

C2000™ Software Frequency Response Analyzer (SFRA) Library and Compensation Designer

User's Guide



Literature Number: SPRUH25A
October 2014—Revised February 2015

1	Introduction	5
2	Installing the SFRA Library	8
	2.1 SFRA Library Package Contents.....	8
	2.2 How to Install the SFRA Library.....	8
3	Module Summary	8
	3.1 SFRA Library Function Summary	8
	3.2 Principle of Operation.....	9
	3.3 Per Unit Format.....	9
	3.4 Fixed Point (IQ).....	9
	3.5 Floating Point.....	16
	3.6 Adding SFRA Library to Assembly Digital Power Library Project	22
	3.7 Script for Importing Frequency Response and Designing Compensation	23
	3.8 SFRA GUI Options and How to Run	24
4	Compensation Designer	25
	4.1 Launching Compensation Designer.....	25
	4.2 Compensator Structure	29
	4.3 Compensation Style and Number	30
5	Case Study	36
	5.1 DPS Workshop Board Overview.....	37
	5.2 Plant TF Extraction	38
	5.3 Designing Compensator Using Compensation Designer.....	44
	5.4 OL Measurement	45
	5.5 Comparing SFRA Measured Frequency Response Versus Modeled	48
6	FAQ	49
	6.1 GUI Will not Connect.....	49
	6.2 GUI Will Connect but Start Button Does not Become Active	49
	6.3 For Measurable Response, What is the Maximum Frequency SFRA?	49
	6.4 What is the Lowest Amplitude Signal Measurable by SFRA?	49
	6.5 Why Does the SFRA Sweep Take so Long?	49
7	About the Author	50
	Revision History	51

List of Figures

1	Digitally Controlled Power Converter.....	5
2	Control Design Using SFRA Library and Compensation Designer	7
3	FRA Principle of Operation	9
4	Compiler Options for a Project Using SFRA IQ Lib	11
5	Adding SFRA Library to the Linker Options in the CCS Project	12
6	Compiler Options for a Project Using SFRA Float Library	17
7	Adding Linker Options to the CCS Project to Include SFRA Library in Floating-Point Project.....	18
8	Project Using Assembly Digital Power Library	22
9	SFRA Added to Assembly Digital Power Library Framework	23
10	SFRA GUI Options	25
11	Standalone Compensation Designer When launched From the controlSUITE/libs/app_libs/SFRA/<version>/GUI Folder.....	27
12	Compensation Designer When Launched From Solution Adapter Page (both Modeled and SFRA data- based Plant Information can be used for compensation design)	28
13	Compensator Coefficients Calculated by the GUI	29
14	Warning Sign and Dialog When Compensation Coeff Range is Exceeded	29
15	Compensation Number and Style Selection	30
16	PID Panel.....	30
17	Two Pole Two Zero Pole Zero Entry in Compensation Designer GUI	31
18	Three Pole Three Zero Pole Zero Entry in Compensation Designer GUI	32
19	Two Pole Two Zero With One Complex Zero Entry in Compensation Designer GUI.....	33
20	Three Pole Three Zero With One Complex Pole Entry in Compensation Designer GUI	33
21	Three Pole Three Zero With One Complex Zero Entry in Compensation Designer GUI	34
22	Three Pole Three Zero With One Complex Zero Entry in Compensation Designer GUI	35
23	DPS Workshop Board Overview	37
24	Software Diagram for Plant tf Extraction	38
25	Plant Frequency Response Plot.....	43
26	Comp Designer Being Used to for the DPS Workshop Board Based on Plant Data From SFRA Run.....	44
27	Closed Loop SFRA Software Diagram	45
28	SFRA GUI With OL Frequency Response Plot	47
29	Plant Frequency Response Modeled Vs Measured on DPSWrkShpKit	48
30	Open Loop Frequency Response Modeled Vs Measured on DPSWrkShpKit	48

List of Tables

1	Library Sub-Directory Structure	8
2	SFRA Cycle and Program Memory Usage	8
3	Fixed Point (IQ) Module Interface Definition	10
4	Floating Point Module Interface Definition	16
5	Different Compensation Style Supported by Compensation Designer	30
6	DPS Workshop Board Key Switches and Components List	37
7	DPS Workshop Board Resource Allocation	38

C2000™ Software Frequency Response Analyzer (SFRA) Library and Compensation Designer

1 Introduction

Texas Instruments' software frequency response analyzer (SFRA) library is designed to enable frequency response analysis on digitally controlled power converters using software only and without the need for an external frequency response analyzer. The optimized library can be used in high frequency power conversion applications to identify the plant and the open loop characteristics of a closed loop power converter, which can be used to get stability information such as bandwidth, gain margin and phase margin to evaluate the control loop performance.

Consider a digitally controlled closed loop power converter, as shown in [Figure 1](#), where:

- H is the transfer function of the plant that needs to be controlled
- G is the digital compensator
- GH is referred to as the open loop transfer function
- r is the instantaneous set point or the reference of the converter
- Ref is the DC set point reference
- y the analog-to-digital converter (ADC) feedback
- e the instantaneous error
- d the sensor noise and disturbance
- u the PWM duty cycle

The key objectives of the compensator in a closed loop system can be summarized as:

- Ensure that the system is stable (for example, the system tracks the reference asymptotically)

$$e = \lim_{t \rightarrow \infty} e(t) = \lim_{t \rightarrow \infty} \frac{r}{1 + GH} \rightarrow 0 \tag{1}$$

- System provides disturbance rejection to ensure robust operation

$$S = \frac{y}{d} = \frac{1}{1 + GH} \rightarrow 0 \tag{2}$$

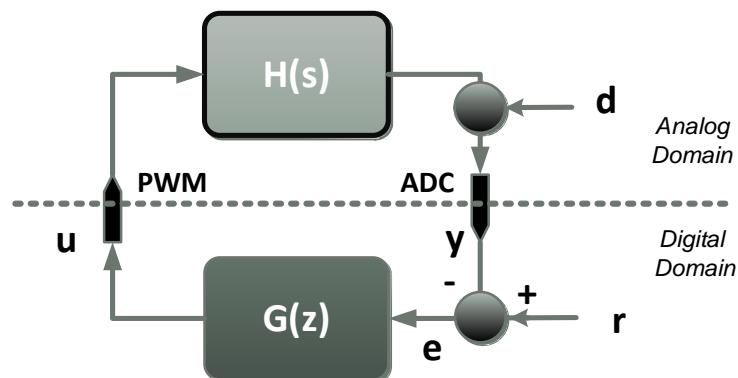


Figure 1. Digitally Controlled Power Converter

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

Whether or not the system meets the objectives can be determined by knowing the open loop transfer function (GH), as shown in [Equation 1](#) and [Equation 2](#).

A Bode plot of the open loop transfer function GH is frequently used for this purpose and quantities such as gain margin (GM), phase margin (PM) and bandwidth (BW) are often used to comment on the stability and robustness of a closed loop power converter.

The SFRA library can enable measurements of the GH and H frequency response by software. This data can be used to:

- Verify the plant model (H) or extract the plant model (H)
- Design a compensator (G) for the closed loop plant
- Verify the close loop performance of the system by plotting the open loop (GH) Bode diagram

As the frequency response of GH and H carry information of the plant, the data can be used to comment on the health of the power stage by periodically measuring the frequency response.

A typical flow of using SFRA library is:

1. The SFRA GUI populates the data in a comma separate value (CSV) file. Identify the plant model for the steady state operating point by running the SFRA library.

NOTE: Optionally, the SFRA GUI can output data to an excel file.

2. This SFRA CSV data can directly be used in the Compensation Designer to design a compensator that meets the system requirements. (Alternatively, if the Excel file was used, the MATLAB script is provided that will curve fit a transfer function and the sisotool can be used to design the compensator).
3. New Compensator values can be copied from the GUI into the Code Composer Studio™ project.
4. Compile and load the code with new coefficients into the microcontroller controlling the power stage. SFRA algorithm (Step 1) can be re-run to verify the closed loop system performance by measuring the open loop gain GH (also referred to as loop gain in many literatures).

In summary, TI's software frequency response analyzer along with the Compensation Designer provides a methodology to tune power supplies in a systematic way and enables quick and easy frequency response analysis for power converters without the need of external connections and equipment, as shown in [Figure 2](#). Since no external connections are used, the SFRA can be run repeatedly to periodically assess the health of the power converter and get diagnostic information.

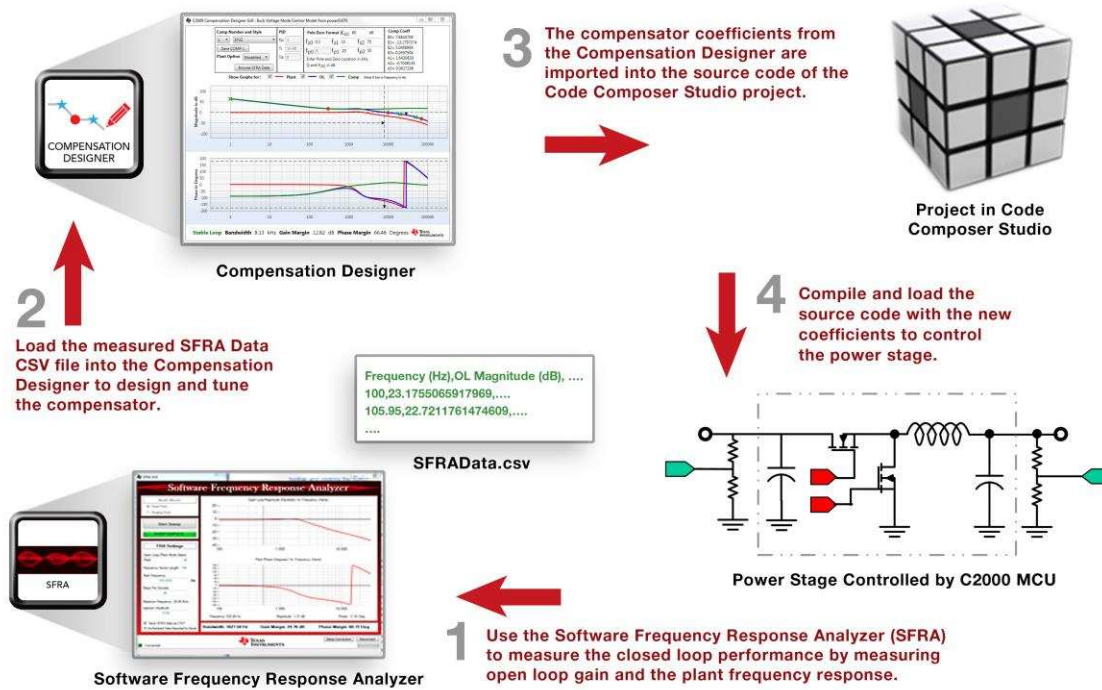


Figure 2. Control Design Using SFRA Library and Compensation Designer

2 Installing the SFRA Library

2.1 SFRA Library Package Contents

The TI SFRA library consists of the following components:

- Header files and software library for fixed- and floating-point computation
- *C2000 Software Frequency Response Analyzer (SFRA) Library User's Guide* ([SPRUHZ5](#))

2.2 How to Install the SFRA Library

The SFRA library is distributed through the controlSUITE™ installer, which can be downloaded at www.ti.com/tool/controlsuite. Select the SFRA Library Checkbox to install the library in the controlSUITE directory. By default, the installation places the library components in the following directory structure:

<base> install directory is C:\ti\controlSUITE\libs\app_libs\SFRA\vx.X

The following sub-directory structure is shown in [Table 1](#).

Table 1. Library Sub-Directory Structure

<base>\Doc	Documentation
<base>\Float	Contains floating-point implementation of the library
<base>\IQ	Contains fixed-point implementation of the library
<base>\GUI	Host side graphical user interface (GUI) for displaying the frequency response and the Compensation Designer. The SFRAData.csv file is also stored here.
<base>\examples	Example using SFRA library
<base>\Scripts	Scripts to import SFRA data for compensation design in MATLAB

3 Module Summary

3.1 SFRA Library Function Summary

The SFRA library consists of modules that enable the user to run SFRA on power converters. [Table 2](#) lists the modules existing in the SFRA library and a summary of cycle counts on each instruction set variant of the SFRA library.

Table 2. SFRA Cycle and Program Memory Usage

	SFRA_INJECT	SFRA_COLLECT	SFRA_Background	Program Size	Data Size
Fixed	45	81	Typical ~50 Max 14,000	737 x 16-bit words	22 x 16-bit words (internal) + 25 x 16-bit words for SFRA object
Float	41	63	Typical ~ 50 Max 2350	665 x 16-bit words	20 x 16-bit words (internal) + 25 x 16-bit words for SFRA object

The numbers reported above are for the data memory usage internal to the library and the memory used by the SFRA object instance. The memory used to store the frequency response data is not included as this is user specific, and depend on the number of frequency points the user wants to run the SFRA over.

NOTE: The SFRA library is non-re-entrant, for example, only one instance of the SFRA library is supported in a particular project. The cycles reported, unless otherwise mentioned, are for maximum and worst case path in the operation.

3.2 Principle of Operation

The software frequency response analyzer is based on the principle of small signal injection. A small signal is injected on the reference of the controller, as shown in Figure 3, and the frequency response on feedback and controller outputs are calculated. This provides the plant frequency response characteristics and the open loop frequency response of the closed loop system.

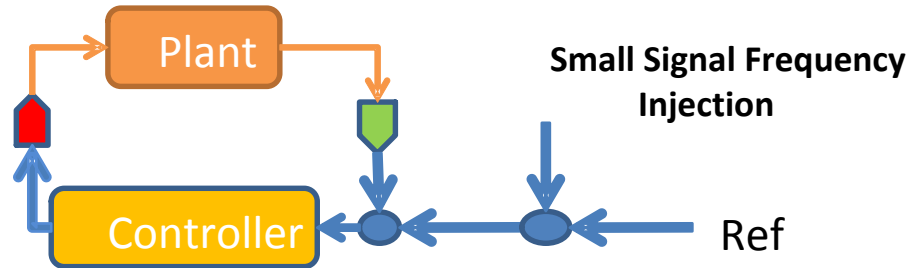


Figure 3. FRA Principle of Operation

3.3 Per Unit Format

The SFRA library supports IQ (fixed point) and floating point based math. Per unit values are typically used for variables in control application and are used by the library. Per unit value is found by dividing the current reading by the maximum value that can be read. For example, if the voltage sense max is 20 V and the instantaneous reading is 5 V, the per unit value is $5/20 = 0.25$.

For IQ based blocks, _IQ24 format is used and all the values are scaled from 0-1 in IQ24 format. For CLA and Float 32, single precision floating point representation is used.

3.4 Fixed Point (IQ)

3.4.1 Object Definition

The SFRA library defines the IQ Math based SFRA structure handle as discussed in the following sections.

```
typedef struct{
    int32_t *H_MagVect;      // Plant Mag SFRA Vector
    int32_t *H_PhaseVect;   // Plant Phase SFRA Vector
    int32_t *GH_MagVect;    // Open Loop Mag SFRA Vector
    int32_t *GH_PhaseVect;  // Open Loop Phase SFRA Vector
    float *FreqVect;        // Frequency Vector
    int32_t amplitude;      // Injection Amplitude
    int32_t ISR_Freq;       // SFRA ISR frequency
    float Freq_Start;
    float Freq_Step;
    int16_t start;          // Command to start SFRA
    int16_t state;         // State of SFRA
    int16_t status;        // Status of SFRA
    int16_t Vec_Length;    // No. of Points in the SFRA
    int16_t FreqIndex;     // Index of the frequency vector
}SFRA_IQ;
```

3.4.2 Module Interface Definition

Table 3. Fixed Point (IQ) Module Interface Definition

Module Element Name	Type	Description	Acceptable Range
H_MagVect,GH_MagVect	Input	Pointer to the array that stores the magnitude of plant and open loop SFRA data of a closed loop plant.	Pointer to 32 bit location
H_PhaseVect,GH_PhaseVect	Input	Pointer to the array that stores the phase of plant and open loop SFRA data of a closed loop plant.	Pointer to 32 bit location
FreqVect	Input	Pointer to array of frequency values where SFRA is performed.	Pointer to 32 bit location
Amplitude	Input	Amplitude of small signal injection in per unit format (pu).	IQ26(-1,1)
SFRA_Freq	Input	Frequency where SFRA is called.	Float32
Freq_Start	Input	Start Frequency.	Float32
Freq_Step	Input	$10^{1/(no\ of\ steps\ per\ decade)}$.	Float32
Start	Input	Command to start SFRA.	int16
State	Output	SFRA state	int16
Status	Output	SFRA injection is active.	int16
Vec_Length	Input	No of points where SFRA is performed.	int16
FreqIndex	Output	Frequency where SFRA is being performed.	int32

3.4.3 Adding SFRA Library to the Project

1. Include library in {ProjectName}-Includes.h. Make sure the math type is specified in the project include file before SFRA lib is included and IQmath libs is included:

```
#define MATH_TYPE 0 //IQ_MATH
#include "SFRA_IQ_Include.h"
```

Add the SFRA library path in the include paths under Project Properties → Build → C2000 Compiler → Include Options (see [Figure 4](#)).

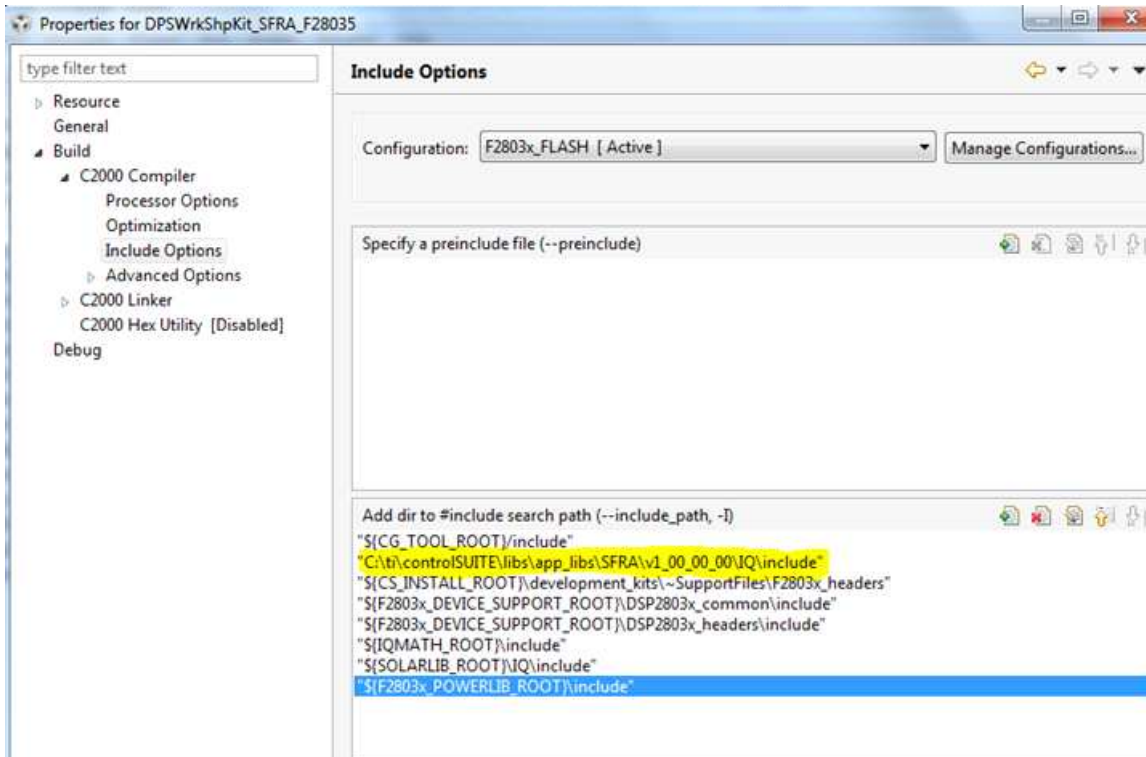


Figure 4. Compiler Options for a Project Using SFRA IQ Lib

NOTE: The exact locations may vary depending on where controlSUITE is installed and which other libraries the project is using.

Link SFRA library: (SFRA_IQ_Lib.lib) to the project, SFRA lib is located at:

controlSUITE\development\libs\app_libs\SFRA\vX\SFRA_IQ\lib

Figure 5 shows the changes to the linker options that are required to include the SFRA library.

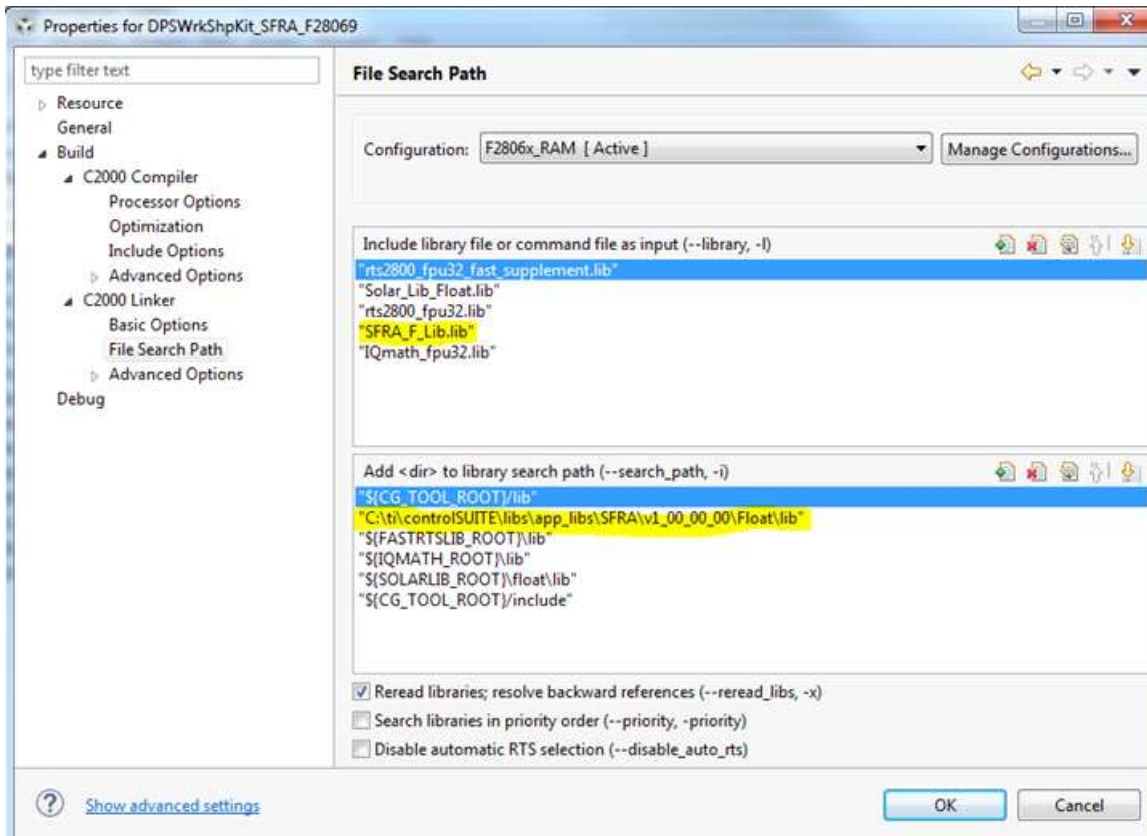


Figure 5. Adding SFRA Library to the Linker Options in the CCS Project

2. Create and add module structure to {ProjectName}-Main.c for SFRA, which also defines the frequency at which the SFRA is called (SFRA_ISR_FREQ), starts the frequency of the SFRA analysis (SFRA_FREQ_START), the number of points in SFRA analysis (SFRA_FREQ_LENGTH) and the step size between individual points (SFREQ_STEP_MULTIPLY). The end frequency of the SFRA analysis is dependent on the start frequency, the length and the step size.

```

...
#define SFRA_ISR_FREQ 200000
#define SFRA_FREQ_START 100
#define SFRA_FREQ_LENGTH 100
//SFRA step Multiply = 10^(1/No of steps per decade(40))
#define SFREQ_STEP_MULTIPLY (float)1.059253
...
SFRA_IQ SFRA1;
//extern to access tables in ROM
extern long IQsinTable[];

```

3. Declare arrays for FRA data storage in {ProjectName}-Main.c.

```

int32 Plant_MagVect[SFRA_FREQ_LENGTH];
int32 Plant_PhaseVect[SFRA_FREQ_LENGTH];
int32 OL_MagVect[SFRA_FREQ_LENGTH];
int32 OL_PhaseVect[SFRA_FREQ_LENGTH];
float32 FreqVect[SFRA_FREQ_LENGTH];

```

4. Initializing the module in {ProjectName}-Main.c.

```
//SFRA Object Initialization
//Specify the injection amplitude
SFRA1.amplitude=_IQ26(0.01);
//Specify the length of SFRA
SFRA1.Vec_Length=SFRA_FREQ_LENGTH;
//Specify the SFRA ISR Frequency
SFRA1.SFRA_Freq=SFRA_ISR_FREQ;
//Specify the Start Frequency of the SFRA analysis
SFRA1.Freq_Start=SFRA_FREQ_START;
//Specify the Frequency Step
SFRA1.Freq_Step=FREQ_STEP_MULTIPLY;
//Assign array location to Pointers in the SFRA object
SFRA1.FreqVect=FreqVect;
SFRA1.GH_MagVect=OL_MagVect;
SFRA1.GH_PhaseVect=OL_PhaseVect;
SFRA1.H_MagVect=Plant_MagVect;
SFRA1.H_PhaseVect=Plant_PhaseVect;

SFRA_IQ_INIT(&SFRA1);
```

5. Using the module, add the code to the ISR to call the SFRA as shown below around the compensator input and outputs:

(a) Using SFRA in a closed loop system to get loop gain(GH) and plant(H) frequency response:

```
interrupt void PWM_ISR(void)
{
    ...

    //Read ADC and computer Fbk Value
    cntl3p3z_vars1.Fdbk= (int32)Vout1R<<12;

    //Add FRA injection into the reference of the controller
    cntl3p3z_vars1.Ref = SFRA_IQ_INJECT(Vout1SetSlewed);

    // Call the controller
    CNTL_3P3Z_IQ_ASM(&cntl3p3z_coeff1,&cntl3p3z_vars1);

    //Update PWM value
    EPwm1Regs.CMPA.half.CMPA=_IQ24mpy((long)(BUCK_PWM_PERIOD),cntl3p3z_vars1.Out);

    SFRA_IQ_COLLECT(&cntl3p3z_vars1.Out,&cntl3p3z_vars1.Fdbk);

    ...
}
```

(b) Using SFRA in open loop to get plant(H) frequency response:

```
interrupt void PWM_ISR(void)
{
    ...

    //Read ADC and computer Fbk Value
    Vout1_Read= (int32)Vout1R<<12;

    //Add FRA injection into the duty cycle for the open loop converter
    Duty_pu=SFRA_IQ_INJECT(Duty_pu_DC);

    //Update PWM value
    EPwm1Regs.CMPA.half.CMPA =_IQ24mpy((long)(BUCK_PWM_PERIOD),Duty_pu);

    SFRA_IQ_COLLECT(&Duty_pu,&Vout1_Read);

    ...
}
```

6. Background task: Call the background function in a slow task:

```
SFRA_IQ_BACKGROUND (&SFRA1);
```

7. Linker command file changes (*.CMD): For the best performance when running from FLASH, the SFRA INJECT and COLLECT operations need to reside in RAM. The following are the changes and additions to CMD file for the SFRA library inclusion and best performance:

```

...
ramfuncs          : LOAD = FLASHD,
                  RUN = RAML0L1,
                  LOAD_START(_RamfuncsLoadStart),
                  LOAD_END(_RamfuncsLoadEnd),
                  RUN_START(_RamfuncsRunStart),
                  PAGE = 0

                  {
                      --library=rts2800_ml.lib<fs_mpy.obj>
                      --library=SFRA_IQ_Lib.lib<SFRA_IQ_INJECT.obj>
                      --library=SFRA_IQ_Lib.lib<SFRA_IQ_COLLECT.obj>
                  }

...

SFRA_IQ_Data      : > dataRAM1, ALIGN = 64, PAGE = 1
  
```

8. Also make sure the ROM location for the IQmath tables is defined correctly. These locations are defined in the device command file, which is available through controlSUITE. The following location is correct for F28035.

```

/* IQ Math Tables in Boot ROM */
IQTABLES        : origin = 0x3FE000, length = 0x000B50
...

/* Allocate IQ math areas: */
/* Math Code */
IQmath           : > FLASHA          PAGE = 0
/* Math Tables In ROM */
IQmathTables     : > IQTABLES        PAGE = 0, TYPE = NOLOAD
  
```

3.4.4 Adding Support for SFRA GUI

A few modifications to the Code Composer Studio™ project are required to connect to the FRA GUI.

1. Select the serial communications interface (SCI) pins. Ensure that pins connected to the FTDI chip for the serial link are configured as SCI-RX and TX (GPIO-28 and GPIO-29 for Piccolo-B and A). Also, make sure that the SCI clock is enabled. Clock configuration and general-purpose input/output (GPIO) muxing is typically handled in [ProjectName]-DevInit_[Target].c
2. Add SciCommsGui_32bit.c to the project. In the project window, right-click on your project and select “Add files to project”, then browse to SciCommsGui_32bit.c found under FRA/version/GUI.
3. Add *SCIA_Init()* and *SerialHostComms()* to your function prototype list.
4. Add *SerialCommsTimer*, *CommsOKflg*, and *initalizationflag* as an int16, if they do not already exist:

```

int16 SerialCommsTimer;
int16 CommsOKflg;
//Flag for reinitializing FRA variables
int16 initializationFlag;
  
```

5. Declare the following variables, if they do not already exist:

```

//GUI support variables
// sets a limit on the amount of external GUI controls - increase as necessary
int16 *varSetTxtList[16]; //16 textbox controlled variables
int16 *varSetBtnList[16]; //16 button controlled variables
int16 *varSetSlidrList[16]; //16 slider controlled variables
int16 *varGetList[16]; //16 variables sendable to GUI
int32 *arrayGetList[16]; //16 arrays sendable to GUI
Uint32 *dataSetList[16]; //16 32-bit textbox or label controlled variables
  
```

NOTE: Values in *ArrayGetList* should be 32-bit values for proper serial communication with the SFRA GUI.

6. Add the following to the project's initialization sequence:

```
// 15000000 is the LSPCLK or the Clock used for the SCI Module
// 57600 is the Baudrate desired of the SCI module
SCIA_Init(15000000, 57600);

CommsOKflg = 0;
SerialCommsTimer = 0;

// "Set" variables
// assign GUI Buttons to desired flag addresses
varSetBtnList[0] = (int16*)&initializationFlag;

// "Get" variables
//-----
// assign a GUI "getable" parameter address
varGetList[0] = (int16*)&(SFRA1.Vec_Length);
varGetList[1] = (int16*)&(SFRA1.status);
varGetList[2] = (int16*)&(SFRA1.FreqIndex);

// "Setable" variables
//-----
// assign GUI "setable" by Text parameter address
dataSetList[0] = (Uint32*)&(SFRA1.Freq_Start);
dataSetList[1] = (Uint32*)&(SFRA1.amplitude);
dataSetList[2] = (Uint32*)&(SFRA1.Freq_Step);

// assign a GUI "getable" parameter array address
arrayGetList[0] = (int32*)FreqVect;
arrayGetList[1] = (int32*)OL_MagVect;
arrayGetList[2] = (int32*)OL_PhaseVect;
arrayGetList[3] = (int32*)Plant_MagVect;
arrayGetList[4] = (int32*)Plant_PhaseVect;
arrayGetList[5] = (int32*)&(SFRA1.Freq_Start);
arrayGetList[6] = (int32*)&(SFRA1.amplitude);
arrayGetList[7] = (int32*)&(SFRA1.Freq_Step);
```

7. Add `SerialCommsTimer++`; to a background task that is executed roughly every 1 msec. The following code shows the addition of the timer increment to a task A0.

```
void A0(void)
{
// loop rate synchronizer for A-tasks
if(CpuTimer0Regs.TCR.bit.TIF == 1)
{
CpuTimer0Regs.TCR.bit.TIF = 1; // clear flag
//-----
(*A_Task_Ptr)(); // jump to an A Task (A1,A2,A3,...)
//-----
VTimer0[0]++; // virtual timer 0, instance 0 (spare)
SerialCommsTimer++; // used by DSP280x_SciCommsGui.c
}
Alpha_State_Ptr = &B0 // Comment out to allow only A tasks
}
```

8. In a further slower task, call the following routine:

```
SerialHostComms();
```

9. To enable change of FRA parameters from the GUI add the following code in a slow background task:

```
if(initializationFlag == 1)
{
SFRA_IQ_INIT(&SFRA1);
initializationFlag = 0;
SFRA1.start = 1;
}
```

10. The project will now connect to the SFRA GUI. For more information on how to run the SFRA GUI, [Section 3.8](#).

3.5 Floating Point

3.5.1 Object Definition

The SFRA library defines the floating point based SFRA structure handle as discussed in the following sections:

```
typedef struct{
    float *H_MagVect;      //Plant Mag FRA Vector
    float *H_PhaseVect;   //Plant Phase FRA Vector
    float *GH_MagVect;    //Open Loop Mag FRA Vector
    float *GH_PhaseVect;  //Open Loop Phase FRA Vector
    float *FreqVect;      //Frequency Vector
    float amplitude;      //Injection Amplitude
    float ISR_Freq;       //FRA ISR frequency
    float Freq_Start;
    float Freq_Step;
    int16_t start;        //Command to start FRA
    int16_t state;        //State of FRA
    int16_t status;       //Status of FRA
    int16_t Vec_Length;   // No. of Points in the FRA
    int16_t FreqIndex;    // Index of the frequency vector
}SFRA_F;
```

3.5.2 Module Interface Definition

Table 4. Floating Point Module Interface Definition

Module Element Name	Type	Description	Acceptable Range
H_MagVect,GH_MagVect	Input	Pointer to the array that stores the magnitude of plant and open loop SFRA data of a closed loop plant.	Pointer to 32 bit location
H_PhaseVect,GH_PhaseVect	Input	Pointer to the array that stores the phase of plant and open loop SFRA data of a closed loop plant.	Pointer to 32 bit location
FreqVect	Input	Pointer to array of frequency values at which SFRA is performed.	Pointer to 32 bit location
amplitude	Input	Amplitude of small signal injection in pu.	Float32(-1,1)
ISR_Freq	Input	Frequency at which SFRA is called.	Float32
Freq_Start	Input	Start Frequency.	Float32
Freq_Step	Input	$10^{(1/(no\ of\ steps\ per\ decade))}$.	Float32
Start	Input	Command to start SFRA.	int16
State	Output	SFRA state.	int16
Status	Output	SFRA injection is active	int16
Vec_Length	Input	No of points for which SFRA is performed.	int16
FreqIndex	Output	Frequency at which SFRA is being performed.	int32

3.5.3 Adding SFRA Library to the Project

1. Include library in {ProjectName}-Includes.h. Make sure the math type is specified in the project include file before SFRA lib is included and IQmath libs is included:

```
#define MATH_TYPE 1 //FLOAT_MATH
#include "SFRA_F_Include.h"
```

Add the SFRA library path in the include paths under Project Properties → Build → C2000 Compiler → Include Options (see Figure 6).

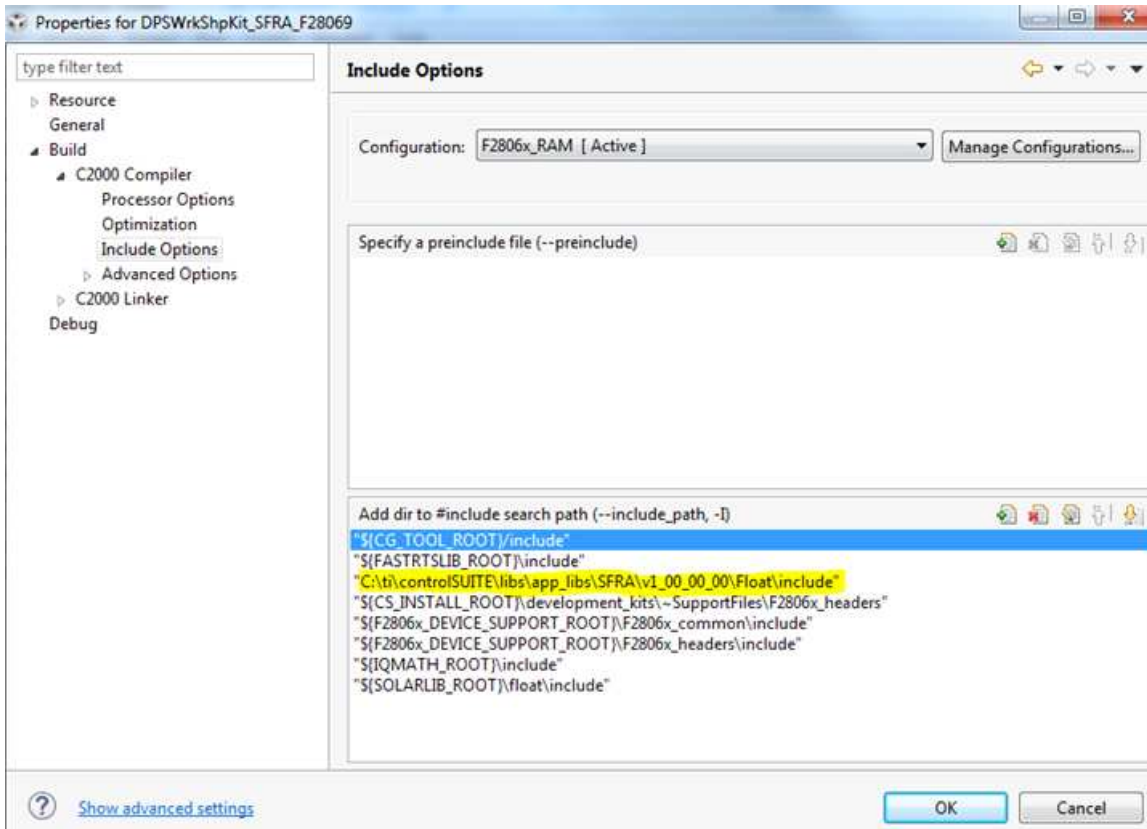


Figure 6. Compiler Options for a Project Using SFRA Float Library

NOTE: The exact locations may vary depending on where controlSUITE is installed and which other libraries the project is using.

Link SFRA library: (SFRA_F_Lib.lib) to the project, SFRA lib is located at:

controlSUITE\development\libs\app_libs\SFRA\vX\Float\lib

Figure 7 shows the changes to the linker options that are required to include the SFRA library.

NOTE: FastRTS library must be included, for steps on including fastRTS library. For more information, see the FastRTS library documentation found at:

C:\ti\controlSUITE\libs\math\FPUfastRTS\V100\doc.

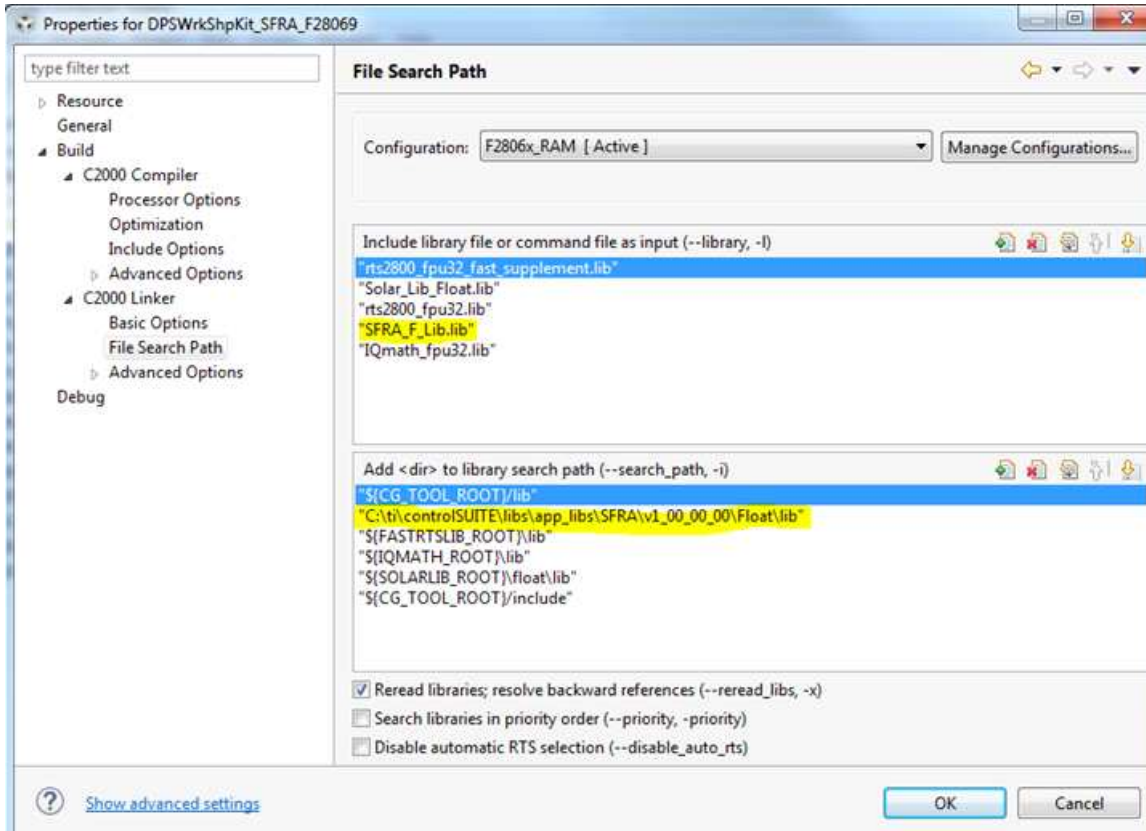


Figure 7. Adding Linker Options to the CCS Project to Include SFRA Library in Floating-Point Project

NOTE: The exact locations may vary depending on where controlSUITE is installed and which other libraries the project is using.

2. Create and add module structure to {ProjectName}-Main.c for SFRA, also define the frequency where SFRA is called (SFRA_ISR_FREQ), start frequency of the SFRA analysis (SFRA_FREQ_START), the number of points in SFRA analysis (SFRA_FREQ_LENGTH) and the step size between individual points (FREQ_STEP_MULTIPLY). The end frequency of the SFRA analysis is dependent on the start frequency, the length, and the step size.

```
#define SFRA_ISR_FREQ 200000
#define SFRA_FREQ_START 100
#define SFRA_FREQ_LENGTH 100
//SFRA step Multiply = 10^(1/No of steps per decade(40))
#define FREQ_STEP_MULTIPLY (float)1.059253
...
SFRA_F SFRA1;
//extern to access tables in ROM
extern long FPUsinTable[];
```

3. Declare arrays for FRA data storage in {ProjectName}-Main.c.

```
float32 Plant_MagVect[SFRA_FREQ_LENGTH];
float32 Plant_PhaseVect[SFRA_FREQ_LENGTH];
float32 OL_MagVect[SFRA_FREQ_LENGTH];
float32 OL_PhaseVect[SFRA_FREQ_LENGTH];
float32 FreqVect[SFRA_FREQ_LENGTH];
```

4. Initializing the module in {ProjectName}-Main.c.

```
//SFRA Object Initialization
//Specify the injection amplitude
SFRA1.amplitude=0.01;
//Specify the length of SFRA
SFRA1.Vec_Length=SFRA_FREQ_LENGTH;
//Specify the SFRA ISR Frequency
SFRA1.SFRA_Freq=SFRA_ISR_FREQ;
//Specify the Start Frequency of the SFRA analysis
SFRA1.Freq_Start=SFRA_FREQ_START;
//Specify the Frequency Step
SFRA1.Freq_Step=FREQ_STEP_MULTIPLY;
//Assign array location to Pointers in the SFRA object
SFRA1.FreqVect=FreqVect;
SFRA1.GH_MagVect=OL_MagVect;
SFRA1.GH_PhaseVect=OL_PhaseVect;
SFRA1.H_MagVect=Plant_MagVect;
```

```
SFRA_F_INIT(&SFRA1);
```

5. Using the module, add the code to the ISR to call the SFRA as shown below around the compensator input and outputs:

(a) Using SFRA in a closed loop system to get loop gain(GH) and plant(H) frequency response:

```
interrupt void PWM_ISR(void)
{
    ...

    //Read ADC and computer Fbk Value
    cnt13p3z_vars1.Fdbk = (float32)Vout1R/(4096.0);

    //Add FRA injection into the reference of the controller
    cnt13p3z_vars1.Ref = SFRA_F_INJECT(Vout1SetSlewed);

    // Call the controller
    CNTL_3P3Z_F_ASM(&cnt13p3z_coeff1,&cnt13p3z_vars1);

    //Update PWM value
    EPwm1Regs.CMPA.half.CMPA = ((long)(BUCK_PWM_PERIOD))*cnt13p3z_vars1.Out;

    SFRA_F_COLLECT(&cnt13p3z_vars1.Out,&cnt13p3z_vars1.Fdbk);
    ...
}
```

(b) Using SFRA in open loop to get plant(H) frequency response:

```
interrupt void PWM_ISR(void)
{
    ...
    //Read ADC and computer Fbk Value
    Vout1_Read = (float32)Vout1R/(4096.0);

    //Add SFRA injection into the duty cycle for the open loop converter
    Duty_pu=SFRA_F_INJECT(Duty_pu_DC);

    //Update PWM value
    EPwm1Regs.CMPA.half.CMPA=((long)(BUCK_PWM_PERIOD))* Duty_pu;

    SFRA_F_COLLECT(&Duty_pu,&Vout1_Read);
    ...
}
```

}

6. Background task: Call the background function in a slow task:

```
SFRA_F_BACKGROUND(&SFRA1);
```

7. Linker command file changes (*.CMD): When running from FLASH for best performance the SFRA INJECT and COLLECT operations need to reside in RAM. The following are the changes and additions to CMD file for the SFRA library inclusion and best performance.

```
...
ramfuncs      : LOAD = FLASHD,
               RUN = RAML0L1,
               LOAD_START(_RamfuncsLoadStart),
               LOAD_END(_RamfuncsLoadEnd),
               RUN_START(_RamfuncsRunStart),
               PAGE = 0
               {
                   --library=SFRA_F_Lib.lib<SFRA_F_INJECT.obj>
                   --library=SFRA_F_Lib.lib<SFRA_F_COLLECT.obj>
               }
...

SFRA_IQ_Data   : > dataRAM1, PAGE = 1
```

8. Also make sure the ROM location for IQmath tables is defined correctly. These locations are defined in the device command file, which is available through controlSUITE. The following location is correct for F28069.

```
/* FPU Math Tables in Boot ROM */
FPUTABLES    : origin = 0x3FD860, length = 0x0006A0
...
/* Allocate FPU math areas: */
FPUmathTables : > FPUTABLES, PAGE = 0, TYPE = NOLOAD
```

3.5.4 Adding Support GUI

A few modifications to a CCS SFRA project are required to connect to the GUI.

1. Select SCI pins. Ensure that pins connected to the FTDI chip for the serial link are configured as SCI-RX and TX (GPIO-28 and GPIO-29 for Piccolo-B and A). Also, make sure that the SCI clock is enabled. Clock configuration and GPIO muxing is typically handled in [ProjectName]-DevInit_[Target].c
2. Add SciCommsGui_32bit.c to the project. In the project window, right-click on your project and select Add files to project, then browse to SciCommsGui_32bit.c found under SFRA/version/GUI.
3. Add *SCIA_Init()* and *SerialHostComms()* to your function prototype list.
4. Add SerialCommsTimer, CommsOKflg, and initalizationflag as an int16, if they do not already exist:

```
int16 SerialCommsTimer;
int16 CommsOKflg;
//Flag for reinitializing FRA variables
int16 initializationFlag;
```

5. Declare the following variables, if they do not already exist:

```
//GUI support variables
// sets a limit on the amount of external GUI controls - increase as necessary
int16 *varSetTxtList[16]; //16 textbox controlled variables
int16 *varSetBtnList[16]; //16 button controlled variables
int16 *varSetSlidrList[16]; //16 slider controlled variables
int16 *varGetList[16]; //16 variables sendable to GUI
int32 *arrayGetList[16]; //16 arrays sendable to GUI
Uint32 *dataSetList[16]; //16 32-bit textbox or label controlled variables
```

NOTE: Values in ArrayGetList should be 32-bit values for proper serial communication with the SFRA GUI.

6. Add the following to the project's initialization sequence:

```
// 20000000 is the LSPCLK or the Clock used for the SCI Module
// 57600 is the Baudrate desired of the SCI module
SCIA_Init(20000000, 57600);
CommsOKflg = 0;
SerialCommsTimer = 0;

// "Set" variables
// assign GUI Buttons to desired flag addresses
varSetBtnList[0] = (int16*)&initializationFlag;

// "Get" variables
//-----
// assign a GUI "getable" parameter address
varGetList[0] = (int16*)&(SFRA1.Vec_Length);
varGetList[1] = (int16*)&(SFRA1.status);
varGetList[2] = (int16*)&(SFRA1.FreqIndex);

// "Setable" variables
//-----
// assign GUI "setable" by Text parameter address
dataSetList[0] = (Uint32*)&(SFRA1.Freq_Start);
dataSetList[1] = (Uint32*)&(SFRA1.amplitude);
dataSetList[2] = (Uint32*)&(SFRA1.Freq_Step);

// assign a GUI "getable" parameter array address
arrayGetList[0] = (int32*)FreqVect;
arrayGetList[1] = (int32*)OL_MagVect;
arrayGetList[2] = (int32*)OL_PhaseVect;
arrayGetList[3] = (int32*)Plant_MagVect;
arrayGetList[4] = (int32*)Plant_PhaseVect;
arrayGetList[5] = (int32*)&(SFRA1.Freq_Start);
arrayGetList[6] = (int32*)&(SFRA1.amplitude);
arrayGetList[7] = (int32*)&(SFRA1.Freq_Step);
```

7. Add `SerialCommsTimer++;` to a background task that is called roughly at 1 msec.

```
void A0(void)
{
  // loop rate synchronizer for A-tasks
  if(CpuTimer0Regs.TCR.bit.TIF == 1)
  {
    CpuTimer0Regs.TCR.bit.TIF = 1; // clear flag
    //-----
    (*A_Task_Ptr)(); // jump to an A Task (A1,A2,A3,...)
    //-----
    VTimer0[0]++; // virtual timer 0, instance 0 (spare)
    SerialCommsTimer++; // used by DSP280x_SciCommsGui.c
  }
  Alpha_State_Ptr = &B0 // Comment out to allow only A tasks
}
```

8. Add the following code in a further slower task (executed every 5 msec):

```
SerialHostComms();
```

9. To enable change of FRA parameters from the GUI, add the following code in a slow background task:

```
if(initializationFlag == 1)
{
  SFRA_IQ_INIT(&SFRA1);
  initializationFlag = 0;
  SFRA1.start = 1;
}
```

10. The project will now connect to the SFRA GUI. For more information on how to run the SFRA GUI, see [Section 3.8](#).

3.6 Adding SFRA Library to Assembly Digital Power Library Project

Texas Instruments reference designs, for digital power application using C2000 devices, typically use an assembly digital power ISR that is based on using assembly blocks from the C2000 Digital Power Library. Figure 8 shows how a typical project using assembly digital power library ISR is constructed.

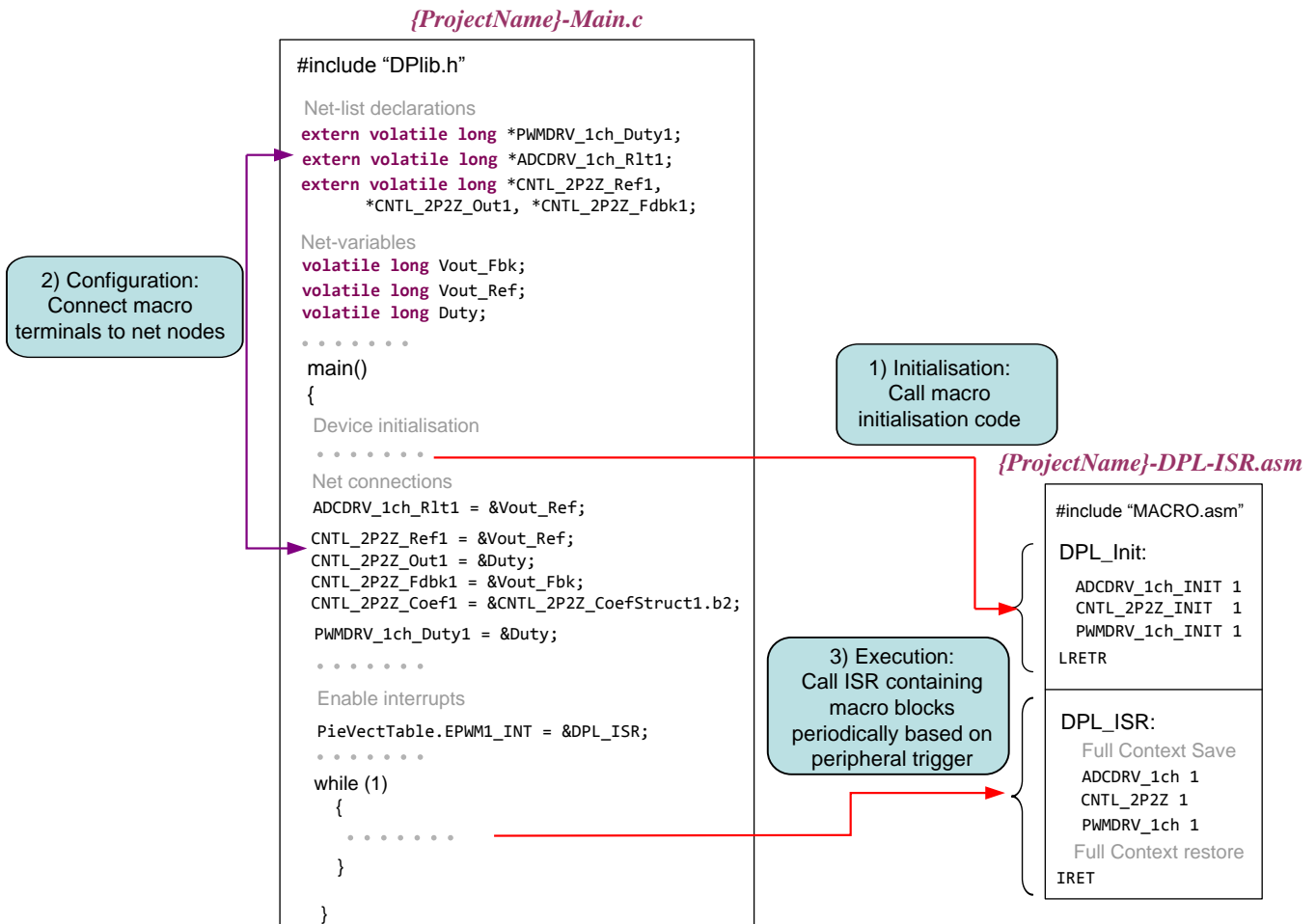


Figure 8. Project Using Assembly Digital Power Library

As the SFRA library is based in C, a C based ISR is needed to call the SFRA routine. Figure 7 shows the modified framework to use the SFRA library with the digital power assembly library. The key changes are listed below:

{ProjectName}-DPL.asm (instead of {ProjectName}-DPL-ISR.asm)

- The assembly file is called {projectname}-DPL.asm instead of {projectname}-DPL-ISR.asm
- The DPL_ISR assembly routine is renamed as DPL_Func
- As DPL_Func is called from the C framework only save on entry registers need to be saved which are XAR1, XAR2 and XAR3.
- Context restore comprises of restoring XAR1, XAR2 and XAR3 only.
- The return from the DPL_Func is LRETR instead of IRETR

{ProjectName}-Main.c

- Add the declaration of the DPL_Func in the main file
- Add declaration for the new C based interrupt called DPL_ISR_wFRA()
- Declare a new variable Vout_ref_wlnj

- Add all the SFRA support items as described in section 3.4 except for the SFRA_Inject and SFRA_Collect calls.
- Change only the connection to the 2p2z macro with Vout_ref_wInj
- Connect the PWM interrupt to *DPL_ISR_wFRA()*
- Inside *DPL_ISR_wFRA()*, add the code as shown in the diagram below to call the SFRA function and DPL_Func.

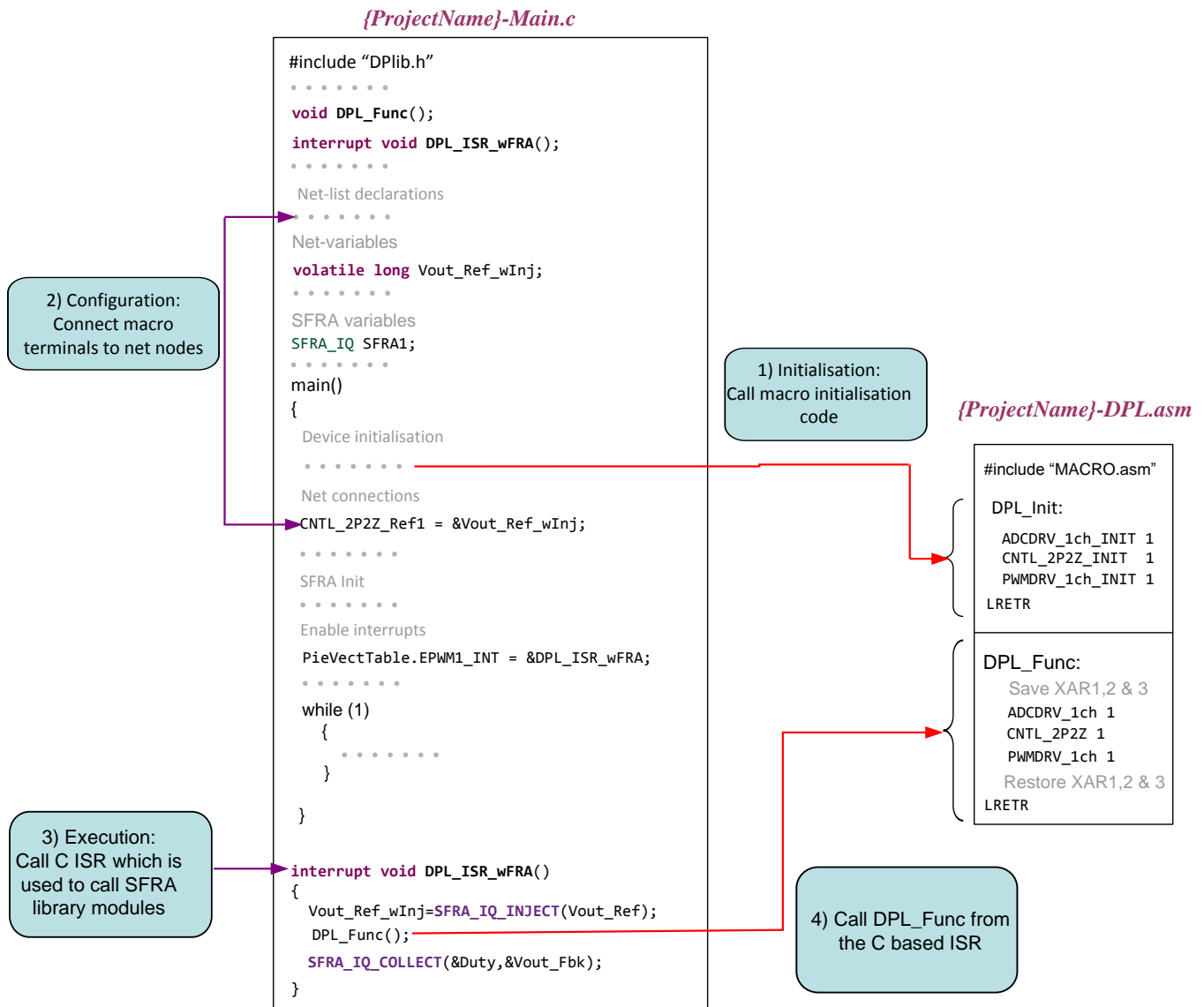


Figure 9. SFRA Added to Assembly Digital Power Library Framework

3.7 Script for Importing Frequency Response and Designing Compensation

The SFRA GUI, upon completing a frequency sweep, can populate a comma separated value(CSV) file or an excel sheet with the frequency response data. When CSV format is selected (which is the default), it can be saved as an excel file. The excel file data This data can be used to fit a pole zero format to extract the transfer function. This transfer function can then be used to design the compensation for the closed loop plant. The scripts for these operations that run on MATLAB are provided under:

SFRA/<version>/Scripts

3.8 SFRA GUI Options and How to Run

The SFRA GUI can be used to start the frequency sweep and display the measured frequency response. Figure 10 illustrates the different options that are presented.

- **MATH Mode Selection Radio Button:** This box must be selected appropriately according to the version of SFRA library being used on the MCU. Because F28035 is a fixed-point device, select fixed point (the project uses SFRA_IQ_Lib.lib) and for F28069, the floating point check box should be selected.
- **Setup Connection Button:** This button is used to select the COM port and set the baud rate. Click on this button and a pop up window requests a Baud Rate selection, which is typically set to 57600. If the device is connected, click “Refresh Comports” and select the serial comport that the target is using.
 - If the comport that the target is connected to is known, please select it.
 - If not, and to find a valid comport, go to:
 - Control Panel → System → Device Manager → Ports (COM and LPT)
 - If using a serial port directly connected to a PC, look for a comport that shows up as “Communications Port” and select this it in the Setup Connection window. If using a USB to Serial adapter, look for the comport that shows “USB Serial Port”, then select this it in the Setup Connection window.

Uncheck “Boot on Connect” to use the GUI in conjunction with a project running from RAM in Code Composer Studio or when using a target that has been flashed with a working SFRA project.
- **Connect and Disconnect Button:** Once the Math mode is selected appropriately and COM port for the communication is selected, click this button to Connect to the MCU. This is the same button that is used for disconnection from the MCU.
- **Connection Status Indicator:** This indicates whether the connection was successfully established or if there were any problems.
- **Frequency Vector Length Label:** Once connected, this label shows the length of the FRA array that is programmed on the MCU.
- **Start Frequency Text Box:** This box shows the start frequency of the SFRA sweep. This can be changed by the user by entering a value in the text box and clicking enter. (Note the value on the controller side is only updated once the user starts a SFRA sweep).
- **Steps Per Decade Text Box:** Specifies the number of frequency points that FRA is performed per decade.
- **Maximum SFRA Frequency Label:** This is the maximum frequency until the SFRA will be performed. This is a function of the start frequency, the steps per decade and the frequency length.
- **Start Sweep Button:** Click this button to begin a frequency sweep on the controller.
- **SFRA Status Bar:** Indicates the progress of SFRA sweep.
- **Drop Down Menu to Select Open Loop or Plant Frequency:** Response to be displayed on the panel to the right.
- **Closed Loop Performance Parameters Panel:** If SFRA is performed on the closed loop, this panel displays the bandwidth, gain margin and the phase margin of the closed loop system.

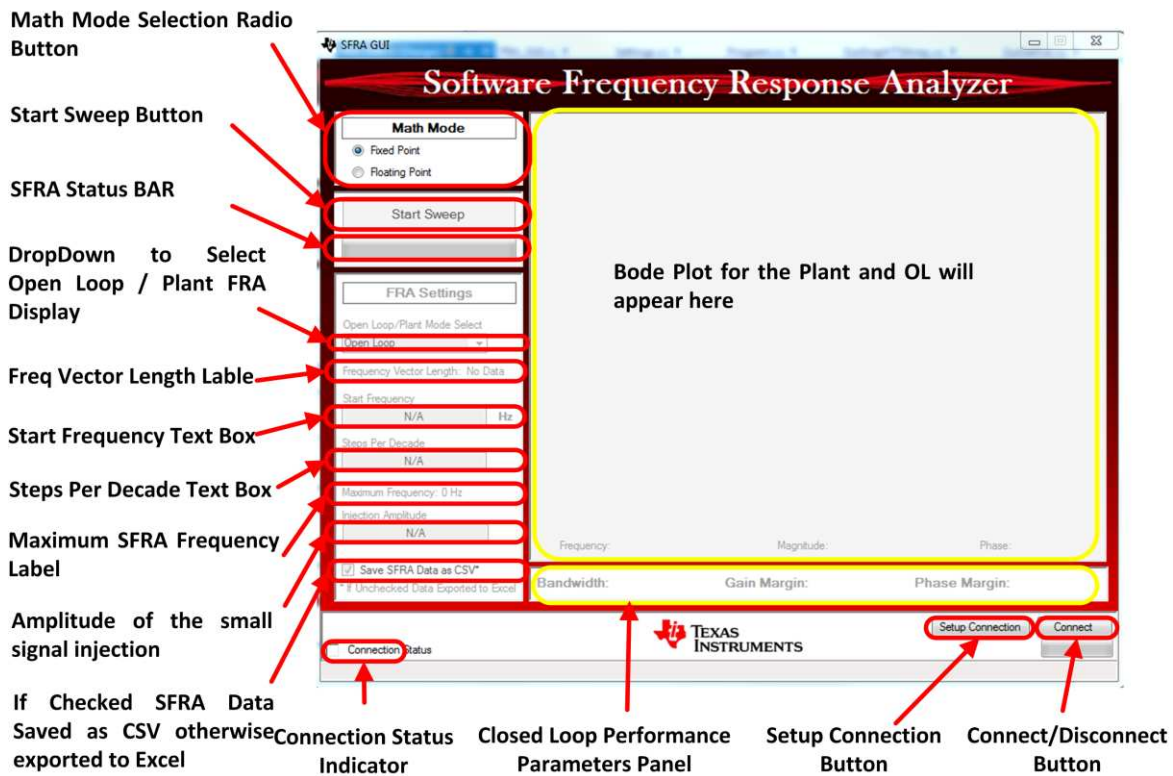


Figure 10. SFRA GUI Options

4 Compensation Designer

TI's Compensation Designer provides an easy GUI interface, to calculate the coefficients that are needed to be programmed onto the micro-controller to implement a compensator as part of a closed loop system. The GUI lets the user select different compensator styles like PID, Two Pole Two Zero, and so forth, and allows the user to save up to five compensator values.

The GUI displays the Plant, the Open Loop and the Compensator Frequency Response. The Pole and Zeroes of the compensator, are marked on the Open Loop Curve on the GUI using a cross and circle, respectively. Critical values such as Bandwidth, Phase Margin and Gain Margin are displayed on the GUI at the bottom, and a label showing "Stable Loop" and "UnStable Loop" is displayed if the system has healthy Bandwidth, Phase Margin and Gain Margin.

For Plant frequency response, depending upon where the Compensation Designer is invoked from, the user can select to use the mathematical model of the power stage or the experimental data gathered using SFRA, to design the compensator.

4.1 Launching Compensation Designer

The compensation design GUI can be launched using two methods.

4.1.1 Standalone From SFRA GUI Folder

Compensation designer can be launched standalone by double clicking on the “CompDesigner.exe” located inside controlSUITE/libs/app_libs/SFRA/<version>/GUI folder. The GUI can be used to design a compensator for desired control performance using plant information that has been gathered using the SFRA GUI in CSV format. The SFRA_GUI.exe puts the latest frequency sweep data inside “controlSUITE/libs/app_libs/SFRA/<version>/GUI/SFRADData.csv” and also keeps all the frequency sweep data with time stamp inside the same folder. The compensation design GUI on launch defaults to this SFRADData.csv file. The user can select, during a session, to use another frequency sweep data file by clicking on “Browse SFRA csv Data”. However on re-launch the GUI will default to the latest SFRADData.csv.

The GUI calculates the coefficients that can be programmed in the Digital Power Library or Solar Library compensator. The user needs to enter the control frequency on the GUI, which is the rate at which the control loop is executed in kHz. This GUI can be used with solutions for which solution adapter GUI does not exist and SFRA has been added.

The GUI lets the user save up to five different compensators, these compensators are saved in the Comp.xml file located inside the same folder from which the GUI is launched. To save a particular compensator setting the user must hit the “Save Comp No” button, otherwise the compensator will not be saved.

When using SFRA data the GUI displays only the frequency response data points for which the SFRA sweep has been performed. If the compensation zero or pole is not in that range it is not marked on the GUI.

Figure 11 shows the Compensation Designer GUI launched from the SFRA folder. Hovering over the drop down box for compensation style select, displays the exact equation used to generate the coefficients.

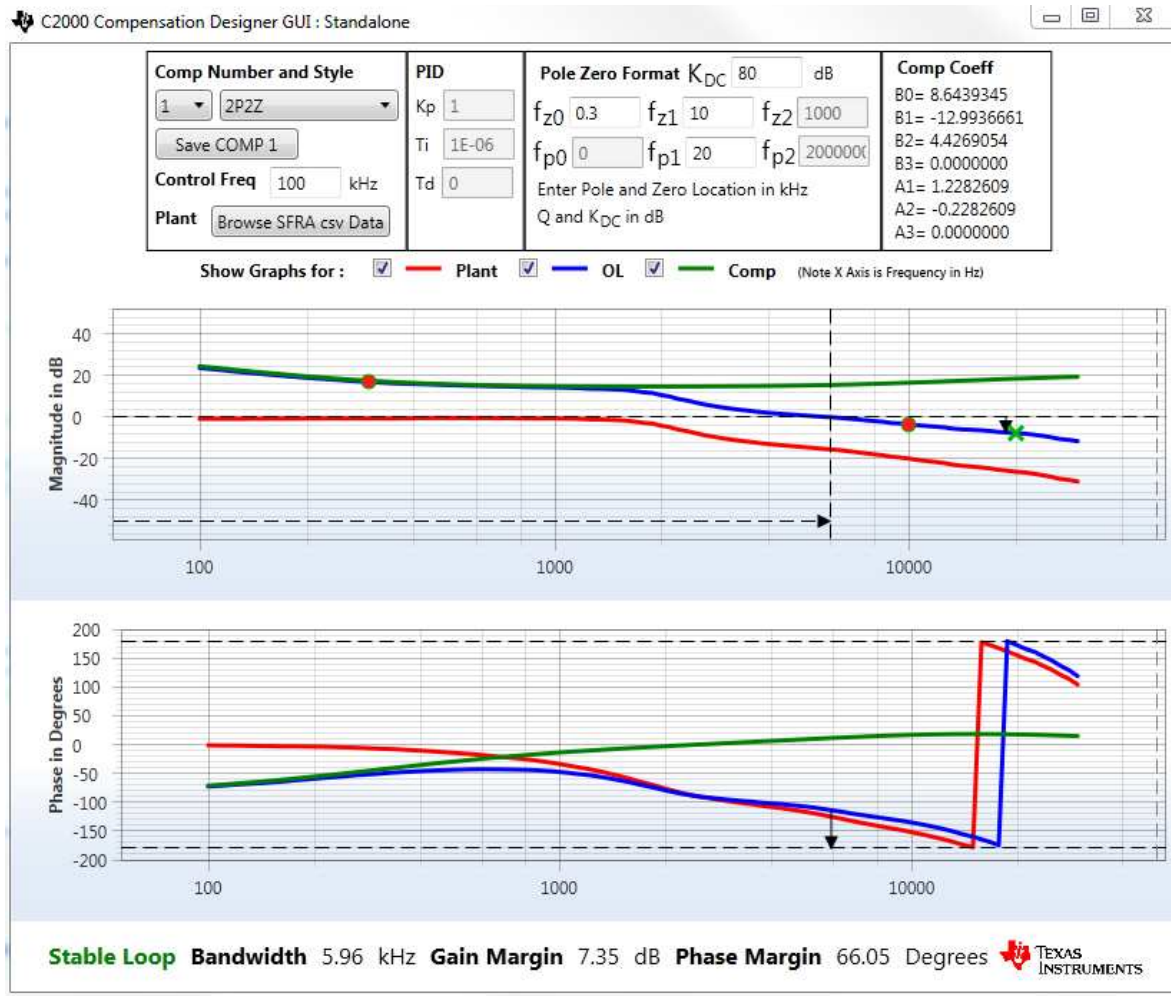


Figure 11. Standalone Compensation Designer When launched From the controlSUITE/libs/app_libs/SFRA/<version>/GUI Folder

4.1.2 From Solution Adapter Page

C2000 kits may support a solution adapter page, which lets the user launch the compensation designer directly from the solution adapter page. When launched from this page, the compensation designer models the power stage based on the parameters specified on the solution adapter page for inductance, capacitor, and so forth.

The control frequency that is used to calculate the compensator coefficients is also specified on the solution adapter page. The modeled plant can serve as a good starting point in the compensation design process before SFRA is run.

The user can also select SFRA data for the Plant Frequency Response and use the measured data to design the compensator just like when launched in standalone fashion.

NOTE: When Modelled Plant Option is selected the browsed SFRA Data csv file is not used. Before selecting SFRA Data, the user must browse to the appropriate SFRA csv file.

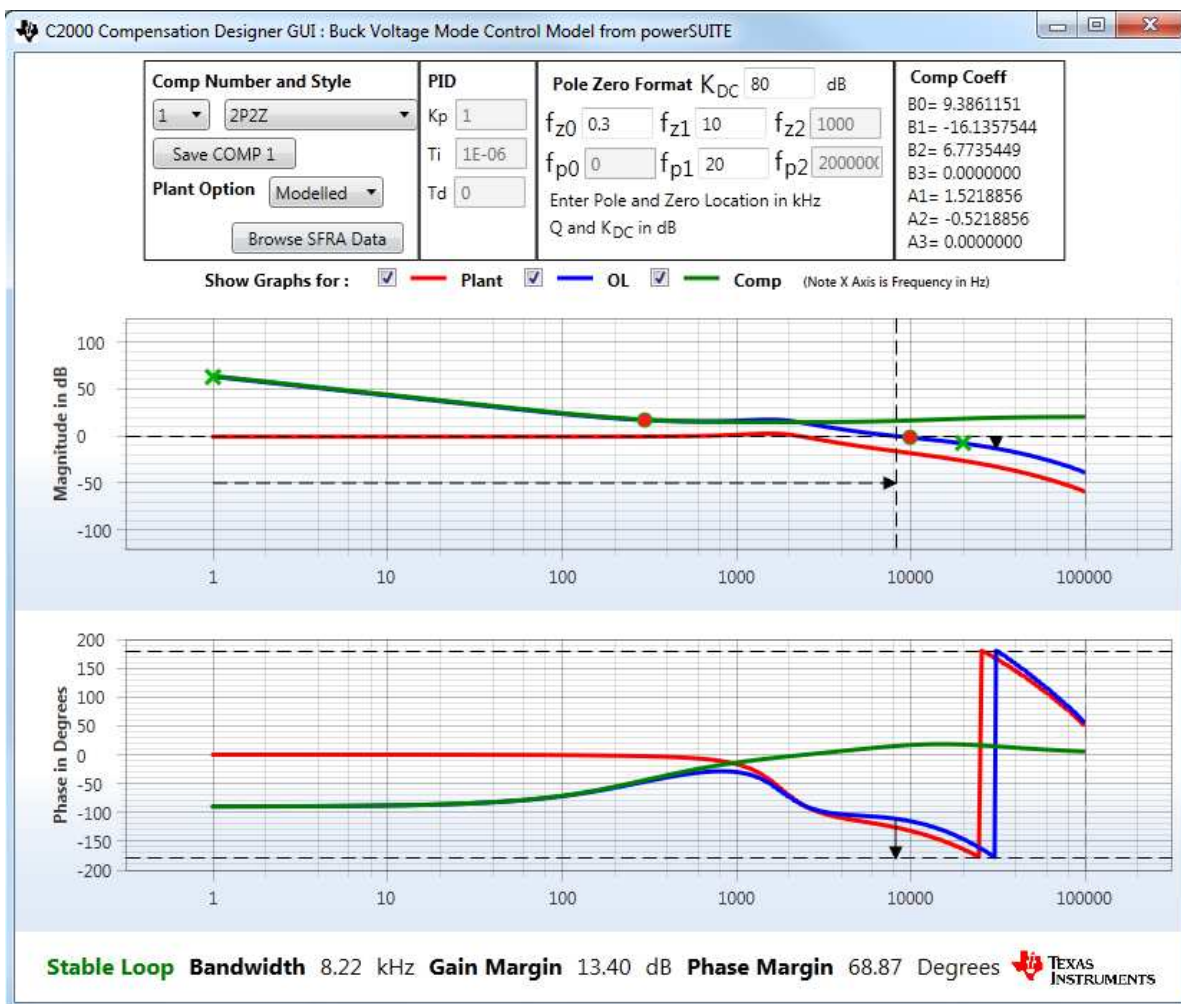


Figure 12. Compensation Designer When Launched From Solution Adapter Page (both Modeled and SFRA data-based Plant Information can be used for compensation design)

4.2 Compensator Structure

The compensation designer generates coefficients for the following structure of the compensator that is implemented in the C2000 Digital Power Library and C2000 Solar Library compensator block.

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3}}{1 - a_1z^{-1} - a_2z^{-2} - a_3z^{-3}} \quad (3)$$

The coefficients that need to be programmed on the controller are displayed on the compensation designer GUI, and can be copied from there.

```

Comp Coeff
B0= 8.6439345
B1= -12.9936661
B2= 4.4269054
B3= 0.0000000
A1= 1.2282609
A2= -0.2282609
A3= 0.0000000
    
```

Figure 13. Compensator Coefficients Calculated by the GUI

The C2000 Digital Power Library based on assembly macros accepts coefficients in IQ26 format, which has a max range of (+32 to -32). The Solar Library accepts compensator coefficients in IQ26 with a range of (+128 to -128). Similarly, the Solar Library, when run on floating-point devices, accepts single precision floating-point variables. The Compensation Designer is un-aware of such restrictions of the controller and, therefore, the user must ensure the coefficients that are generated are reasonable for the micro-controller that they will finally be run on.

When the compensation designer is invoked from the solution adapter page of a solution, because this is device and implementation specific, the compensation designer will show a warning in case the compensation coefficients exceed the limits of what can be implemented on the controller. A warning sign is displayed on the compensation coefficient panel; and when a save attempt is made for these values of the compensator, a warning dialog box is shown (see [Figure 14](#)).

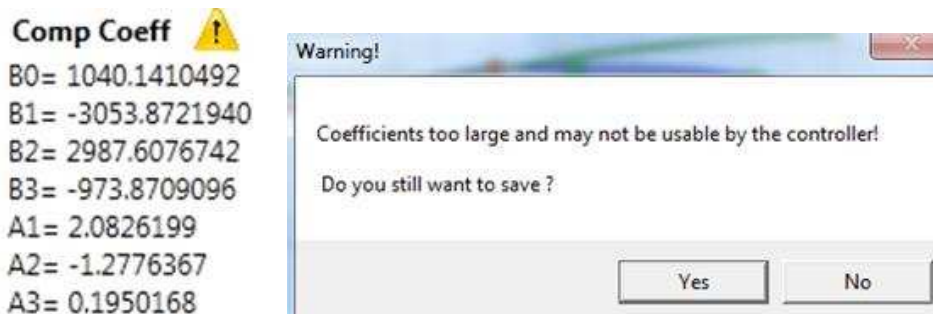


Figure 14. Warning Sign and Dialog When Compensation Coeff Range is Exceeded

4.3 Compensation Style and Number

Different styles of compensation are supported by the Compensation Designer and are listed in [Table 5](#).

Table 5. Different Compensation Style Supported by Compensation Designer

Compensator Number	Style	Description
1	PID	Proportional-Integral-Derivative Controller
2	2P2Z	Two Pole Two Zero
3	3P3Z	Three Pole Three Zero
4	2P2Z 1CZ	Two Pole Two Zero with One Complex Zero
5	3P3Z 1CP	Three Pole Three Zero with One Complex Pole
6	3P3Z 1CZ	Three Pole Three Zero with One Complex Zero
7	3P3Z 1CZ 1CP	Three Pole Three Zero with One Complex Pole and One Complex Zero

The GUI enables designing up to five compensators each of which can be tuned using any of the different compensation styles. The Compensation Style and Number can be selected using the drop down box.

Any changes to the compensation value are not saved until the button “Save COMP<number> is selected (see [Figure 15](#)). This prevents any inadvertent change to the compensation values.

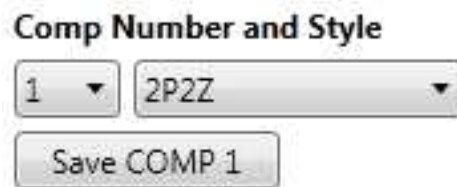


Figure 15. Compensation Number and Style Selection

Depending on which compensation style is selected different panels on the GUI will become active. The following is the description of each compensation style.

NOTE: Assuming F_s is the control loop frequency or the rate at which the control loop is executed and $T=1/F_s$.

4.3.1 PID (Proportional-Integral Derivative Controller)

When this compensation style is selected, the PID panel becomes active.



Figure 16. PID Panel

The user can enter values such as K_p , T_i and T_d and in the GUI, which are used to implement a compensator given in analog form as:

$$G(s) = K_P + \frac{K_I}{s} + K_D s \quad (4)$$

Where, $K_I = \frac{K_P}{T_I}$ and $K_D = K_P * T_d$.

To discretize this compensator, using Tustin/Trapezoidal Transformation, for example, $s = 2F_S \frac{(z-1)}{(z+1)}$ for the integral term and Backward Euler Transformation $s = F_S \frac{(z-1)}{z}$ for the derivative term the coefficients for a digital compensator can be written as shown in Equation 5.

$$G(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-1}} \tag{5}$$

Where,

$$\begin{aligned} b_0 &= K'_P + K'_I + K'_D \\ b_1 &= -K'_P + K'_I - 2K'_D \\ b_2 &= K'_D \end{aligned} \tag{6}$$

and

$$\begin{aligned} K'_P &= K_P \\ K'_I &= \frac{T}{2} K_I \\ K'_D &= \frac{1}{T} K_D \end{aligned} \tag{7}$$

4.3.2 Two Pole Two Zero (2P2Z)

Two pole two zero is a common compensator style used in power stage design. Following is the structure of the analog compensator.

$$G(s) = K_{DC} \left(\frac{p_1}{z_0 z_1} \right) \frac{(s + z_0)(s + z_1)}{s(s + p_1)} \tag{8}$$

Where,

$$z_0 = 2\pi * f_{z0}, \quad z_1 = 2\pi * f_{z1}, \quad p_1 = 2\pi * f_{p1} \tag{9}$$

When this compensator is selected from the compensation style drop box, the Pole Zero Format Panel becomes active and can be used to enter the pole and zero locations (in kHz). The K_{DC} value is entered in the gain field in dB's.

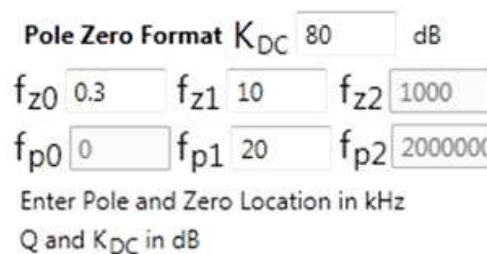


Figure 17. Two Pole Two Zero Pole Zero Entry in Compensation Designer GUI

For digital implementation, Tustin transform is used $s = 2F_S \frac{(z-1)}{(z+1)}$ and the coefficients of the digital compensator are derived as shown in Equation 10.

$$G(z) = \left(K_{DC} \left(\frac{p_1}{z_0 z_1} \right) \right) \left(\frac{1}{2F_S * (p_1 + 2F_S)} \right) \frac{((z_1 + 2F_S) * (z_0 + 2F_S)) + ((z_0 + 2F_S) * (z_1 - 2F_S) + (z_1 + 2F_S) * (z_0 - 2F_S))z^{-1} + ((z_1 - 2F_S) * (z_0 - 2F_S))z^{-2}}{1 + \left(\frac{-4F_S}{(p_1 + 2F_S)} \right) z^{-1} + \left(\frac{2F_S - p_1}{(2F_S + p_1)} \right) z^{-2}} \quad (10)$$

4.3.3 Three Pole Three Zero (3P3Z)

Three pole three zero is a common compensator style used in power stage design. Equation 11 is the structure of the analog compensator.

$$G(s) = \left(K_{DC} \frac{p_1 p_2}{z_0 z_1 z_2} \right) \frac{(s + z_0)(s + z_1)(s + z_2)}{s(s + p_1)(s + p_2)} \quad (11)$$

Where,

$$z_0 = 2\pi * f_{z0}, \quad z_1 = 2\pi * f_{z1}, \quad z_2 = 2\pi * f_{z2}, \quad p_1 = 2\pi * f_{p1}, \quad p_2 = 2\pi * f_{p2} \quad (12)$$

When this compensator is selected from the compensation style drop box, the Pole Zero Format Panel becomes active and can be used to enter the pole and zero locations (in kHz). The K_{DC} value is entered in the gain field in dB's.

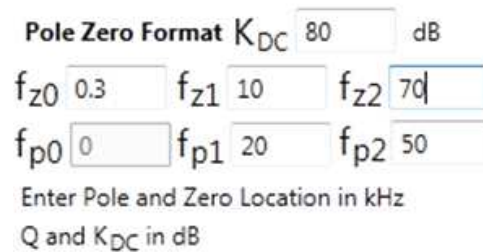


Figure 18. Three Pole Three Zero Pole Zero Entry in Compensation Designer GUI

For digital implementation, Tustin transform is used $s = 2F_S \frac{(z-1)}{(z+1)}$ and the coefficients of the digital compensator are derived as shown in Equation 13:

$$G(z) = \left(K_{DC} \frac{p_1 p_2}{z_0 z_1 z_2} \right) \left(\frac{1}{2F_S * (p_1 + 2F_S) * (p_2 + 2F_S)} \right) \frac{((z_2 + 2F_S) * (z_1 + 2F_S) * (z_0 + 2F_S)) + ((z_0 + 2F_S) * (z_1 + 2F_S) * (z_2 - 2F_S) + (z_1 + 2F_S) * (z_2 + 2F_S) * (z_0 - 2F_S) + (z_0 + 2F_S) * (z_2 + 2F_S) * (z_1 - 2F_S))z^{-1} + ((z_2 + 2F_S) * (z_1 - 2F_S) * (z_0 - 2F_S) + (z_1 + 2F_S) * (z_2 - 2F_S) * (z_0 - 2F_S) + (z_0 + 2F_S) * (z_2 - 2F_S) * (z_1 - 2F_S))z^{-2} + ((z_2 - 2F_S) * (z_1 - 2F_S) * (z_0 - 2F_S))z^{-3}}{1 + \left(\frac{(p_2 - 2F_S) * (p_1 + 2F_S) - 4F_S * (p_2 + 2F_S)}{(p_1 + 2F_S) * (p_2 + 2F_S)} \right) z^{-1} + \left(\frac{-(p_1 - 2F_S) * (p_2 + 2F_S) - 4F_S * (p_2 - 2F_S)}{(p_1 + 2F_S) * (p_2 + 2F_S)} \right) z^{-2} + \left(\frac{-(p_1 - 2F_S) * (p_2 - 2F_S)}{(p_1 + 2F_S) * (p_2 + 2F_S)} \right) z^{-3}} \quad (13)$$

4.3.4 Two Pole Two Zero With One Complex Zero (2P2Z + 1CZ)

Two Pole Two Zero with one complex zero compensator structure is shown in Equation 14.

$$G(s) = K_{DC} p_1 \frac{\left(\frac{s^2}{\omega_r z} + \frac{s}{\omega_r z Q r z} + 1 \right)}{s(s + p_1)} \quad (14)$$

Where,

$$\omega_{rZ} = 2\pi * f_{rZ}, \quad p_1 = 2\pi * f_{p1} \tag{15}$$

is the quality factor of the resonant zero.

When this compensator is selected from the compensation style drop box the Pole Zero Format Panel becomes active and can be used to enter the pole and zero locations (in kHz) and the Quality factor Q_z of the complex zero (ω_{rZ}). The K_{DC} value is entered in the gain field in dB's.

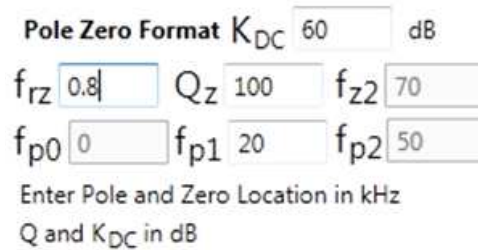


Figure 19. Two Pole Two Zero With One Complex Zero Entry in Compensation Designer GUI

For digital implementation, Tustin transform is used $s = 2F_S \frac{(z-1)}{(z+1)}$ and the coefficients of the digital compensator derived as shown in Equation 16.

$$G(z) = (K_{DC} * p_1) * \left(\frac{1}{Q_z \omega_{rZ}^2 * 2F_S * (p_1 + 2F_S)} \right) * \left(\frac{(4F_S^2 Q_z + 2F_S \omega_{rZ} + Q_z \omega_{rZ}^2) + (-8F_S^2 Q_z + 2Q_z \omega_{rZ}^2)z^{-1} + (4F_S^2 Q_z - 2F_S \omega_{rZ} + Q_z \omega_{rZ}^2)z^{-2}}{1 + \left(\frac{-4F_S}{(p_1 + 2F_S)} \right)z^{-1} - \left(\frac{p_1 - 2F_S}{(p_1 + 2F_S)} \right)z^{-2}} \right) \tag{16}$$

4.3.5 Three Pole Three Zero With One Complex Pole (3P3Z + 1CP)

Three Pole Three Zero with one complex pole compensator structure is shown in Equation 17.

$$G(s) = \frac{K_{DC} (s+z_0)(s+z_1)(s+z_2)}{z_0 z_1 z_2} s \left(\frac{\frac{s^2}{\omega_{rp}^2} + \frac{s}{\omega_{rp} Q_p} + 1 \right) \tag{17}$$

Where,

$$\omega_{rZ} = 2\pi * f_{rZ}, \quad z_2 = 2\pi * f_{z2}, \quad p_1 = 2\pi * f_{p1}, \quad p_2 = 2\pi * f_{p2}, \quad Q_z \text{ is the quality factor of the resonant zero.}$$

When this compensator is selected from the compensation style drop box, the Pole Zero Format Panel becomes active and can be used to enter the pole and zero locations (in kHz) and the Quality factor Q_z of the complex zero (ω_{rp}). The K_{DC} value is entered in the gain field in dB's.

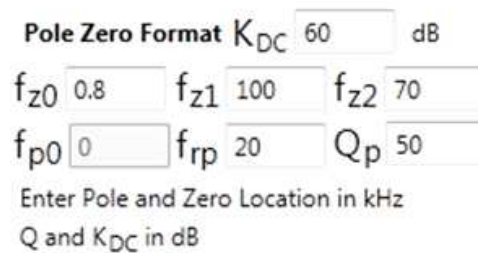


Figure 20. Three Pole Three Zero With One Complex Pole Entry in Compensation Designer GUI

For digital implementation, Tustin transform is used $s = 2F_S \frac{(z-1)}{(z+1)}$ and the coefficients of the digital compensator derived as shown in Equation 18.

$$G(z) = \left(\frac{K_{DC} p_1^2 * Q_p}{2F_S * z_0 * z_1 * z_2} \right) \frac{1}{(4F_S^2 Q_p + 2F_S \omega_{rp} + Q_p \omega_{rp}^2)} * \left(\begin{array}{l} ((z_2 + 2F_S) * (z_1 + 2F_S) * (z_0 + 2F_S)) + \\ ((z_0 + 2F_S) * (z_1 + 2F_S) * (z_2 - 2F_S) + (z_1 + 2F_S) * (z_2 + 2F_S) * (z_0 - 2F_S) + (z_0 + 2F_S) * (z_2 + 2F_S) * (z_1 - 2F_S))z^{-1} + \\ ((z_2 + 2F_S) * (z_1 - 2F_S) * (z_0 - 2F_S) + (z_1 + 2F_S) * (z_2 - 2F_S) * (z_0 - 2F_S) + (z_0 + 2F_S) * (z_2 - 2F_S) * (z_1 - 2F_S))z^{-2} + \\ ((z_2 - 2F_S) * (z_1 - 2F_S) * (z_0 - 2F_S))z^{-3} \end{array} \right) \frac{1}{\left(\begin{array}{l} + \left(\frac{-12F_S^2 Q_p - 2F_S \omega_{rp} + Q_p \omega_{rp}^2}{4F_S^2 Q_p + 2F_S \omega_{rp} + Q_p \omega_{rp}^2} \right) z^{-1} \\ + \left(\frac{12F_S^2 Q_p - 2F_S \omega_{rp} - Q_p \omega_{rp}^2}{4F_S^2 Q_p + 2F_S \omega_{rp} + Q_p \omega_{rp}^2} \right) z^{-2} \\ + \left(\frac{-4F_S^2 Q_p + 2F_S \omega_{rp} - Q_p \omega_{rp}^2}{4F_S^2 Q_p + 2F_S \omega_{rp} + Q_p \omega_{rp}^2} \right) z^{-3} \end{array} \right)} \quad (18)$$

4.3.6 Three Pole Three Zero With One Complex Zero (3P3Z + 1 CZ)

Three Pole Three Zero with one complex zero compensator structure is shown in Equation 19

$$G(s) = \frac{K_{DC} p_1 p_2}{z^2} \left(\frac{\frac{s^2}{\omega_{rz}^2} + \frac{s}{\omega_{rz} Q_z} + 1}{s(s+p_1)(s+p_2)} \right) (s+z_2) \quad (19)$$

Where,

$\omega_{rz} = 2\pi * f_{rz}$, $z_2 = 2\pi * f_{z2}$, $p_1 = 2\pi * f_{p1}$, $p_2 = 2\pi * f_{p2}$, Q_z is the quality factor of the resonant zero.

When this compensator is selected from the compensation style drop box, the Pole Zero Format Panel becomes active and can be used to enter the pole and zero locations (in kHz) and the Quality factor Q_z of the complex zero (ω_{rz}). The K_{DC} value is entered in the gain field in dB's.

Pole Zero Format K_{DC} dB

f_{rz} Q_z f_{z2}

f_{p0} f_{p1} f_{p2}

Enter Pole and Zero Location in kHz

Q and K_{DC} in dB

Figure 21. Three Pole Three Zero With One Complex Zero Entry in Compensation Designer GUI

For digital implementation, Tustin transform is used $s = 2F_S \frac{(z-1)}{(z+1)}$ and the coefficients of the digital compensator derived as shown in Equation 20.

$$G(z) = \left(\frac{K_{DC} p_1 * p_2}{2F_S * Q_z * \omega_p^2 * z_2} \right) \frac{1}{(2F_S + p_1) * (2F_S + p_2) *} \left(\begin{array}{l} (4F_S^2 Q_z + 2F_S \omega_{rZ} + Q_z \omega_{rZ}^2)(2F_S + z_2) \\ + \left((4F_S^2 Q_z + 2F_S \omega_{rZ} + Q_z \omega_{rZ}^2)(-2F_S + z_2) + (-8F_S^2 Q_z + 2Q_z \omega_{rZ}^2)(2F_S + z_2) \right) z^{-1} \\ + \left((4F_S^2 Q_z - 2F_S \omega_{rZ} + Q_z \omega_{rZ}^2)(2F_S + z_2) + (-8F_S^2 Q_z + 2Q_z \omega_{rZ}^2)(-2F_S + z_2) \right) z^{-2} \\ + (4F_S^2 Q_z - 2F_S \omega_{rZ} + Q_z \omega_{rZ}^2)(-2F_S + z_2) z^{-3} \\ 1 + \left(\frac{(p_2 - 2F_S) * (p_1 + 2F_S) - 4F_S * (p_2 + 2F_S)}{(p_1 + 2F_S) * (p_2 + 2F_S)} \right) z^{-1} + \left(\frac{-(p_1 - 2F_S) * (p_2 + 2F_S) - 4F_S * (p_2 - 2F_S)}{(p_1 + 2F_S) * (p_2 + 2F_S)} \right) z^{-2} + \\ \left(\frac{-(p_1 - 2F_S) * (p_2 - 2F_S)}{(p_1 + 2F_S) * (p_2 + 2F_S)} \right) z^{-3} \end{array} \right) \quad (20)$$

4.3.7 Three Pole Three Zero With One Complex Pole and One Complex Zero (3P3Z 1CP 1CZ)

Three Pole Three Zero with one complex pole and one complex zero compensator structure is shown in Equation 21.

$$G(s) = \frac{K_{DC}}{z_2} \frac{\left(\frac{s^2}{\omega_{rZ}^2} + \frac{s}{\omega_{rZ} Q_z} + 1 \right) (s + z_2)}{s \left(\frac{s^2}{\omega_{rp}^2} + \frac{s}{\omega_{rp} Q_p} + 1 \right)} \quad (21)$$

Where,

$\omega_{rZ} = 2\pi * f_{rZ}$, $z_2 = 2\pi * f_{z2}$, $\omega_{rp} = 2\pi * f_{rp}$, Q_p is the quality factor of the resonant pole, Q_z is the quality factor of the resonant zero.

When this compensator is selected from the compensation style drop box, the Pole Zero Format Panel becomes active and can be used to enter the pole and zero locations (in kHz) and the Quality factor Q_z of the complex zero (cp or ω_{rp}) and the Quality factor Q_p of the complex pole (cp or ω_{rp}). The K_{DC} value is entered in the gain field in dB's.

Pole Zero Format K_{DC} dB

f_{z0} f_{z1} f_{z2}

f_{p0} f_{rp} Q_p

Enter Pole and Zero Location in kHz
Q and K_{DC} in dB

Figure 22. Three Pole Three Zero With One Complex Zero Entry in Compensation Designer GUI

For digital implementation, Tustin transform is used $s = 2F_S \frac{(z-1)}{(z+1)}$ and the coefficients of the digital compensator derived as shown in Equation 20.

$$G(z) = \left(\frac{KDC}{z^2} \right) \left(\frac{p_1^2 * Q_p}{2F_S * Q_Z \omega_r^2} \right) \frac{1}{(4F_S^2 Q_p + 2F_S \omega_r p + Q_p \omega_r^2)} * \left(\begin{array}{l} (4F_S^2 Q_Z + 2F_S \omega_r Z + Q_Z \omega_r^2)(2F_S + z^2) z^3 \\ + \left((4F_S^2 Q_Z + 2F_S \omega_r Z + Q_Z \omega_r^2)(-2F_S + z^2) + (-8F_S^2 Q_Z + 2Q_Z \omega_r^2)(2F_S + z^2) \right) z^2 \\ + \left((4F_S^2 Q_Z - 2F_S \omega_r Z + Q_Z \omega_r^2)(2F_S + z^2) + (-8F_S^2 Q_Z + 2Q_Z \omega_r^2)(-2F_S + z^2) \right) z^1 \\ + (4F_S^2 Q_Z - 2F_S \omega_r Z + Q_Z \omega_r^2)(-2F_S + z^2) \end{array} \right) \frac{1}{z^3} + \left(\frac{-12F_S^2 Q_p - 2F_S \omega_r p + Q_p \omega_r^2}{4F_S^2 Q_p + 2F_S \omega_r p + Q_p \omega_r^2} \right) z^2 + \left(\frac{12F_S^2 Q_p - 2F_S \omega_r p - Q_p \omega_r^2}{4F_S^2 Q_p + 2F_S \omega_r p + Q_p \omega_r^2} \right) z + \left(\frac{-4F_S^2 Q_p + 2F_S \omega_r p - Q_p \omega_r^2}{4F_S^2 Q_p + 2F_S \omega_r p + Q_p \omega_r^2} \right)$$

(22)

5 Case Study

To understand the use of SFRA, the SFRA library is integrated into a software example that runs on the TI DPS workshop board. More details on this board can be found on the [C2000 DPSWorkshop wiki page](#).

The following section provides a brief overview to facilitate running of the SFRA library project on DPSWorkshop board.

5.1 DPS Workshop Board Overview

The DPS Workshop board has two buck channels. The example code for this board using SFRA is provided in controlSUITE at:

- Fixed point IQ Math

```
libs/app_libs/SFRA/vX/examples/DPSWrkShpKit_F28035x_SFRA
```

- Float Math

```
libs/app_libs/SFRA/vX/examples/DPSWrkShpKit_F28069_SFRA
```

The concept of using SFRA to extract the plant and measure the closed loop frequency characteristics is illustrated in this document using the fixed-point library. The example code for the floating point is provided but not discussed here for succinctness. The document assumes that the user is familiar with running the DPS workshop board. If needed, the workshop detailed on the wiki link mentioned in Section 5 provides a good introduction to this board.

The DPS workshop board consists of two identical DC-DC buck power stages. The input bus voltage for both stages is 9 V. Figure 23 shows the location of the switches and power stages on the EVM board.

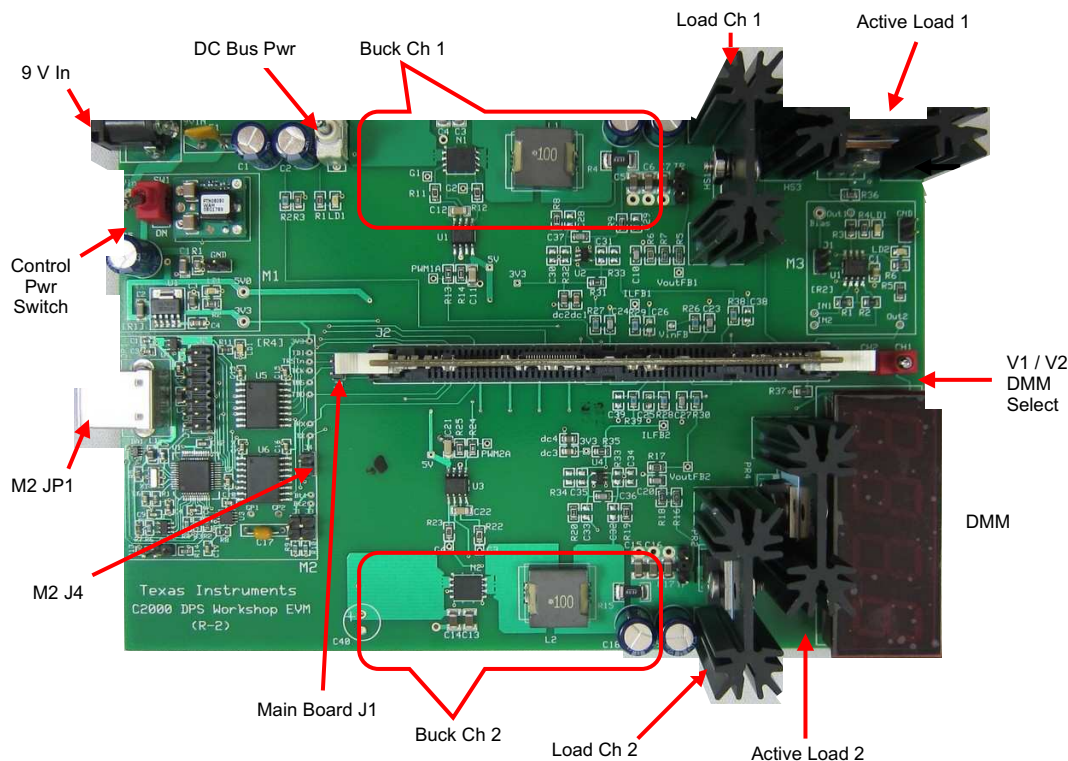


Figure 23. DPS Workshop Board Overview

Table 6. DPS Workshop Board Key Switches and Components List

9V In	DC power supply from plug pack
Control Pwr	SW1-M1 - Power switch for control logic and driver circuit
DC Bus Pwr	SW1-Main - Power switch for Vin to buck stages only. When off F28035 DIMM controller card can still operate.
Buck 1, 2	Synchronous buck power stage (includes TI NEXFET modules)
Load 1, 2	7.5 Ω resistive loads permanently connected across the two outputs
Active Load 1, 2	Software controlled switched load (2 Ω each)
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 DMM Select	SW2 (Main) - Selects between output voltage of buck 1 and 2 to be displayed on the DMM

The software provided with the library only uses the buck channel 1. The key signal connections between the F28035 microcontroller and the buck1 stage are listed in [Table 7](#).

Table 7. DPS Workshop Board Resource Allocation

Signal Name	Description	Connection to F28035
EPWM-1A	PWM duty control signal for buck stage 1	GPIO-00
EPWM-2A	PWM duty control signal for buck stage 2	GPIO-02
VoutFB-1	Output voltage feedback for buck stage 1	ADC-A6
ILFB-1	Inductor current feedback for buck stage 1	ADC-A2
VinFB	Input voltage feedback	ADC-B1

The projects are to be used with CCSv6 or later.

5.2 Plant TF Extraction

If the plant model is not known the plant model can be identified by running the SFRA in open loop. For this, a DC operating point is chosen by providing a fixed duty cycle in case of a buck stage. The SFRA_INJECT function is used to add small signal injection to the duty value. The SFRA_COLLECT function is used to analyze data from the excitation and calculate the plant transfer function, for example, (y/u) which, in case of a voltage controlled power supply, will be $Vout/Duty$. [Figure 24](#) illustrates the software diagram for the SFRA inclusion.

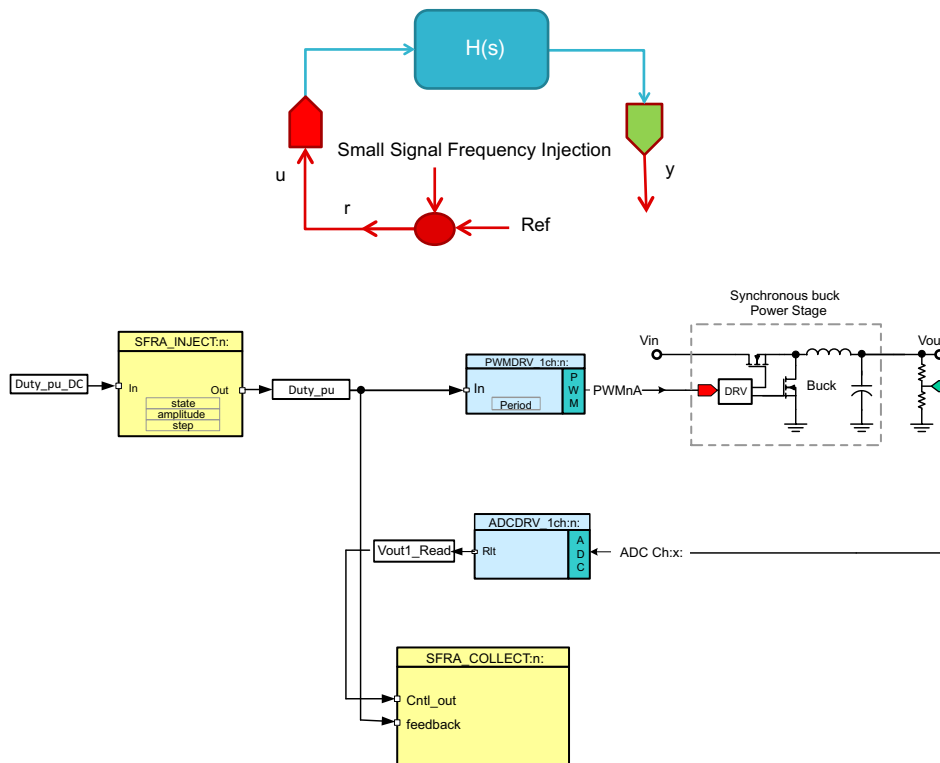


Figure 24. Software Diagram for Plant tf Extraction

1. Import the project under SFRA/vX/examples/DPSWrkShpKit_SFRA_F28035 in CCS6.0 or later. (Make sure that the check box *Copy Project into workspace* is unchecked.) *DPSWrkShpKit* will be referred to as <ProjectName> in the document. Go to the <ProjectName>-Settings.h file.
2. Set the incremental build level to 1.

```
//=====
// Incremental Build options for System check-out
//=====
// BUILD 1 Open Loop PWMDRV_lch using PWM 1 on C28x with FRA
// BUILD 2 Closed Loop PWMDRV_lch using PWM1 on C28x with FRA
```

```
#define INCR_BUILD 1
```

3. In the Main.c file, locate and inspect the definitions and the object declaration of the SFRA library:

```
#include "SFRA_F_Include.h"
#define SFRA_ISR_FREQ 200000
#define SFRA_FREQ_LENGTH 100
#define SFRA_FREQ_START 100
//SFRA step Multiply = 10^(1/No of steps per decade(40))
#define FREQ_STEP_MULTIPLY (float)1.059253
```

```
...
// SFRA lib Object
SFRA_IQ SFRA1;
// SFRA Variables
int32 Plant_MagVect[SFRA_FREQ_LENGTH];
int32 Plant_PhaseVect[SFRA_FREQ_LENGTH];
int32 OL_MagVect[SFRA_FREQ_LENGTH];
int32 OL_PhaseVect[SFRA_FREQ_LENGTH];
float32 FreqVect[SFRA_FREQ_LENGTH];
//Flag for reinitializing SFRA variables
int16 initializationFlag;
//extern to access tables in ROM
extern long IQsinTable[];
```

The buck stage on the DPS Workshop board switches at 200 KHz, the length of the frequency sweep is chosen to be 100, and the start frequency is selected as 100Hz. The step value is chosen to provide fixed number of points in a decade of frequency.

4. Observe that in order to enable connection with the SFRA GUI, the SCI port is initialized for a 57600 baud rate and connection to the GUI arrays is done as shown in the following code:

```
// 20000000 is the LSPCLK or the Clock used for the SCI Module
// 57600 is the Baudrate desired of the SCI module
SCIA_Init(20000000, 57600);
CommsOKflg = 0;
SerialCommsTimer = 0;
//"Set" variables
// assign GUI Buttons to desired flag addresses
varSetBtnList[0] = (int16*)&initializationFlag;

//"Get" variables
//-----
// assign a GUI "getable" parameter address
varGetList[0] = (int16*)&(SFRA1.Vec_Length); //int16
varGetList[1] = (int16*)&(SFRA1.status); //int16
varGetList[2] = (int16*)&(SFRA1.FreqIndex); //int16

//"Setable" variables
//-----
// assign GUI "setable" by Text parameter address
dataSetList[0] = (Uint32*)&(SFRA1.Freq_Start); //Float 32
dataSetList[1] = (Uint32*)&(SFRA1.amplitude); //Float32
dataSetList[2] = (Uint32*)&(SFRA1.Freq_Step); //Float32
```

```

// assign a GUI "getable" parameter array address
arrayGetList[0] = (int32*)FreqVect;           //Float 32
arrayGetList[1] = (int32*)OL_MagVect;        //
arrayGetList[2] = (int32*)OL_PhaseVect;     //
arrayGetList[3] = (int32*)Plant_MagVect;    //
arrayGetList[4] = (int32*)Plant_PhaseVect;  //
arrayGetList[5] = (int32*)&(SFRA1.Freq_Start); //Float32
arrayGetList[6] = (int32*)&(SFRA1.amplitude); //Float32
arrayGetList[7] = (int32*)&(SFRA1.Freq_Step); //Float32

```

- Next, look at the SFRA initialization code. First, the amplitude of the small signal injection is specified. This amplitude is in per unit format. In the following code, the injection amplitude is specified at 1%, which means the reference of the power stage will be perturbed by 100mV if the maximum measurable voltage value is 10 V. Second, the vector length or the number of points on which the FRA is performed is specified. This is also the length of the arrays defined in the previous step. The rate at which the SFRA library is called is specified next. The frequencies at which the response is calculated are uniquely specified by the frequency start variable and the frequency step multiply. The combination of the frequency sweep start, step multiply and length determine the frequency values at which the frequency response is computed. The arrays used to store the response are then linked to the SFRA object.

NOTE: The SFRA GUI enables changing the start frequency, the steps per decade and injection amplitude at run time.

```

SFRA1.amplitude=_IQ26(0.01);
//Specify the length of SFRA
SFRA1.Vec_Length=SFRA_FREQ_LENGTH;
//Specify the SFRA ISR Frequency
SFRA1.SFRA_Freq=SFRA_ISR_FREQ;
//Specify the Start Frequency of the SFRA analysis
SFRA1.Freq_Start=SFRA_FREQ_START;
//Specify the Frequency Step
SFRA1.Freq_Step=FREQ_STEP_MULTIPLY;
//Assign array location to Pointers in the SFRA object
SFRA1.FreqVect=FreqVect;
SFRA1.GH_MagVect=OL_MagVect;
SFRA1.GH_PhaseVect=OL_PhaseVect;
SFRA1.H_MagVect=Plant_MagVect;
SFRA1.H_PhaseVect=Plant_PhaseVect;

SFRA_IQ_INIT(&SFRA1);

```

- To enable changing of the SFRA parameters from the GUI, the following code is added in the background task:

```

//SFRA Object Initialization
//Specify the injection amplitude
if(initializationFlag == 1)
{
    SFRA_F_INIT(&SFRA1);
    initializationFlag = 0;
    SFRA1.start = 1;
}

```

- Also, the SFRA background task is called in a background task A1:

```
SFRA_IQ_BACKGROUND(&SFRA1);
```

- Note that for the GUI communication, the following task is called every 1 msec or so in task A2.

```
SerialHostComms();
```


- Observe the PWM ISR code. Here, the SFRA_INJECT routine is called to add a sinusoidal injection on the DC operating point. Later in the code, the feedback is used to collect the response for the sinusoidal injection.

```

interrupt void PWM_ISR(void)
{
...

//Read ADC and computer Fbk Value
Vout1_Read= (int32)Vout1R<<12;


//Add SFRA injection into the duty cycle for the open loop converter
Duty_pu=SFRA_IQ_INJECT(Duty_pu_DC);

//Update PWM value
EPwm1Regs.CMPA.half.CMPA=_IQ24mpy((long)(BUCK_PWM_PERIOD),Duty_pu);

SFRA_IQ_COLLECT(&Duty_pu,&Vout1_Read);
...
}

```




- It is time to setup the hardware. Verify that the TMS320F28035 control card SW1 is pointing down (towards the off position). SW3-control card is pointing down and SW2-control card is pointing up.
- Insert the TMS320F28035 control car into the DIM100 connector on the DPS Workshop board.
- Make sure J1 on the main board and J4 on the M2 macro are populated. For location on the board, see [Figure 23](#).
- Make sure the power switch SW1-Main is in the OFF position. Make sure the SW2-Main switch is in the CH1 position. For these locations on the board, see [Figure 23](#).
- Connect a USB cable from the DPS board (JP1 of the M2 Macro) to a host computer.
- Power up the DPS board, by connecting the 9 V power supply at JP1 Vin.
- Switch the power switch SW1-M1 to the ON position. You will see LD1 in M1 macro on the board and LD1 on the control card light up.
- Go to the Project → Build All menu and watch the tools run in the build window to compile the project. The project will compile successfully.
- Go to the Run → Debug menu. The program is then loaded into the Flash of the F28035 device. The code will run to the start of *Main()*.
- Once loaded, enable the real-time mode by hovering your mouse on the buttons on the horizontal


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

A message box may appear. If so, select YES to enable debug events.

- Add watch variables to CCS expressions by clicking under View → Scripting Console. The console will appear in CCS and on the top left corner click on open command and browse to DPSWrkShpKit_SFRA_F28035.js file located inside the project folder.

NOTE: Make sure you select .js as the extension when browsing the folders.

- Click on the Continuous Refresh button  for the watch view to enable the variables to update continuously.
- Run the project by clicking the .
- The Duty_pu_DC value is set to zero and one will see Gui_Vout1 is almost zero. The code is running now and you should see update happening periodically in the watch window. If not, make sure to select the Continuous Fresh icon  in the expressions view.
- Switch the DC Bus power switch ON, and the LED LD1 next to the switch will light up. For the location on the board, see [Figure 23](#).

25. The software is configured for open loop, for example, there is no regulation of the voltage. Hence start entering value for Duty_pu_DC in the watch window by going in steps of `_IQ24(0.05)` from `_IQ24(0)` to `_IQ24(0.22)`. This makes the output voltage `Gui_Vout1` go close to 2 V, which is close to where this board is designed to operate. Set `Active_LD1_EN` to 1, which loads the power stage and the voltage may drop slightly. Adjust the duty to get the voltage back at 2 V if the correct voltage (~2V) does not appear on the `Gui_Vout1` variable when the `Duty_pu_DC` variable is set to `_IQ24(0.22)`. Check if `EPwm1Regs.TZFLG.all` is all zero. If it is not zero that means PWM is tripped, in this case reset the board, terminate the debug session by pressing the stop sign button , and start again from Step 13 and make sure the duty cycle is changed slowly.
26. Now that the DC point is established, open the SFRA gui.exe located at `SFRA\version\GUI`.
27. For instruction on how to connect and what each panel on the SFRA GUI box means, see [Section 3.8](#).
28. Select fixed point math.
29. Click Setup Connection and set the baud rate to be 57600 on the pop up window.
30. Uncheck boot on connect and select the appropriate COM port. For procedures to find out which COM port to select, see [Section 3.8](#).
31. Click OK to close the pop-up window and return to the main screen.
32. On Main Window, click Connect. Once connected the GUI will parse the current settings for the FRA sweep from the controller, these include the Start Frequency of the sweep, the length of the frequency sweep array (this is fixed in the code and hence cannot be changed through the GUI), injection amplitude and steps per decade. Leave these as default for now.
33. Press the Start Sweep button.
34. Wait for the status bar in the GUI to change to Sweep Complete.
35. The results of the SFRA sweep are now displayed on the window. The open loop graph result has no meaning as the plant is not in in closed loop operation. Therefore, select Plant in the drop down menu to the left. Once Plant is selected in the drop down menu, the GUI will look similar to [Figure 25](#).

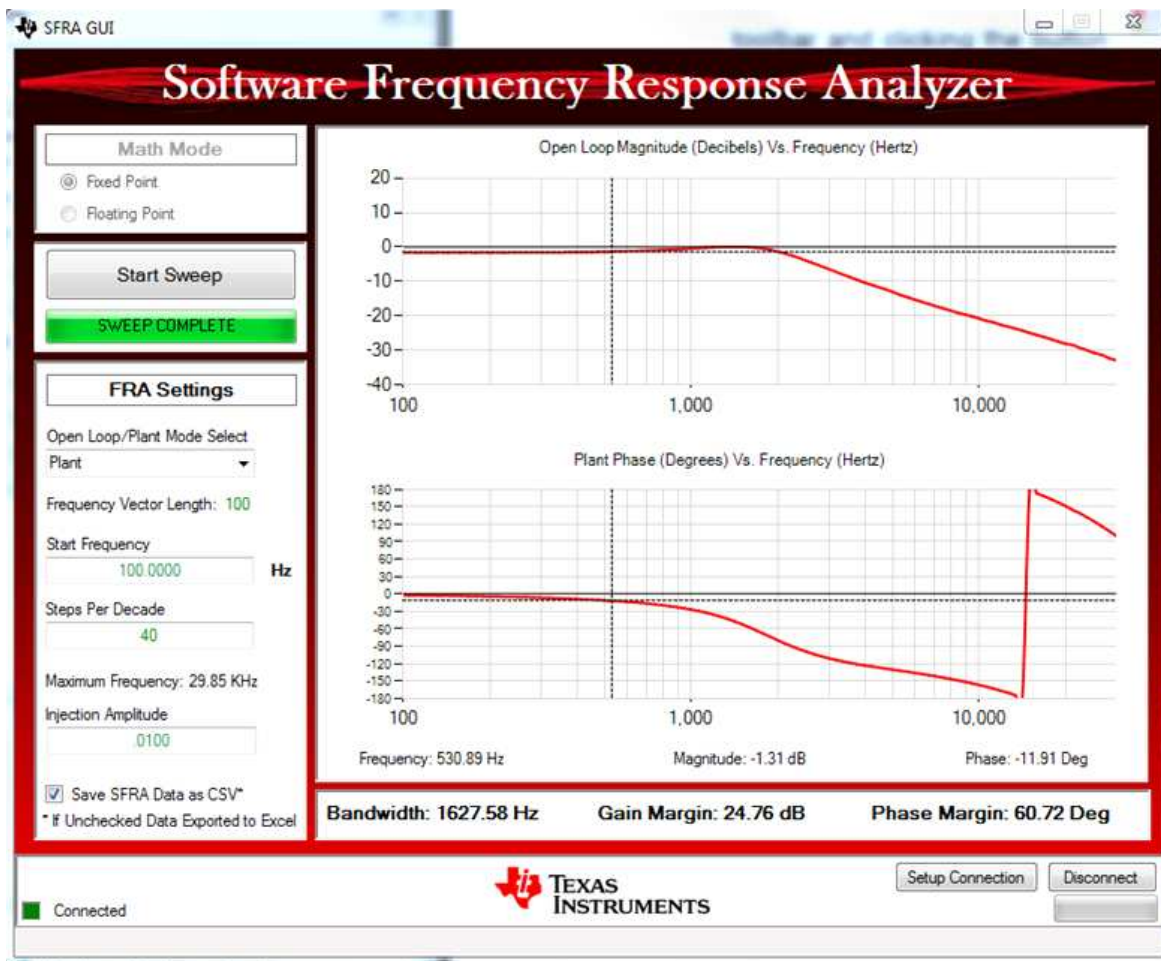





Figure 25. Plant Frequency Response Plot

36. Drag your mouse across either of the graphs to locate the values at specific frequencies used in the Frequency Response Analysis.
37. A SFRAData.csv file inside the GUI folder is updated with the latest run of the SFRA library. All runs of the SFRA library and time stamped are saved under that folder. If the checkbox "Save SFRA Data as CSV" is unchecked, the excel sheet will pop up after each sweep with the data of the frequency response. Each subsequent sweep is added as a new page on the excel sheet.

NOTE: the change to the checkbox will only affect the save of the next run of the SFRA, and hence the selection must be done before hitting "Start Sweep".

38. One can also try to change the Duty_pu_DC value and change the operating point and run the frequency sweep again. Additionally, the active load can be enabled to increase the load of the power stage and frequency response done under loaded conditions. To enable the active load just write 1 to the Active_LD1_EN. (Make sure you turn the active load off once finished evaluating at that operating point, as it can get fairly hot).
39. Once done with the evaluation, write zero to the Duty_pu_DC value, make the SW1 (on the main board) off.
40. Close the SFRA GUI.
41. Fully halting the MCU when in real-time mode is a two-step process. Now, halt the processor by using the Halt button  on the toolbar, or by using Target → Halt. Then, take the MCU out of real-time mode by clicking on . Finally, reset the MCU.

42. Close CCS debug session by clicking on the Terminate Debug Session button  or select Target → Terminate all.
43. The frequency response measured can then be used to design the compensation by importing the frequency data in MATLAB. The Scripts were described in [Section 3.7](#).

5.3 Designing Compensator Using Compensation Designer

The SFRA_GUI.exe puts the latest frequency sweep data inside “controlSUITE/libs/app_libs/SFRA/<version>/GUI/SFRADData.csv”. The compensation design GUI can be launched by clicking on the “CompDesigner.exe” located at “controlSUITE/libs/app_libs/SFRA/<version>/GUI”. The Compensation Designer defaults to using the latest run of the SFRADData.csv file. A run of SFRA was completed in [Section 5.2](#), the CompDesigner will use that SFRA measurement to design the compensation.

Figure 26 shows the Compensation Designer with two pole two zero compensation style selected and used to design a stable closed loop system, based on plant data from SFRA run in the previous section. The compensator coefficients can be copied from the compensation designer GUI into the C code. This will be done in the next section.

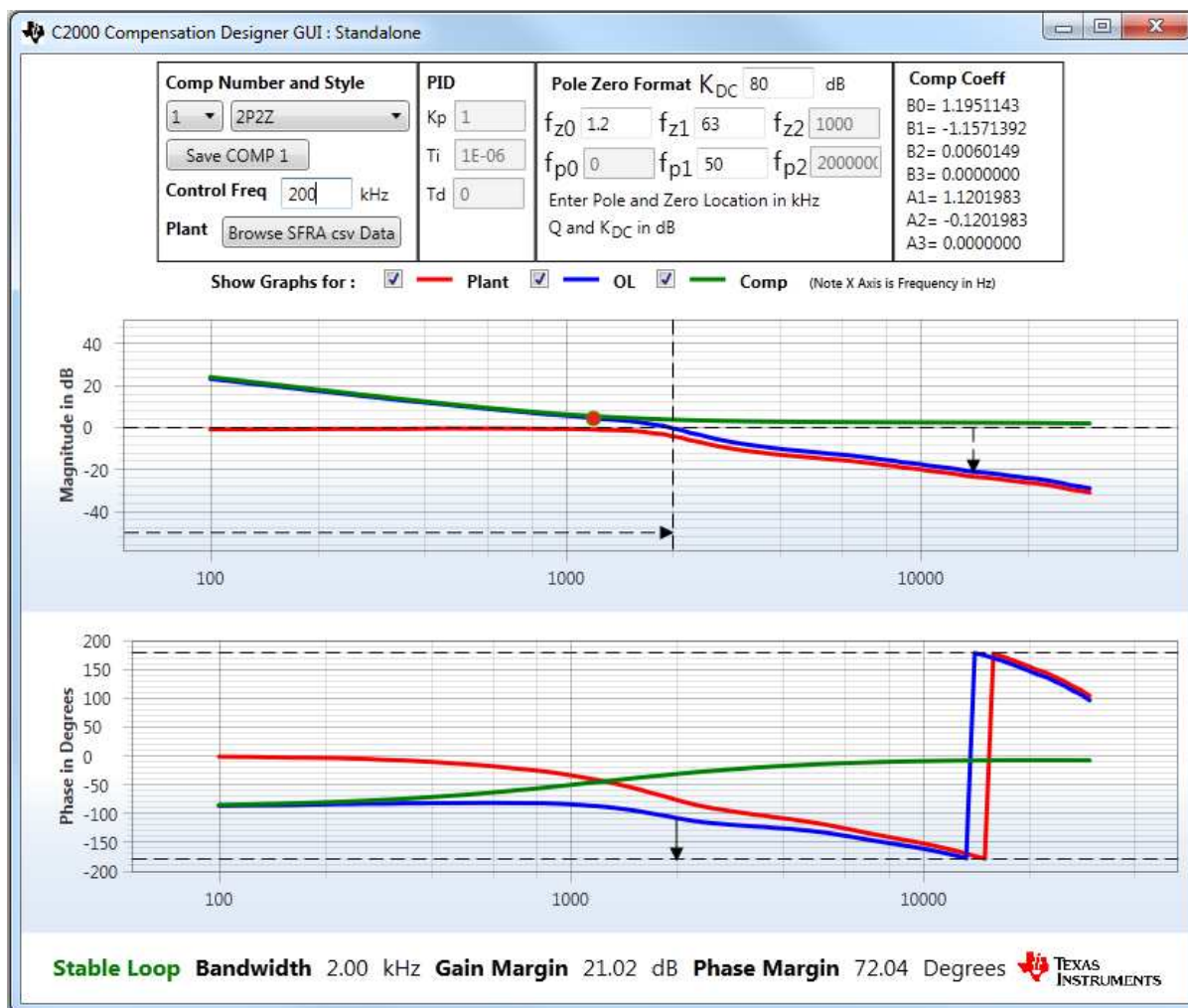


Figure 26. Comp Designer Being Used to for the DPS Workshop Board Based on Plant Data From SFRA Run

5.4 OL Measurement

The SFRA library is used to measure the open loop frequency response of the closed loop controlled power converter in this example. The compensation can be designed using the compensation designer based on plant information gathered in the previous section by the SFRA run. By default, the Compensation Designer GUI uses the SFRAData.csv file that is saved under the GUI folder. Figure 27 describes the software connections for a closed loop system using SFRA.

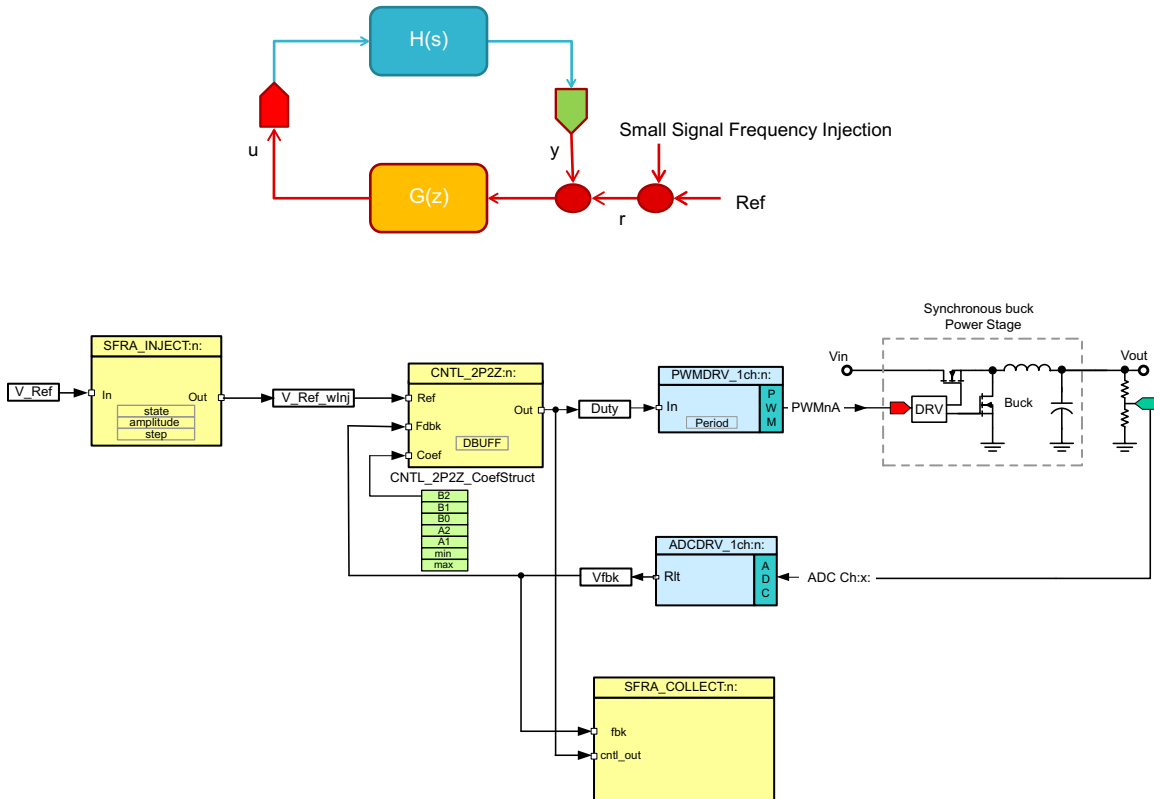


Figure 27. Closed Loop SFRA Software Diagram

1. If not already in workspace, import the project under

```
SFRA/vX/examples/DPSWrkShpKit_SFRA_F28035
```

in CCS6.0 or later. “DPSWrkShpKit” will be referred to as <ProjectName> in the document. Go to the <ProjectName>-Settings.h file and set the incremental build level to 2.

```
//=====
// Incremental Build options for System check-out
//=====
// BUILD 1      Open Loop PWMDRV_1ch using PWM 1 on C28x with FRA
// BUILD 2      Closed Loop PWMDRV_1ch using PWM1 on C28x with FRA
```

2. In the Main.c file locate the ISR and observe the connections for the SFRA:

```
interrupt void PWM_ISR(void)
{
...

//Read ADC and computer Fbk Value
cntl3p3z_vars1.Fdbk= (int32)Vout1R<<12;

//Add SFRA injection into the reference of the controller
cntl3p3z_vars1.Ref= SFRA_IQ_INJECT(Vout1SetSlewed);

// Call the controller
CNTL_3P3Z_IQ_ASM(&cntl3p3z_coef1,&cntl3p3z_vars1);

//Update PWM value
EPwm1Regs.CMPA.half.CMPA=_IQ24mpy((long)(BUCK_PWM_PERIOD),cntl3p3z_vars1.Out);

SFRA_IQ_COLLECT(&cntl3p3z_vars1.Out,&cntl3p3z_vars1.Fdbk);
```

3. Make sure the DC Bus power switch, SW1-Main, is in the off position. For the location on the board, see [Figure 23](#). If ON, turn this switch to the OFF position.

NOTE: The code is set to put Gui_Vset to 2.0 V automatically, you must switch the DC Bus power switch ON before this happens.

4. Build and load the project similar to that in [Section 5.2](#), enable real time and click Run.
5. Once the code is running, immediately put the DC Bus power switch, SW1-Main, in the ON position. For the location on the board, see [Figure 23](#). The voltage can be seen being regulated at 2 V.
6. To run SFRA, open the SFRA gui.exe, located at SFRA\version\GUI. for instruction on how to connect and what each panel on the SFRA GUI box means, see [Section 3.8](#).
7. Select fixed point math.
8. Click Setup Connection and set the baud rate to be 57600 on the pop up window.
9. Uncheck boot on connect and select the appropriate COM port. For the procedure to find out which COM port to select, see [Section 3.8](#).
10. Click OK to close the pop-up window and return to the main screen.
11. On the Main Window, click Connect. Once connected, the GUI will parse the current settings for the FRA sweep from the controller. These include the Start Frequency of the sweep, the length of the frequency sweep array (this is fixed in the code and cannot be changed through the GUI), injection amplitude and steps per decade. Leave these as default for now.
12. Press the Start Sweep button.
13. Wait for the status bar in the GUI to change to Sweep Complete.
14. The results of the SFRA sweep is displayed on the large panel in the SFRA GUI. The control performance parameter like bandwidth, gain margin and phase margin are also reported in a panel on the SFRA GUI. It is noted that the values reported match closely to what the system was designed for using the Plant Frequency Response Data from the open loop run.

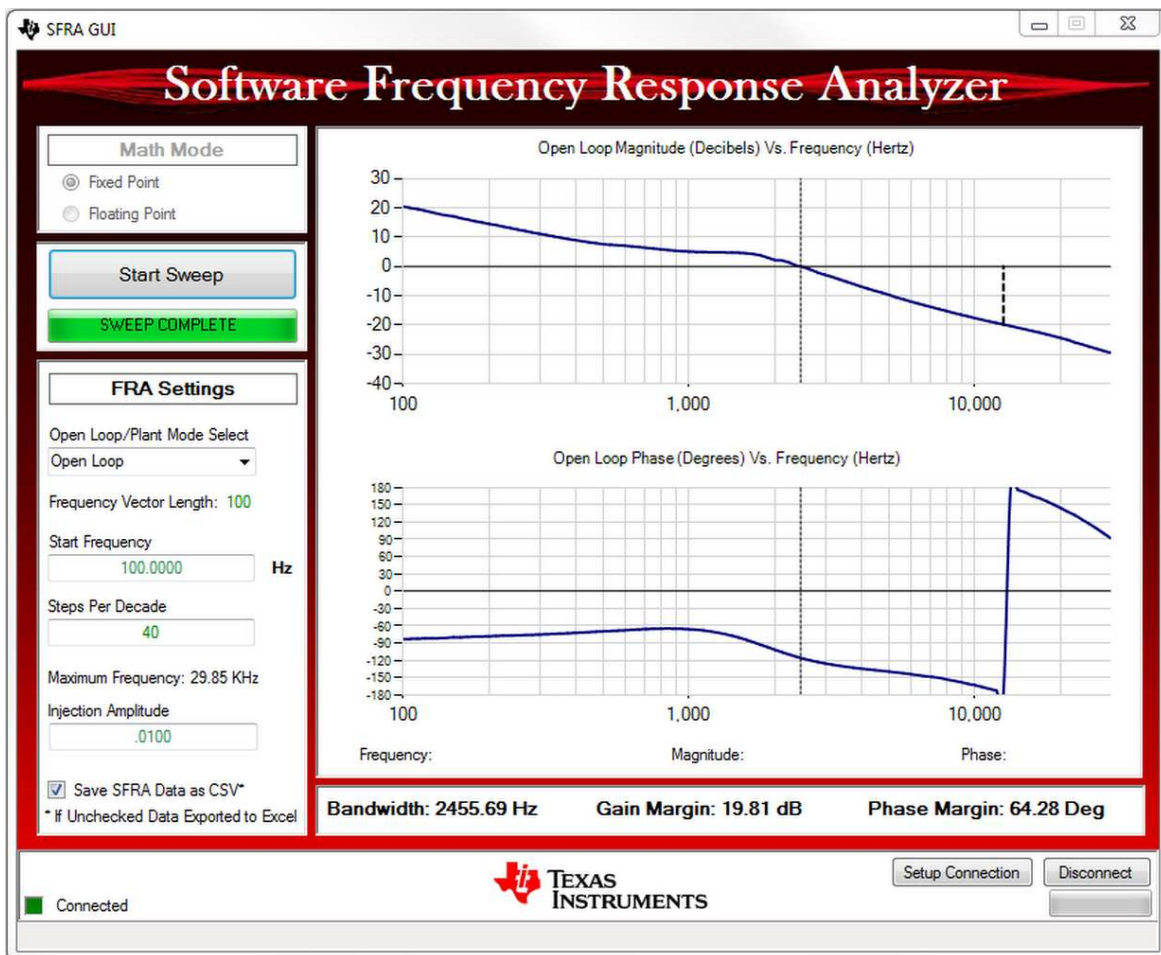


Figure 28. SFRA GUI With OL Frequency Response Plot

15. Drag your mouse across either of the graphs to locate the values at specific frequencies used in the Frequency Response Analysis.
16. A SFRADData.csv file inside the GUI folder is updated with the latest run of the SFRA library. All runs of the SFRA library are time stamped and saved under that folder. If the checkbox "Save SFRA Data as CSV" is unchecked, an excel sheet will pop up after each sweep with the data of the frequency response. Each subsequent sweep is added as a new page on the excel sheet.

NOTE: the change to the checkbox will only affect the save of the next run of the SFRA, and hence the selection must be done before hitting "Start Sweep".

17. Critical values like Bandwidth, gain margin and phase margin are brought out on the GUI. The values can be used to verify whether or not the designed compensation really met the goals.
18. Once finished, put SW1 in the OFF position, and the code can be halted and CCS debug session stopped.

5.5 Comparing SFRA Measured Frequency Response Versus Modeled

The SFRA library has been run successfully on a number of power topologies. The following section shows the comparison between modeled and measured frequency responses on the DPSWorkshop board, as shown in Figure 29 and Figure 30. This clearly shows that the modeling was close enough to what was measured and vice versa. Some margin of error is expected due to parameter estimation error and non idealities of the model.

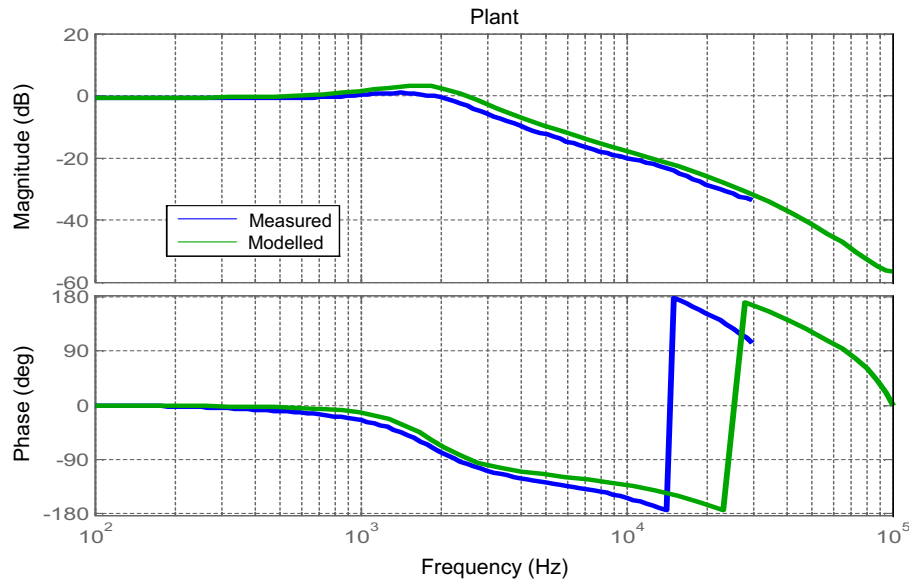


Figure 29. Plant Frequency Response Modeled Vs Measured on DPSWrkShpKit

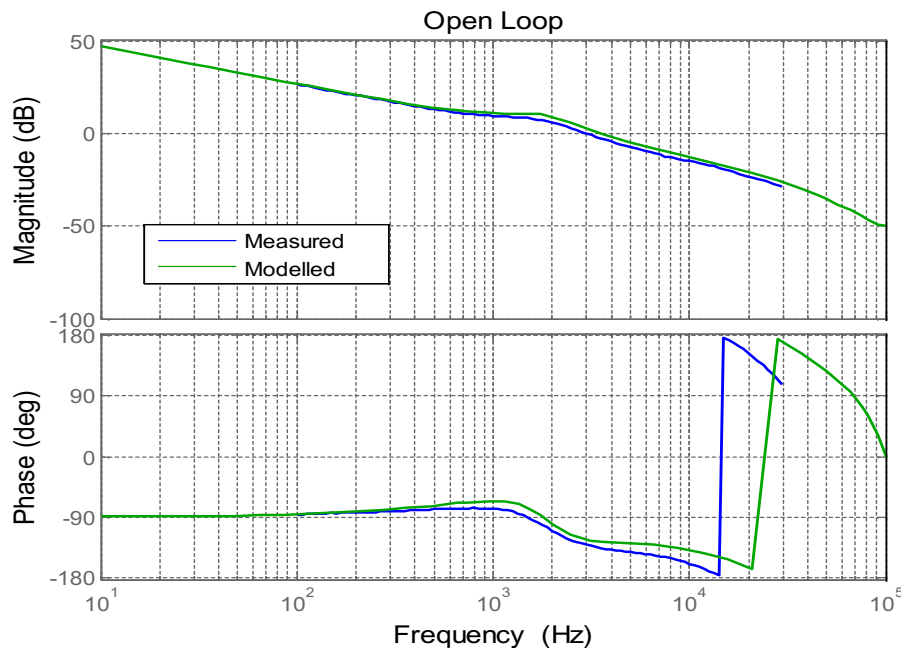


Figure 30. Open Loop Frequency Response Modeled Vs Measured on DPSWrkShpKit

6 FAQ

6.1 *GUI Will not Connect*

Make sure the SCI pins are configured to be used as SCI and the SCI module clock rate value provided to the SCI init routine is correct.

6.2 *GUI Will Connect but Start Button Does not Become Active*

If your native language is not English (USA) and the above behavior is observed, try selecting the English (USA). To do this, go to "Region and Language" using search in the Start Button → select under Formats → English (USA) → Apply.

6.3 *For Measurable Response, What is the Maximum Frequency SFRA?*

Maximum frequency measurable is determined by the SFRA_ISR_FREQ and is limited by the Nyquist criterion (half of the switching frequency or the frequency at which the SFRA routine is called).

6.4 *What is the Lowest Amplitude Signal Measurable by SFRA?*

SFRA uses the on-chip ADC whose noise floor for a 12-bit ADC is at approximately 60dB-70dB. Therefore, readings of the FRA below 60dB should be interpreted with caution as there may be significant noise elements.

6.5 *Why Does the SFRA Sweep Take so Long?*

The time taken by the SFRA depends on two factors:

- The ISR frequency in which the SFRA routine is called.
- The speed at which the background task is called, therefore, it is expected that SFRA will run much quicker when used to measure the response of a 200 kHz power converter compared to a one running at 10 kHz-20 kHz.

7 About the Author

MANISH BHARDWAJ is a Systems Application Engineer at Texas Instruments where he is responsible for developing system solutions for C2000 microcontrollers. Manish has been with TI since 2009 before which he received his Masters of Science in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta and Bachelor of Engineering from Netaji Subhash Institute of Technology, University of Delhi, India. He can be reached at mbhardwaj@ti.com.

Revision History

Changes from Original (October 2014) to A Revision	Page
• Updated information in Section 1	5
• Update was made to Table 1	8
• Updated information in Section 3.4.3	11
• New Section 3.6 was added.....	22
• Updated information was added to Section 3.7	23
• Figure 10 was updated in Section 3.8	24
• Added new Section 4	25
• Updated information made to Section 4.1.1	26
• Updated information made to Section 4.1.2	28
• Updated information in Section 4.2	29
• Updated information in Section 4.3	30
• Updated information in Section 5.2	38
• Figure 25 was updated in Section 5.2	42
• Added new Section 5.3	44
• Updated information in Section 5.4	45
• Updated Figure 28 in Section 5.4	46
• Updated information in Section 6	49

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

Products		Applications	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	TI E2E Community	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		