STPM32F407 and STPM3x sampling firmware

Introduction

The STPM3x is a family of ASSP devices for high accuracy energy metering applications, integrating up to four sigma-delta ADC and a hardwired DSP.

From the 24-bit samples of voltage and current provided by the sigma-delta ADC, the device DSP implements most of the metrology calculations, as voltage and current RMS, active (wideband and fundamental), reactive and apparent power and energy.

The ADC samples of wideband and fundamental components of voltage and current are available at the 7.8125 kHz sampling rate for further application post processing, if required. For example, the RMS of the signal fundamental component and the THD are not calculated by the DSP.

This document describes a simple STPM32F407 firmware that reads the STPM3x ADC data synchronously to the device sampling clock and calculates the RMS values (both wideband and fundamental) and the THD of the voltage and current signals.

The firmware has been developed for the STM32F4DISCOVERY board and can be interfaced to any EVALSTPM3x.

The firmware is intended as a basic metrology example and integrates the already existing firmware for STPM3x devices.

Table 1. Applicable products

Type	Reference products
Metrology	STPM3x
Microcontroller	STM32F407VG

For more information, refer to the STM32F4DISCOVERY discovery kit user manual, to the EVALSTPM3x boards user manual, and to the STPM3x datasheet.

1 Getting started

1.1 System requirements

To run the application on the board, the minimum requirements are as follows:

- Host PC with IDE “IAR Embedded Workbench for Arm” version 8.5.
- 'USB Type-A to micro-B' cable, used to power and connect the STM32F4DISCOVERY board to a host PC for debugging and programming.
- A 3.3 DC voltage supply to power on the EVALSTPM3x board.

1.2 Hardware connection

To run the firmware, the following steps must be followed:

- Connect the STM32F4DISCOVERY board to a PC with a 'USB Type-A to micro-B' cable through USB connector CN1 to power the board and connect to the ST-LINK/V2-A onboard.
- Connect the two boards as follows:

Table 2. Hardware connection

STM32F4DISCOVERY	EVALSTPM3x	Function
GND	SPI conn. PIN 3	GND
PB14	SPI conn. PIN 4	MISO
PB15	SPI conn. PIN 2	MOSI
PB13	SPI conn. PIN 6	SCL
PA1	Digital conn. PIN 8	CLKOUT

- Connect the EVALSTPM3x board to a 3.3 V supply and to a load. For this purpose, refer to UM1748: “EVALSTPM34, EVALSTPM33, EVALSTPM32 evaluation board”.

2 Firmware description

The STPM3x devices sample voltage and current data at 7.8125 kHz. At this rate, as soon as a new ADC data is available, a pulse is output on the CLKOUT pin of the device.

To synchronize the samples reading with the ADC sampling rate, the firmware uses the pin PA1 of the MCU connected to the CLKOUT pin to trigger an interrupt.

The interrupt routine sets a flag that starts the SPI reading through DMA.

Four 5-byte frames are sent to the device, constituted by 4-byte data and CRC byte, to:

- Set the latch bits into the STPM3x configuration registers. This command updates the SPI registers with the latest ADC data.
- Set the reading address to receive the four data registers of wideband and fundamental ADC samples of voltage and current. The channel (primary or secondary) to read from can be set by changing the aTxBuffer[] content in the main.c file.

Overall, 625 samples are taken from the device (four periods at 50 Hz), configured in the main.h file. It is suggested to modify the number of samples according to the line frequency to get a precise RMS and THD calculation.

To manage received data, the Signal_HandleTypeDef structured data type is defined in the main.h file, containing, for each signal, buffers to store ADC wideband and fundamental samples, plus RMS and THD data.

A timer is configured to calculate RMS and THD at the desired rate, not to overload the MCU if a real-time calculation of these parameters is not required.

When all the data are read and if the timer interval has elapsed, the RMS and THD of the voltage and current channels are calculated in the function CalculateTHD() in the main.c file, using the functions of the Arm library defined in arm_math.h.

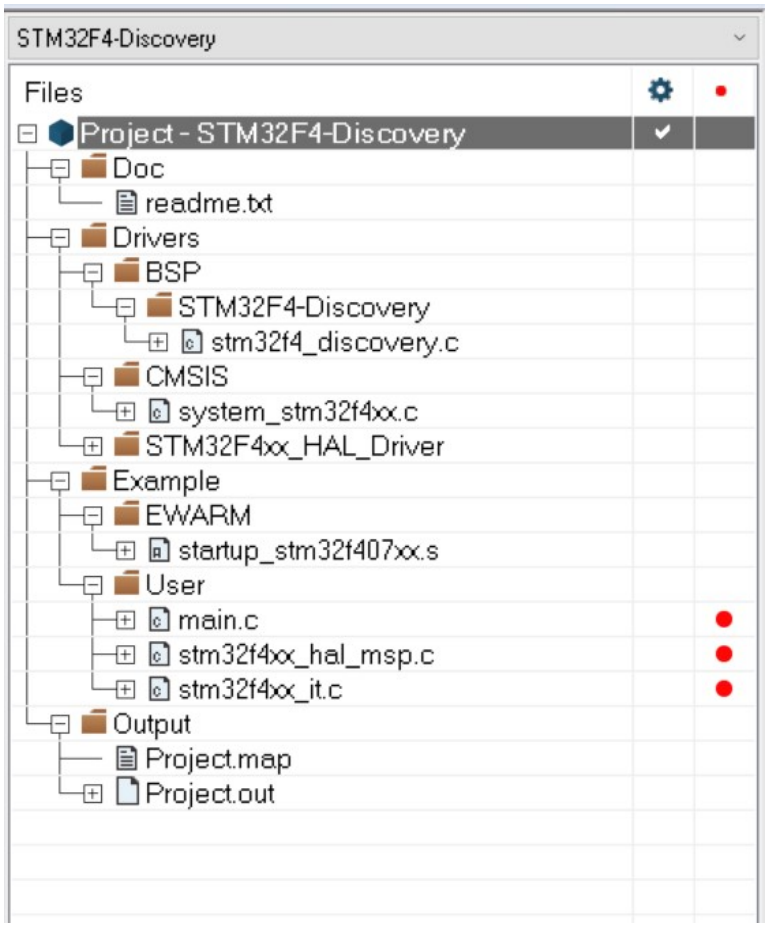
For this purpose, the use of the CMSIS DSP library is enabled in the project options.

3 Firmware package

The firmware has been developed starting from the STM32Cube_FW_F4_V1.16.0 libraries and developed using the IDE “IAR Embedded Workbench® for Arm” version 8.5.

The firmware applications are provided in one single package and supplied in one single zip file. The extraction of the zip file generates one folder, which contains the following subfolders:

Figure 1. Firmware package



3.1 Application folder

3.1.1 EWARM subfolder

This folder contains the file startup_stm32f4xx.s that provides the Cortex-M4F startup code and interrupt vectors for all STM32F4xx device interrupt handlers.

It also contains the preconfigured project for the EWARM toolchain.

3.1.2 User subfolder

This folder contains the main.c file, the stm32f4xx_hal_msp.c file for user specific implementation of peripheral drivers, and the stpm32f4xx_it.c with the interrupt routines implementation.

In the **main.c** file, two sets of frames are defined in aTxBuffer[] array, to read from the primary or secondary channel respectively. The user should uncomment those related to the channel to read.

The set of frames contains four frames, which send the read and write commands below to the device. As the first transaction just sets the data latch bits, the first four frames received are disregarded. From the second transaction on, the received data is stored in aRxBuffer[] array.

1. First frame:
   a. Read address: 0xFF to increment address pointer and read next data register.
   b. Write command: set latch bits to update register content.
   c. Received register contains the voltage fundamental sample.
2. Second frame:
   a. Read address: 0x30 or 0x34, to set address pointer to voltage wideband register, for primary or secondary channel.
   b. Write command: none.
   c. The second frame received is the current fundamental data.
3. Third frame:
   a. Read address: 0xFF to increment address pointer and read next data register.
   b. Write command: none.
   c. Received register contains the voltage wideband sample.
4. Fourth frame:
   a. Read address: 0x38 or 0x3C, to set address pointer to voltage fundamental register, for primary or secondary channel.
   b. Write command: none.
   c. Received register contains the current wideband sample.

For more information on STPM3x communication protocol, refer to the datasheet.

In the CalculateTHD functions, the RMS of the signals are calculated, based on the samples collected. These data are then multiplied for voltage and current LSB values.

The **stpm32f4xx_it.c** contains the following interrupt routines:

- TIMx_IRQHandler
- SPIx_DMA_RX_IRQHandler
- SPIx_DMA_TX_IRQHandler
- SPIx_IRQHandler
- EXTI1_IRQHandler.

Each interrupt is configured in the stm32f4xx_hal_msp.c file.

3.2 Drivers folder

3.2.1 BSP subfolder

This folder contains the stm32f4_discovery.c file that provides a set of firmware functions to manage LEDs and push buttons available on the STM32F4DISCOVERY board.

3.2.2 CMSIS subfolder

This subfolder contains the file system_stm32f4xx.c.

This file contains the system clock configuration for STM32F4xx devices. It exports the SystemInit() function, which sets up the system clock source, PLL multiplier, and divider factors, AHB AHB/APBx prescalers, and flash settings. This function is called at startup just after reset and before connecting to the main program. The call is made inside the startup_stm32f4xx.s file.

The stm32f4xx.h include file contains the definitions of all peripheral registers, bits, and memory mapping for STM32F4xx devices.

3.2.3 STM32F4xx_HAL_driver

This subfolder contains the sources of STM32F4xx peripheral drivers.

Each driver consists of a set of routines and data structures covering all peripheral functionalities.

The development of each driver is driven by a common API (application programming interface) which standardizes the driver structure, the functions, and the parameter names.

Each peripheral has a source code file, stm32f4xx_ppp.c, and a header file, stm32f4xx_ppp.h. The stm32f4xx_ppp.c file contains all the firmware functions required to use the PPP peripheral.

3.3 Output

Here there are the generated .map and .out files.

3.4 Peripheral configuration

In the main.c and in the stm32f4xx_hal_msp.c files, the used peripherals are configured as shown in Table 3 below.

Table 3. Peripheral configuration

Peripheral	Instance	Configuration
Timer	TIM3	• Period = 10000 - 1 • Prescaler = ((SystemCoreClock/2)/10000) - 1 • ClockDivision = 0 • Counter direction = up
SPI	SPI2	• Prescaler 16 • CLK polarity high, phase 2nd edge • SCS hardware • No CRC check • MSB first • Master mode
GPIO	EXTI Line 1	• Mode falling • No pull
DMA	DMA1_Stream4	• SPI TX • MEMORY_TO_PERIPH; • PINC_DISABLE; • DMA_MINC_ENABLE; • Mode NORMAL; • DMA_PRIORITY_LOW;
	DMA1_Stream3	• SPI RX • PERIPH_TO_MEMORY • PINC_DISABLE; • DMA_MINC_ENABLE; • Mode NORMAL; • DMA_PRIORITY_HIGH;

The pin used to connect to the SPI pins of the STPM3x boards are defined in main.h:

- SPIx_SCK_PIN: GPIO_PIN_13
- SPIx_SCK_GPIO_PORT: GPIOB
- SPIx_SCK_AF: GPIO_AF5_SPI2
- SPIx_MISO_PIN: GPIO_PIN_14
- SPIx_MISO_GPIO_PORT: GPIOB
- SPIx_MISO_AF: GPIO_AF5_SPI2
- SPIx_MOSI_PIN: GPIO_PIN_15
- SPIx_MOSI_GPIO_PORT: GPIOB
- SPIx_MOSI_AF: GPIO_AF5_SPI2.

The pin used to connect the CLKOUT pin of the STPM3x devices is PA1 and it is configured in main.c file.

Revision history

Table 4. **Document revision history**

Date	Version	Changes
26-Mar-2023	1	Initial release.

Contents

List of tables

List of figures