

C2000 Software Frequency Response Analyzer (SFRA) Library & Compensation Designer

v1.10

Feb-15

Module User's Guide

C28x Foundation Software

**Manish Bhardwaj
C2000 System Solution**

IMPORTANT NOTICE

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

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

Customers are responsible for their applications using TI components.

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

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

Reproduction of 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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with statements different from or beyond the parameters stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

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

Copyright ©2015, Texas Instruments Incorporated

Trademarks

TMS320, C2000, Piccolo are the trademarks of Texas Instruments Incorporated.
All other trademarks mentioned herein are property of their respective companies

Abbreviation

C28x: Refers to devices with the C28x CPU core.

SFRA: Software Frequency Response Analyzer.

IQmath: Fixed-point mathematical functions in C.

Q-math: Fixed point numeric format defining the binary resolution in bits.

Float: IEEE single precision floating point number

H: Plant Model as seen by the digital compensator (include analog to digital converter feedback scaling and plant model).

G: Compensator

CSV: Comma Separated Value

Contents

Chapter 1. Introduction	6
1.1. Introduction.....	6
Chapter 2. Installing the SFRA Library	8
2.1. SFRA Library Package Contents	8
2.2. How to Install the SFRA Library	8
Chapter 3. Module Summary	9
3.1. SFRA Library Function Summary	9
3.2. Principle of Operation	9
3.3. Per Unit Format	10
3.4. Fixed Point (IQ).....	10
Object Definition:	10
Module interface Definition:	10
Adding SFRA Lib to the Project.....	11
Adding Support for SFRA GUI.....	16
3.5. Floating Point	19
Object Definition:	19
Module interface Definition:	19
Adding SFRA Lib to the Project.....	20
Adding Support for GUI	24
3.6. Adding SFRA Library to Assembly Digital Power Library Project.....	27
3.7. Script for Importing Frequency Response & Designing Compensation.....	29
3.8. SFRA GUI Options and How to Run.....	29
Chapter 4. Compensation Designer.....	31
4.1. Launching Compensation Designer	31
4.1.1 Standalone from SFRA GUI folder	31
4.1.2 From Solution Adapter page page	32
4.2. Compensator Structure	34
4.3. Compensation Style & Number	34
1. PID (Proportional-Integral Derivative Controller)	35
2. Two Pole Two Zero (2P2Z).....	36
3. Three Pole Three Zero (3P3Z)	37
4. Two Pole Two Zero with One Complex Zero (2P2Z + 1CZ).....	38
5. Three Pole Three Zero with One Complex Pole (3P3Z + 1CP.....	39
6. Three Pole Three Zero with One Complex Zero (3P3Z + 1 CZ)	40
7. Three Pole Three Zero with One Complex Pole and One Complex Zero (3P3Z 1CP 1CZ) 41	
Chapter 5. Case Study	43
5.1. DPS Workshop Board Overview	43
5.2. Plant TF extraction	45
5.3. Designing Compensator using Compensation Designer	51

5.4	OL Measurement.....	52
5.4.	Comparing SFRA Measured Frequency Response Versus Modelled.....	56
Chapter 6. FAQ		59
6.1.	GUI Will not connect.....	59
6.2.	GUI will connect but Start Button Does not become Active	59
6.3.	What is the max frequency SFRA can measure response for?	59
6.4.	What is the lowest amplitude signal measurable by SFRA?	59
6.5.	Why does the SFRA sweep take so long?	59
Chapter 7. Revision History.....		60

Chapter 1. Introduction

1.1. Introduction

Texas Instruments Software Frequency Response Analyzer (SFRA) library is designed to enable frequency response analysis on digitally controlled power converters using software only. This enables performing frequency response analysis of the power converter with relative ease as no external connections or equipment is required. 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 ADC feedback,
- e the instantaneous error,
- d the sensor noise/disturbance,
- u the PWM duty cycle.

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

1. Ensure system is stable i.e. system tracks the reference asymptotically:

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

2. System provides disturbance rejection to guarantee robust operation

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

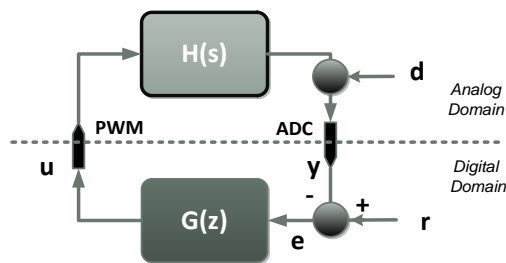


Figure 1 Digitally Controlled Power Converter

It is clear from (1) and (2) that by knowing the open loop transfer function (GH), one can determine if the system meets the objectives. 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.

SFRA library can enable measurement 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:

- Identify the plant model for the steady state operating point by running the SFRA library. The SFRA Gui will populate the data in a comma separate value (CSV) file. (Note: optionally the SFRA GUI can output data to an excel file).
- This SFRA CSV data can directly be used in the Compensation Designer to design a compensator that meets the system requirements. (Alternatively if Excel file was used MATLAB script is provided that will curve fit a transfer function and sisotool can be used to design the compensator).
- New Compensator values can be copied from the GUI into the CCS project.
- 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, 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.

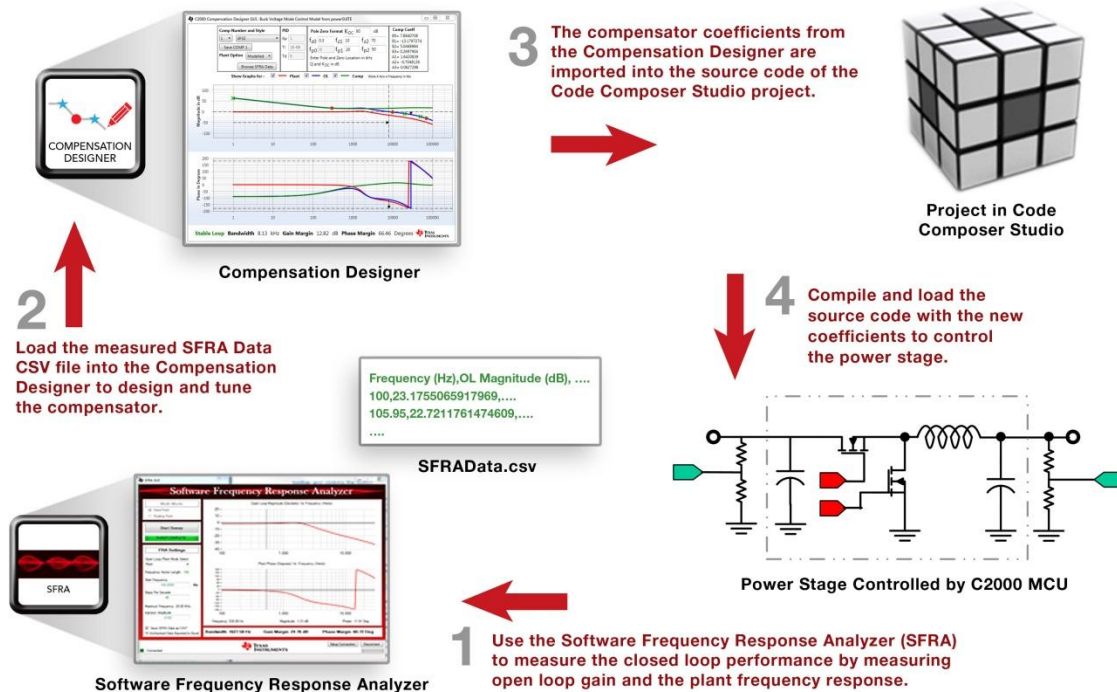


Figure 2 Control Design Using SFRA Library & Compensation Designer

Chapter 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
- Documentation SFRA Library User Guide

2.2. How to Install the SFRA Library

The SFRA Library is distributed through the controlSUITE installer. The user must 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 used:

<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 GUI for displaying the frequency response and the Compensation Designer. SFRADData.csv file is also stored here.
<base>\examples	Example using SFRA library
<base>\Scripts	Scripts to import SFRA data for compensation design in MATLAB

Chapter 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. The following table lists the modules existing in the SFRA library and a summary of cycle counts on each instruction set variant of the SFRA library.

Table 1 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 x16bit words	22x16 bit words (internal) + 25x16 bot words for SFRA object
Float	41	63	Typical ~ 50 Max 2350	665x16 bit words	20x16 bit words (internal) + 25x16 bot 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 i.e. only one instance of the SFRA library is supported in a particular project. The cycles reported, unless otherwise mentioned, are for maximum/worst case path in the operation.

3.2. Principle of Operation

Software Frequency Response Analyzer is based on the principle of small signal injection. A small signal is injected on the reference of the controller, 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.

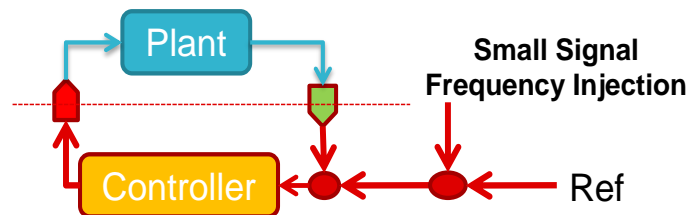


Figure 3 FRA Principle of Operation

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 20V and the instantaneous reading is 5V, 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)

The SFRA library defines the IQ Math based SFRA structure handle as below:

Object Definition:

```
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;
```

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 per unit format(pu).	IQ26(-1,1)
ISR_Freq	Input	Frequency at which SFRA is called.	Float32
Freq_Start	Input	Start Frequency.	Float32
Freq_Step	Input	$10^{1/(\text{no of steps per decade})}$.	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

Adding SFRA Lib 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

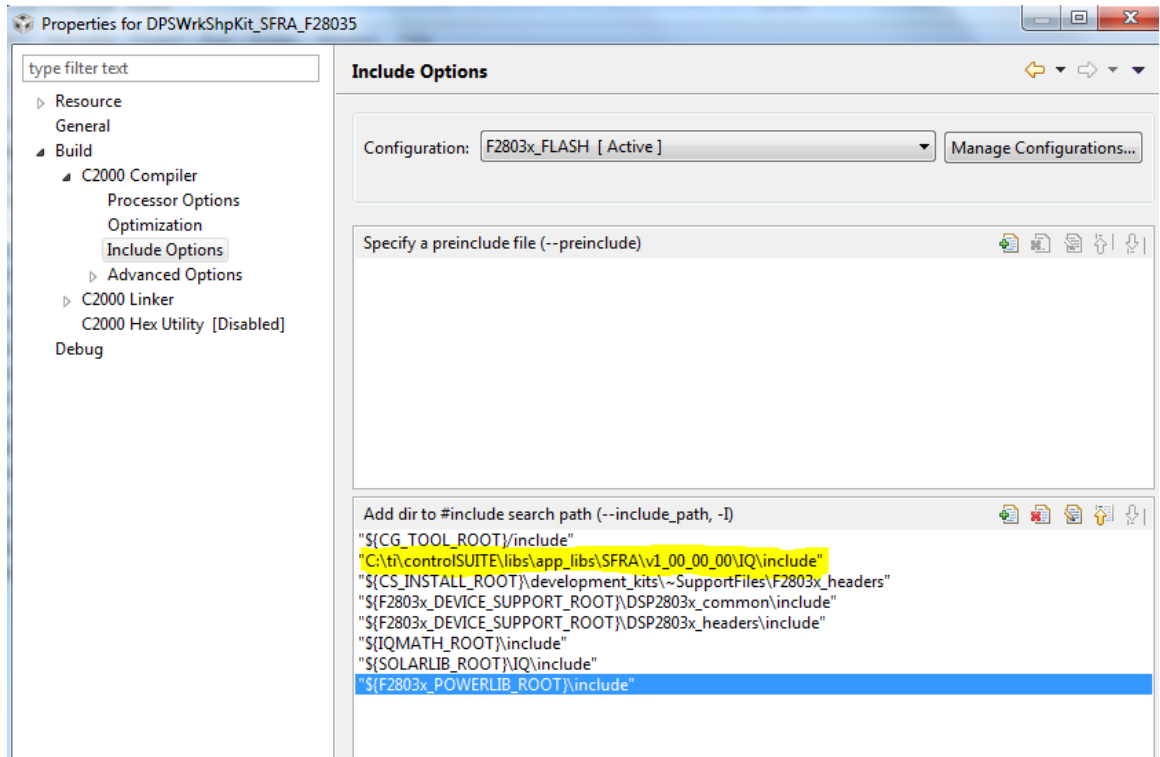


Figure 4 Compiler options for a project using SFRA IQ lib

(Please note exact location 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

Following is a snapshot that shows the changes to the linker options that are required to include the SFRA library:

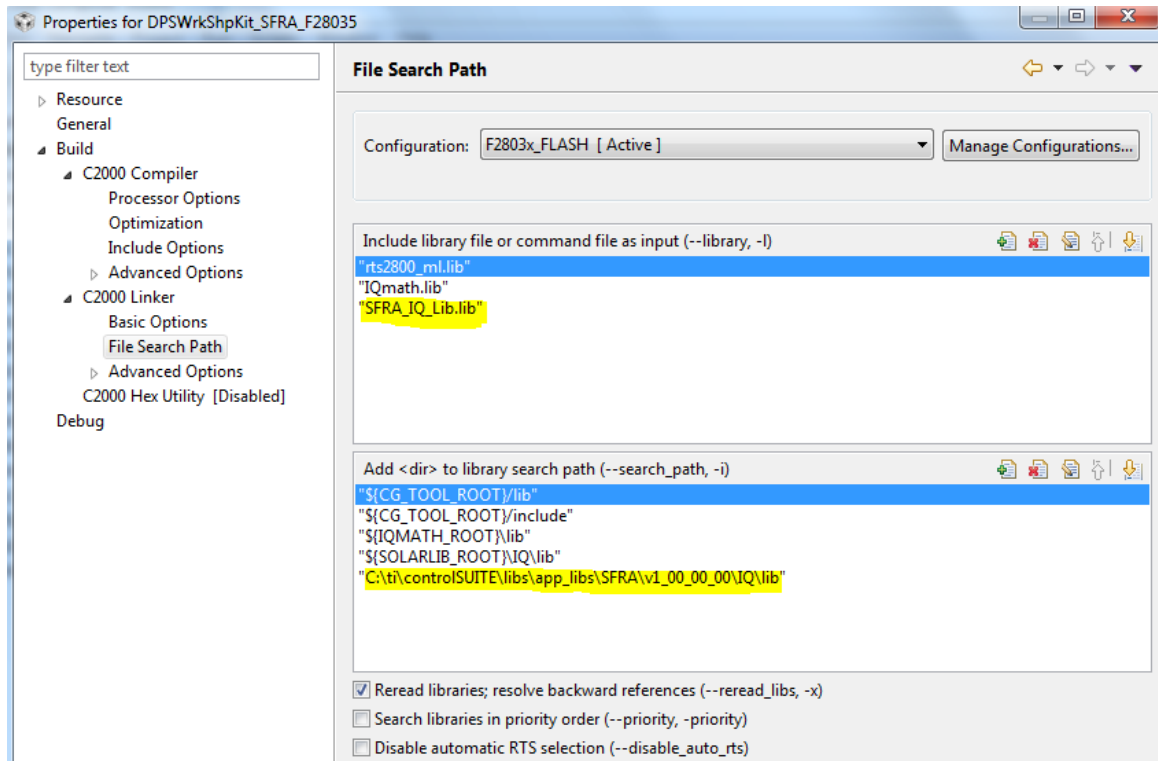


Figure 5 Adding SFRA library to the linker options in CCS project

(Please note exact location 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 at which 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 (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.ISR_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:

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);
```

```
...
}
```

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):** When running from FLASH for best performance the SFRA INJECT and COLLECT operations need to reside in RAM. Following is the changes/addition to CMD file for SFRA library inclusion and best performance

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

{
                --library=rtss2800_m1.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 IQmath Tables is defined correctly. These location are defined in the device command file which are 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
```

Adding Support for SFRA GUI

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

9. **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
10. **Add SciCommsGui_32bit.c to the project** : Add SciCommsGui_32bit.c to your project. In the project window, right-click on your project, select “Add files to project”, then browse to SciCommsGui_32bit.c found under FRA/version/GUI.
11. Add SCIA_Init() and SerialHostComms() to your function prototype list
12. Add SerialCommsTimer, CommsOKflg, and initializationflag as an int16 if they do not already exist.

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

13. 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.

14. 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);
```

15. Add SerialCommsTimer++; to a background task that is executed roughly every 1 msec. 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  
}
```

16. In a further slower task call the following routine:

```
SerialHostComms();
```

17. 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;  
}
```

18. The project will now connect to the SFRA GUI. See section 3.8 for how to run the SFRA GUI.

3.5. Floating Point

The SFRA library defines the floating point based SFRA structure handle as below:

Object Definition:

```
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;
```

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	$10^{(1/(\text{no of steps per decade}))}$.	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

Adding SFRA Lib 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

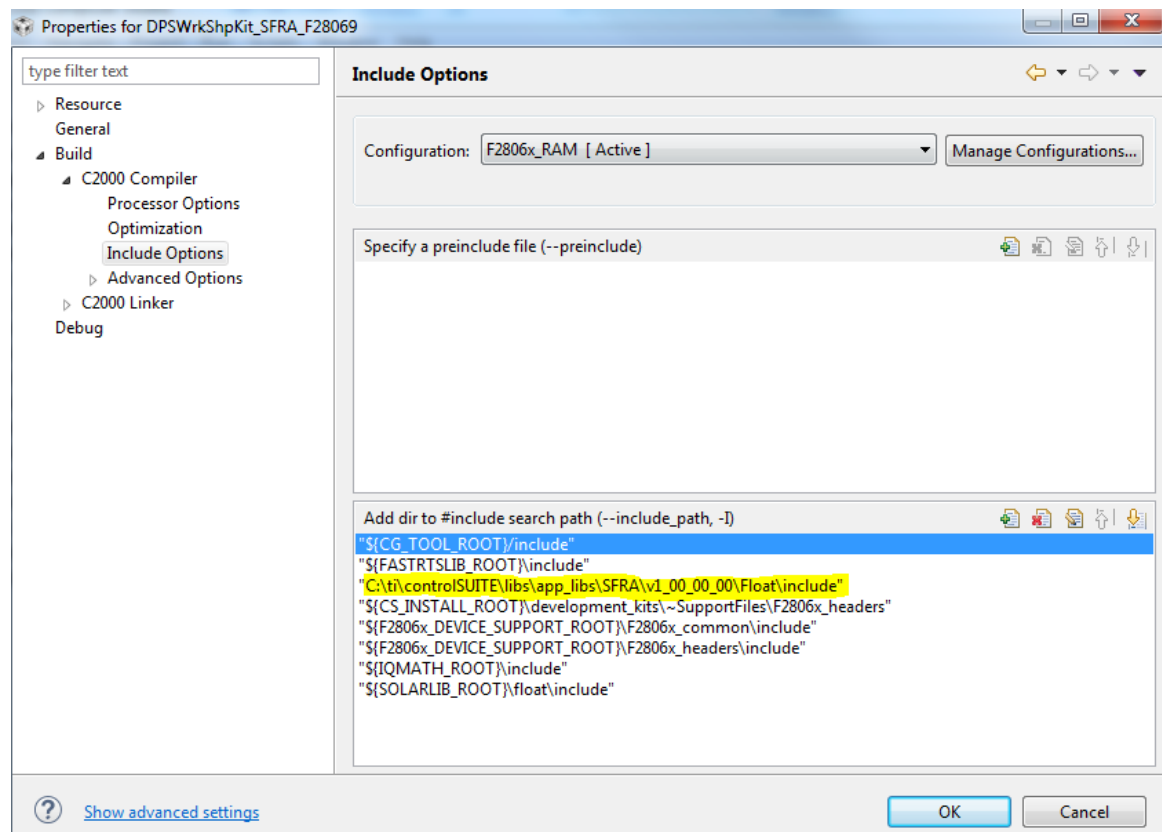


Figure 6 Compiler options for a project using SFRA Float lib

(Please note exact location 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

Following is a snapshot that 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 please refer to the FastRTS Library documentation found at

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

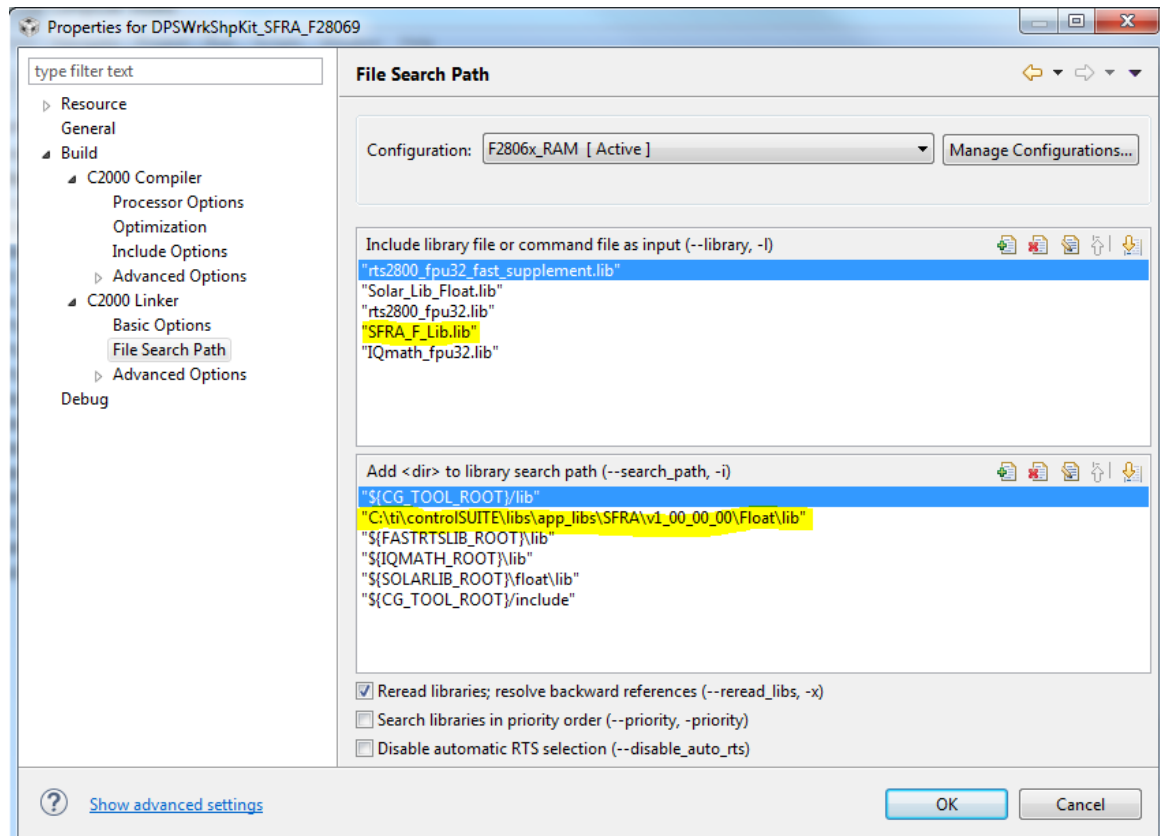


Figure 7 Adding linker options to the CCS project to include SFRA library in floating point project

(Please note exact location 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 at which 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.ISR_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_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:

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= (float32)Vout1R/(4096.0);

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

// Call the controller
CNTL_3P3Z_F_ASM(&cntl3p3z_coeff1,&cntl3p3z_vars1);

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

SFRA_F_COLLECT(&cntl3p3z_vars1.Out,&cntl3p3z_vars1.Fdbk);
...
}

```

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. Following is the changes/addition to CMD file for 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_F_Data      : > dataRAM1, PAGE = 1

```

8. Also make sure the ROM location for IQmath Tables is defined correctly. These location are defined in the device command file which are 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

```

Adding Support for GUI

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

9. **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
10. **Add SciCommsGui_32bit.c to the project** : Add SciCommsGui_32bit.c to your project. In the project window, right-click on your project, select “Add files to project”, then browse to SciCommsGui_32bit.c found under SFRA/version/GUI.
11. Add SCIA_Init() and SerialHostComms() to your function prototype list
12. Add SerialCommsTimer, CommsOKflg, and initializationflag as an int16 if they do not already exist.

```

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

```

13. 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 FRA GUI.

14. 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] = &(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);

```

15. Add SerialCommsTimer++; to a 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
}

```

16. Add the following code in a further slower task, i.e. executed every 5 msec:

```
SerialHostComms();
```

17. 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;
}

```

18. The project will now connect to the SFRA GUI. See section 3.8 for how to run the SFRA GUI.

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 which 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.

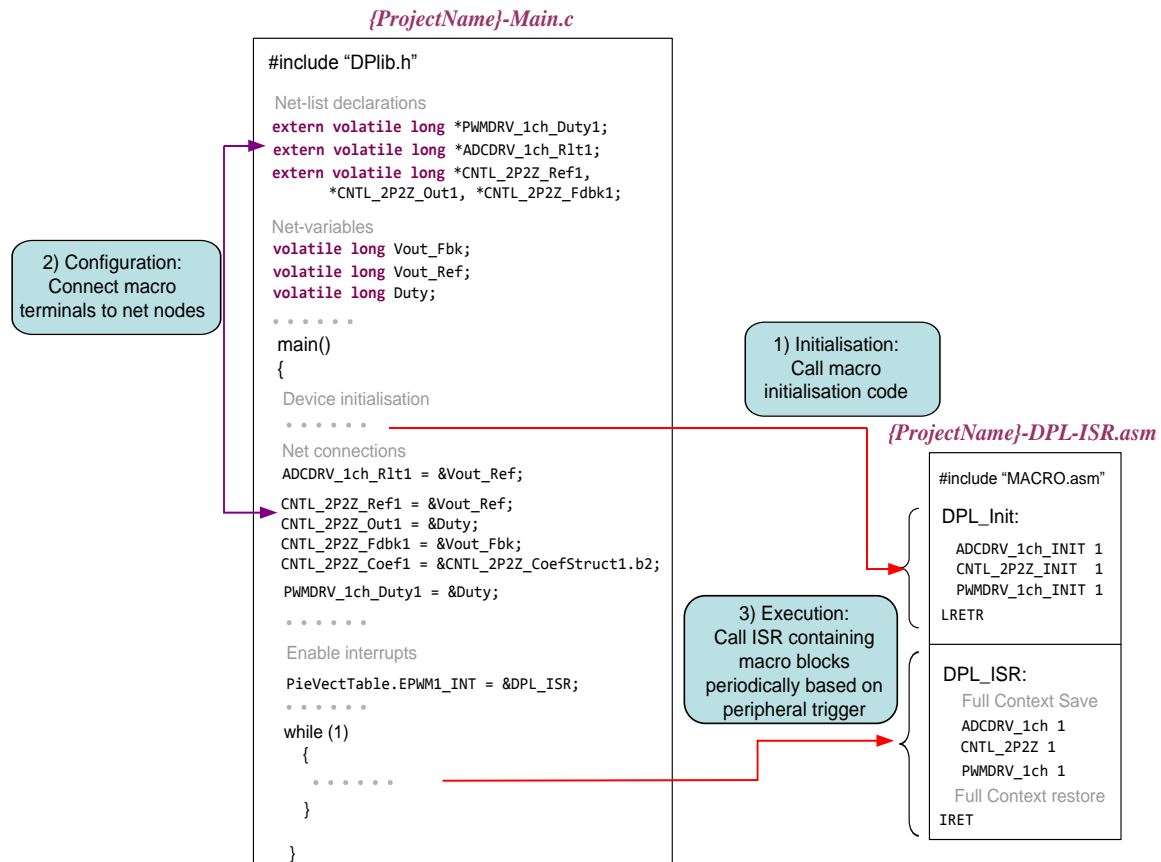


Figure 8 Project Using Assembly Digital Power Library

As the SFRA library is based in C, a C based ISR is needed to call the SFRA routine. Figure 9 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)

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

{ProjectName}-Main.c

1. Add the declaration of the DPL_Func in the main file

2. Add declaration for the new C based interrupt called DPL_ISR_wFRA()
3. Declare a new variable Vout_ref_wlnj
4. Add all the SFRA support items as described in section 3.4 except for the SFRA_Inject and SFRA_Collect calls.
5. Change only the connection to the 2p2z macro with Vout_ref_wlnj
6. Connect the PWM interrupt to DPL_ISR_wFRA()
7. Inside DPL_ISR_wFRA() add the code as shown in the diagram below to call the SFRA function and DPL_Func.

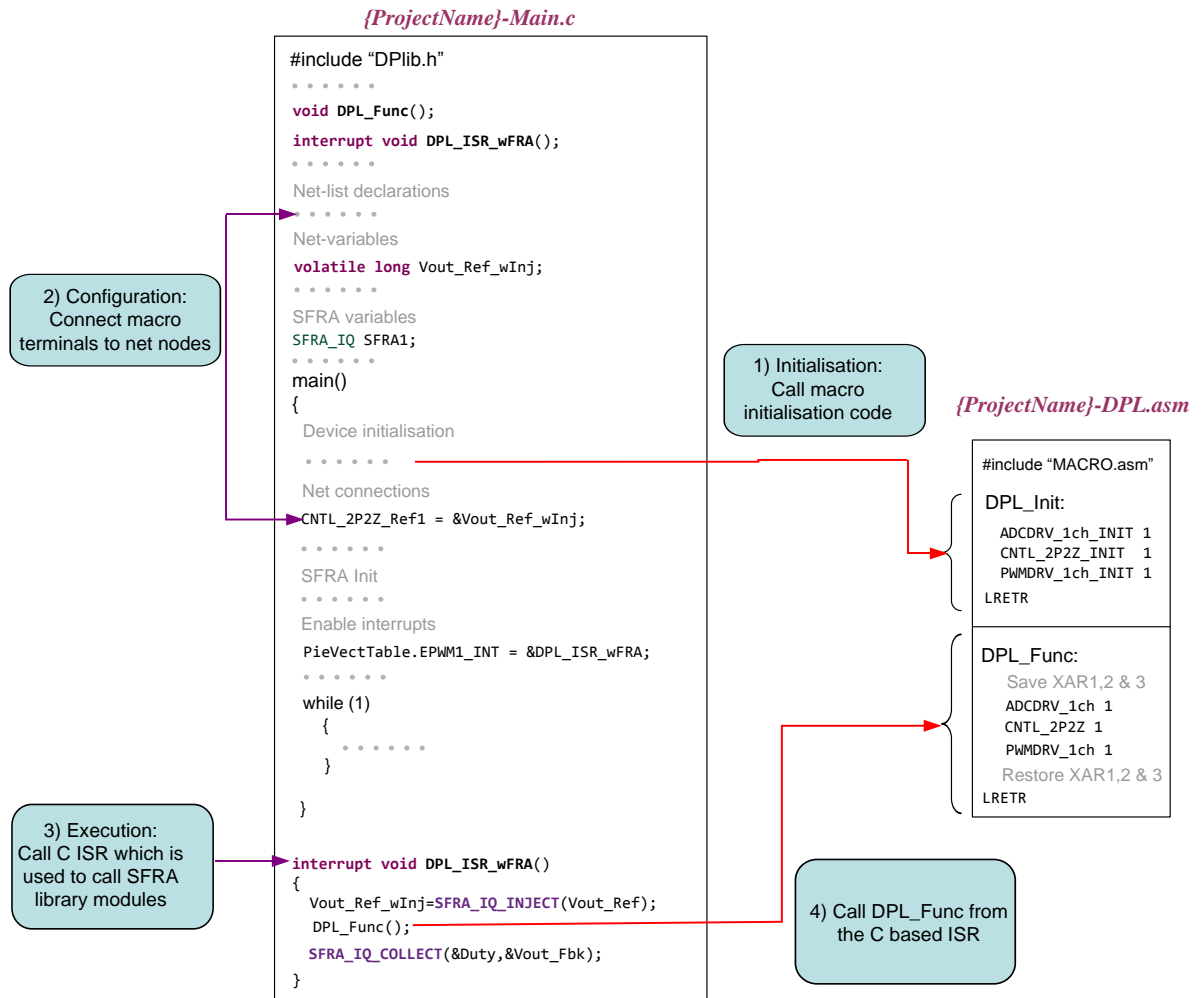


Figure 9 SFRA added to assembly Digital Power Library Framework

3.7. Script for Importing Frequency Response & Designing Compensation

SFRA GUI, on 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) the user may save it as an excel file. The excel file 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\vx\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. The following diagram, Figure 10, illustrates the different options that are presented to the user.

1. **MATH Mode Selection Radio Button:** This box must be selected appropriately according the version of SFRA library being used on the MCU. For example for F28035 because it is a fixed point device we select fixed point (the project will use SFRA_IQ_Lib.lib) and for F28069 floating point check box should be selected.
2. **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 will request for 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.
 - a. If the comport that the target is connected to is known, please select it.
 - b. If not, to find a valid comport, go to:
 - i. Control Panel->System-> Device Manager->Ports(COM & LPT
 - ii. If using a serial port directly connected to a PC, look for a comport which shows up as "Communications Port" and select this comport in the Setup Connection window. If using a USB to Serial adapter look for the comport which shows "USB Serial Port", then select this comport 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.

3. **Connect/ Disconnect Button:** Once the Math mode is selected appropriately and COM port for the communication is selected hit this button to Connect to the MCU. This is the same button that is used for disconnection from the MCU.
4. **Connection Status Indicator:** This indicates if the connection was successfully established or if there were any problems.
5. **Freq Vector Length Label:** Once connected this label shows the length of the FRA array that is programmed on the MCU.
6. **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 hitting enter. (Note the value on the controller side is only updated once the user starts a SFRA sweep).
7. **Steps Per Decade Text Box:** Specifies the number of frequency points that FRA is performed per decade.

8. **Maximum SFRA Frequency Label:** This is the max frequency till which the SFRA will be performed. This is a function of the start frequency, the steps per decade and the frequency length.
9. **Checkbox to save SFRA Data as CSV:** If checked the SFRA GUI will populated the SFRA data in a CSV file. If SFRA is run from the controlSUITE folder the file is saved at controlSUITE/libs/app_libs/SFRA/<version>/GUI/SFRADData.csv. When invoked from the powerSUITE solution GUI the SFRA run data is saved under SFRADData folder under the project directory of the powerSUITE project. The latest SFRA run is saved as SFRADData.csv, all the runs of the SFRA are time stamped and saved as SFRADData<date><time>.csv. This enabled the user to go back and use a previous run of the SFRA data in the compensation designer GUI.
10. **Start Sweep Button:** Click this button to begin a frequency sweep on the controller.
11. **SFRA Status Bar:** Indicated the progress of SFRA sweep.
12. **Drop Down Menu to Select Open Loop or Plant Frequency response** to be displayed on the panel to the right.
13. **Closed Loop Performance Parameters Panel:** If SFRA is performed on the closed loop this panel will display the bandwidth, gain margin and the phase margin of the closed loop system.

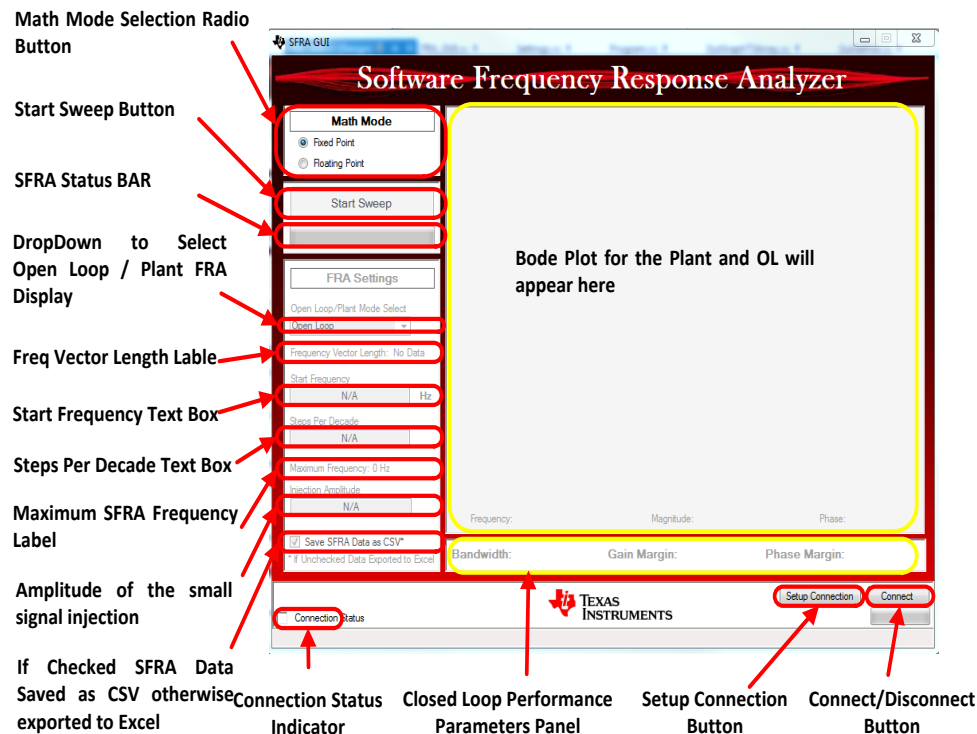


Figure 10 SFRA GUI Options

Chapter 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 etc.. 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 let's 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. **Error! Reference source not found.** 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.

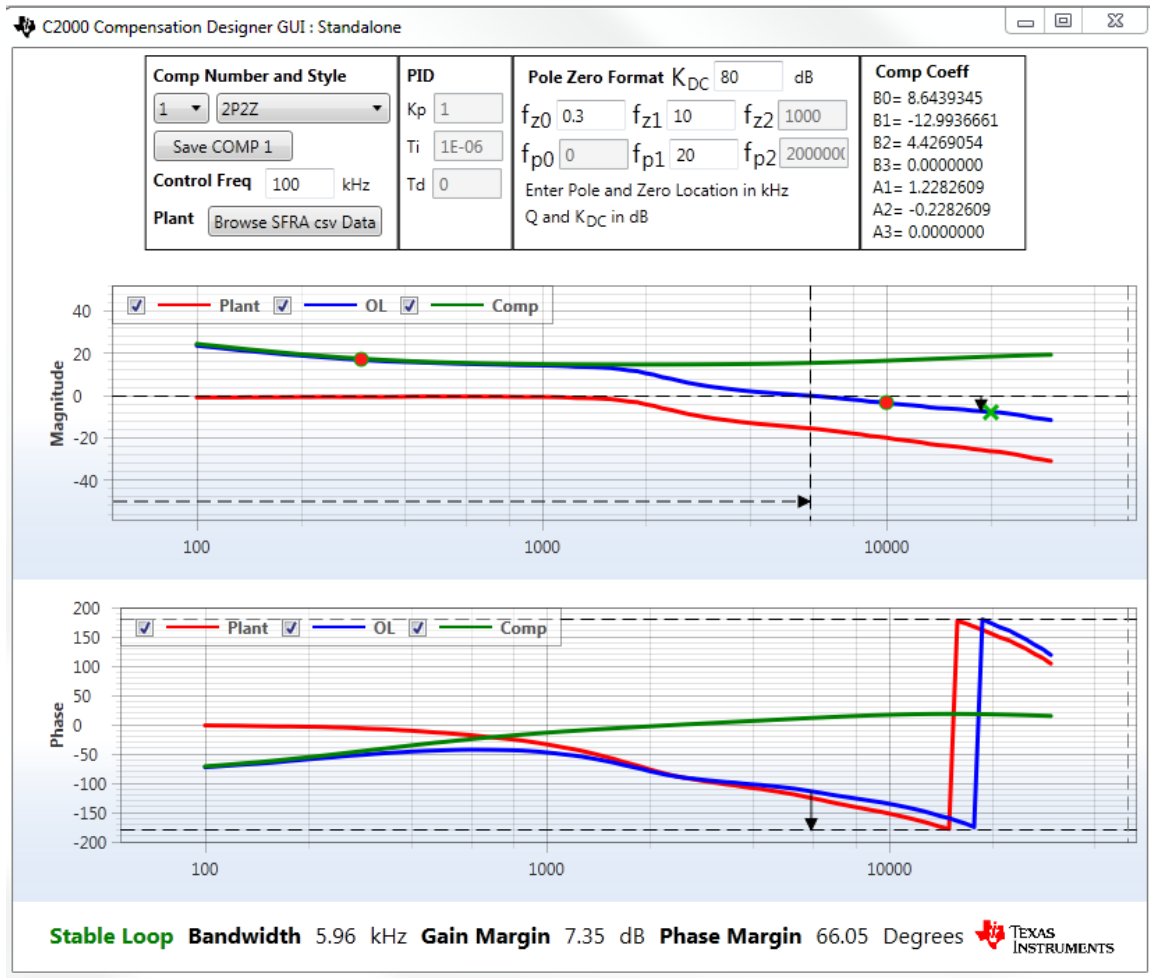


Figure 11 Standalone Compensation Designer when launched from the controlSUITE/libs/app_libs/SFRA/<version>/GUI folder

4.1.2 From Solution Adapter page

C2000 kits may support a solution adapter page, which let's 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 modelled 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.

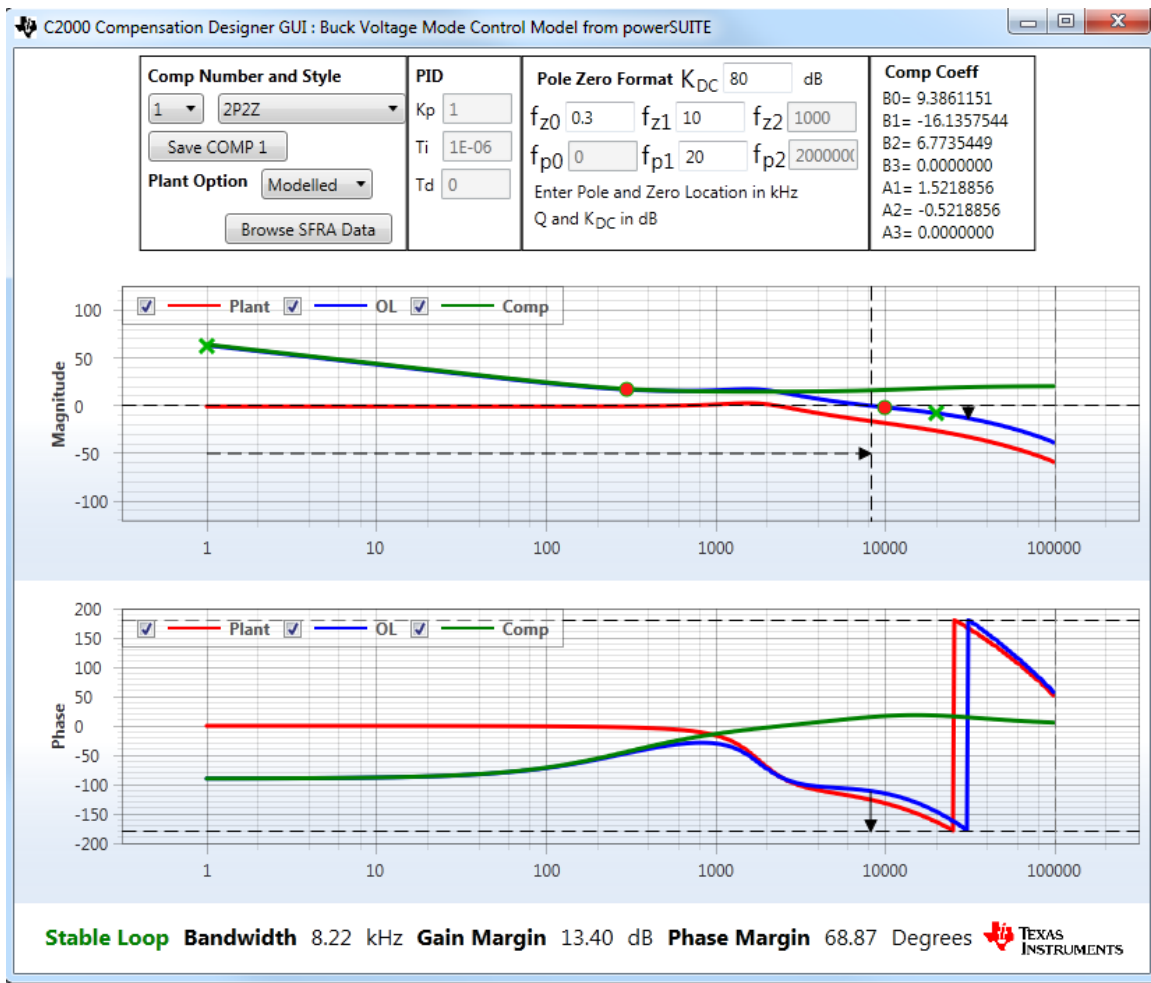


Figure 12 Compensation Designer when launched from solution adapter page
(Both Modelled and SFRA data based Plant Information can be used for compensation design)

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.

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 - a_1 z^{-1} - a_2 z^{-2} - a_3 z^{-3}}$$

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

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 hence the user must ensure the coefficients that are generated are reasonable for the micro-controller they will finally be run on.

When the compensation designer is invoked from the solution adapter page of a solution, because this page 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.

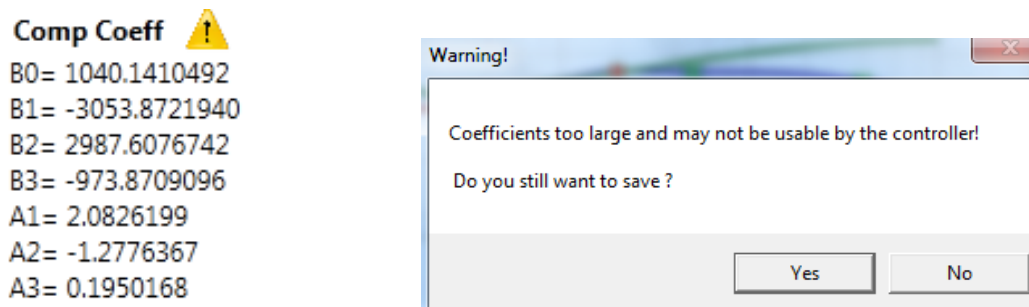


Figure 14 Warning Sign and Dialog when Compensation Coeff Range is exceeded

4.3. Compensation Style & Number

Different styles of compensation are supported by the Compensation Designer and are listed in Table 2.

Table 2 Different Compensation Style Supported by Compensation Designer

Compensation 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 user hits the button “Save COMP<number>”. This prevents any inadvertent change to the compensation values.

Comp Number and Style

1 ▼

2P2Z ▼

Save COMP 1

Figure 15 Compensation Number and Style Selection

Depending on which compensation style is selected different panels on the GUI will become active. 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 $T=1/F_s$.

1. PID (Proportional-Integral Derivative Controller)

When this compensation style is selected the PID panel becomes active

PID

K_p

T_i

T_d

Figure 16 PID Panel

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

$$G(s) = K_p + \frac{K_I}{s} + K_D s$$

Where, $K_I = \frac{K_p}{T_i}$ and $K_D = K_p * T_d$

To discretize this compensator, using Tustin/Trapezoidal Transformation i.e. $s = 2F_s \frac{(z-1)}{(z+1)}$ 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:

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

Where:

$$b_0 = K'_p + K'_I + K'_D$$

$$b_1 = -K'_p + K'_I - 2K'_D$$

$$b_2 = K'_D$$

and

$$K'_p = K_p$$

$$K'_I = \frac{T}{2} K_I$$

$$K'_D = \frac{1}{T} K_D$$

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.

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

Where: $z_0 = 2\pi * f_{z0}$, $z_1 = 2\pi * f_{z1}$, $p_1 = 2\pi * f_{p1}$

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.

Pole Zero Format K_{DC} dB

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

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

Enter Pole and Zero Location in kHz

Q and K_{DC} in dB

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 derived as follows:

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

3. Three Pole Three Zero (3P3Z)

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

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

Where: $z_0 = 2\pi * f_{z0}$, $z_1 = 2\pi * f_{z1}$, $z_2 = 2\pi * f_{z2}$, $p_1 = 2\pi * f_{p1}$, $p_2 = 2\pi * f_{p2}$

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.

Pole Zero Format K_{DC} dB

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

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

Enter Pole and Zero Location in kHz

Q and K_{DC} in dB

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 derived as follows:

$$G(z) = \left(K_{DC} \frac{p_1 p_2}{z_0 z_1 z_2} \right) \left(\frac{1}{2F_s^* (p_1 + 2F_s)^* (p_2 + 2F_s)} \right) \left(\begin{aligned} &((z_2 + 2F_s)^* (z_1 + 2F_s)^* (z_0 + 2F_s)) + \\ &((z_0 + 2F_s)^* (z_1 + 2F_s)^* (z_2 - 2F_s) + (z_1 + 2F_s)^* (z_2 + 2F_s)^* (z_0 - 2F_s) + (z_0 + 2F_s)^* (z_2 + 2F_s)^* (z_1 - 2F_s)) z^{-1} + \\ &((z_2 + 2F_s)^* (z_1 - 2F_s)^* (z_0 - 2F_s) + (z_1 + 2F_s)^* (z_2 - 2F_s)^* (z_0 - 2F_s) + (z_0 + 2F_s)^* (z_2 - 2F_s)^* (z_1 - 2F_s)) z^{-2} + \\ &((z_2 - 2F_s)^* (z_1 - 2F_s)^* (z_0 - 2F_s)) z^{-3} \end{aligned} \right) \left(\begin{aligned} &1 + \left(\frac{(p_2 - 2F_s)^* (p_1 + 2F_s) - 4F_s^* (p_2 + 2F_s)}{(p_1 + 2F_s)^* (p_2 + 2F_s)} \right) z^{-1} + \left(\frac{-(p_1 - 2F_s)^* (p_2 + 2F_s) - 4F_s^* (p_2 - 2F_s)}{(p_1 + 2F_s)^* (p_2 + 2F_s)} \right) z^{-2} + \\ &\left(\frac{-(p_1 - 2F_s)^* (p_2 - 2F_s)}{(p_1 + 2F_s)^* (p_2 + 2F_s)} \right) z^{-3} \end{aligned} \right)$$

4. Two Pole Two Zero with One Complex Zero (2P2Z + 1CZ)

Two Pole Two Zero with one complex zero compensator structure is shown below:

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

Where: $\omega_{rz} = 2\pi * f_{rz}$, $p_1 = 2\pi * f_{p1}$, 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 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 follows:

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

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

Three Pole Three Zero with one complex pole compensator structure is shown below:

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

Where: $z_0 = 2\pi * f_{z0}$, $z_1 = 2\pi * f_{z1}$, $z_2 = 2\pi * f_{z2}$, $\omega_{rp} = 2\pi * f_{rp}$, Q_p is the quality factor of the resonant pole.

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_p of the complex zero (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 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 follows:

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

6. Three Pole Three Zero with One Complex Zero (3P3Z + 1 CZ)

Three Pole Three Zero with one complex zero compensator structure is shown below:

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

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 follows:

$$G(z) = \left(\frac{K_{DC} p_1 * p_2}{2F_s * Q_z * \omega_r^2 * z_2} \right) \frac{1}{(2F_s + p_1) * (2F_s + p_2)} * \left(\begin{aligned} & (4F_s^2 Q_z + 2F_s \omega_{rz} + Q_z \omega_{rz}^2)(2F_s + z_2) \\ & + \left((4F_s^2 Q_z + 2F_s \omega_{rz} + Q_z \omega_{rz}^2)(-2F_s + z_2) + (-8F_s^2 Q_z + 2Q_z \omega_{rz}^2)(2F_s + z_2) \right) z^{-1} \\ & + \left((4F_s^2 Q_z - 2F_s \omega_{rz} + Q_z \omega_{rz}^2)(2F_s + z_2) + (-8F_s^2 Q_z + 2Q_z \omega_{rz}^2)(-2F_s + z_2) \right) z^{-2} \\ & + (4F_s^2 Q_z - 2F_s \omega_r + Q_z \omega_r^2)(-2F_s + z_2) z^{-3} \\ & 1 + \left(\frac{(p_2 - 2F_s) * (p_1 + 2F_s) - 4F_s * (p_2 + 2F_s)}{(p_1 + 2F_s) * (p_2 + 2F_s)} \right) z^{-1} + \left(\frac{-(p_1 - 2F_s) * (p_2 + 2F_s) - 4F_s * (p_2 - 2F_s)}{(p_1 + 2F_s) * (p_2 + 2F_s)} \right) z^{-2} + \\ & \left(\frac{-(p_1 - 2F_s) * (p_2 - 2F_s)}{(p_1 + 2F_s) * (p_2 + 2F_s)} \right) z^{-3} \end{aligned} \right)$$

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 below:

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

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), the Quality

factor Q_z of the complex zero (ω_{rz}) and the Quality factor Q_p of the complex pole (ω_{rp}). 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_{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 follows:

$$G(z) = \left(\frac{K_{DC}}{z_2} \right) \left(\frac{p_1^2 * Q_p}{2F_s * Q_z \omega_{rz}^2} \right) \frac{1}{(4F_s^2 Q_p + 2F_s \omega_{rp} + Q_p \omega_{rp}^2)} *$$

$$\left(\frac{(4F_s^2 Q_z + 2F_s \omega_{rz} + Q_z \omega_{rz}^2)(2F_s + z_2)z^3}{z^3} + \left(\frac{(-12F_s^2 Q_p - 2F_s \omega_{rp} + Q_p \omega_{rp}^2)}{4F_s^2 Q_p + 2F_s \omega_{rp} + Q_p \omega_{rp}^2} \right) z^2 + \left(\frac{12F_s^2 Q_p - 2F_s \omega_{rp} - Q_p \omega_{rp}^2}{4F_s^2 Q_p + 2F_s \omega_{rp} + Q_p \omega_{rp}^2} \right) z + \left(\frac{-4F_s^2 Q_p + 2F_s \omega_{rp} - Q_p \omega_{rp}^2}{4F_s^2 Q_p + 2F_s \omega_{rp} + Q_p \omega_{rp}^2} \right) \right)$$

Chapter 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 DPS Workshop Wiki Page:

(http://processors.wiki.ti.com/index.php/C2000_DPSWorkshop)

The following section provides a brief overview to facilitate running of the SFRA library project on DPS Workshop 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:

```
libs/app_libs/SFRA/vX/examples/DPSWrkShpKit_F28035x_SFRA (Fixed point IQ Math)
/DPSWrkShpKit_F28069_SFRA (Float Math)
```

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 the user is familiar with running the DPS workshop board. If needed, the workshop detailed on the wiki link mentioned earlier 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 9V. Figure 23 shows the location of the switches and power stages on the EVM board.

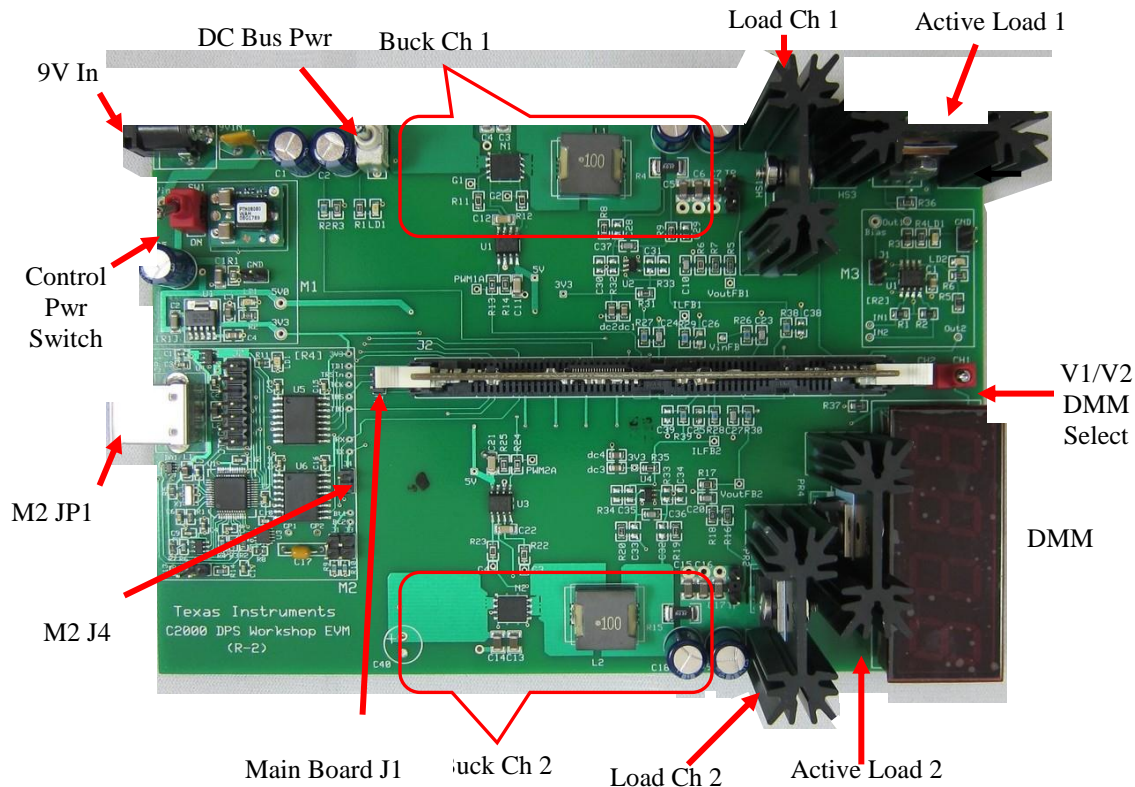


Figure 23 DPS Workshop Board Overview

Table 3 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 ohm resistive loads permanently connected across the two outputs
Active Load 1, 2	Software controlled switched load (2 ohms 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 4.

Table 4 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 i.e (y/u) which, in case of a voltage controlled power supply, will be Vout/Duty. Figure below illustrates the software diagram for the SFRA inclusion.

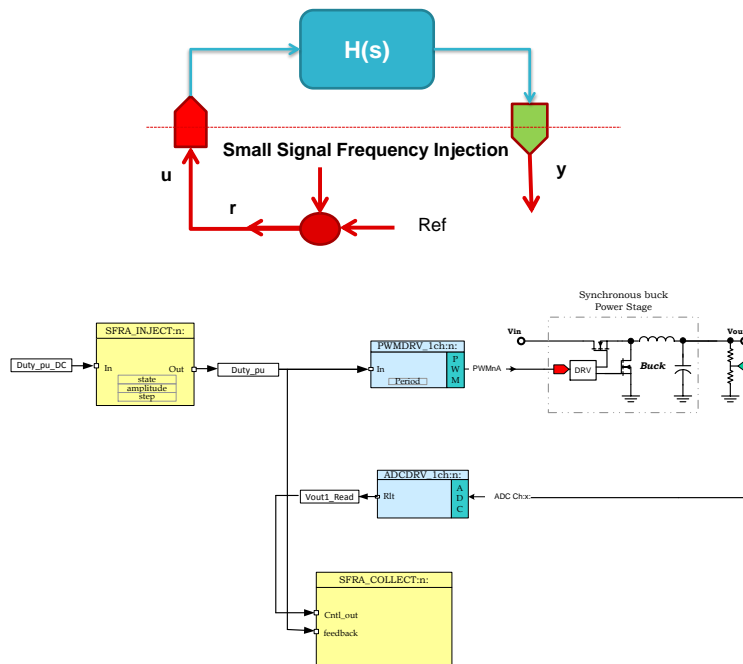


Figure 24 Software diagram for plant tf extraction

1. Import the project under SFRA/vX/examples/DPSWrkShpKit_SFRA_F28035 in CCS6.0 or later. "DPSWrkShpKit" will be referred to as <ProjectName> in the document. Go to the <ProjectName>-Settings.h file and set the incremental build level to 1.

```
//=====
// 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
```

```
#define INCR_BUILD 1
```

2. 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 200Khz, 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.

3. 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 below:

```
// 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
```

4. 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 below 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. Next the vector length or the number of points on which FRA is performed is specified. This is also the length of the arrays defined in the previous step. The rate at which SFRA library is called is specified next. Frequencies at which the response is calculated are uniquely specified by the frequency start variable and the frequency step multiply. Combination of the frequency sweep start, step multiply and length determine the frequency values at which 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);
```

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

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

6. Also the SFRA background task is called in a background task A1

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

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

```
SerialHostComms();
```

8. Next, 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);

...
}
```

9. Now it is time to setup the hardware. First verify that the SW1 on the TMS320F28035 control card is pointing down (towards the off position). SW3 on the control card is pointing down and SW2 on the control card is pointing up.
10. Insert the TMS320F28035 control car into the DIM100 connector on the DPS Workshop board.
11. Next make sure J1 on the main board and J4 on the M2 macro are populated (See Figure 23 for location on the board)
12. Make sure the DC Bus power switch (See Figure 23 for location on the board), is in off position. This is indicated by the LED LD1 next to the switch being off.
13. Connect a USB cable from the DPS board (JP1 of the M2 Macro) to a host computer.
14. Power up the DPS board, by connecting the 9V power supply at JP1 Vin. Switch the power switch SW1 to the ON position. You will see LD1 in M1 macro on the board and LD1 on the control card light up.
15. Now 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.
16. Then go to the Target → ”Debug Active Project” menu. The program will then be loaded into the flash of the F28035 device. The code will run to the start of Main().
17. Once loaded, enable real-time mode by hovering your mouse on the buttons on the horizontal toolbar and clicking button



Enable Silicon Real-time Mode (service critical interrupts when halted, allow debugger accesses while running)

- A message box *may* appear. If so, select YES to enable debug events.
18. Now 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).
 19. Click on Continuous Refresh buttons  for the watch view to enable the variables to be updated continuously.
 20. Run the project by clicking .
 21. 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 the continuous refresh icon () in the expressions view is selected.
 22. Now switch the, DC Bus power switch (see Figure 23 for location) ON, LED LD1 next to the switch will light up.
 23. The software is configured for open loop i.e. 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 will make the output voltage Gui_Vout1 go close to 2V, which is close to where this board is designed to operate. Set Active_LD1_EN to 1, this will load the power stage and the voltage may drop slightly. Adjust the duty to get the voltage back at 2V.
 24. If the correct voltage (~2V) does not appear on the Gui_Vout1 variable, when 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  and start again from step 13 and make sure the duty cycle is changed slowly.
 25. Now that the DC point is established, open the SFRA gui.exe, located at SFRA\version\GUI
 26. Refer to Section 3.8, for instruction on how to connect and what each panel on the SFRA GUI box means.
 27. Select fixed point math.
 28. Click Setup Connection and set the baud rate to be 57600 on the pop up window.
 29. Uncheck boot on connect and select the appropriate COM port. For procedure to find out which COM port to select refer to section 3.8.
 30. Click "OK" to close the pop-up window and return to the main screen.

31. 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.
32. Press the “Start Sweep” button.
33. Wait for the status bar in the GUI to change to “Sweep Complete”
34. 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 closed loop operation. Hence 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 below:

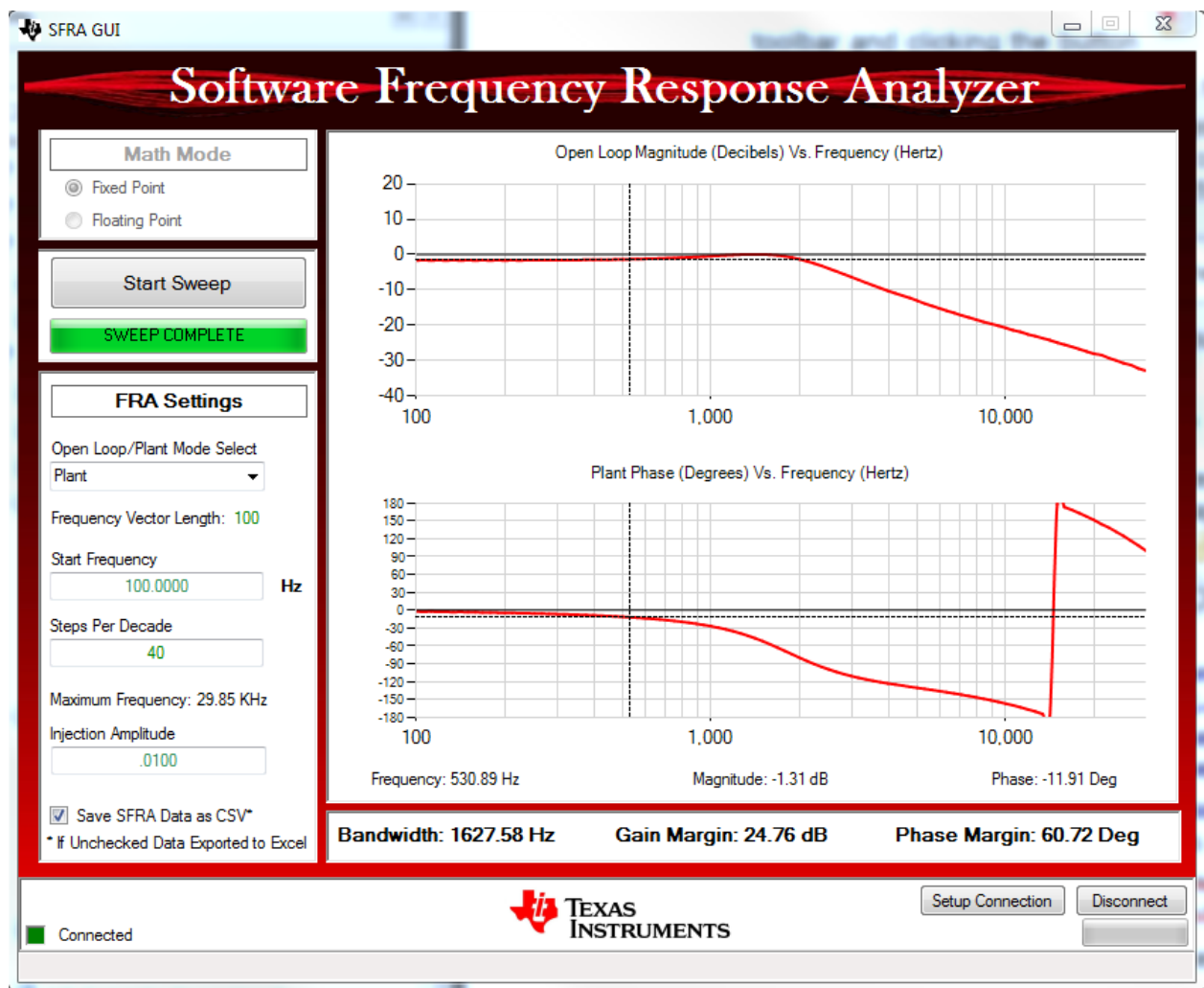






Figure 25 Plant Frequency Response Plot

35. Drag your mouse across either of the graphs to locate the values at specific frequencies used in the Frequency Response Analysis.
36. 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".
37. 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).
38. Once done with the evaluation, write zero to the Duty_pu_DC value, make the SW1 (on the main board) off.
39. Close the SFRA GUI.
40. 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 .
41. Close CCS debug session by clicking on Terminate Debug Session  (Target->Terminate all).
42. The frequency response measured can then be used to design the compensation by importing the frequency data into MATLAB. Scripts for which were described in 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. In the previous section (5.2) a run of SFRA was completed, the CompDesigner will use that SFRA measurement to design the compensation.

Figure below 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.

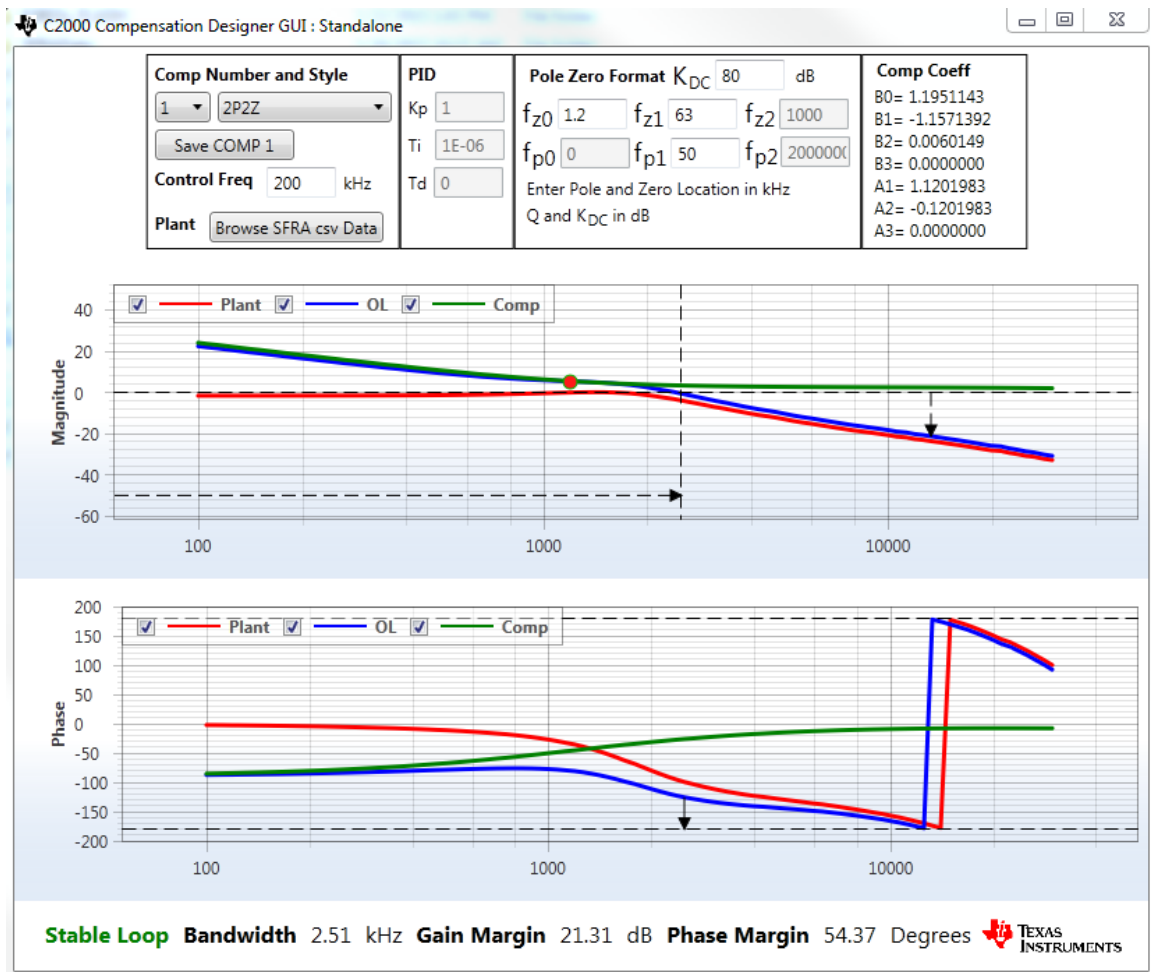
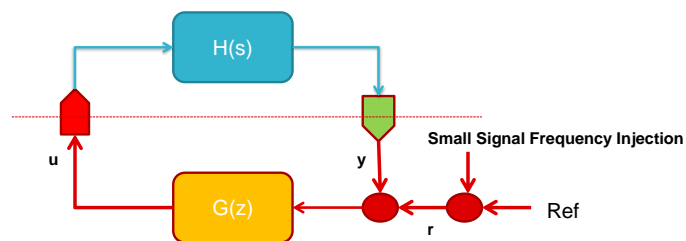


Figure 26 Comp Designer being used to for the DPS workshop board based on plant data from SFRA run

5.4 OL Measurement

In this example the SFRA library is used to measure the open loop frequency response of the closed loop controlled power converter. 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 SFRADData.csv file that is saved under the GUI folder. The diagram below describes the software connections for a closed loop system using SFRA.



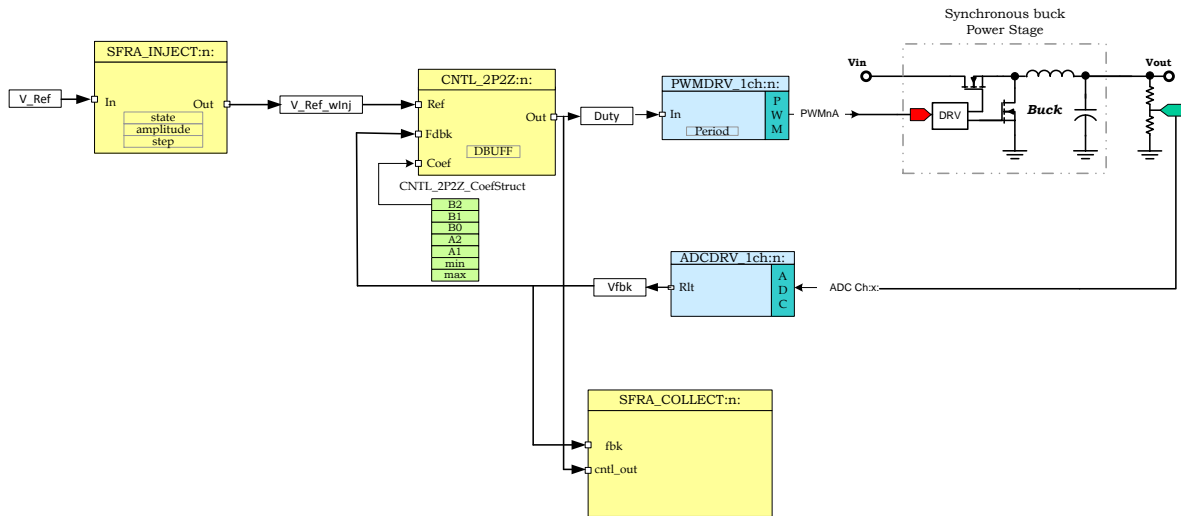


Figure 27 Closed Loop SFRA software diagram

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
```

```
#define INCR_BUILD 2
```

1. Assuming the user stepped through the Plant TF Extraction section the steps to setup are not repeated.
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_coeff1,&cntl3p3z_vars1);

    //Update PWM value
```

```

    EPwm1Regs.CMPA.half.CMPA=_IQ24mpy((long)(BUCK_PWM_PERIOD),cnt13p3
z_vars1.Out);

    SFRA_IQ_COLLECT(&cnt13p3z_vars1.Out,&cnt13p3z_vars1.Fdbk);
...
}

```

3. Also note the compensation coefficients that are programmed onto the device, these are the same coefficients as were designed in the section "Designing Compensator using Compensation Designer":

```

cnt13p3z_coeff1.Coeff_A1=_IQ24(1.1201983);
cnt13p3z_coeff1.Coeff_A2=_IQ24(-0.1201983);
cnt13p3z_coeff1.Coeff_A3=_IQ24(0.0);
cnt13p3z_coeff1.Coeff_B0=_IQ24(1.1951143);
cnt13p3z_coeff1.Coeff_B1=_IQ24(-1.1571392);
cnt13p3z_coeff1.Coeff_B2=_IQ24(-0.0060149);
cnt13p3z_coeff1.Coeff_B3=_IQ24(0.0);

```

4. Make sure DC Bus power switch,(see Figure 23 for location) is in off position. If ON turn this switch to off position. Note: The code is set to put Gui_Vset to 2.0V automatically, you must switch DC Bus power switch, Figure 23 on before this happens.
5. Now build and load the project similar to in section 5.2, enable real time and hit run.
6. Once the code is running put the DC Bus power switch, (see Figure 23 for location) in ON position immediately.
7. Now you will see the voltage being regulated at 2V.
8. To run SFRA, open the SFRA gui.exe, located at SFRA\version\GUI
9. Refer to Section 3.8, for instruction on how to connect and what each panel on the SFRA GUI box means.
10. Select fixed point math.
11. Click Setup Connection and set the baud rate to be 57600 on the pop up window.
12. Uncheck boot on connect and select the appropriate COM port. For procedure to find out which COM port to select refer to section 3.8.
13. Click "OK" to close the pop-up window and return to the main screen.
14. 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.
15. Press the "Start Sweep" button.
16. Wait for the status bar in the GUI to change to "Sweep Complete"

17. The results of the SFRA Sweep will be 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 the values reported match closely to what the system was designed for using the Plant Frequency Response Data from open loop run.

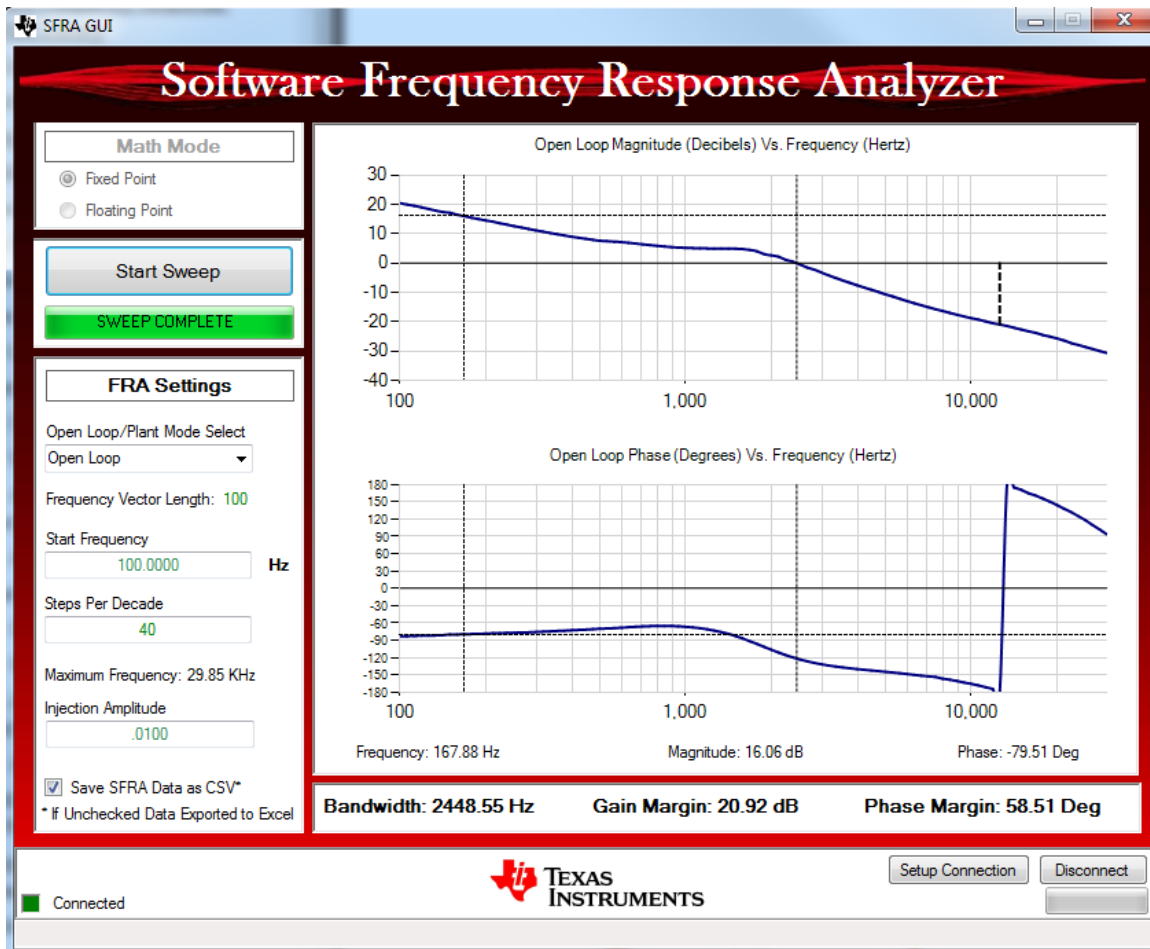


Figure 28 SFRA GUI with OL Frequency Response Plot

18. Drag your mouse across either of the graphs to locate the values at specific frequencies used in the Frequency Response Analysis.
19. 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".

20. Critical values like Bandwidth, gain margin and phase margin are brought out on the GUI. The values can be used to verify if the designed compensation really met the goals.
21. Once finished, put SW1 in off position, and the code can be halted and CCS debug session stopped.

5.4. Comparing SFRA Measured Frequency Response Versus Modelled

The SFRA library has been run successfully on a number of power topologies. The following section shows the comparison between modelled and measured frequency response on the DPSWorkshop board Figure 28 & Figure 29. This clearly shows that the modelling 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.

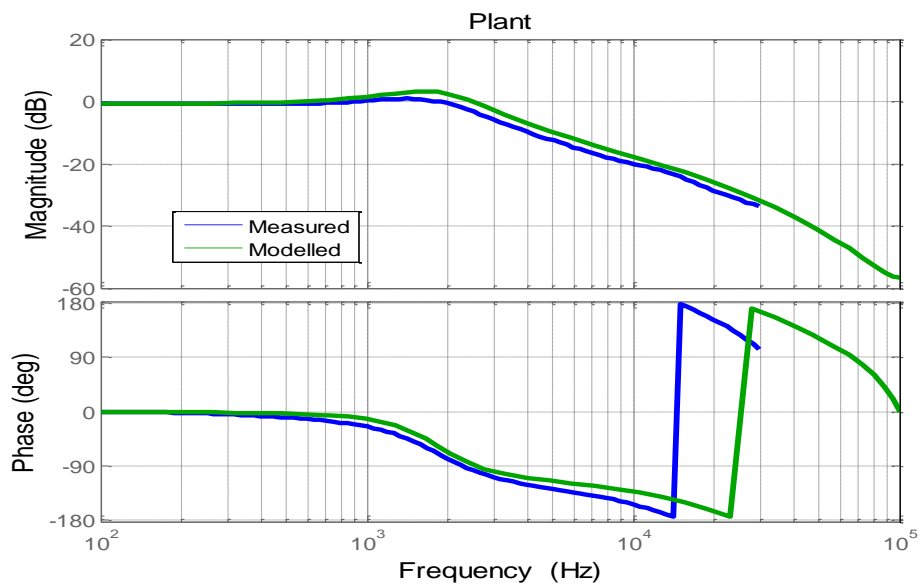


Figure 29 Plant Frequency Response Modelled Vs Measured on DPSWrkShpKit

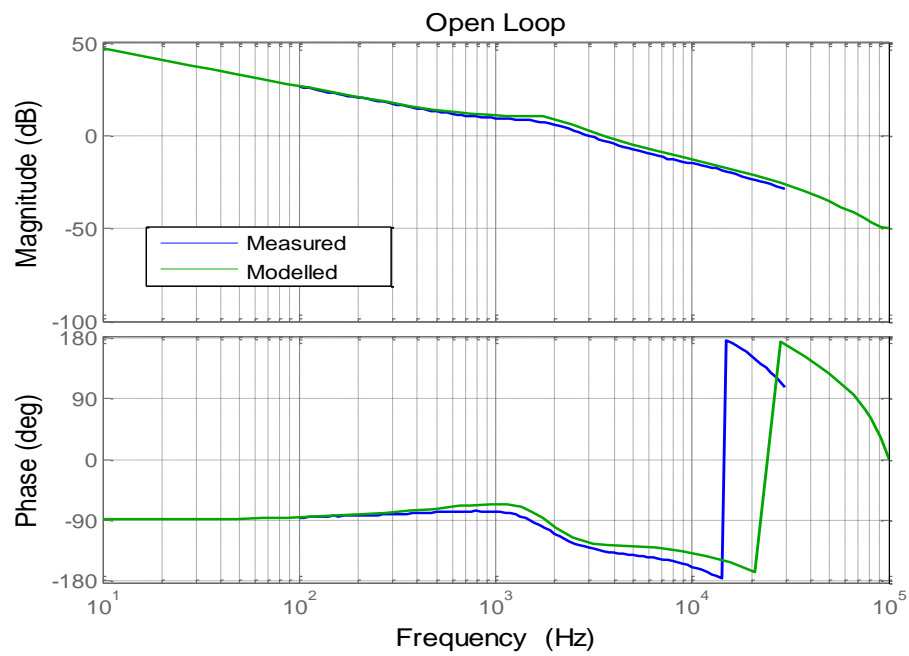


Figure 30 Open Loop Frequency Response Modelled Vs Measured on DPSWrkShpKit

Chapter 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 selection is not English (USA) and the above behavior is observed, please try selection the language to be English(USA). To do this go to “Region and Language” using the search in the Start Button , and select under Formats -> English (USA) and apply.

6.3. What is the max frequency SFRA can measure response for?

Max frequency measurable is determined by the SFRA_ISR_FREQ and is milted by the nyquist criterion i.e. half of the switching frequency or the frequency at which 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 ~60-70dB. Hence 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

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

Chapter 7. Revision History

Version	Date	Notes
V1_10_00_00	February, 2015	<ol style="list-style-type: none">1. Added documentation to add SFRA to Digital Power Library based on assembly macros.2. Added documentation for Compensation Designer and included it as part of the SFRA install.3. Re-ordered chapters4. SFRA GUI now defaults to storing data in CSV format. The user can choose to select Excel if needed. When launched from powerSUITE GUI CSV is the default and cannot be changed.
V1_00_00_00	October, 13 2014	First Release of SFRA Library