Application Report

Application Software (SW) Architecture for MSP430FR6043-Based Ultrasonic Gas Flow Meter

Luis Reynoso *MSP430 Applications*

ABSTRACT

This document explains the software architecture used in the implementation of an ultrasonic gas flow meter with the MSP430FR6043 MCU. The document describes the software libraries used by the application as well as the file structure and all relevant source, library, and project files. It describes all resources and peripherals used by the application.

Table of Contents

Trademarks

MSP430™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Software Architecture

The MSP430FR6043 ultrasonic gas flow meter application is a portable and modular software designed to guide developers in the implementation and customization of their own gas meter solutions.

Figure 1-1 shows the software architecture.

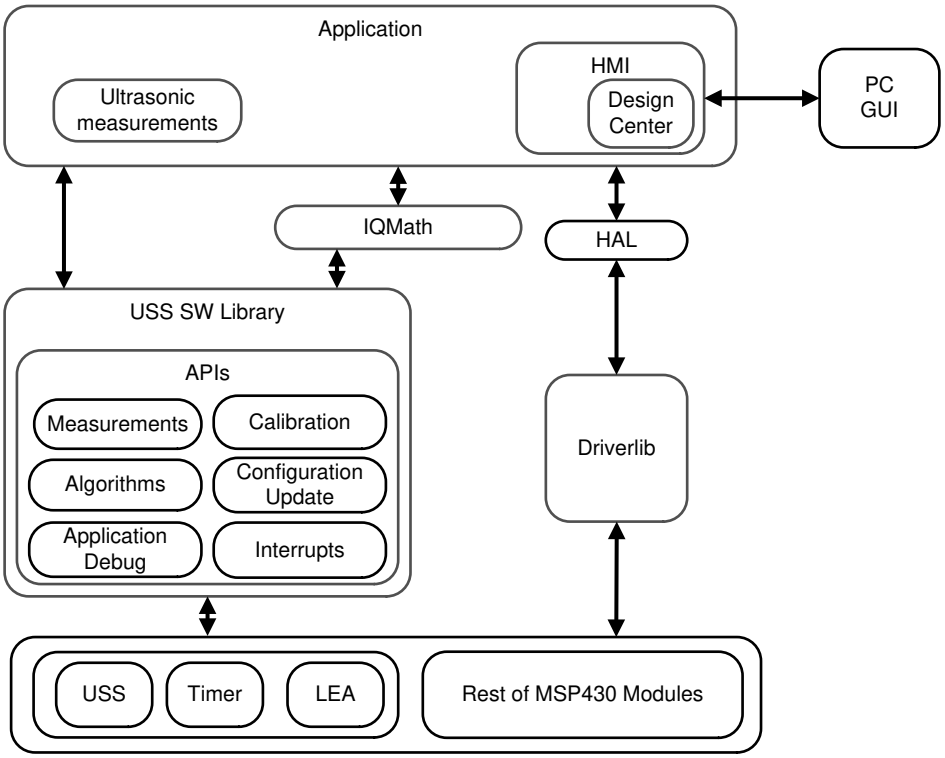

Figure 1-1. Software Architecture

Section 1.1 describes the application residing in the top layer in more detail.

Section 1.2 describes the IQMath library, which is used to perform fixed point operations in an effective and efficient way.

Section 1.3 describes the Ultrasonic Sensing Software Library (USS SW Library), which is used to implement ultrasonic captures and measurements.

Section 1.4 and Section 1.5 describe the hardware-abstraction layer (HAL) and MSP430™ DriverLib, respectively, which provide a modular and portable implementation.

1.1 Application

The application layer has two main tasks: to perform ultrasonic captures and measurements, and to interface with the user.

Ultrasonic captures and measurements are performed by calling APIs from the USS SW library. The application layer handles all responses from the library including results and errors.

User interaction is implemented in the human-machine interface (HMI) layer. This layer includes support for communication with a graphical user interface (GUI), LCD, push buttons, and LEDs.

Figure 1-2 shows the flow diagram of the application.

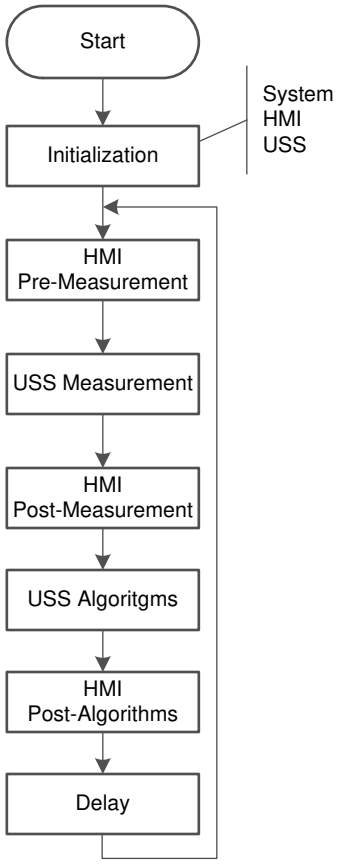

Figure 1-2. Application Flow Diagram

The application starts by initializing the basic functionality of the system including peripherals, clocks and IOs, followed by the initialization of the HMI peripherals including LCD, GUI communication, buttons and LEDs. Finally, the system performs the initial configuration of the ultrasonic subsystem and its algorithms.

After initialization, the application stays in a continuous loop performing the following actions:

- HMI pre-measurement: Performs user interaction functions before an ultrasonic measurement, such as checking if the GUI has a new configuration to send.
- USS measurement: Performs an ultrasonic measurement. The result of this function is an ADC-sampled waveform.
- HMI post-measurement: Performs user interaction functions after a measurement such as sending the ADC waveform to the GUI if requested.
- USS algorithms: Runs ultrasonic algorithms on the ADC-sampled waveform to calculate data such as the time-of-flight (TOF) and the volume flow rate.
- HMI post-algorithms: Performs user interaction functions after the algorithms are executed. This includes sending the results to the GUI or displaying them in the LCD.
- Delay: The device goes to a low-power state waiting for the next iteration.

Although the configuration of ultrasonic sensing parameters is implemented in the Ultrasonic Sensing Software Library (see Section 1.3), the application includes the implementation of several features which can be configured as described in Section 3.2.

1.1.1 Ultrasonic Sensing Design Center

One of the functions of the HMI layer is to interface with a PC GUI known as the Ultrasonic Sensing Design Center (see Figure 1-3).

This GUI was developed to allow for an easier configuration of the USS subsystem, enabling developers to observe the effect of each parameter in the ADC-sampled waveform or in real-time plots of the time-of-flight and volume flow rate.

For more information about the Ultrasonic Sensing Design Center, see the MSP430FR6043 ultrasonic design center user's guide.

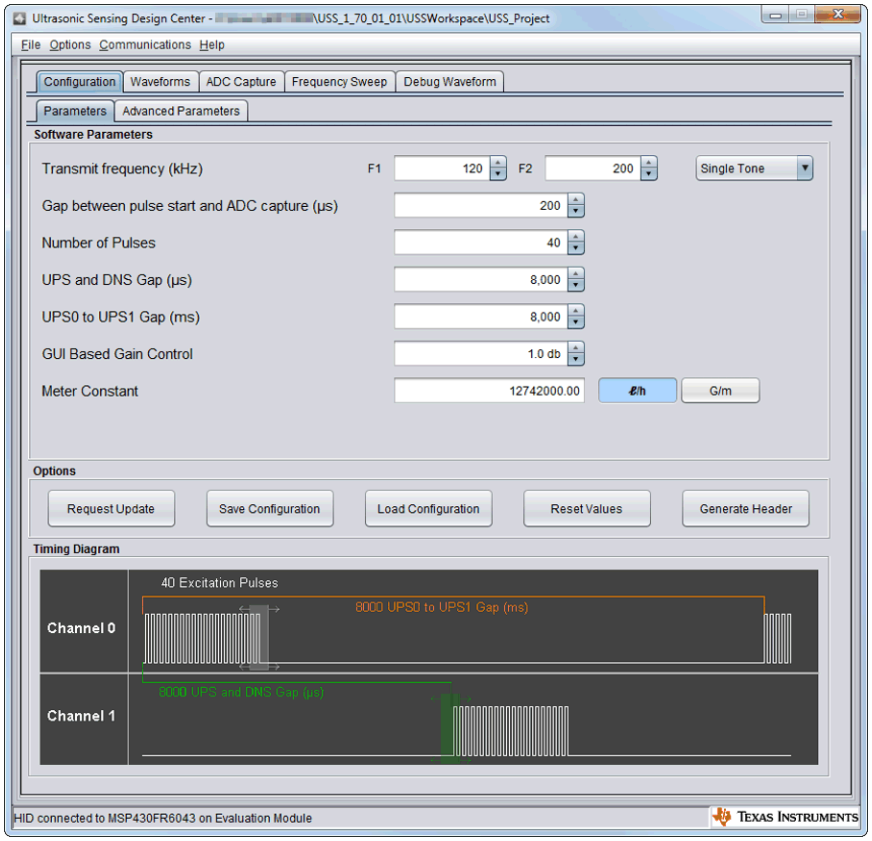

Figure 1-3. Ultrasonic Sensing Design Center

1.2 IQMath Library

The MSP IQMath and QMath libraries are collections of highly optimized and high-precision mathematical functions for C programmers to seamlessly port a floating-point algorithm into fixed-point code on MSP430 devices. By using IQMath and QMath libraries, it is possible to achieve execution speeds considerably faster and energy consumption considerably lower than equivalent code written using floating-point math.

Both the application and the USS SW Library make use of the IQMath library to perform fixed point operations in an effective and efficient way.

For more information about IQMath and QMath libraries, see MSP-IQMATHLIB.

1.3 Ultrasonic Sensing Software Library (USS SW Library)

The ultrasonic software library is a set of APIs and proprietary algorithms created to configure the ultrasonic subsystem of the MSP430FR6043 MCU, perform ultrasonic captures, and process the signal received from the transducers to calculate output information such as the time-of-flight (TOF) and flow rate.

The library includes an easy-to-implement set of fully documented APIs that hide the complexity behind ultrasonic measurement calculations and allow for a faster implementation of the application.

The library's APIs are grouped in the following categories:

- Measurement APIs: Configure the USS subsystem and perform signal captures.
- Algorithms APIs: Process signal captures and generate output flow information.
- Application debug APIs: Functions to help during the development and debugging stages.
- Calibration APIs: Implement calibration of USS subsystem including modules like the HSPLL or SDHS.
- Configuration update APIs: Reconfigure the USS subsystem if a new configuration is sent by the application layer (for example, from a GUI).
- Interrupt APIs: Registers and handles interrupts used by the application.

The algorithm module of the USS SW Library is provided in CCS and IAR library format, while the rest of the modules are provided in source code. This flexibility enables developers not only to understand the implementation but to customize it according to their needs.

A comprehensive list of parameters allows developers to configure the system according to different system and transducer requirements. The configuration and customization of the ultrasonic parameters is explained in Section 3.1.

For more details about the USS SW library, including a comprehensive description of the available APIs, see MSP-USSSWLIB.

1.4 Hardware Abstraction Layer (HAL)

All hardware interactions are implemented in the hardware abstraction layer (HAL) allowing for a more modular and portable solution.

The HAL layer includes the following functionality:

- ADC: implements battery voltage measurement and temperature measurement using the MSP430FR6043's ADC12 module.
- LCD: implements functions to configure and control the LCD_C module in the MSP430FR6043 MCU.
- System: includes functions to configure and control the watchdog (WDT), clock system (CS), and GPIOs.
- UART: includes functions to configure and communicate using the eUSCI_A module.

1.5 MSP430 Driver Library (DriverLib)

Driver Library (or DriverLib) includes APIs for selected MSP430 device families providing easy-to-use function calls. Each API is thoroughly documented through a user's guide, API guide and code examples.

The ultrasonic flow meter application software uses DriverLib to interface with all hardware modules used by the application. This allows for an easier migration to other MSP MCUs and makes the code easier to read and understand by using common language APIs.

All DriverLib files used by this application are included in source code.

More information and documentation about DriverLib is available at MSPDRIVERLIB.

2 File Structure

Example 2-1 shows the file structure of the MSP430FR6043 ultrasonic gas flow meter application.

Example 2-1. File Structure

```
UltrasonicGasFR6043>
|
+---driverlib
|       \---MSP430FR5xx_6xx
|
+---examples
|   +---common
|   |   +---DesignCenter
|   |   +---gui
|   |   \---hal
|   |       \--fr6043EVM
|   |
|   +---mtr_gui_config
|   |
|   \---MSP430FR6043EVM_USS_Gas_Demo
|       +---CCS
|       +---IAR
|       +---USS_Config
|       \---USSLibGUIApp
|
+---image
|
+---include
|
\---lib
    +---IQMathLibrary
    \---USS
        +---gas
        |   +---all
        |   +---hilbertwide
        |   \---lobewide
        \---source
```

The most relevant folders are:

- *driverlib*: Source code of MSP DriverLib files supporting the MSP430FR6043 MCU
- *examples*: Contains code examples using the USS SW library
- *common*: files shared by USS applications. They include the HAL layer and files for used for communication with Design Center. These files can be modified to customize the system according to specific hardware or communication requirements. More information available in Section 3.3.
- *mtr_gui_config*: Includes files to configure the USS subsystem with Design Center to support some sample meters and transducers
- *MSP430FR6043EVM_USS_Gas_Demo: Main application folder including source code and IAR and CCS project files for the application described in this document.*
  - *CCS*: Project files for CCS
  - *IAR:* Project files for IAR
  - *USS_Config*: Includes a header file with definition of the structure passed to the USS SW library and the definition of the default configuration of the application. These files can be modified to set the default configuration of the system. For more information, see Section 3.1 and Section 3.2.
  - *USSLibGUIApp*: Source code for main functionality of USS gas meter application
- *image*: Pre-built binaries for the application
- *include*: Header files for IQMath and USS SW Library
  - *USS_HAL*: Hardware Abstraction Layer for Ultrasonic USS Library using the MSP430FR6043 MCU
- *lib:* Library files
  - *IQMathLibrary*: Library files for IQMath
  - *USS*: Library files and public source code for USS SW Library.

2.1 Application Layer Files

2.1.1 main.c

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\main.c

Description

Main application file. Initializes the system and jumps to the main application loop.

Relevant Functions

```
void main(void)
```

Main function for ultrasonic gas meter application.

2.1.2 system_pre_init.c

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\system_pre_init.c

Description

Function called after reset to initialize time-critical modules in the MSP430 MCU before jumping to the main function. This application is only using this file to initialize the Watchdog.

Relevant Functions

```
int _system_pre_init(void)
```

System initialization for CCS.

```
int __low_level_init(void)
```

System initialization for IAR.

2.1.3 USS_Config.c, USS_Config.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USS_Config\USS_userConfig.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USS_Config\USS_userConfig.h

Description

Default configuration of the USS subsystem. Pre-configures all parameters used by the ultrasonic software library, including transducer frequency, algorithm thresholds, or the sampling frequency, among others.

These files should be modified by developers to set the default configuration of the system. See Section 3.1 for more details.

Relevant Variables

```
__persistent USS_SW_Library_configuration gUssSWConfig
```

Structure used to initialize USS SW Library and defining the default configuration of USS subsystem and algorithms.

2.1.4 USS_App_userConfig.c, USS_App_userConfig.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USS_Config\USS_App_userConfig.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USS_Config\USS_App_userConfig.h

Description

Default configuration of the gas metering application. Allows developers to enable, disable and configure several functions implemented in the application layer.

These files can be modified by developers to set the default configuration of the application as described in Section 3.2.

Relevant Variables

```
__persistent USS_App_userConfig_t USS_App_userConfig
```

Structure used to define the default configuration of gas metering application.

2.1.5 USSLibGUIApp.c, USSLibGUIApp.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\USSLibGUIApp.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\USSLibGUIApp.h

Description

Integrates the application together with ultrasonic library, HMI and GUI. Calls initialization of the USS library and implements the application's main loop.

Relevant Functions

```
void USSLibGUIApp_Init(void)
```

Initialization of USS gas meter application.

```
void USSLibGUIApp_Engine(void)
```

Main loop of USS gas meter application calling USS library APIs to get new measurements and execute the algorithms, and calling HMI functions to get and send information to the user.

2.1.6 masterIncludes.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\masterIncludes.h

Description

Common definitions used by application files.

2.1.7 hmi.c, hmi.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\hmiDC\hmi.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\hmiDC\hmi.h

Description

Human-machine interface files implementing interactions with user including GUI, LCD, buttons and LEDs. These files can be modified by developers to customize interactions with user.

Relevant Functions

```
void HMI_Init(void)
```

Initialization of HMI peripherals.

```
void HMI_updateUSSParameters(void)
```

Function called when the GUI requests a parameter update. This function reconfigures the USS sub-system and its algorithms.

```
void HMI_reportError_sendErrorMessage(uint16_t error_type, uint16_t error_info)
```

Handles errors which are reported to the user. These errors are sent to the GUI, but developers can also log them or report them in any other way.

```
void HMI_guiInputValidation(void)
```

Validates inputs from the GUI.

Can be modified by developers to change the values of parameters accepted by the application.

```
void HMI_PreMeasurement_Update(void)
```

Function called before an ultrasonic measurement to check for a new configuration from the GUI.

```
void HMI_PostMeasurement_Update(void)
```

Function called after an ultrasonic measurement to send the ADC waveform to the GUI if requested.

```
void HMI_PostAlgorithm_Update(USS_Algorithms_Results * pt_alg_res)
```

Function called after the ultrasonic algorithms are executed. Used to send algorithm results to the GUI if requested.

Relevant Variables

```
HMI_App_Config_t HMI_App_Config
```

Application configuration. Defines the sleep time between ultrasonic captures.

2.1.8 lcd_statemachine.c, lcd_statemachine.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\lcd_statemachineDC\lcd_statemachine.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\lcd_statemachineDC\lcd_statemachine.h

Description

Implements the LCD state machine.

Relevant Functions

```
void lcd_satemachine_stateUpdate(void)
```

Handles the LCD state machine, checking for button presses and updating the LCD.

```
void lcd_statemachine_stateAction(void)
```

Function executed after the state machine is updated to perform a corresponding action on the LCD.

Relevant Variables

```
LCD_STATEMACHINE_t g_lcd_statemachine
```

State machine of the LCD.

2.1.9 mathematics.c, mathematics.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\mathematicsDC\mathematics.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\mathematicsDC\mathematics.h

Description

Implements calculations used by the HMI and LCD display.

2.1.10 results.c, results.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\resultsDC\results.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\resultsDC\results.h

Description

Updates the information displayed on LCD.

Relevant Functions

```
void results_Reset(bool unit)
```

Clears all the results of data displayed on LCD.

```
void results_Update(float dtofData, float volrateData, uint16_t rate)
```

Updates all results of data displayed on LCD.

Relevant Variables

```
RESULTS_AVERAGEDATA g_ResultsOfLastMeasurement
```

Structure containing the data which is displayed on LCD.

2.1.11 testing.c, testing.h

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\testingDC\testing.c

~\examples\MSP430FR6043EVM_USS_Gas_Demo\USSLibGUIApp\fr6043_USS_app\testingDC\testing.c

Description

File used to configure the ultrasonic subsystem.

Developers are encouraged to use this file to customize the behavior of the system before and after ultrasonic measurements if needed.

Relevant Functions

```
void Testing_PreMeasurement_Update(void)
```

Function called before an ultrasonic measurement. Developers can use this function to configure the USS subsystem before a measurement.

```
void Testing_PostMeasurement_Update(void)
```

Function called after an ultrasonic measurement. Developers can use this function to configure the USS subsystem after a measurement.

```
void Testing_PostAlgorithm_Update(void)
```

Function called after running ultrasonic algorithms. Developers can use this function to configure the USS subsystem after the algorithms are executed.

```
void Testing_GUIUpdate_PreUSSConfig(void)
```

Function called when the GUI requests an update, before the USS Library is reconfigured.

```
void Testing_GUIUpdate_PostUSSConfig(void)
```

Function called when the GUI requests an update, after the USS Library is reconfigured.

```
void Testing_Update_Results(HMI_DesignCenterALG_Results *results)
```

Function called after the algorithms are executed and just before the results are sent to the GUI. Can be used to modify the data sent to the GUI if needed.

2.2 Ultrasonic Sensing Design Center Files

2.2.1 ussDCCommandHandlers.c, ussDCCommandHandlers.h

Location

~\examples\common\DesignCenter\ussDC\ussDCCommandHandlers.c

~\examples\common\DesignCenter\ussDC\ussDCCommandHandlers.h

Description

Implements the communication with Design Center. Registers and implements the command listeners called when a command is received.

Relevant Functions

```
void CommandHandler_registerCmdListeners()
```

Registers the command listeners which are called when commands are received from GUI.

```
void CommandHandler_transmitResults(Packet_t *txPacket, USS_Algorithms_Results* algRes)
```

Transmit all algorithm results to Design Center GUI.

```
void CommandHandler_transmitCaptures(Packet_t *txPacket)
```

Transmits the ADC waveforms to the Design Center GUI.

```
void CommandHandler_transmittDebugData(Packet_t *packet, float data)
```

Transmits debug data to the Design Center. Can be used by developers to send custom data to the GUI which will be displayed in a debug plot.

2.2.2 comm.c, comm.h

Location

~\examples\common\DesignCenter\comm\comm.c

~\examples\common\DesignCenter\comm\comm.h

Description

Communication module top level API. These are the communication functions that will

be called directly by the application.

Relevant Functions

```
void Comm_setup(void)
```

Setup the communication module for operation.

```
bool Comm_addCmdListener(uint8_t cmd0, uint8_t cmd1, Listener_handler_t handler)
```

Adds command listeners which are called when a command is received from Design Center.

Relevant Variables

```
const tUARTPort Comm_uartPort
```

The UART port definition for the UART driver. Used only when UART is enabled for communication with Design Center.

```
const tI2CSlavePort Comm_i2cSlavePort
```

The I^2C slave port definition for the I^2C slave driver. Used only when I^2C is enabled for communication with Design Center.

2.2.3 Comm_config.c, Comm_config.h

Location

~\examples\common\DesignCenter\comm\comm_config.c

~\examples\common\DesignCenter\comm\comm_config.h

Description

Communication module configuration file. Can be used to configure the communication interface (UART or I^2C) used for communication with Design Center.

2.2.4 drivers\

Location

~\examples\common\DesignCenter\comm\drivers*.*

Description

Implementation of lower-level drivers used to communicate with Design Center. These files are not expected to be modified by developers using Design Center.

2.2.5 protocol\

Location

~\examples\common\DesignCenter\comm\protocol*.*

Description

Protocol used for communication with Design Center. These files are not expected to be modified by developers using Design Center.

2.2.6 utils\

Location

~\examples\common\DesignCenter\comm\utils*.*

Description

Queue and ping-pong buffer implementations used by other design center files. These files are not expected to be modified by developers using Design Center.

2.3 IQMath Library Files

2.3.1 include\

Location

~\include\IQmathLib.h

~\include\QmathLib.h

Description

Header files for IQMath (32-bit) and QMath (16-bit) fixed point libraries. Includes all definitions and function prototypes accessible by the application.

For more information about IQMath and QMath libraries, visit MSP-IQMATHLIB.

2.3.2 lib\

Location

~\lib\IQMathLibrary\libraries\CCS*.*

~\lib\IQMathLibrary\libraries\IAR*.*

Description

Library files for IAR and CCS.

For more information about IQMath and QMath libraries, visit MSP-IQMATHLIB.

2.4 Ultrasonic Software Library Files

2.4.1 include\

Location

~\include\ussSwLib.h

Description

Header files for USS SW Lib. Includes all definitions and function prototypes accessible by the application.

For more details about the USS SW library, see MSP-USSSWLIB.

2.4.2 lib\

Location

~\lib\USS\gas*\CCS*.*

~\lib\USS\gas*\IAR*.*

Description

Library files for IAR and CCS.

For more details about the USS SW library, see MSP-USSSWLIB.

Note

The software package includes library files supporting all or individual available gas. The demo includes the complete library supporting all algorithm but developers can use an optimized version to reduce the footprint of their application.

2.4.3 ussSwLibMeasurement.c

Location

~\lib\USS\source\ussSwLibMeasurement.c

Description

Configures the USS module and performs ultrasonic measurements.

Relevant Functions

```
USS_message_code USS_configureUltrasonicMeasurement(USS_SW_Library_configuration *config)
```

Configure the USS module based on user configuration.

```
USS_message_code USS_startLowPowerUltrasonicCapture(USS_SW_Library_configuration *config)
```

Performs a power-optimized ultrasonic capture.

2.4.4 ussSwLibConfiguration.c

Location

~\lib\USS\source\ussSwLibConfiguration.c

Description

Configures the different USS modules

Relevant Functions

```
USS_message_code USS_updateSDHSConfiguration(USS_SW_Library_configuration *config)
```

Configures the SDHS.

```
USS_message_code USS_updateHSPLLConfiguration(USS_SW_Library_configuration *config)
```

Configures HSPLL.

```
USS_message_code USS_updateSAPHConfiguration(USS_SW_Library_configuration *config)
```

Configures SAPH.

```
USS_message_code USS_updateUUPSConfiguration(USS_SW_Library_configuration *config)
```

Configures UUPS.

2.4.5 USSSWLib Common files

Location

~\lib\USS\source\common*.*

Description

Drivers for different USS submodules -including HSPLL, SAPH, SDHS, USS-, interrupts, and a timer used by USS library.

2.4.6 USSSWLib HAL files

Location

~\include\USS_HAL\FR6043\USS_Lib_HAL.*

Description

Includes the declaration of interrupts, GPIOs and peripherals used by the USS library.

2.5 HAL Files

2.5.1 hal.h

Location

~\examples\common\hal\fr6043EVM\hal.h

Description

Main HAL header file including other HAL files used by the application.

2.5.2 hal_adc.c. hal_adc.h

Location

~\examples\common\hal\fr6043EVM\hal_adc.c

~\examples\common\hal\fr6043EVM\hal_adc.h

Description

HAL files for ADC implementing temperature sensor and supply voltage measurements using the internal ADC12 in the MSP430FR6043 MCU.

Relevant Functions

```
void hal_adc_init(void)
```

Initializes the ADC12 in MSP430FR6043 to read the internal temperature sensor and supply voltage.

```
float hal_adc_tempsensor_readCelsius(void)
```

Reads the internal temperature and returns the value in Celsius degrees.

```
uint16_t hal_adc_voltagesupply_readmV(void)
```

Reads the supply voltage and returns the value in millivolts.

2.5.3 hal_lcd.c, hal_lcd.h

Location

~\examples\common\hal\fr6043EVM\hal_lcd.c

~\examples\common\hal\fr6043EVM\hal_lcd.h

Description

HAL files supporting the FH-1138P segmented LCD using the LCD_C module on the MSP430FR6043 MCU.

Relevant Functions

```
void hal_lcd_Init(void)
```

Initializes the LCDC_C in MSP430FR6043 including I/Os, voltage reference, and charge pump.

```
void hal_lcd_turnonLCD(void)
```

Turn on LCD.

```
void hal_lcd_turnoffLCD(void)
```

Turn off LCD.

2.5.4 hal_system.c, hal_system.h

Location

~\examples\common\hal\fr6043EVM\hal_system.c

~\examples\common\hal\fr6043EVM\hal_system.h

Description

HAL file for system modules including the clock system (CS), watchdog (WDT), and GPIOs.

Relevant Functions

```
void hal_system_Init(void)
```

Initializes the system, including GPIOs, clock system, watchdog, LCD, and ADC.

```
void hal_system_WatchdogInit(void)
```

Initializes the watchdog.

2.5.5 hal_uart.c, hal_uart.h

Location

~\examples\common\hal\fr6043EVM\hal_uart.c

~\examples\common\hal\fr6043EVM\hal_uart.h

Description

HAL file for UART communication module. The application is not using the UART module but it can be used to communicate with PC using the ezFET backchannel UART available in the MSP430FR6043 EVM.

2.6 DriverLib Files

Location

~\driverlib\MSP430FR5xx_6xx

Description

DriverLib source and header files supporting the MSP430FR6043 MCU.

For more details about DriverLib, see MSPDRIVERLIB

2.7 IDE Files

2.7.1 CCS Project File

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\CCS\MSP430FR6043EVM_USS_Gas_Demo.spec

Description

This file can be imported into CCS, creating a copy of the USS gas meter project in the user's workspace. See the CCS documentation for more information on how to import and build projects.

The project contains the following configuration:

- EVM_E1_AFE3v3: Configured to use EVM E1 using edefault configuration with external 3.3V AFE.
- EVM_E2_AFE3v3: Configured to use EVM E2 using edefault configuration with external 3.3V AFE.

2.7.2 CCS Linker File

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\CCS\lnk_msp430fr6043.cmd

Description

Linker file used by CCS project. This file can be modified by developers to change the memory layout used by the application.

A copy of this file is made by CCS when importing the project to the user's workspace.

2.7.3 IAR Project Files

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\IAR\MSP430FR6043EVM_USS_Gas_Demo.*

Description

IAR workspace and project files.

Developers can import the project file (MSP430FR6043EVM_USS_Gas_Demo.ewp) to their workspace, or they can open the provided workspace (MSP430FR6043EVM_USS_Gas_Demo.eww). See the IAR documentation for more information on how to import and build projects.

The project contains the following configuration:

- EVM_E1_AFE3v3: Configured to use EVM E1 using edefault configuration with external 3.3V AFE.
- EVM_E2_AFE3v3: Configured to use EVM E2 using edefault configuration with external 3.3V AFE.

2.7.4 IAR Linker File

Location

~\examples\MSP430FR6043EVM_USS_Gas_Demo\IAR\lnk430fr6043.xcl

Description

Linker file used by IAR project. This file can be modified by developers to change the memory layout used by the application.

3 Customizing the Application

The ultrasonic gas flow meter application described in this document is provided in source code, allowing developers to customize it according to their needs. Although developers can customize the functionality and behavior of the application as needed, some of the most relevant customizations include: changing the default configuration of the USS SW library, changing the default configuration of the application, making hardware modifications, modifying the ultrasonic capture sequence, and modifying the algorithms.

3.1 Changing the Default USS Configuration

The Ultrasonic Sensing Design Center GUI can be used to modify the configuration of the system without any software changes; however, developers have the option to customize the default configuration of the system without using the GUI.

The default USS configuration is defined in the following files:

~>examples\MSP430FR6043EVM_USS_Gas_Demo\USS_Config\USS_userConfig.c

~>examples\MSP430FR6043EVM_USS_Gas_Demo\USS_Config\USS_userConfig.h

These files contain the following global structure which sets the default configuration of the system:

```
__persistent USS_SW_Library_configuration gUssSWConfig
```

As observed, gUssSWConfig is defined as a persistent structure, meaning that it can be modified as a RAM variable but it resides in FRAM and retains its contents after a power cycle.

gUssSWConfig contains pointers to the structures in Table 3-1.

Table 3-1. Contents of gUssSWConfig Structure

Structure	Description	Example Members
USS_System_Configuration .systemConfig	Configuration of system parameters	.mCLKFrequency .LFXTFrequency .timerBaseAddress
USS_Meter_Configuration .meterConfig	Configuration of the meter and transducer characteristics	.volumeScaleFactor .acousticLength .transducerFreq
USS_Measurement_Configuration .measurementConfig	Configuration of ultrasonic measurements	.pulseConfig .startPPGCount .startADCsamplingCount
USS_HSPLL_Configuration .pllConfiguration	Configuration of PLL	.pllXtalFreq_inKHz .pllOutputFreq .hspllTolerance
USS_Capture_Configuration .captureConfig	Configuration of capture of ultrasonic waveform	.overSampleRate .gainRange .sampleSize
USS_Trigger_Configuration .triggerConfig	Configuration of USS trigger	.triggerConfig
USS_Interrupt_Configuration .interruptConfig	Configuration of PLL	.enableUUPSPREQIGInterrupt .enableSAPHPingTransmitDoneInterrupt
USS_Algorithms_User_Configuration .algorithmsConfig	Configuration of USS algorithms	.pAlgorithmUserConfig .algorithmOption .filerCoeffs

A more detailed and complete definition of all the parameters is included in MSP-USSSWLIB.

USS_userConfig.c contains the definition of this structure; however, all the parameters are initialized with a default value defined in USS_userConfig.h.

For example, the number of samples taken per capture is defined as follows:

USS_userConfig.c:

```
__persistent USS_Capture_Configuration ussCaptureConfig =
{
    .overSampleRate = USS_OVER_SAMPLE_RATE,
    .sampleSize = USS_USER_CONFIG_NUMBER_OF_SAMPLES_PER_CAPTURE,
    .gainRange = USS_GAIN_RANGE,
    .enableWindowHiComp = USS_ENABLE_WINDOW_HI_COMP,
    .enableWindowLoComp = USS_ENABLE_WINDOW_LO_COMP,
    ...
};
```

USS_userConfig.h

```
#define USS_USER_CONFIG_NUMBER_OF_SAMPLES_PER_CAPTURE 400
```

3.1.1 Using Design Center to Generate Header Files

The Ultrasonic Sensing Design Center GUI can be used not only to modify the runtime configuration of the device, but also to create header files defining the default configuration.

For more information, see the MSP430FR6043 ultrasonic design center user's guide.

The Design Center allows developers to modify the most common parameters; however, developers might need to modify the header files manually to change the default configuration of other parameters.

3.2 Configuring Application Features

The gas metering application for MSP430FR6043 includes several features that can be enabled, disabled, or customized by developers. These features include: capture delay, resonator calibration, automatic gain control, and DC offset cancellation.

3.2.1 Capture Delay

The application is configured to take ultrasonic measurements periodically. This periodicity is defined by the following definition in USS_App_userConfig.h:

```
#define USS_APP_CAPTURE_DELAY_ACLK (32768)
```

The period is defined in ACLK counts, which is derived from the 32.768-kHz crystal.

For example, a value of 32768 will be equivalent to 1 second.

The value can be changed at runtime by modifying the following structure member:

```
USS_App_userConfig.u16CaptureDelayACLK
```

3.2.2 Resonator Calibration

The application can be enabled to calibrate the USS oscillator periodically using the 32.768-kHz crystal as a reference. The resulting value is used as a correction term for TOF calculations.

Note that this feature is useful for resonators or crystals with lower accuracy (for example, wide variations over temperature), but it might not be needed when using more accurate crystals/resonators.

This feature can be enabled and configured using the following definitions in USS_App_userConfig.h:

```
#define USS_APP_RESONATOR_CALIBRATE_ENABLE
#define USS_APP_RESONATOR_CALIBRATE_INTERVAL (30)
```

When defined, USS_APP_RESONATOR_CALIBRATE_ENABLE enables this feature. The periodicity given in USS measurements is defined by USS_APP_RESONATOR_CALIBRATE_INTERVAL.

For example, an interval of 30 means that the calibration will be performed every 30 measurements (30 seconds if taking one measurement per second).

When enabled, the calibration interval can be modified using the following structure member:

```
USS_App_userConfig.u16ResonatorCalibrateInterval
```

The demo application uses User Param #8 to configure the resonator calibration interval as explained in MSP430FR6043 ultrasonic design center user's guide.

3.2.3 AGC Calibration

During runtime, the amplitude of the captured ultrasonic signal can vary depending on different factors, such as temperature, gas composition, or transducer aging.

The application can be enabled to calculate the optimal gain setting periodically to compensate for this variation in amplitude.

This feature can be enabled and configured using the following definitions in USS_App_userConfig.h:

```
#define USS_APP_AGC_CALIBRATE_ENABLE
#define USS_APP_AGC_CALIBRATE_INTERVAL (10)
```

When defined, USS_APP_AGC_CALIBRATE_ENABLE enables this feature. The periodicity – given in USS measurements- is defined by USS_APP_AGC_CALIBRATE_INTERVAL.

For example, an interval of 10 means that the calibration will be performed every 10 measurements (10 seconds if taking one measurement per second).

When enabled, the calibration interval can be modified using the following structure member:

```
USS_App_userConfig.u16AGCCalibrateInterval
```

The demo application uses User Param #8 to configure the resonator calibration interval as explained in MSP430FR6043 ultrasonic design center user's guide.

3.2.4 DC Offset Estimation and Cancellation

The received ADC signal might include an offset due to factors such as external hardware or PGA settings.

The application can be configured to estimate and cancel this offset before processing the signal in the algorithms layer.

This feature can be enabled and configured using the following definitions in USS_App_userConfig.h:

```
#define USS_APP_DC_OFFSET_CANCELLATION_ENABLE
#define USS_APP_DC_OFFSET_ESTIMATE_INTERVAL (10)
```

When defined, USS_APP_DC_OFFSET_CANCELLATION_ENABLE enables this feature. The periodicity – given in USS measurements- is defined by USS_APP_DC_OFFSET_ESTIMATE_INTERVAL.

For example, an interval of 10 means that the calibration is performed every 10 measurements (10 seconds if taking one measurement per second).

When enabled, the calibration interval can be modified using the following structure member:

```
USS_App_userConfig.u16DCOffsetEstimateInterval
```

The demo application uses User Param #8 to configure the resonator calibration interval as explained in MSP430FR6043 ultrasonic design center user's guide.

3.3 Customizing the System Hardware

The application example was developed and tested using the MSP430FR6043 EVM; however, developers can customize the software to their own hardware.

As explained in Section 1.4, hardware interaction from the application is implemented in the HAL layer.

Table 3-2 lists the most relevant HAL functions and their corresponding source files.

Table 3-2. Functions in Application HAL

Function	File	Description	Possible Uses
hal_system_Init	hal_system.c	Initializes the system	• Change the order of initialization of the system • Add initialization of additional peripherals
hal_system_GPIOInit	hal_system.c	Defines the default configuration and state of GPIOs	• Change the pins used for communication purposes (for example, LEDs or buttons) • Set default state of all pins to avoid floating inputs • Add functionality to other pins not used by the application.
hal_system_ClockInit	hal_system.c	Defines the default configuration of the system clocks including the low-frequency crystal	• Change the ACLK, MCLK, SMCLK frequencies for CPU or peripherals • Change the configuration of the low frequency crystal (LFXT) • Enable a high-frequency crystal (HFXT)
hal_system_WatchdogInit	hal_system.c	Sets default configuration of watchdog	• Change watchdog timeout • Disable or enable watchdog by default
hal_adc_init	hal_adc.c	Configures the ADC to take temperature and voltage supply measurements	• Change ADC channel used for voltage supply measurement • Enable ADC to read other channels
hal_lcd_Init	hal_lcd.c	Defines the default configuration of the LCD	• Enable or disable the LCD charge pump • Change the I/Os used for LCD • Enable or disable the external resistor ladder for contrast control

The Ultrasonic SW Library defines a separate USS_HAL layer defining the interaction with hardware (see Table 3-3).

Table 3-3. Hardware Definitions in USS HAL Layer

Definition	File	Description
USSSWLIB_TIMER_BASE_ADDRESS	USS_Lib_HAL.h	Defines the timer used by the USS SW Library
USS_AFE_RX_SELx_PORT USS_AFE_RX_SEL1_PIN USS_AFE_RX_SEL2_PIN	USS_Lib_HAL.h	Defines the GPIOs used to control RxSel lines.
USS_AFE_RXEN_PORT USS_AFE_AMP_PIN	USS_Lib_HAL.h	Defines the GPIO used to control the RXEn line of the external amplifier.
USS_AFE_RXPWR_PORT USS_AFE_RXPWR_PIN	USS_Lib_HAL.h	Defines the GPIO used to control RxPwr.
USS_AFE_TXPWR_PORT USS_AFE_TXPWR_PIN	USS_Lib_HAL.h	Defines the GPIO used to control TxPwr.

3.4 Customizing Data Processing

The USS SW Library includes proprietary algorithms which can be used to process captured ADC waveforms and obtain flow rate information. While developers are encouraged to use the algorithms implemented in the library, it is also possible to implement custom algorithms to add any proprietary IP or to optimize and improve the performance of the system.

This can be achieved in several ways, including:

- Processing raw ADC waveform:

  The USS subsystem uses the sigma-delta high-speed ADC (SDHS) to sample the signal coming from the ultrasonic transducers on two directions (upstream and downstream). These raw signals are used by the USS algorithms to calculate the absolute and differential time-of-flight but developers can process the data using custom algorithms.

  The functions USS_startLowPowerUltrasonicCapture and USS_startUltrasonicMeasurement perform a capture according to the predefined USS configuration and generate upstream and downstream ADC waveforms as a result.

  These waveforms have a size of:

  ```
  gUssSWConfig.captureConfig.sampleSize
  ```

  The upstream and downstream waveforms are stored in gUssSWConfig.captureConfig.pCapturesBuffer but they are easily accessible using the following USS library functions:

  ```
  uint8_t* pUPSCap = (uint8_t*)(USS_getUPSPtr());
  uint8_t* pDNSCap = (uint8_t*)(USS_getDNSPtr());
  ```

- Perform custom calculation of flow rate:

  While the USS algorithms can calculate the flow rate, developers have the flexibility to perform their own calculation to compensate for things like noise or temperature.

  This is achieved by using the output generated by the function USS_runAlgorithms. This function uses the upstream and downstream ADC waveforms as an input and it calculates the following information:
- Differential time-of-flight in seconds:

  ```
  USS_Algorithms_Results.deltaTOF
  ```

- Absolute time-of-flight of upstream signal in seconds:

  ```
  USS_Algorithms_Results.totalTOF_UPS
  ```

- Absolute time-of-flight of downstream signal in seconds:

  ```
  USS_Algorithms_Results.totalTOF_DNS
  ```

- Volume flow rate (in units defined by meter constant):

  ```
  USS_Algorithms_Results.volumeFlowRate
  ```

The volume flow rate is calculated using Equation 1.

$$v = \frac{L}{2} \times \left(\frac{\Delta t}{T_{12} T_{21}} \right) \quad (1)$$

where

- v = Volume flow rate
- L = Propagation length of pipe. Defines the meter constant.
- Δt = Differential time-of-flight (TOF)
- T12 = Upstream absolute TOF
- T21 = Downstream absolute TOF

By having direct access to the data shown above, developers can implement custom algorithms to compensate for other factors and calculate the flow rate.

4 Resources Used by the Application

4.1 I/Os

Table 4-1 lists the I/Os used by the application in EVM430-FR6043 revision E1.

Table 4-2 lists the I/Os used by the application in EVM430-FR6043 revision E2.

Table 4-1. I/Os Used by the Application in EVM430-FR6043-E1

Port	I/O	Initialization	Use by Application
P1	P1.0	Output low	TxPwr. Optional signal to control transmit power. Not used for EVM.
	P1.1	Output low	Tx5vEn. Not used. Optional signal to control 5V level shifter.
	P1.2	Output low	Not used. Can be used as TXD UART for communication using eZ-FET back-channel, communication with Design Center, or J7/J8 connectors.
	P1.3	Output low	Not used. Can be used as UART RXD for communication using eZ-FET back-channel, communication with Design Center, or J7/J8 connectors.
	P1.4	Input	IRQ for communication with Design Center
	P1.5	Input with pull-up	Select button
	P1.6	I^2C SDA	I^2C communication with Design Center
	P1.7	I^2C SCL	I^2C communication with Design Center
P2	P2.0	LCD	LCD segment 19
	P2.1	LCD	LCD segment 18
	P2.2	Output low	RxPwr. Controls power to external RX circuitry.
	P2.3	Output low	TxSel. Not used. Optional control of transmit switch.
	P2.4	LCD	LCD segment 24
	P2.5	LCD	LCD segment 21
	P2.6	LCD	LCD segment 23
	P2.7	LCD	LCD segment 22
P3	P3.0	LCD	LCD segment 20
	P3.1	Input with pull-up	Left button. Can also be used as MTIF_OUT_IN, or J7/J8 connectors.
	P3.2	LCD	LCD segment 28
	P3.3	LCD	LCD segment 25
	P3.4	LCD	LCD segment 27
	P3.5	LCD	LCD common 3
	P3.6	Output low	RxSel1. Used to control RX switches.
	P3.7	Output low	RxSel2. Used to control RX switches.

Table 4-1. I/Os Used by the Application in EVM430-FR6043-E1 (continued)

Port	I/O	Initialization	Use by Application
P4	P4.0	Input with pull-up	Right button. Can also be used as MTIF_PIN_EN, or J7/J8 connectors.
	P4.1	LCD	LCD segment 15
	P4.2	LCD	LCD segment 14
	P4.3	LCD	LCD segment 13
	P4.4	LCD	LCD segment 12
	P4.5	LCD	LCD segment 11
	P4.6	LCD	LCD segment 10
	P4.7	LCD	LCD segment 9
P5	P5.0	LCD	LCD segment 8
	P5.1	LCD	LCD segment 7
	P5.2	LCD	LCD segment 6
	P5.3	LCD	LCD segment 5
	P5.4	LCD	LCD segment 4
	P5.5	LCD	LCD segment 3
	P5.6	LCD	LCD segment 2
	P5.7	LCD	LCD segment 1
P6	P6.0	LCD	LCD segment 0
	P6.1	LCD	LCD resistor ladder R03
	P6.2	LCD	LCD resistor ladder R13
	P6.3	LCD	LCD resistor ladder R23
	P6.4	LCD	LCD common 0
	P6.5	LCD	LCD common 1
	P6.6	LCD	LCD common 2
	P6.7	Output low	RxEn. Controls external amplifier.
P7	P7.0	Output low	XPB0. Not used.
PJ	PJ.0	Output low	Not used. Can be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.1	Output low	LED2 (Green). Optionally, it be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.2	Output low	Not used. Can be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.3	Output low	LED1 (Red). Optionally, it be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.4	LFXT	Low-frequency crystal input
	PJ.5	LFXT	Low-frequency crystal output
	PJ.6	Output low	Not used. Can be used for HFXT.
	PJ.7	Output low	Not used. Can be used for HFXT.
Not general purpose	TEST/SBWTCK	-	Can be used by JTAG/SBW.
	RST/NMI/SBWTDIO	-	Reset. Can be used by JTAG/SBW.
	R33/LCDCAP	-	LCD resistor ladder R33
USS	USSXTIN	-	USS crystal input
	USSXTOUT	-	USS crystal output
	CH0_IN	-	Input from transducer 1
	CH0_OUT	-	Output to transducer 1
	CH1_IN	-	Input from transducer 2
	CH1_OUT	-	Output to transducer 2

Table 4-2. I/Os Used by the Application in EVM430-FR6043-E2

Port	I/O	Initialization	Use by Application
P1	P1.0	Output low	TxPwr. Signal to control transmit power.
	P1.1	Output low	Tx5vEn. Not used. Optional signal to control 5V level shifter.
	P1.2	Output low	Not used. Can be used as TXD UART for communication using eZ-FET back-channel, communication with Design Center, or J7/J8 connectors.
	P1.3	Output low	Not used. Can be used as UART RXD for communication using eZ-FET back-channel, communication with Design Center, or J7/J8 connectors.
	P1.4	Input	IRQ for communication with Design Center
	P1.5	Output low	LED2.
	P1.6	I^2C SDA	I^2C communication with Design Center
	P1.7	I^2C SCL	I^2C communication with Design Center
P2	P2.0	LCD	LCD segment 19
	P2.1	LCD	LCD segment 18
	P2.2	Output low	RxPwr. Controls power to external RX circuitry.
	P2.3	Output low	Not used.
	P2.4	LCD	LCD segment 24
	P2.5	LCD	LCD segment 21
	P2.6	LCD	LCD segment 23
	P2.7	LCD	LCD segment 22
P3	P3.0	LCD	LCD segment 20
	P3.1	Output low	Not used. Can be used as MTIF_OUT_IN, or J7/J8 connectors.
	P3.2	LCD	LCD segment 28
	P3.3	LCD	LCD segment 25
	P3.4	LCD	LCD segment 27
	P3.5	LCD	LCD common 3
	P3.6	Output low	RxSel1. Used to control RX switches.
	P3.7	Output low	RxSel2. Used to control RX switches.
P4	P4.0	Output low	Not used. Can be used as MTIF_PIN_EN, or J7/J8 connectors.
	P4.1	LCD	LCD segment 15
	P4.2	LCD	LCD segment 14
	P4.3	LCD	LCD segment 13
	P4.4	LCD	LCD segment 12
	P4.5	LCD	LCD segment 11
	P4.6	LCD	LCD segment 10
	P4.7	LCD	LCD segment 9
P5	P5.0	LCD	LCD segment 8
	P5.1	LCD	LCD segment 7
	P5.2	LCD	LCD segment 6
	P5.3	LCD	LCD segment 5
	P5.4	LCD	LCD segment 4
	P5.5	LCD	LCD segment 3
	P5.6	LCD	LCD segment 2
	P5.7	LCD	LCD segment 1

Table 4-2. I/Os Used by the Application in EVM430-FR6043-E2 (continued)

Port	I/O	Initialization	Use by Application
P6	P6.0	LCD	LCD segment 0
	P6.1	LCD	LCD resistor ladder R03
	P6.2	LCD	LCD resistor ladder R13
	P6.3	LCD	LCD resistor ladder R23
	P6.4	LCD	LCD common 0
	P6.5	LCD	LCD common 1
	P6.6	LCD	LCD common 2
	P6.7	Output low	RxEn. Controls external amplifier.
P7	P7.0	Output low	XPB0. Not used.
PJ	PJ.0	Input with pull-up	Left button. Can be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.1	Input with pull-up	Right button. Can be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.2	Input with pull-up	Select button. Can be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.3	Output low	LED1. Can be used for 4-wire JTAG, or J7/J8 connectors.
	PJ.4	LFXT	Low-frequency crystal input
	PJ.5	LFXT	Low-frequency crystal output
	PJ.6	Output low	Not used. Can be used for HFXT.
	PJ.7	Output low	Not used. Can be used for HFXT.
Not general purpose	TEST/SBWTCK	-	Can be used by JTAG/SBW.
	RST/NMI/SBWTDIO	-	Reset. Can be used by JTAG/SBW.
	R33/LCDCAP	-	LCD resistor ladder R33
USS	USSXTIN	-	USS crystal input
	USSXTOUT	-	USS crystal output
	CH0_IN	-	Input from transducer 1
	CH0_OUT	-	Output to transducer 1
	CH1_IN	-	Input from transducer 2
	CH1_OUT	-	Output to transducer 2

4.2 Peripherals

Table 4-3 lists the peripherals and modules used by the application.

Table 4-3. Peripherals and Modules Used by the Application

Module	Initialization	Use in Application
USS	Used and initialized by USS SW Library	The application uses and initializes the 6 USS submodules (UUPS, HSPLL, ASQ, PPG, PGA, SDHS) to implement an efficient measurement of ultrasonic signals
CS	LFXT enabled at 32.768 kHz Enable DCO at 8 or 16 MHz ACLK = LFXT SMCLK = DCO MCLK = DCO	Configure clocks used by application.
WDT	Watchdog enabled using VLO (approximately 10 kHz) with a timeout of apprximately 3.2 seconds	Configure and feed watchdog
ADC12	Initialized to read temperature sensor and external voltage. Uses internal MODOSC	Read internal temperature sensor and external supply voltage
LCD_C	Using 4-MUX mode. Internal charge pump disabled. External resistor ladder enabled for contrast control.	Display information to the user
eUSCI_B0	I^2C slave with address 0x0A	Communication with Design Center
TA2	Used and initialized by USS SW Library	Time-keeping used by USS SW library
TA1	Used and initialized by Design Center drivers	Design Center communication timeouts
TA0	Used and initialized by USS SW Library	Control external amplifier
LEA	Used and initialized by USS SW Library	Efficient implementation of algorithms
DMA2	Used and initialized by USS SW Library	Implement multi-tone generation
DMA3	Used and initialized by USS SW Library	Implement multi-tone generation

4.3 Memory Footprint

Table 4-4 summarizes the memory footprint of the application.

Table 4-4. Application Memory Footprint

Memory Area	Description	Library(2) (1)	Total demo(1)
Code	Code used by demo application.	23212 bytes	46954 bytes
Const	Nonvolatile constants.	400 bytes	1204 bytes
Data	Read-Write variables including persistent variables.	11631 bytes	14326 bytes

(1) Using CCS 8.0 with optimization level 3 using EVM_E2_AFE3v3 configuration
(2) Includes ussSwLib source, USS_HAL, USS_Config, and USS Software library. Excludes application files and common/shared files (e.g. IQMath, CCS library).

Table 4-5 summarizes the RAM usage of the application.

Table 4-5. Application RAM Usage on MSP430FR6043

Memory Area(1)	Description	Available on MSP430FR6043	Total Demo(2)
RAM	RAM not shared with LEA	4KB	3829 bytes
LEA RAM	RAM shared with LEA	8KB	5703 bytes

(1) The MSP430FR6043 MCU has 12KB of total RAM. 8KB are shared with LEA
(2) Using CCS 8.0 with optimization level 3 using EVM_E2_AFE3v3 configuration

5 Terminology

AbsTOF	Absolute time-of-flight
APIs	Application programming interface
dTOF	Delta (differential) time-of-flight
GUI	Graphical user interface
HAL	Hardware abstraction layer
HMI	Human-machine interface
LEA	Low-energy accelerator
TOF	Time-of-flight
USS	Ultrasonic sensing module
USSSWLib	Ultrasonic sensing software library

6 References

1. MSP430FR6043-Based Ultrasonic Gas Meter Quick Start Guide
2. MSP430FR6043 Ultrasonic Design Center User's Guide
3. EVM430-FR6043 Hardware Guide
4. MSP430FR604x, MSP430FR504x Ultrasonic Sensing MSP430™ Microcontrollers for Gas and Water Flow Metering Applications Data Sheet
5. MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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