISR Files

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

3. Open and inspect `TwoChannel-Main.c` by double clicking on the filename in the project window. Notice the incremental build option 2 (i.e. IB2). A section of code is shown here for convenience. Comments have been added in italics.

```c
//=================================================================
#if (IB2) // Closed Loop Ch-1, with SoftStart using separate lib blocks
//=================================================================
#define prd  400 // Period count = 250 KHz @ 100 MHz
#define NumActvCh 1 // Number of Active Channels

// "Raw" (R) ADC measurement name defines
#define  VoutR1  AdcMirror.ADCRESULT0 //
#define  VoutR2  AdcMirror.ADCRESULT8 //
#define  IoutR1  AdcMirror.ADCRESULT9 //
#define  IoutR2  AdcMirror.ADCRESULT10 //
#define  TempR1  AdcMirror.ADCRESULT11 //
#define  TempR2  AdcMirror.ADCRESULT12 //
#define  VinR   AdcMirror.ADCRESULT13 //

Soft-Start parameters for channel 1
OnDelay[1] = 0;
OffDelay[1] = 0;
Vsoft[1] = 10900;  // 1.8 V
SlewStep[1] = 200;

Used for Scope feature via Graph window
DataLogTrigger = 980000;
ScopeGain = 1;
ScopeACmode = 0;     // DC mode initially

ADC Sequencer 1 - VoutR1 used every PWM cycle
// Channel Selection for Sequencer-1
ChSel[0] = 8;     // B0 - Vout1

ADC Sequencer 2 - Instrumentation only, round robin scheme
// Channel Selection for Sequencer-2
ChSel[8] = 0;    // A0 - Vout2
ChSel[9] = 9;    // B1 - Iout1
ChSel[10] = 1;   // A1 - Iout2
```
Lab3: Closed-Loop Control

ChSel[12] = 2;  // A2 - Temperature-2
ChSel[13] = 11; // B3 - Vin

PWM and ADC configure functions

BuckSingle_CNF(1, prd, 1, 0);   // ePWM1, Period=prd, Master, Phase=0
ADC_DualSeqCNF(ChSel, 1, 1, 1); // ACQPS=1, Seq1#Conv=1, Seq2#Conv=1
EPwm1Regs.CMPB = 193;   // tCMPB1 - ISR trigger point
ISR_Init();           // ASM ISR init

// Lib Module connection to "nets"
//------------------------------
// ADC1CH_DRV connections
ADC_Rslt = &Vfdbk;
// CNTL_2P2Z connections
CNTL_2P2Z_Ref1 = &VrefNetBus[1];   // point to Vref from SlewRate limiter
CNTL_2P2Z_Out1 = &Uout; // point to Uout
CNTL_2P2Z_Fdbk1 = &Vfdbk;  // point to Vfdbk
CNTL_2P2Z_Coef1 = &Coef2P2Z[0];    // point to first coeff for Single Loop

// BUCK_DRV connections
Buck_In1 = &Uout;

Datalogger is an optional feature, not required for loop to run
// Data Logger connections, DLTST = DataLogTimeStampTrigger
DLTST_In1 = &Vfdbk;
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
$/)endif // (IB2)

4. Open and inspect TwoChannel-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 Lab3, incremental build option IB2 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.

.if(IB2) ; Init time
ADC_NchDRV_INIT 1 ; 1 Channel, N-1
ControlLaw_2P2Z_INIT 1
BuckSingleHR_DRV_INIT 1 ; EPWM1A
DataLogTST_INIT 1 ; 1 Channel Data logger
.endif

.if(IB2) ; Run time
ADC_NchDRV 1 ; 1 Channel, N-1  ; Measure
ControlLaw_2P2Z 1 ; Compensate
BuckSingleHR_DRV 1 ; Update
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

52 C28x Digital Power Supply Workshop
5. Optionally, you can close the inspected files.

**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 “Type” and “Radix” as needed.

![Watch Window](image)

**Note:** ScopeGain, ScopeACmode, and ActiveLoad will be explained and used in the next lab exercise.
The following table gives a description for the variable names:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelEnable</td>
<td>Channel enable array</td>
</tr>
<tr>
<td>Vsoft</td>
<td>Voltage target array</td>
</tr>
<tr>
<td>OnDelay</td>
<td>DelayOn array</td>
</tr>
<tr>
<td>OffDelay</td>
<td>DelayOff array</td>
</tr>
<tr>
<td>SlewStep</td>
<td>Ramp step size array</td>
</tr>
<tr>
<td>StartUp</td>
<td>Global turn-on/turn-off integer flag</td>
</tr>
</tbody>
</table>

9. Open and setup a time graph windows to plot the data log buffer (ADC result register). 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 Lab3.wks and clicking:

File ➔ Workspace ➔ Save Workspace As...

and saving in C:\C28x_DPS\LABS\LAB3.
Run the Code – TwoChannel

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 12-volt DC bus (SW2) on the 2-channel buck EVM and observe variable VinMeas in the watch-window. It should now be approximately 12V. Also the Graph window should show a trace hovering at “0”.

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 OnDelay and OffDelay parameters since synchronization to a global trigger is not utilized.

17. Open slider V1softSlider (GEL → 2-Channel Sliders →) and move the slider into the workspace area. This slider is used to directly change Vsoft[1]. Increase the value of Vsoft[1] to approximately 10900 (decimal). Power stage buck 1 module output voltage should be approximately 1.8V. When you are done turn off Channel 1 by setting ChannelEnable[1]=0.

18. Channel 1 can also be enabled by using the global turn-on flag – StartUp. In this case OnDelay and OffDelay parameters are used. Both of these delays are set to zero by default, but can be modified via the watch-window. For example, modify these values as follows:

   OnDelay[1]=1000
   StartUp=1

   This will trigger a global turn on and Channel 1 will start ramping up after 1000 time units. Using StartUp=0 will trigger a global turn off and Channel 1 will ramp down after 2000 time units.

19. The ramp-up and ramp-down rates can also be modified. In this lab code, up and down
Lab3: Closed-Loop Control

rates are the same and set by parameter SlewStep[1] for channel 1. The default value in Lab3 is 200 units per step. Change it to 40 in the watch window and see the result. Follow these steps:

\[
\begin{align*}
\text{SlewStep}[1] &= 40 \\
\text{ChannelEnable}[1] &= 1 \quad \text{Output should ramp up at slower rate} \\
\text{ChannelEnable}[1] &= 0 \quad \text{Output should ramp down at slower rate}
\end{align*}
\]

20. Turn off the 12-volt DC bus (SW2) on the 2-channel buck EVM. *Do not turn off the main power SW1.*

21. 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 Æ Halt. Then click Debug Æ Real-time Mode and uncheck the “Real-time mode” to take the DSP out of real-time mode.

22. In the project window right click on Lab3.pjt and select Close.

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

End of Exercise
4 – Tuning the Loop for Good Transient Response

Tuning the Loop for Good Transient Response

- Digital Power Supply Control Theory
- Intuitive Loop Tuning – “Visually without Math”
- Active Load Feature of the Power EVM
- Multi-Loop Control

Digital Power Supply Control Theory

The Digital Control System

Advantages
- Immunity from environmental effects
- Advanced control strategies possible
- Immunity from component errors
- Improved noise immunity
- Ability to modify and store control parameters
- Ability to implement digital communications
- System fault monitoring and diagnosis
- Data logging capability
- Ability to perform automated calibration

Considerations
- Sample rate
- Quantization
- Ease of programming
- Controller design
- Cost
- Processor selection
- Requires data converters
- Numeric issues
4 – Tuning the Loop for Good Transient Response

**PID Control Review**

\[ u(t) = K_P e(t) + K_I \int e(t) \, dt + K_D \frac{de(t)}{dt} \]

- **Proportional term** controls loop gain
- **Integral action** increases low frequency gain and reduces/eliminates steady state errors
- **Derivative action** adds phase lead which improves stability and increases system bandwidth

\[ G_C(s) = \frac{K_P + K_I s + K_D s^2}{s} \]

Usually written in “parallel” form:

\[ G_C(s) = \frac{1 + \frac{1}{T_1 s} + T_2 s}{s} \]

- \( K_P = K_C \)
- \( K_I = K_C T_1 \)
- \( K_D = K_C T_2 \)

**Tuning the Step Response**

- Performance of the control loop can be determined from the output response to a step change in load
- We will adjust PID coefficients to minimise deviation from steady state and settling time of the output voltage

![Transient Response Plot](image-url)

- Steady state
- Peak deviation
- Acceptable error
- Settling time
Intuitive Loop Tuning – “Visually without Math”

Loop Tuning – Good First Step

\[ U(n) = B_0 E(n) + B_1 E(n-1) + B_2 E(n-2) + A_1 U(n-1) + A_2 U(n-2) \]

PID – Intuitive / Interactive

We can also write the controller in transfer function form:

\[
\frac{U(z)}{E(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - z^{-1}} = \frac{B_0 z^2 + B_1 z + B_2}{z^2 - z}
\]

Compare with the General 2P2Z transfer function:

\[
\frac{U(z)}{E(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 + A_1 z^{-1} + A_2 z^{-2}} = \frac{B_0 z^2 + B_1 z + B_2}{z^2 + A_1 z + A_2}
\]

We can see that PID is nothing but a special case of 2P2Z control where:

\[ A_1 = -1 \quad \text{and} \quad A_2 = 0 \]

Change PID coeff. “on fly” in background loop

```c
// Coefficient init
Conf2P2Z_1[0] = Dgain * 67108; // B2
Conf2P2Z_1[1] = (Igain - Pgain - Dgain - Dgain)*67108; // B1
Conf2P2Z_1[2] = (Pgain + Igain + Dgain)*67108; // B0
Conf2P2Z_1[3] = 0; // A2
Conf2P2Z_1[4] = GT108864; // A1
Conf2P2Z_1[6] = 0x00000000; // Clamp Lo
```
Control Law Computation

\[ U(n) = B_2E(n) + B_1E(n-1) + B_0E(n-2) + A_2U(n-1) + A_1U(n-2) \]

### Voltage control law

\[
\begin{align*}
\text{MOVU} & \quad \text{ACC}, \text{Vref} \\
\text{SUBU} & \quad \text{ACC}, \star \text{XAR2}++ \\
\text{LSL} & \quad \text{ACC}, \#8 \quad \text{acc} = e(n) \quad \text{(Q24)} \\
\text{MOVL} & \quad \text{@VCNTL_DBUFF+4, ACC} \\
\text{ZAPA} & \quad \text{Voltage control law} \\
\text{MOVL} & \quad \text{XT, @VCNTL_DBUFF+8} \quad \text{XT} = e(n-2) \\
\text{QMPYAL} & \quad P, XT, \star \text{XAR7}++ \quad b_2e(n-2) \\
\text{MOVDL} & \quad XT, @VCNTL_DBUFF+6 \quad XT = e(n-1), e(n-2) = e(n-1) \\
\text{QMPYAL} & \quad P, XT, \star \text{XAR7}++ \quad ACC = b_2e(n-2), P = b_1e(n-1) \\
\text{MOVL} & \quad XT, @VCNTL_DBUFF+2 \quad XT = u(n-2) \\
\text{QMPYAL} & \quad P, XT, \star \text{XAR7}++ \quad P = a_2u(n-2) \\
\text{MOVL} & \quad XT, @VCNTL_DBUFF \quad XT = u(n-1), u(n-2) = u(n-1) \\
\text{QMPYAL} & \quad P, XT, \star \text{XAR7}++ \quad ACCa_2u(n-2) \\
\text{ADDL} & \quad ACC, P \quad ACC = a_2u(n-2) + a_1u(n-1) \\
\text{LSL} & \quad ACC, \#(23 - \text{VCNTL} \_QF + 8) \quad \text{(Q23)} \\
\text{ADLB} & \quad ACC, ACC \quad \text{(Q24)} \\
\text{MOVL} & \quad @VCNTL_DBUFF, ACC \quad ACC = u(n) \\
\text{MINL} & \quad ACC, \star \text{XAR7}++ \quad \text{Saturate the result [Min,Max]} \\
\text{MAXL} & \quad ACC, \star \text{XAR7}++ \\
\text{Duty Cycle Modulation} \\
\text{MOVL} & \quad XT, ACC \\
\text{QMPYL} & \quad P, XT, \star \text{XAR7}++ \quad \text{(Q0)} \\
\text{MOV} & \quad \star \text{XAR3}++, P
\end{align*}
\]
4 – Tuning the Loop for Good Transient Response

**Type II Controller**

\[
G_c(s) = \frac{1}{RC_1} \left( \frac{s + \frac{1}{RC_2}}{s + \frac{C_1 + C_2}{RC_2}} \right)
\]

- \( R_1 = 4.12\Omega \)
- \( R_2 = 124\Omega \)
- \( C_1 = 8.2pF \)
- \( C_2 = 2.2nF \)

---

**Digital Type II Controller**

\[
G_c(z) = \frac{B_2 + B_1z^{-1} + B_0z^{-2}}{1 + A_1z^{-1} + A_0z^{-2}}
\]

- \( B_2 = 9.927 \)
- \( B_1 = 0.03632 \)
- \( B_0 = 9.891 \)
- \( A_1 = 1.339 \)
- \( A_0 = 0.3391 \)

(Tustin’s transform, \( T_s = 1\ mu s \))
Type III Controller

\[ G(s) = \frac{R_1 + R_2}{R_1 C_1} \left( s + \frac{1}{R_1 C_1} \right)^2 + \frac{1}{R_1 C_2} \left( s + \frac{1}{R_1 C_2} \right)^2 \]

\[ R_1 = 4.12\Omega \]
\[ R_2 = 20.5\Omega \]
\[ R_3 = 150\Omega \]
\[ C_1 = 0.22nF \]
\[ C_2 = 2.7nF \]
\[ C_3 = 6.8nF \]

Digital Type III Controller

\[ G(z) = \frac{B_1 + B_2 z^{-1} + B_3 z^{-2} + B_4 z^{-3}}{1 + A_1 z^{-1} + A_2 z^{-2} + A_3 z^{-3}} \]

\[ B_1 = 9.658 \]
\[ B_2 = 9.158 \]
\[ B_3 = 9.652 \]
\[ B_4 = 9.164 \]
\[ A_1 = 2.128 \]
\[ A_2 = 1.397 \]
\[ A_3 = 0.2689 \]

(Tustin's transform, \( T_s = 1 \) us)
Active Load Feature of the Power EVM

2-Channel Buck EVM

TI Power-Train
PTD08A010W
10A module
• Current meas.
• Temp meas.
• Over Current Prot.
• Over Current Flag
• No Heat-sink needed

2-Channel Buck EVM Schematic
Lab4: 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 sliders. 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.

Project Overview

The software code used in Lab4 is exactly the same code as used in Lab3. All of the files and build options are identical. The five coefficients to be modified are stored in the array Coef2P2Z[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 Lab3 provides a convenient way to
observe the output transient while the built-in active load on the EVM can provide the periodic
disturbance.

The compensator block (macro) used is CNTL_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 function 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
CNTL_2P2Z control where:

\[
a_1 = -1 \quad \text{and} \quad a_2 = 0
\]

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

**Lab Exercise Overview**

In Lab3 the software has been configured to provide closed-loop voltage control for Ch1 of the
buck EVM and datalogging of the output which was displayed in a CCS graph window. Lab4
will additionally allow modification of the five coefficients associated with the 2nd order
CNTL_2P2Z compensator block. This modification will be done “on the fly” by using 3 sliders
(P, I and D) while the buck output is put under transient using an active load which is switched
periodically by the ECAP peripheral.

The following figure is the system diagram for Lab4.
The default coefficient settings chosen for Lab3 provide very poor performance (low gains). Initially Lab4 will use the same settings. The control loop will be soft-started to the target Vout value, the same way it was done in Lab3. 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pgain</td>
<td>Proportional gain; value adjustment: 0 ~ 1000</td>
</tr>
<tr>
<td>Igain</td>
<td>Integral gain; value adjustment: 0 ~ 1000</td>
</tr>
<tr>
<td>Dgain</td>
<td>Derivative gain; value adjustment: 0 ~ 1000</td>
</tr>
<tr>
<td>ActiveLoad</td>
<td>Enable (value=1) / Disable (value=0) flag for the active load circuit</td>
</tr>
<tr>
<td>ScopeACmode</td>
<td>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</td>
</tr>
<tr>
<td>ScopeGain</td>
<td>Vertical gain adjustment for the CCS scope (much like a real oscilloscope)</td>
</tr>
</tbody>
</table>
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 debug environment from the previous lab exercise.*

2. A project named Lab4.pjt has been created for this lab exercise. Open the project by clicking:

   Project → Open...

   and look in C:\C28x_DPS\LABS\LAB4.

   The TwoChannel.gel file, watch windows and graph window should still be loaded from the previous lab.

Device Initialization, Main, and ISR Files

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

3. Open and inspect TwoChannel-Main.c by double clicking on the filename in the project window. This file is identical to the one used in Lab3. Notice the coefficient initialization and P, I, and D mapping equation. A section of code is shown here for convenience.

   \[
   \begin{align*}
   \text{Pgain} &= 1; \quad \text{Igain} = 1; \quad \text{Dgain} = 5; \quad \text{\textit{ // very "loose"}} \\
   \text{Coef2P2Z[0]} &= \text{Dgain} \times 67108; \quad \text{\textit{ // B2}} \\
   \text{Coef2P2Z[1]} &= (\text{Igain} - \text{Pgain} - \text{Dgain} - \text{Dgain}) \times 67108; \quad \text{\textit{ // B1}} \\
   \text{Coef2P2Z[2]} &= (\text{Pgain} + \text{Igain} + \text{Dgain}) \times 67108; \quad \text{\textit{ // B0}} \\
   \text{Coef2P2Z[3]} &= 0; \quad \text{\textit{ // A2}} \\
   \text{Coef2P2Z[4]} &= 67108864; \quad \text{\textit{ // A1 = 1 in Q26}} \\
   \text{Coef2P2Z[5]} &= \text{Dmax[1]} \times 67108; \quad \text{\textit{ // Clamp Hi limit (Q26)}} \\
   \text{Coef2P2Z[6]} &= 0x00000000; \quad \text{\textit{ // Clamp Lo}}
   \end{align*}
   \]

   (Note: 67108 \approx 0.001 in Q26 format)

4. Optionally, you can close the inspected file.

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 Lab4.wks and clicking:

File → Workspace → Save Workspace As...

and saving in C:\C28x_DPS\LABS\LAB4.

Run the Code – TwoChannel

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 12-volt DC bus (SW2) on the 2-channel buck EVM and observe variable VinMeas in the watch-window. It should now be approximately 12V. 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:
15. Open the sliders for P, I and D adjustment (GEL → 2-Channel Sliders →) Pgain_Slider, Igain_Slider and Dgain_Slider. Optionally, if you want to adjust the output voltage via a slider, then open VsoftSlider, too. Move the sliders into the workspace area.

16. By observing the transient in real time, gradually adjust each slider to get the best transient response (i.e. least perturbation from the zero line). Gradual adjustment can be best achieved by selecting the slider (mouse click) and then using the up/down arrows. A large movement of the slider may cause the system to go unstable. A suggested procedure is given below:

- Start with Igain first, increase gradually 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 is achieved – this may look something like the graph shown below:
17. Reduce the scope vertical gain back to 1 (\(\text{ScopeGain} = 1\)) and set the graph back in DC mode (\(\text{ScopeACmode} = 0\)). The voltage output response should look something like this now:

![](image)

18. 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.

19. Turn off the 12-volt DC bus (SW2) on the 2-channel buck EVM. (\textit{Do not turn off the main power SW1}).

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

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

\textbf{End of Exercise}
Multi-Loop Control

Supervisory BG

Control Engine(s)

Loop-1 mgmt

Loop-2 mgmt

Loop-N mgmt

Diag, slow adjustment.

PFC (2PHIL) Software Control Flow

Voltage Controller

Current Controller

PFC ISHARE

ADC-GenPurp
DC-DC (ZVSFB) Software Control Flow

CPU Bandwidth Utilization

BG Function / Activity # Inst. Tot. Cycles Stats

Comms + Supervisory 400 434

% ISR utilization = 87%

Spare ISR space = 128

BG loop rate (kHz) = 26.8 34.4
5 – Summary and Conclusion

Summary and Conclusion

- Review of Workshop Topics and Exercises
- TI Digital Power Products
  - C2000 Digital Signal Controller Family
  - UCD9xxx Digital Power Controller Family
- Where to Find More Information

Review of Workshop Topics and Exercises

Workshop Topics and Exercises Review

- C28x DSC family provides ideal controller for Digital Power Supply design
  - Scalable ePWM peripherals, ADC and fault management support
  - Code Composer Studio, DPS Library and TI Buck EVM
- Controlled Buck output voltage using PWM waveform and duty cycle without feedback
- Controlled Buck output using Voltage Mode Control with feedback
- Tuned closed-loop Buck power stage visually using CCS features
TI Digital Power Products

TI Is The Right Digital Power Partner

TI solutions cover the spectrum of power applications

System Complexity

Power-Optimized Controllers

UCD9240

UCD9111

UCD9112

TMS320F282x

TMS320F280x

Fully Programmable, Control Focused

Flexibility

UCD9080 Power Supply Sequencer and Monitor

UCD91xx Single-Output Digital Controller

UCD92xx Multi-Output Digital Controller

TMS320C2000 Digital Signal Controllers

TMS320C2000 32-bit controller solutions for green energy (solar, wind, fuel cells) and UPS battery backup

TI’s Digital Power Solutions Span the Industry

Non-Isolated DC/DC POL

• UCD91xx Single-Output Digital Controller
• UCD92xx Multi-Output Digital Controller
• TMS320C2000 Digital Signal Controllers

Isolated DC/DC & Offline AC/DC

• TMS320C2000 Digital Signal Controllers

DC/AC Inverters

• TMS320C2000 32-bit controller solutions for green energy (solar, wind, fuel cells) and UPS battery backup

System Management Only

• UCD9080 Power Supply Sequencer and Monitor
C2000 Digital Signal Controller Family

C2000 Family Roadmap

Future C28xxx

F283xx
300 MFLOPS
FPU, DMA

F282xx
150 MHz
DMA

F281x
150 MHz
8 Devices

F280xx
60 MHz
150ps PWM

TMS320F280xx Digital Signal Controllers

High Performance Signal Processing
- Up to 100 MHz performance
- Single cycle 32 x32 MAC (or dual 16 x16 MAC)
- Very Fast Interrupt Response
- Single cycle read-modify-write

Memory Sub-System
Fast program execution out of both RAM and Flash memory
- 80 MIPS with Flash Acceleration Technology
- 100 MIPS out of RAM for time-critical code

Control Peripherals
Up to 16 PWM channels and 4 event captures
150 ps High-Resolution PWM
Ultra-Fast 12-bit ADC
- 6.25 MSPS throughput
- Dual sample&holds enable simultaneous sampling
- Auto Sequencer, up to 16 conversions w/o CPU

Communications Ports
Multiple standard communication ports provide simple interfaces to other components

## F280xx Controller Portfolio

All Devices are 100% Hardware, Software & Pin Compatible

<table>
<thead>
<tr>
<th>TMS320</th>
<th>MHz</th>
<th>Flash KB</th>
<th>RAM KB</th>
<th>12-bit 16-ch ADC</th>
<th>PWM/Hi-Res.</th>
<th>CAP/QEP</th>
<th>Communication Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>F28015</td>
<td>60</td>
<td>32</td>
<td>12</td>
<td>267ns</td>
<td>10/4</td>
<td>2/0</td>
<td>SPI, SCI, I²C</td>
</tr>
<tr>
<td>F28016</td>
<td>60</td>
<td>32</td>
<td>12</td>
<td>267ns</td>
<td>10/4</td>
<td>2/0</td>
<td>SPI, SCI, CAN, I²C</td>
</tr>
<tr>
<td>F2801-60</td>
<td>60</td>
<td>32</td>
<td>12</td>
<td>267ns</td>
<td>8/3</td>
<td>2/1</td>
<td>2x SPI, SCI, CAN, I²C</td>
</tr>
<tr>
<td>F2802-60</td>
<td>60</td>
<td>64</td>
<td>12</td>
<td>267ns</td>
<td>8/3</td>
<td>2/1</td>
<td>2x SPI, SCI, CAN, I²C</td>
</tr>
<tr>
<td>F2801</td>
<td>100</td>
<td>32</td>
<td>12</td>
<td>160ns</td>
<td>8/3</td>
<td>2/1</td>
<td>2x SPI, SCI, CAN, I²C</td>
</tr>
<tr>
<td>F2802</td>
<td>100</td>
<td>64</td>
<td>12</td>
<td>160ns</td>
<td>8/3</td>
<td>2/1</td>
<td>2x SPI, SCI, CAN, I²C</td>
</tr>
<tr>
<td>F2806</td>
<td>100</td>
<td>64</td>
<td>20</td>
<td>160ns</td>
<td>16/4</td>
<td>4/2</td>
<td>4x SPI, 2x SCI, CAN, I²C</td>
</tr>
<tr>
<td>F2808</td>
<td>100</td>
<td>128</td>
<td>36</td>
<td>160ns</td>
<td>16/4</td>
<td>4/2</td>
<td>4x SPI, 2x SCI, 2x CAN, I²C</td>
</tr>
<tr>
<td>F2809</td>
<td>100</td>
<td>256</td>
<td>36</td>
<td>80ns</td>
<td>16/6</td>
<td>4/2</td>
<td>4x SPI, 2x SCI, 2x CAN, I²C</td>
</tr>
<tr>
<td>F28044</td>
<td>100</td>
<td>128</td>
<td>20</td>
<td>80ns</td>
<td>16/16</td>
<td>0</td>
<td>SPI, SCI, I²C</td>
</tr>
</tbody>
</table>

100-pin LQFP and u*BGA; Also available in -40 to 125 C and Automotive Q100

---

## TMS320F283xx Digital Signal Controllers

### High Performance Signal Processing
- Up to 150 MIPS performance with 32-bit floating-point unit
- Six-channel DMA speeds data throughput
- Very Fast Interrupt Response

### Memory Sub-System
Fast program execution out of both RAM and Flash memory
- 120 MIPS with Flash Acceleration Technology
- 150 MIPS out of RAM for time-critical code

### Control Peripherals
- Up to 16 PWM channels and 4 event captures
- 150 ps High-Resolution PWM
- Ultra-Fast 12-bit ADC
- 12.5 MSPS throughput
- Dual sample&holds enable simultaneous sampling
- Auto Sequencer, up to 16 conversions w/o CPU

### Communications Ports
Multiple standard communication ports provide simple interfaces to other components

### F283xx & F282xx Controller Portfolio

<table>
<thead>
<tr>
<th>TMS320</th>
<th>MHz</th>
<th>FPU</th>
<th>Flash KB</th>
<th>RAM KB</th>
<th>12-bit 10-ch ADC</th>
<th>DMA</th>
<th>PWM/HRPWM</th>
<th>CAP/QEP</th>
<th>Communication Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>F28335</td>
<td>150</td>
<td>Yes</td>
<td>512</td>
<td>68</td>
<td>80 ns</td>
<td>Yes</td>
<td>18/6</td>
<td>6/2</td>
<td>SPI, 3x SCI, P'C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28334</td>
<td>150</td>
<td>Yes</td>
<td>256</td>
<td>68</td>
<td>80 ns</td>
<td>Yes</td>
<td>18/6</td>
<td>4/2</td>
<td>SPI, 3x SCI, P'C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28332</td>
<td>100</td>
<td>Yes</td>
<td>128</td>
<td>52</td>
<td>80 ns</td>
<td>Yes</td>
<td>16/4</td>
<td>4/2</td>
<td>SPI, 2x SCI, P'C, McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28235</td>
<td>150</td>
<td>No</td>
<td>512</td>
<td>68</td>
<td>80 ns</td>
<td>Yes</td>
<td>18/6</td>
<td>6/2</td>
<td>SPI, 3x SCI, P'C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28234</td>
<td>150</td>
<td>No</td>
<td>256</td>
<td>68</td>
<td>80 ns</td>
<td>Yes</td>
<td>18/6</td>
<td>4/2</td>
<td>SPI, 3x SCI, P'C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28232</td>
<td>100</td>
<td>No</td>
<td>128</td>
<td>52</td>
<td>80 ns</td>
<td>Yes</td>
<td>16/4</td>
<td>4/2</td>
<td>SPI, 2x SCI, P'C, McBSP, 2x CAN</td>
</tr>
</tbody>
</table>

- 176-pin/ball LQFP/PBGA; 179-ball u*BGA; -40 to 125 C and Q100 in PBGA
- IQMath library provides software compatibility between floating-point and fixed-point!

### C2000 controlCARDS

- New low cost single-board controllers perfect for initial software development and small volume system builds.
- Small form factor (9cm x 2.5cm) with standard 100-pin DIMM interface
- F28x analog I/O, digital I/O, and JTAG signals available at DIMM interface
- Isolated RS-232 interface
- Single 5V power supply required
- controlCARDS available for 100MHz fixed-point TMS320F2808, TMS320F28044, and 150 MHz TMS320F28335 floating-point controller
- controlCARDS are available individually through TI distributors and on the web:
  - Part Number: TMDSCNCD2808, TMDSCNCD28044, TMDSCNCD28335

**F2808 only $59!**

**F28335 only $69!**
Digital Power Experimenter Kit

- DPEK includes
  - 2-rail DC/DC EVM using TI PowerTrain™ modules (10A)
  - On-board digital multi-meter and active load for transient response tuning
  - F2808 controlCARD
  - C2000 Applications Software CD with example code and full hardware details
  - Digital Power Supply Workshop teaching material and lab software
  - Code Composer Studio v3.3 with code size limit of 32KB
  - 9VDC power supply
- DPEK available through TI authorized distributors and on the web
  - Part Number: TMDSDC2KIT

C2000 DC/DC Developer’s Kit

- DC/DC Kit includes
  - 8-rail DC/DC EVM using TI PowerTrain™ modules (10A)
  - F28044 controlCARD
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v3.3 with code size limit of 32KB
  - 9VDC power supply
- Available through TI authorized distributors and on the web
  - Part Number: TMDSDCDC8KIT
C2000 AC/DC Developer’s Kit

- AC/DC Kit includes
  - AC/DC EVM with interleaved PFC and phase-shifted full-bridge
  - F2808 controlCARD
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v3.3 with code size limit of 32KB

- AC/DC EVM features
  - 12VAC in, 80W/10A output
  - Primary side control
  - Synchronous rectification
  - Peak current mode control
  - Two-phase PFC with current balancing

- AC/DC Kit available through TI authorized distributors and on the web
  - Part Number: TMDSACDCKIT

Emulation Solutions for C2000 Controllers

- BlackHawk USB2000 Controller only $299
  - Full CCS compatibility
  - Bi-Color Status LED (red/green)
  - 3.3/5.0 volt device I/O
  - Optional Isolation Adaptor for $299

- Spectrum Digital XDS510-LC only $249
  - Full CCS compatibility
  - Supports SDFlash programming utility
  - Supports XMLGUI for interfacing to ‘C’ – provides scripting capability
  - http://www.spectrumdigital.com
VisSim Graphical Programming for C2000

- Model based design for simulation, code generation, and interactive debugging
- Efficient code generation near hand code quality
- Automatic code generation for F28xx peripherals: ADC, SCI, SPI, I²C, CAN, ePWM, GPIO
- High speed target acquisition for waveform display on PC
- Watch ‘how to’ tutorials on Visual Solutions web site

www.vissim.com

UCD9xxx Digital Power Controller Family

UCD9xx Digital Power Controller Family

- UCD92xx
  - 4 ind outputs 64 & 80 pin
- UCD9230
  - 3 ind outputs 48 pin
- UCD9220
  - 2 ind outputs 32 pin
- UCD91xx
  - 1 output, 2 phase 32 pin
- UCD9112
  - 1 output, 2 phase 32 pin
- UCD9111
  - 1 output, 1 phase 32 pin
Where to Find More Information

**Recommended Next Step: One-day Training Course**

**Introduction to TMS320F2808 Design and Peripheral Training**

<table>
<thead>
<tr>
<th>TMS320C28x 1-Day Workshop Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Workshop Introduction</td>
</tr>
<tr>
<td>- Architecture Overview</td>
</tr>
<tr>
<td>- Programming Development Environment</td>
</tr>
<tr>
<td>- Peripheral Register Header Files</td>
</tr>
<tr>
<td>- Reset, Interrupts and System Initialization</td>
</tr>
<tr>
<td>- Control Peripherals</td>
</tr>
<tr>
<td>- IQ Math Library and DSP/BIOS</td>
</tr>
<tr>
<td>- Flash Programming</td>
</tr>
<tr>
<td>- The Next Step…</td>
</tr>
</tbody>
</table>

**Recommended Next Step: Multi-day Training Course**

<table>
<thead>
<tr>
<th>TMS320C28x Multi-day Workshop Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Architectural Overview</td>
</tr>
<tr>
<td>- Programming Development Environment</td>
</tr>
<tr>
<td>- Peripheral Register Header Files</td>
</tr>
<tr>
<td>- Reset and Interrupts</td>
</tr>
<tr>
<td>- System Initialization</td>
</tr>
<tr>
<td>- Analog-to-Digital Converter</td>
</tr>
<tr>
<td>- Control Peripherals</td>
</tr>
<tr>
<td>- Numerical Concepts and IQmath</td>
</tr>
<tr>
<td>- Using DSP/BIOS</td>
</tr>
<tr>
<td>- System Design</td>
</tr>
<tr>
<td>- Communications</td>
</tr>
<tr>
<td>- Support Resources</td>
</tr>
</tbody>
</table>
For More Information . . .

Internet
Website:  http://www.ti.com

FAQ:  http://www-k.ext.ti.com/sc/technical_support/knowledgebase.htm
   ♦ Device information    ♦ my.ti.com
   ♦ Application notes    ♦ News and events
   ♦ Technical documentation    ♦ Training

Enroll in Technical Training:  http://www.ti.com/sc/training

USA - Product Information Center (PIC)
Phone:  800-477-8924 or 972-644-5580
Email: support@ti.com
   ♦ Information and support for all TI Semiconductor products/tools
   ♦ Submit suggestions and errata for tools, silicon and documents

European Product Information Center (EPIC)
Web:  http://www-k.ext.ti.com/sc/technical_support/pic/euro.htm

Phone:  Language   Number
Belgium (English)   +32 (0) 27 45 55 32
France   +33 (0) 1 30 70 11 64
Germany   +49 (0) 8161 80 33 11
Israel (English)   1800 949 0107 (free phone)
Italy   800 79 11 37 (free phone)
Netherlands (English)   +31 (0) 546 87 95 45
Spain   +34 902 35 40 28
Sweden (English)   +46 (0) 8587 555 22
United Kingdom   +44 (0) 1604 66 33 99
Finland (English)   +358(0) 9 25 17 39 48

Fax:  All Languages   +49 (0) 8161 80 2045
Email:  epic@ti.com
   ♦ Literature, Sample Requests and Analog EVM Ordering
   ♦ Information, Technical and Design support for all Catalog TI
   ♦ Semiconductor products/tools
   ♦ Submit suggestions and errata for tools, silicon and documents