Application Software (SW) Architecture for MSP430FR6047-Based Ultrasonic Water Flow Meter

Luis Reynoso *MSP Systems Applications*

ABSTRACT

This document explains the software architecture used in the implementation of an ultrasonic water flow meter with MSP430FR6047. It includes a description of 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.

Contents

List of Figures

List of Tables

Trademarks

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

1 Software Architecture

The MSP430FR6047 ultrasonic water flow meter application is a portable and modular software designed to guide developers in the implementation and customization of their own water meter solutions.

Figure 1 shows the software architecture.

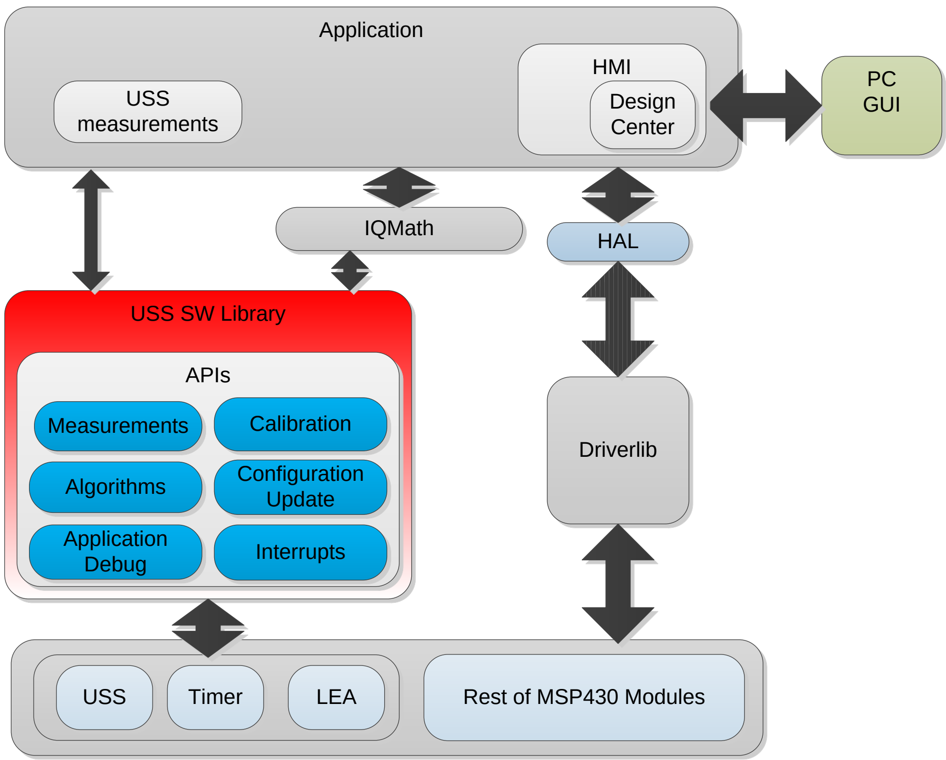

Figure 1. Software Architecture

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 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 2 shows the flow diagram of the application.

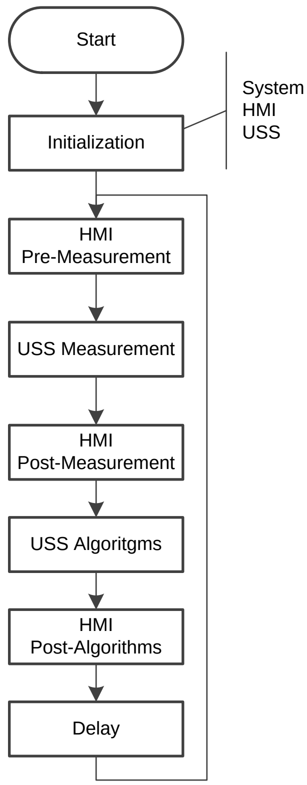

Figure 2. Application Flow Diagram

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.

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 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 Design Center Graphical User Interface (DCGUI) user guide.

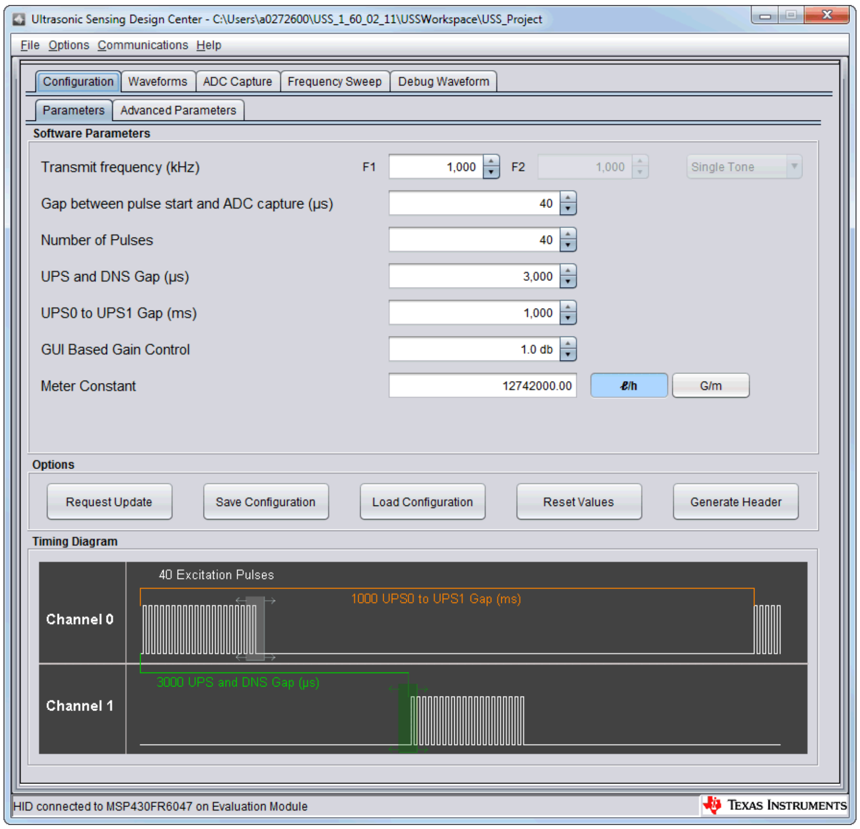

Figure 3. Ultrasonic Sensing Design Center

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 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 MSP430FR6047, perform ultrasonic captures, and process the signal received from the transducers to calculate output information such as the time-of-flight (TOF) and water 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.

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.

The ultrasonic software library used by the application is included in CCS and IAR library format.

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 MSP430FR6047's ADC12 module.
- LCD: implements functions to configure and control the LCD_C module in MSP430FR6047.
- 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

Figure 4 shows the file structure of the MSP430FR6047 ultrasonic water flow meter application.

```
UltrasonicWaterFR6047>
|
+---driverlib
|   \---MSP430FR5xx_6xx
|
+---examples
|   +---common
|   |   +---DesignCenter
|   |   +---gui
|   |   \---hal
|   |       \---fr6047EVM
|   |
|   +---mtr_gui_config
|   |
|   \---USSLib_GUI_Demo
|       +---CCS
|       +---IAR
|       +---USS_Config
|       \---USSLibGUIApp
|
+---image
|
+---include
|
\---lib
    +---IQMathLibrary
    \---USS
```

Figure 4. File Structure

The most relevant folders are:

- *driverlib*: Source code of MSP DriverLib files supporting MSP430FR6047.
- *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 3.2.
  - *mtr_gui_config*: Includes files to pre-configure the USS subsystem with Design Center to support some sample meters and transducers.
  - *USSLib_GUI_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. These files can be modified to set the default configuration of the system. For more information, see Section 3.1.
    - *USSLibGUIApp*: Source code for main functionality of USS water meter application.
- *image*: Prebuilt binaries of the application.
- *include*: Header files for IQMath and USS SW Library.
- *lib:* Library files for IQMath and USS SW Library.

2.1 Application Layer Files

2.1.1 main.c

Location

~\examples\USSLib_GUI_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 water meter application.

2.1.2 system_pre_init.c

Location

~\examples\USSLib_GUI_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_userConfig.c, USS_userConfig.h

Location

~\examples\USSLib_GUI_Demo\USS_Config\USS_userConfig.c

~\examples\USSLib_GUI_Demo\USS_Config\USS_userConfig.h

Description

Default initialization 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 USSLibGUIApp.c, USSLibGUIApp.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\USSLibGUIApp.c

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_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 water meter application.
- `void USSLibGUIApp_Engine(void)`

  Main loop of USS water 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.

2.1.5 masterIncludes.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\masterIncludes.h

Description

Common definitions used by application files.

2.1.6 hmi.c, hmi.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\hmiDC\hmi.c

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_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 subsystem 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.

2.1.7 lcd_statemachine.c, lcd_statemachine.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\lcd_statemachineDC\lcd_statemachine.c

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_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.

2.1.8 mathematics.c, mathematics.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\mathematicsDC\mathematics.c

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\mathematicsDC\mathematics.h

Description

Implements calculations used by the HMI and LCD display.

2.1.9 results.c, results.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\resultsDC\results.c

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_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.

2.1.10 testing.c, testing.h

Location

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\testingDC\testing.c

~\examples\USSLib_GUI_Demo\USSLibGUIApp\fr6047_USS_app\testingDC\testing.h

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 the 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 I2C slave port definition for the I2C slave driver. Used only when I2C 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 I2C) 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, see 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, see 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\CCS*.*

~\lib\USS\IAR*.*

Description

Library files for IAR and CCS.

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

2.5 HAL Files

2.5.1 hal.h

Location

~\examples\common\hal\fr6047EVM\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\fr6047EVM\hal_adc.c

~\examples\common\hal\fr6047EVM\hal_adc.h

Description

HAL files for ADC implementing temperature sensor and supply voltage measurements using the internal ADC12 in MSP430FR6047.

Relevant functions

- `void hal_adc_init(void)`

  Initializes the ADC12 in MSP430FR6047 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\fr6047EVM\hal_lcd.c

~\examples\common\hal\fr6047EVM\hal_lcd.h

Description

HAL files supporting the FH-1138P segmented LCD using the LCD_C module on MSP430FR6047.

Relevant functions

- `void hal_lcd_Init(void)`

  Initializes the LCDC_C in MSP430FR6047 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\fr6047EVM\hal_system.c

~\examples\common\hal\fr6047EVM\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\fr6047EVM\hal_uart.c

~\examples\common\hal\fr6047EVM\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 MSP430FR6047 EVM.

2.6 DriverLib Files

Location

~\driverlib\MSP430FR5xx_6xx

Description

DriverLib source and header files supporting the MSP430FR6047.

For more details about DriverLib, see MSPDRIVERLIB.

2.7 IDE Files

2.7.1 CCS Project File

Location

~\examples\USSLib_GUI_Demo\CCS\USSLib_GUI_Demo.projectspec

Description

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

The project contains three configurations:

- LPM

  Default configuration. Uses *USS_startLowPowerUltrasonicCapture* to perform USS captures while going to the lowest-power mode possible between upstream and downsteam. The device also goes to LPM3 between measurements.
- Disable_LPM

  Uses *USS_startUltrasonicMeasurement* to perform USS captures and stays in LPM0 between upstream and downstream. The device also goes to LPM0 between measurements.
- LPM_LEARAM

  In addition to the same functionality as the LPM configuration, the device executes LEA functions from RAM.

2.7.2 CCS Linker File

Location

~\examples\USSLib_GUI_Demo\CCS\lnk_msp430fr6047.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\USSLib_GUI_Demo\IAR\USSLib_GUI_Demo.*

Description

IAR workspace and project files.

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

The project contains two configurations:

- LPM

  Default configuration. Uses *USS_startLowPowerUltrasonicCapture* to perform USS captures while going to the lowest-power mode possible between upstream and downsteam. The device also goes to LPM3 between measurements.
- Disable_LPM

  Uses *USS_startUltrasonicMeasurement* to perform USS captures and stays in LPM0 between upstream and downstream. The device also goes to LPM0 between measurements.

2.7.4 IAR Linker File

Location

~\examples\USSLib_GUI_Demo\IAR\lnk430fr6047.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 water 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, making hardware modifications, and modifying algorithms and processing of data.

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:

```
Root>examples>USSLib_GUI_Demo>USS_Config>USS_userConfig.c
Root>examples>USSLib_GUI_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 listed in Table 1.

Table 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	.numOfExcitationPulses .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	.absTOFInterval .dcOffset .enableWindowing

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 excitation pulses sent to the transducer is defined as follows:

USS_userConfig.c:

```
__persistent USS_Measurement_Configuration ussMeasurementConfig =
{
  .ch0DriveStrength = USS_CH0_DRIVE_STRENGHT,
  .ch1DriveStrength = USS_CH1_DRIVE_STRENGHT,
  .pauseState = USS_PAUSE_STATE,
  .pulsePolarity = USS_PULSE_POLARITY,
  .pulseHighPhasePeriod= USS_PULSE_HIGH_PHASE_PERIOD,
  .pulseLowPhasePeriod = USS_PULSE_LOW_PHASE_PERIOD,
  .numOfExcitationPulses = USS_NUM_OF_EXCITATION_PULSES,
  .numOfStopPulses = USS_NUM_OF_STOP_PULSES,
  .sequenceSelection = USS_SEQUENCE_SELECTION,
  .eofSequenceState = USS_EOF_SEQUENCE_SELECTION,
  .startPPGCount = USS_START_PPG_COUNT,
  .turnOnADCCount = (uint_least16_t) USS_TURN_ON_ADC_COUNT,
  .startPGAandINBiasCount = USS_PGA_IN_BIAS_COUNT,
  .startADCsamplingCount = USS_ADC_SAMP_COUNT,
  .restartCaptureCount = (uint_least16_t)USS_RESTART_CAP_COUNT,
  .captureTimeOutCount = USS_TIME_OUT_COUNT,
  .ulpBiasDelay = USS_ULP_BIAS_DELAY,
  .biasImpedance = USS_BIAS_IMPEDANCE,
  .muxChargePumpMode = USS_MUX_CHARGE_PUMP_MODE,
  .restartLowPowerCaptureCount = (uint_least16_t)USS_LOW_POWER_RESTART_CAP_COUNT,
};
```

USS_userConfig.h:

```
#define USS_PULSE_DUTYPERCENT               0.50
#define USS_NUM_OF_EXCITATION_PULSES        20
#define USS_NUM_OF_STOP_PULSES              0
```

3.2 Customizing the Hardware

The application example was developed and tested using the MSP430FR6047 EVM; however, developers can customize the software to their own hardware.

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

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

Table 2. HAL Functions

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, LEDs, buttons, and so on. • 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)

Table 2. HAL Functions (continued)

Function	File	Description	Possible Uses
hal_system_WatchdogInit	hal_system.c	Sets default configuration of watchdog.	• Change watchdog time-out • Disable or enable watchdog by default
hal_adc_init	hal_adc.c	Preconfigures 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

3.3 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:

1. 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* or *USS_startUltrasonicMeasurement* perform a capture according to the predefined USS configuration and will 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());
   ```

   By having direct access to the ADC sampled waveforms, developers can implement their own algorithms to process the data as desired.

2. 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)
- T_{12} = Upstream absolute TOF
- T_{21} = Downstream absolute TOF

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

4 Resources Used by the Application

4.1 I/Os

Table 3 lists the device I/Os that are used in the application.

Table 3. I/Os Used by the Application

Port	I/O	Initialization	Use by application
P1	P1.0	Output low	LED1
	P1.1	Output low	LED2
	P1.2	Output low	Not used. Can be used as TXD for communication using back-channel UART.
	P1.3	Output low	Not used. Can be used as RXD for communication using back-channel UART.
	P1.4	Output low	LED3
	P1.5	Input	IRQ for communication with Design Center.
	P1.6	I2C SDA	I^2C communication with Design Center.
	P1.7	I2C SCL	I^2C communication with Design Center.
P2	P2.0	Output low	Not used. Can be used as TXD for UART communication with design center.
	P2.1	Output low	Not used. Can be used as RXD for UART communication with design center.
	P2.2	Input	Enables measurement of VCC when set as output low.
	P2.3	Analog Input	Used to measure external VCC using voltage divider.
	P2.4	Output low	Not used.
	P2.5	Input with pullup	Left button.
	P2.6	Input with pullup	Up button.
	P2.7	LCD	LCD segment 21
P3	P3.0	Input with pullup	Down button
	P3.1	Input with pullup	Select button
	P3.2	Input with pullup	Right button
	P3.3	LCD	LCD segment 26
	P3.4	LCD	LCD segment 25
	P3.5	LCD	LCD segment 24
	P3.6	LCD	LCD segment 23
	P3.7	LCD	LCD segment 22

Table 3. I/Os Used by the Application (continued)

Port	I/O	Initialization	Use by application
P4	P4.0	LCD	LCD segment 16
	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	LCD	LCD common 3
P7	P7.0	Output low	Not used. Accessible using J5/J6 connectors.
	P7.1	Output low	Not used. Accessible using J5/J6 connectors.
	P7.2	Output low	Not used. Accessible using J5/J6 connectors.
	P7.3	Output low	Not used. Accessible using J5/J6 connectors.
	P7.4	Output low	Not used. Accessible using J5/J6 connectors.
	P7.5	Output low	Not used. Accessible using J5/J6 connectors.
	P7.6	Output low	Not used. Accessible using J5/J6 connectors.
	P7.7	Output low	Not used. Accessible using J5/J6 connectors.
P8	P8.0	Output low	Not used. Accessible using J5/J6 connectors.
	P8.1	Output low	Not used. Accessible using J5/J6 connectors.
	P8.2	Output low	Not used. Accessible using J5/J6 connectors.
	P8.3	Output low	Not used. Accessible using J5/J6 connectors.
	P8.4	Output low	Not used. Accessible using TP3 testpoint.
	P8.5	Output low	Not used. Accessible using TP2 testpoint.
	P8.6	Output low	Not used. Accessible using J5/J6 connectors.
	P8.7	Output low	Not used. Accessible using J5/J6 connectors.
P9	P9.0	LCD	LCD segment 20
	P9.1	LCD	LCD segment 19
	P9.2	LCD	LCD segment 18
	P9.3	LCD	LCD segment 17

Table 3. I/Os Used by the Application (continued)

Port	I/O	Initialization	Use by application
PJ	PJ.0	Output low	Not used. Can be used by JTAG.
	PJ.1	Output low	Not used. Can be used by JTAG.
	PJ.2	Output low	Not used. Can be used by JTAG.
	PJ.3	Output low	Not used. Can be used by JTAG.
	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 and Modules

Table 4 summarizes the device modules used by the application.

Table 4. Peripherals and Modules Used by the Application

Module	Initialization	Use by Application
USS	Used and initialized by USS SW Library	The application uses and initializes the six USS submodules (UUPS, HSPLL, ASQ, PPG, PGA, SDHS) to implement an efficient measurement of ultrasonic signals.
UCS	LFXT enabled at 32.768 kHz. Enable DCO at 8 MHz. ACLK = LFXT SMCLK = DCO MCLK = DCO	Configure clocks used by application
WDT	Watchdog enabled using VLO (approximately 10 kHz) with a time-out of approximately 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	Delays used by USS SW library
TA1	Used and initialized by Design Center drivers	Design Center communication time-outs
LEA	Used and initialized by USS SW Library	Efficient implementation of algorithms

4.3 *Memory Footprint*

Table 5 summarizes the memory footprint of the application.

Table 5. Application Memory Footprint

Memory Area	Description	CCS[1]	IAR[2]
Code	Total code used by application	43KB	37.2KB
Const	Nonvolatile constants and persistent variables used by application	3.87KB	3.65KB
Data	Volatile variables used by the application	5.96KB	5.75KB

(1) Using "LPM" configuration with CCS 6.2.0 with optimization level 3
(2) Using "LPM" configuration with IAR 6.50.2 with optimization level Medium

Table 6 shows the RAM usage of the application.

Table 6. Application RAM Usage on MSP430FR6047

Memory Area[1]	Description	Available on MSP430FR6047	Used by Application	
			CCS[2]	IAR[3]
RAM[4]	RAM not shared with LEA	4KB	3672 bytes	3252 bytes
LEA RAM	RAM shared with LEA	4KB	2748 bytes	2748 bytes

(1) The MSP430FR6047 has 8KB of total RAM. 4KB are shared with LEA.
(2) Using "LPM" configuration with CCS 7.2.0 with optimization level 3
(3) Using "LPM" configuration with IAR 7.10.5 with optimization level Medium
(4) Developers can also place variables in FRAM if needed.

5 References

1. MSP430FR6047 Ultrasonic Sensing Design Center Quick Start Guide
2. Quick Start Guide for MSP430FR6047-Based Ultrasonic Water Flow Meter
3. EVM430-FR6047 Hardware Guide
4. MSP430FR6047 and Ultrasonic Software Based Water Flow Meter Measurement Results

Revision History

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

Changes from June 2, 2017 to February 2, 2018 **Page**

IMPORTANT NOTICE AND DISCLAIMER

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