Getting started with STSW-STPM004 firmware package

Introduction

This document describes the STSW-STPM004 firmware that implements a three-phase meter with shunts current sensors, using STPMS2 sigma-delta modulator and STISO621 isolated interface. The FW is developed for the EVALSTPM-3PHISO evaluation board on the STPM32F413 microcontroller.

The FW implements hardware access layer, metrology calculations and a basic metrology application of three-phase solution interfacing STPMS2 analog front-end.

It also embeds a simple shell communication to access relevant metrology parameter with a terminal or to interface the software GUI for a fast solution evaluation.

Figure 1. EVALSTPM-3PHISO eco system

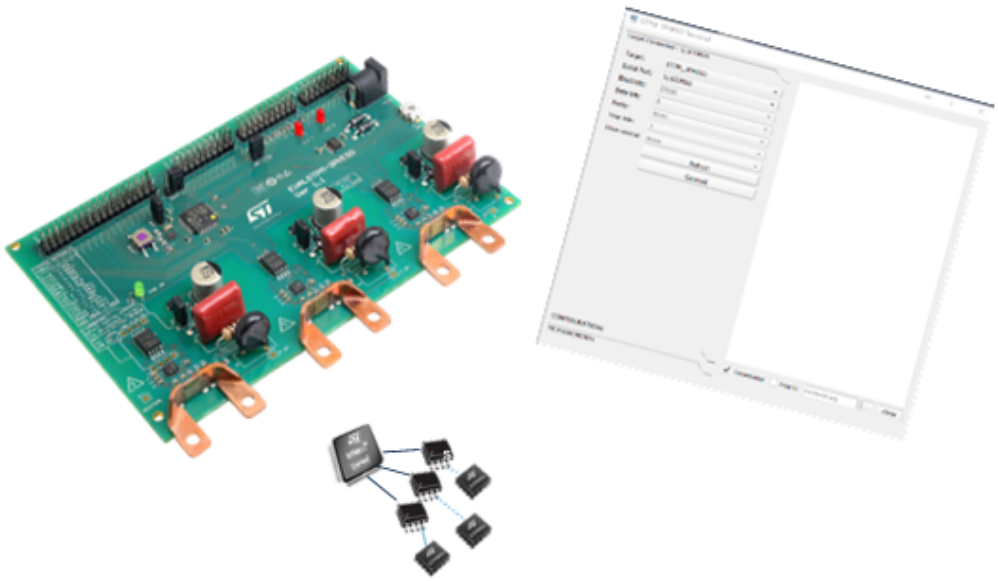


Table 1. Applicable products

Type	Reference products
Evaluation board	EVALSTPM-3PHISO
Sigma-delta modulator	STPMS2
Isolated interface	STISO621W
Microcontroller	STM32F413RH

1 Metrology application

1.1 Hardware configuration

The STPMS2 devices in the EVALSTPM-3PHISO board are configured as follows:

Table 2. STPMS2 HW configuration

PIN	Connection	Description
MS0	VCC	HPR, amplifier GAIN selection g3 = 16
MS1	GND	TC = 50 ppm/°C
MS2	GND	Voltage channel ON, DATn = ~ [DAT =(CLK) ? bsV : bsC)]
MS3	GND	Hard mode, BIST mode OFF

For further information on STPMS2 configuration please refer to device datasheet (see www.st.com).

The metrology section analog front-end component values are as follows:

Table 3. AFE Components

Component	Value	Description
Shunt	0.3 mOhm	Current sensor
R1	810 kOhm	Voltage divider resistor (actually 3 x 270 kOhm)
R2	470 Ohm	Voltage divider resistor

1.2 Application parameters and data conversion

Given the AFE components in Table 3 and the STPMS2 constant values below:
Application constants

Table 4. Application constants

Parameter	Value	Unit	Description
Vref	1.2	V	Voltage reference value
Ai	16		Current channel gain
Au	2		Voltage channel gain
Cal_i	0.875		Calibrator mid value
Cal_v	0.875		Calibrator mid value
Dclk	5000	Hz	Decimation frequency

The following parameters can be calculated:

Table 5. Application parameters

Component	Value	Unit	Description
LSB_E	1.41E-07	Wh Varh Vah	LSB of energy registers
LSB_P	1.24E-03	W Var VA	LSB of power registers
LSB_VRMS	35.60 E-03	V	LSB of RMS voltage register
LSB_IRMS	2.18 E-03	A	LSB of RMS current register
Vmax	360.92	V	Maximum RMS voltage
Imax	88.39	A	Maximum RMS current
LSB_VMOM	1.39 E-04	V	LSB of instantaneous voltage register
LSB_IMOM	3.41E-05	A	LSB of instantaneous current register
CP	27000	Pulses/kWh	Pulse constant

It is possible to calculate these application parameters, using the formulas in application note AN4470 “The STPM3x and the STCOMET application calibration” (see www.st.com).

All the design and calibration formulas apply to the STPMS2 application as well, but the application constants must be changed as in Table 4 .

The calculation follows the formulas given for fixed shunt sensitivity, R2 and CP.

Using these formulas, it is also possible to change the AFE components according to the application needs (voltage and current range and resolution) or to calibrate the board to get the given pulse constant.

From the LSB of voltage and current, the conversion factors to be input in the code can be calculated from the following formulas:

Table 6. Application constants

Parameter	Formulas	Value	Description
powerFact	$LSB\_P*2^{28}*100$	33333333	Power register conversion factor
voltageFact	$LSB\_VRMS*2^{15}*100$	116667	Voltage register conversion factor
currentFact	$LSB\_IRMS\ 2^{*17}*100$	28571	Current register conversion factor
energyFact	$LSB\_E\ 2^{*32}*100$	60681	Energy register conversion factor

The first three of the above factors should be input in the handler_metrology.c, metroDefaultNvm constant, as indicated in the comment.

Since power and energy LSBs differ by a constant factor, the energy conversion is done by the FACTOR_POWER_ON_ENERGY constant, set in metroTask.c file.

This factor is calculated as follows:

FACTOR_POWER_ON_ENERGY = powerFact / energyFact = 549.

An excel file with all the above formulas is available on request.

1.3 Metrology registers

The data can be accessed in STPM32-like registers; data mapping in the registers is shown in Figure 2. Not all the STPM32 registers are filled; only the used registers are shown.

Figure 2. Registers map

	Address	(R)ead (W)rite (L)atch	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	Name	Default Value
0	00	RW	DSP Control Register #1 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 LCS1 [1:0] / LPS1 [1:0] / LPW1 [3:0] / BHPFC1 / BHPFV1								DSPCTRL1	C4000000
1	02	RW	DSP Control Register #2 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 LCS2 [1:0] / LPS2 [1:0] / LPW2 [3:0]								DSPCTRL2	E4000000
2	04	RW	DSP Control Register #3 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 REF_FREQ / LED2OFF / LED1OFF								DSPCTRL3	00000000
4	08	RW						CHV [11:0]			DSPCTRL5	00000800
5	0A	RW						CHC [11:0]			DSPCTRL6	00000800
6	0C	RW	Padding		OFFV [23:0]						DSPCTRL7	00000000
7	0E	RW	Padding		OFFC [23:0]						DSPCTRL8	00000000
8	10	RW	OFAF [9:0]			OFA [9:0]					DSPCTRL9	00000000
9	12	RW	OFS [9:0]			OFR [9:0]					DSPCTRL10	00000000
14	1C	RW		V Freq ERR / V Signal Stuck		C Signal Stuck	RE_S / AF_S / AWB_S				DSPIRQ1	03107000
16	20	RWL		V Freq ERR / V Signal Stuck		C Signal Stuck	RE_S / AF_S / AWB_S				DSPSR1	00000000
21	2A	RL		V Freq ERR / V Signal Stuck		C Signal Stuck	RE_S / AF_S / AWB_S				DSPEVENT1	00000000
23	2E	RL						V Period [11:0]			DSP_REG1	00000000
24	30	RL	Padding		V Sample [23:0]						DSP_REG2	00000000
25	32	RL	Padding		C Sample [23:0]						DSP_REG3	00000000
28	38	RL	Padding		V Fund Sample [23:0]						DSP_REG6	00000000
29	3A	RL	Padding		C Fund Sample [23:0]						DSP_REG7	00000000
32	40	RL	THD V								DSP_REG10	00000000
33	42	RL	THD C								DSP_REG11	00000000
36	48	RL	C RMS Data [16:0]				V RMS Data [14:0]				DSP_REG14	00000000
37	4A	RL	C Fund RMS Data [16:0]				V Fund RMS Data [14:0]				DSP_REG15	00000000
38	4C	RL					V_V Delay[15:0]				DSP_REG16	00000000
39	4E	RL		V-C Angle [11:0]							DSP_REG17	00000000
42	54	RL	1-PH Active Energy								CH1_REG1	00000000
43	56	RL	1-PH Fundamental Energy								CH1_REG2	00000000
44	58	RL	1-PH Reactive Energy								CH1_REG3	00000000
45	5A	RL	1-PH Apparent Energy								CH1_REG4	00000000
46	5C	RL	Padding	1-PH Active Power[28:0]							CH1_REG5	00000000
47	5E	RL	Padding	1-PH Fundamental Power[28:0]							CH1_REG6	00000000
48	60	RL	Padding	1-PH Reactive Power[28:0]							CH1_REG7	00000000
49	62	RL	Padding	1-PH Apparent RMS Power[28:0]							CH1_REG8	00000000
51	66	RL	Padding	1-PH Momentary Active Power[28:0]							CH1_REG10	00000000
52	68	RL	Padding	1-PH Momentary Fundamental Power[28:0]							CH1_REG11	00000000
66	84	RL	3-PH Active Energy								TOT_REG1	00000000
67	86	RL	3-PH Fundamental Energy								TOT_REG2	00000000
68	88	RL	3-PH Reactive Energy								TOT_REG3	00000000
69	8A	RL	3-PH Apparent Energy								TOT_REG4	00000000

For a full register description please refer to the EVALSTPM-3PHISO getting started user manual UM2847 on www.st.com

2 Firmware architecture

The STPMS2 firmware library has been developed as a basic metrology application for the EVALSTPM-3PHISO evaluation board and it is part of a metrology ecosystem, able to interface a PC GUI for fast solution evaluation and application development.

Figure 3. Virtual STPM36

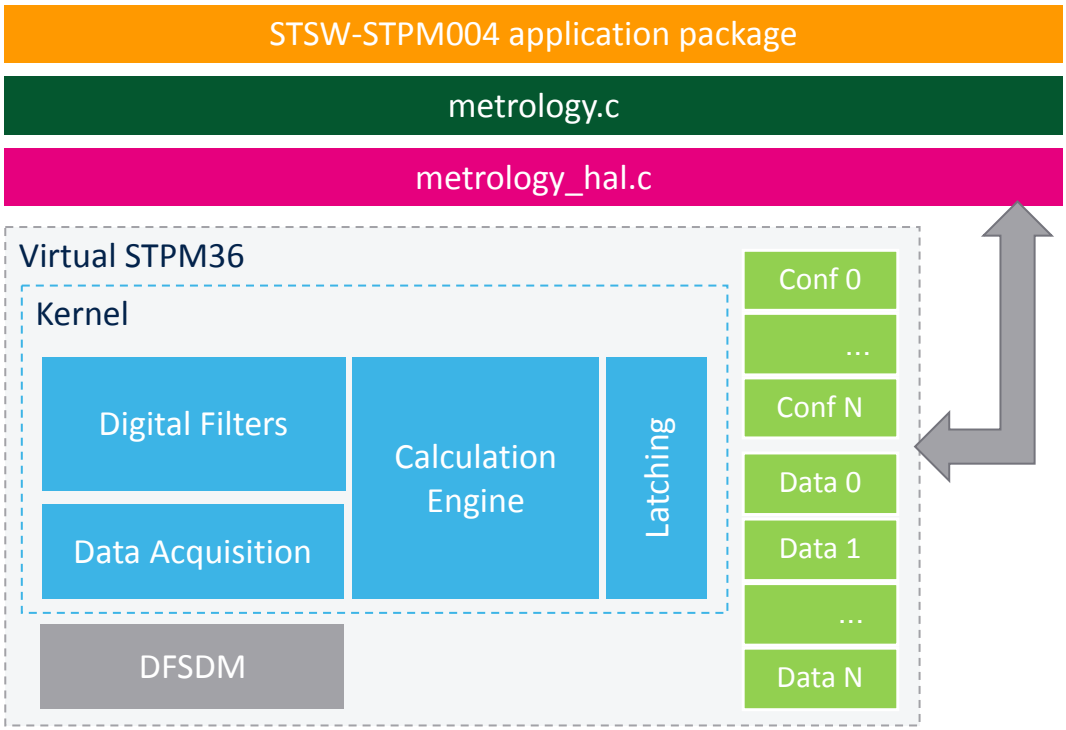


Figure 3 shows the FW library, based on a “virtual STPM36” metrology device with:

- Signal acquisition from up to six channels (three phases, each with voltage and current monitoring)
- a processing kernel implementing digital filtering of signals and data processing
- a set of registers, containing metrology measurements and calibration/configuration data.

The STPM3x-like registers allow the application access to the virtual device (as 3xSTPM32 devices), which can be read/configured by the hardware interface layer contained in the metrology_hal.c file.

The metrology.c file uses the metrology data for application processing and makes them available to the user by the terminal or GUI.

The metroTask.c file implements the functions for a simple metrology application.

Each part of the “virtual STPM36” implementation is explained in detail in the following sections.

For the metrology GUI and related documentation please refer to STSW-STPM005 on www.st.com.

2.1 Application configuration

The firmware workspace contains two different configurations. The configuration differs for some compiling options, to enable or disable calculation of the THD:

- Release_S2: the metrology_STPMS2.a library is included in the workspace. THD and fundamental RMS of voltage and current signals are not computed. The overall signal processing time is reduced.
- Release_S2_THD: the metrology_STPMS2_THD.a library is included in the workspace. THD and fundamental RMS of voltage and current signals are computed.

2.2 Application workflow

The main.c file calls all the generic initialization functions and starts the metrology processes.

In particular, the METRO_Init() function in the metroTask.c file configures all the metrology related data:

- MET_Conf() in handler_metrology.c copies the default configuration and calibration data, and the conversion factors for metrology registers from the constant metroDefaultNvm in the metroData variable, which stores all metrology information
- Metro_Setup() configures the topology of the metrology system and the device type in the variable Tab_METRO_internal_Devices_Config (this is the interface variable to the virtual STPM device - see Section 4.1 for full description). This information, stored in metroData.nvm->config and previously copied from metroDefaultNvm, corresponds to a three-phase system with STM32 and 3xSTPM32.
- MET_RestoreConfigFromNVM () writes calibration/configuration data from metroData.nvm->data to the low level STPM virtual device, and loads them back to Tab_METRO_internal_Devices_Config[].metro_stpm_reg structure
- Configures hardware factors for registers conversion
- Configures data latch type (LATCH_SW in this application example, or LATCH_AUTO)
- Configures LED0 and LED1 output energies
- STPMS2_MetroInit configures DFSDM peripheral and DSP filters
- STPMS2_MetroStart starts data acquisition and processing.

A 2 sec timer (Timer2) sets a variable whose value is checked in the while loop. If the time has elapsed, these functions are executed:

- METRO_Latch_Measures() sets a variable that triggers the copy from the internal metrology kernel registers to the STPMS2_MetroRegs[] low level variable.
- METRO_Get_Measures() reads the metrology registers from STPMS2_MetroRegs[] and updates the Tab_METRO_internal_Devices_Config[].metro_stpm_reg structure.
- METRO_UpdateData() fills the metroData variable with the high level metrology information calculated from the Tab_METRO_internal_Devices_Config[].metro_stpm_reg structure.

A global interrupt variable, set by the metrology processing kernel according to the event monitored, could be checked in the main while loop.

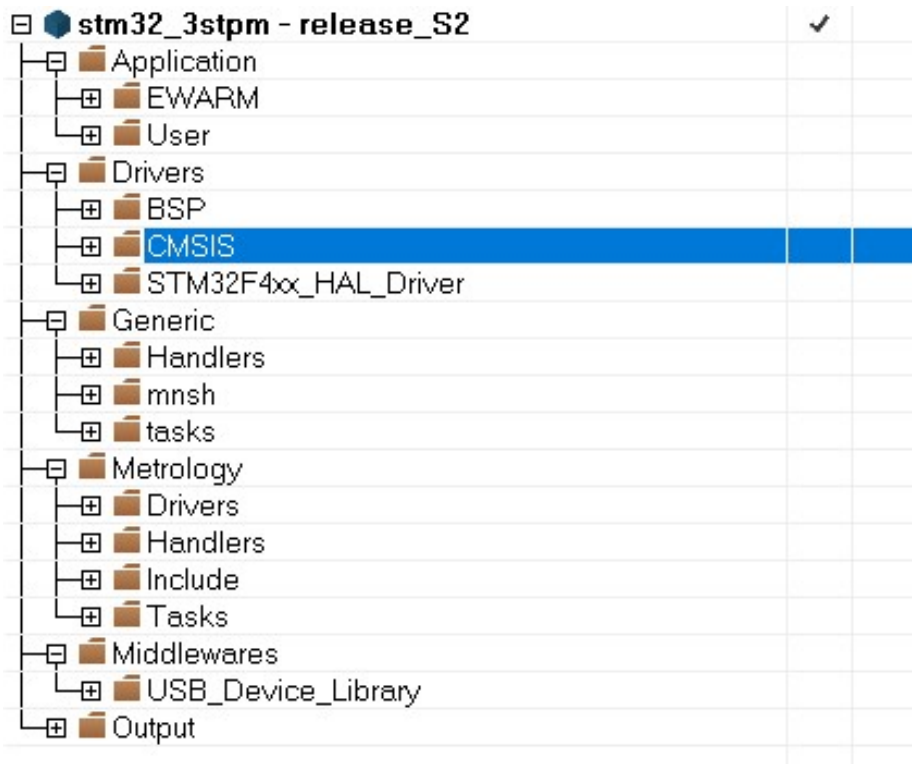
To monitor a specific event the related bit should be set in the DSP_IRQ1 register. For further details please refer to the full register description in the EVALSTPM-3PHISO getting started user manual on www.st.com.

3 Description of the firmware package

The FW package is developed using the IDE “IAR embedded workbench for ARM” version 8.5.

The firmware applications is 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 4. 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 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.

The stpm32f4xx_it.c contains the following DMA interrupt routines:

- DMA2_Stream6_IRQHandler
- DMA2_Stream1_IRQHandler
- DMA2_Stream0_IRQHandler
- DMA2_Stream5_IRQHandler
- DMA2_Stream2_IRQHandler
- DMA2_Stream3_IRQHandler

These functions are linked to the six DFSDM filters.

The interrupt call starts the acquisition of a voltage or current converted sample from DFSDM peripheral and all related metrology calculation.

Each interrupt is defined and linked to its filter in the HAL_DFSDM_FilterMspInit() in stm32f4xx_hal_msp.c file, and each interrupt has the highest priority (0).

Since the interrupt priority impacts the measurement accuracy and bandwidth, it must be chosen carefully according to the application needs.

This folder also contains USB driver files.

3.2 Drivers folder

3.2.1 BSP subfolder

This folder contains the STM32F413_3STPMS2.c file that provides a set of firmware functions to manage the LEDs available on the STM32F413_3STPMS2 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 file contains the definitions of all peripheral registers, bits, and memory mapping for STM32F4xx devices.

3.2.3 STM32F4xx_HAL_Driver

This subfolder contains 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 Generic

This folder contains generic handlers for EEPROM management, and contains implementation of a minishell protocol to communicate with the Metrology GUI.

3.3.1 Minishell commands

The commands implemented to communicate through the shell are reported in the table below.

For the full list of commands implemented by the minishell please refer to EVALSTPM-3PHISO getting started user manual on www.st.com.

3.4 Middlewareness

This folder contains standard USB device library.

3.5 Output

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

3.6 Metrology

3.6.1 Drivers

This folder contains the metrology libraries, described in detail in the next section.

3.6.2 Handlers

The files handler_eeprom.c and handler_nvram.c manage the storage of data in non-volatile memory. To use this feature the EEPROM_PRESENT macro must be defined in the project C/C++ compiler options, among preprocessor defined symbols.

This feature is not enabled in this project.

3.6.3 Include

Contains metrology devices related definition.

3.6.4 Tasks

metroTask.c implements metrology application high level functions:

- METRO_Init() function takes care of metrology variables initialization, set-up of the DFSDM peripheral and acquisition start of metrology data.
- METRO_Task() function manages commands coming from the metrology GUI interfaced by the minishell
- METRO_Latch_Measures() function sends a command to copy updated internal metrology data, calculated in metrology_STPMS2.a (or metrology_STPMS2_THD.a) library, into the STPMS2_MetroRegs[] variable, which represents a metrology device full register map. Two data latch type can be configured:
  - LATCH_SW: Data is updated in the registers once when the function is called
  - LATCH_AUTO: Data is updated at the data processing rate, every 200 us, after the function is called the first time. This option increases FW execution time.
- METRO_Get_Measures() function, called after the latch, takes device data registers from STPMS2_MetroRegs[] variable and copies them into the Tab_METRO_internal_Devices_Config (see below for full description) global variable.
- METRO_UpdateData() function takes the metrology raw data from the Tab_METRO_internal_Devices_Config global variable (see below for full description), and processes them to get final power and energy information.

Data is then updated in metroData_t metroData, storing calculated data for each phase (power, energy, voltage and current rms and thd, …).

These metrology functions are regularly called in the main while loop, with a time base of 2 seconds defined by Timer2 (configured in main.c).

In metroTask.h the metroData variable, containing metrology application data, is defined.

It is structured to contain three-phase data:

- int32_t powerActive; // three-phase power;
- int32_t powerActiveFund;
- int32_t powerReactive;
- int32_t powerApparent;
- uint32_t energyCumul[4]; // three-phase active, active fund, reactive and apparent energy

and single-phase data:

- int32_t chanPower[4][METRO_MAX_PHASES];
- uint32_t energy[4][METRO_MAX_PHASES];
- uint32_t rmsvoltage[METRO_MAX_PHASES];
- uint32_t rmscurrent[METRO_MAX_PHASES].

If THD calculation is enabled, it contains related information:

- uint32_t rmsfundvoltage[METRO_MAX_PHASES];
- uint32_t rmsfundcurrent[METRO_MAX_PHASES];

- uint32_t thdvoltage[METRO_MAX_PHASES];
- uint32_t thdcurrent[METRO_MAX_PHASES];

4 Metrology library

4.1 Metrology.c/.h files

This file implements functions taking care of:

- initialization and update of the device related data structures
- configuration of the metrology conversion factor to convert device calculated data into metrology values
- read and write access of metrology data to read, configure and calibrate the devices

These functions are the entry point for metrology application development. All the prototypes can be found in its header file.

To access metrology data two global variables are used, as described in the sections below.

4.1.1 METRO_Device_Config_t Tab_METRO_internal_Devices_Config[NB_MAX_DEVICE]

This is an array of maximum five elements.

Each element of the array contains all relevant data for each single voltage/current phase of the meter architecture.

The first element is the host MCU that, in this architecture, does not contain any metrology relevant data, so sensitive information is contained in devices 1, 2 and 3.

In this data structure, the two data:

- METRO_Device_t device
- uint8_t channels_mask

are used to keep backward compatibility with the metrology ecosystem and the external metrology GUI. They indicate the device p/n and at which device channel the phase corresponds.

In this topology, the STPMS2 is represented by STPM32 which is equivalent, having just one V – C channel.

This data structure also contains the conversion factor to translate calculated data into metrology values, the latch type (upon software request or automatic).

All configuration, calibration and metrology data from each phase is contained in the metro_stpm_reg data structure, whose seventy 32-bit data have the same structure as the STPM3x registers map.

In this way the “Virtual STPM36” is represented as 3 x STPM32 architecture.

The above organization allows keeping the compatibility with metrology ecosystem.

4.1.2 METRO_Data_Energy_t METRO_Data

This variable is internal to the module and contains two software energy accumulators for each phase and each energy type.

This is necessary since internal 32-bit energy registers length allow the storage of a small amount of energy, with a high resolution to keep accuracy for LED generation, but not enough for the accumulation during the meter lifetime. Then the metrology application takes care of integration of energy in this data structure.

4.2 Metrology_hal.c/.h files

This file contains low level function to access all configuration, calibration and measurement information inside the “Virtual STPM36”.

4.3 Metrology_STPMS2.a / Metrology_STPMS2_THD.a file

This library file implements the sigma-delta bit streams filtering and the calculation of all metrology data. A block diagram of the calculation chain is shown in Figure 5.

Main functions of the library are:

- Initializes and implements filters for each stream
- Implements all metrology calculations on filtered signals

- Calculates signals period and phase shift
- Calculates RMS and THD value for each signal
- Calculates power and energy for each phase
- Copies all data in register structures upon latch request

Each measurement (except THD) is performed in real-time on a 200 us basis.

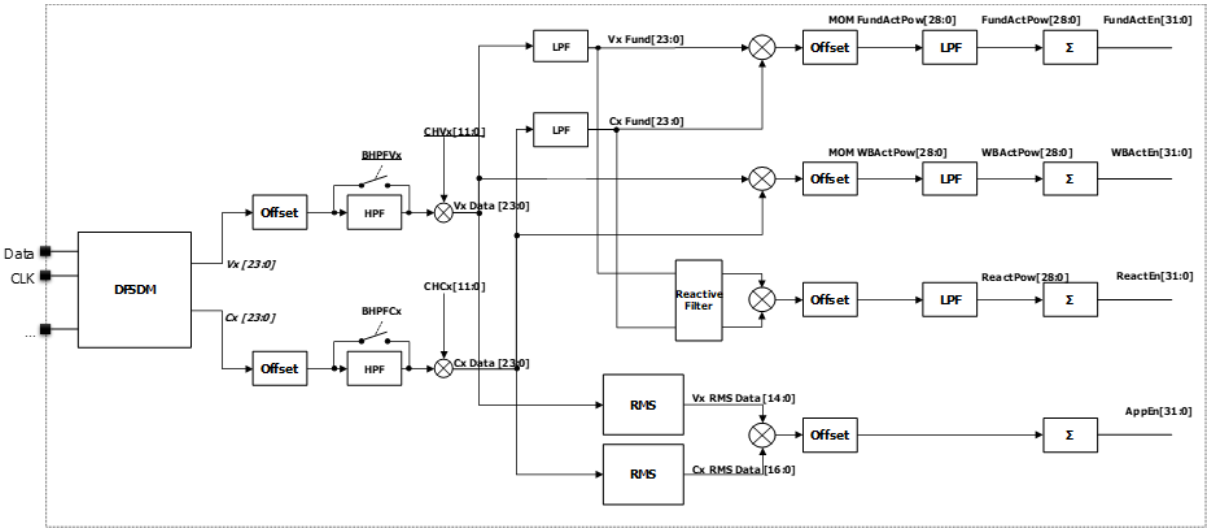
Since THD calculation increases the burden of the CPU utilization, it can be enabled or disabled by the define macro CALCULATE_THD in the metrology_STPMS2.h file. THD calculation is performed with a low rate, on a 2 s basis (defined by Timer2 in main.c).

Calculations are based on the CMSIS DSP Software Library, which features a suite of common signal processing functions for use on Cortex-M processor based devices.

The CMSIS DSP library is completely written in C and is fully CMSIS compliant. High performance is achieved through maximum use of Cortex-M4F intrinsics. A check in the general options of the project enables the use of this library.

Measured data for each phase is available in the variable Tab_METRO_internal_Devices_Config[x]. metro_stpm_reg with x = 1, 2 and 3 for the 3 phases respectively, packed as reported in Section 1.3 .

Figure 5. Sigma-delta bitstream processing block



Revision history

Table 7. **Document revision history**

Date	Version	Changes
12-Apr-2021	1	Initial release.
03-May-2021	2	Updated Figure 2. Registers map, Section 2.2 Application workflow and Section 3.6.4 Tasks.

Contents

List of tables

List of figures