User's Manual

16

RL78/I1C Group

User's Manual: 1PH2W Metrology

RENESAS MCU
RL78 Family / I1C Series

— Preliminary —

Renesas Electronics
www.renesas.com

Rev5.01 April 20, 2023

Table of Contents

1. Overview

1.1 Introduction

The RL78/I1C Metrology Library provides functions used to build up the firmware for energy meter, supports implementations of core features, and meter measurement (e.g., VRMS, IRMS, Energy Accumulation…), for many kinds of current sensor: Shunt, Current Transformer (CT) and Rogowski coil. Customization is also provided to better align software code with meter usage.

This library is a special build usable only on the RL78/I1C Group.

To minimize memory footprint while including specific meter features, some versions of the library are provided based on the following naming rules shown below. Please choose a library that best fits your usage.

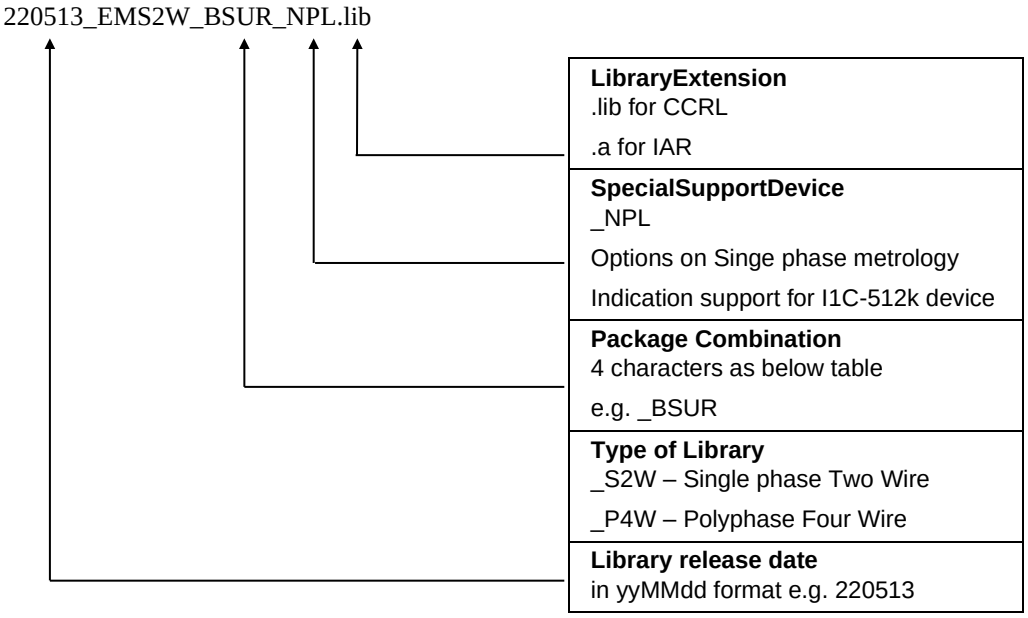


Table 1-1 Package Combination

Suffix	Description	Include
BSUR	Single Gain Measurement	EM Basic
BQUR	Single Gain with Reactive Measurement	EM Basic + Reactive
BQFR	Single Gain with Reactive and Fundamental Measurement	EM Basic + Reactive + Fundamental
WSUR	Dual Gain Measurement	EM Basic + Gain Switch
WQUR	Dual Gain with Reactive Measurement	EM Basic + Gain Switch + Reactive
WQFR	Dual Gain with Reactive and Fundamental Measurement	EM Basic + Gain Switch + Reactive + Fundamental

Each package contains different library builds, thus has different levels of ROM, RAM, and CPU load; please refer to Occupied RAM, ROM, CPU Load for more details.

EXAMPLE

- The 220513_EMS2W_BSUR.lib is the single-phase two-wire metrology library for CCRL compiler, without reactive measurement for common I1C device
- The 220513_EMS2W_BSUR_NPL.lib is the single-phase two-wire metrology library for CCRL compiler, without reactive measurement specialized for I1C-512k device

1.2 Specification

The following tables list specifications of the library.

Table 1-2 Basic specification

Item	Specification
Support microcontroller	RL78 / I1C Group
Data-endian	Little-endian
MCU Frequency	6-24MHz
Environment + Compiler	CS+_CCRL: - Integrated development Environment: CS+ V8.07 - C Compiler: CCRL v1.11.0 e2studio_CCRL: - Integrated development Environment: e2studio 202204 - C Compiler: CCRL v1.11.0 IAR: - Integrated development Environment: Embedded Workbench v8.5 - C Compiler: 4.21.1
Meter Library Type	Single Phase 2 Wires (1P2W) No. of channels: • 1 Voltage Channel: V • 2 Current Channels: I1, I2 (Support Shunt, CT and Rogowski coil sensor)
Library required modules	• I1C except NPL: Hardware acceleration of arithmetic calculation (MACL) • I1C NPL: Hardware acceleration of arithmetic calculation (24 channel MACL) • Interval timer (40ms) with interrupt. • DSAD continuous conversion with interrupt (highest priority) • PORT, 3 I/O pins for pulse output • *RTC, WDT (optional).*

Table 1-3 Library features

Item	Specification
Gain Switch	Support to sense the sampling counter; switch to higher amplifier gain on I1 and I2 for more accurate on measurement output at low current; switch to lower gain at high current.
Fundamental power	Support to measure fundamental active power in the event signal contains harmonic.
Sampling Frequency	Depends on sampling trigger input, support max. 3906Hz
Pulse output	3 channels, IEC standard, configurable pulse on duration in ms
Sampling resolution	Max 24 bits counter
Measurement output	• VRMS, I1RMS, I2RMS • Power: Active, Fundamental Active, Reactive, Apparent Note • Power2: Active, Fundamental Active, Reactive, Apparent Note • Energy Accumulation: Active, Reactive, Apparent • Power Factor • Power Factor sign (Lead, Lag, Unity) • Line Frequency • Meter calibration information • Event Status Bit: Meter no-load status, voltage sag & swell

Note Reactive and Fundamental Active is only available on some library version that supported.

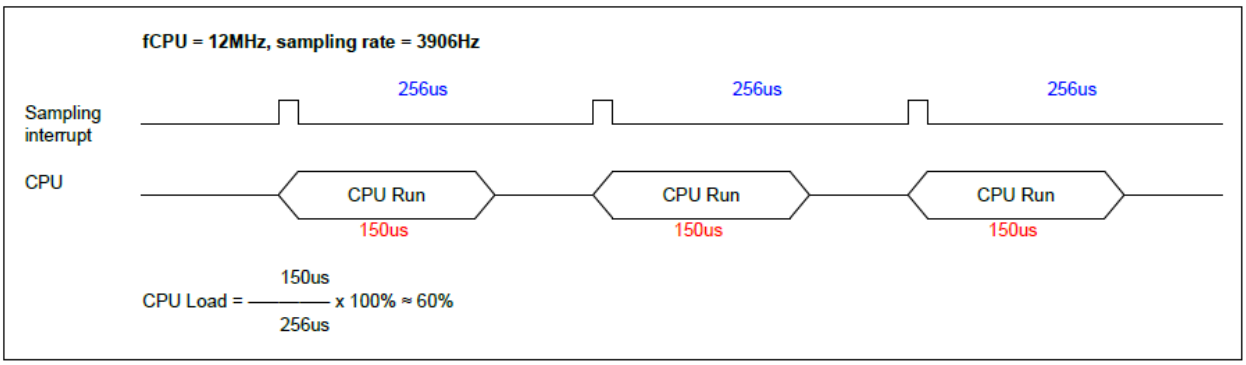
1.3 Occupied RAM, ROM, CPU Load

How is the CPU load of library measured?

In overview, the library runs in the background, on DSAD interrupt (1/fs interval), TIMER interrupt (40ms interval).

DSAD interrupt priority is at the highest and nested interrupt is disabled (EI=0), then all CPU load of the library can be estimated by measuring based on DSAD interrupt, with how much time spend for processing of 1 sampling set (Voltage, I1, I2).

For example, the following result shows a 60% CPU load.



Below is a summary for **version 221124**.

Table 1-4 ROM, RAM, and CPU load of library of CCRL, sampling frequency 3906Hz, CPU @ 6MHz

Suffix	Library with included modules	ROM (in bytes)	RAM (in bytes)	% Max CPU Load
BSUR	EM Basic	14473	1618	39.323
BQUR	EM Basic + Reactive	15510	1626	57.422
BQFR	EM Basic + Reactive + Fundamental	16392	1778	74.805 Note1
WSUR	EM Basic + Gain Switch	15802	1618	44.727
WQUR	EM Basic + Gain Switch + Reactive	17107	1626	63.737
WQFR	EM Basic + Gain Switch + Reactive + Fundamental	18254	1778	86.263 Note1

Table 1-5 ROM, RAM, and CPU load of library of NPL CCRL, sampling frequency 3906Hz, CPU @ 6MHz

Suffix	Library with included modules	ROM (in bytes)	RAM (in bytes)	% Max CPU Load
BSUR	EM Basic	14210	1514	31.901
BQUR	EM Basic + Reactive	15139	1522	44.401
BQFR	EM Basic + Reactive + Fundamental	15919	1642	63.086 Note1
WSUR	EM Basic + Gain Switch	15253	1514	37.240
WQUR	EM Basic + Gain Switch + Reactive	16286	1522	55.078
WQFR	EM Basic + Gain Switch + Reactive + Fundamental	17191	1642	74.349 Note1

Note1: Recommended to use with 12Mhz as CPU load is high. These figures already included 2 biquad filter for voltage and 1 current channel on wrapper layer

1.4 Meter firmware structure

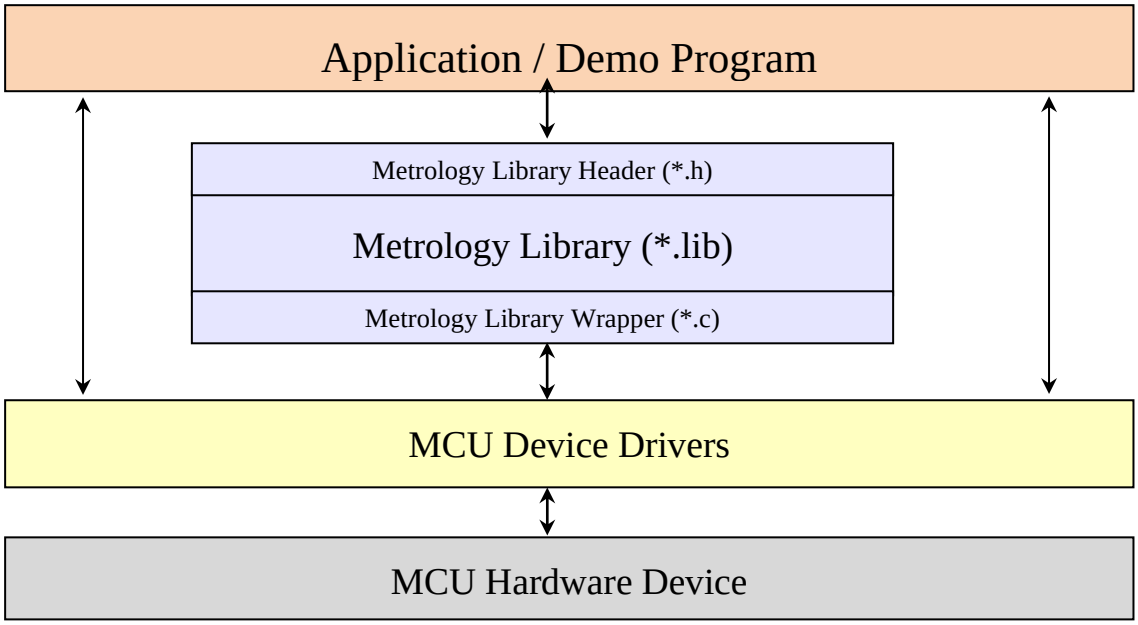


Figure 1-1 Meter firmware structure

In Energy Measurements Application, the Metrology Library is a core component. The figure above shows the software layering and project structure of a simple Energy Measurements Application. It is roughly divided into 4 layers which are: MCU Device Drivers layer, Metrology Wrapper layer, Metrology Library and Application layer.

MCU Device Driver layer contains all the device drivers source code used to configure the MCU peripherals. The source code of this layer must be created before integrating the meter library according to the hardware design and features for controller configurations. MCU Device Driver layer can be generated using the Code Generator (CG) or hand code if customization of peripheral operation is required.

Metrology Wrapper layer contains modifiable APIs to adapt the APIs in the Metrology Library to MCU Device Driver layer. Metrology Wrapper layer adaptation is required to link the device driver to the Metrology Library API. Additional signal processing can also be performed in Metrology Wrapper layer before parsing the sampled signal to the Metrology Library.

Metrology Library, a core component, uses the parsing sampled data to calculate Metering Parameters such as VRMS, IRMS, Line Frequency, Power, Energy, and some tamper detection flag. The APIs provided in the Metrology Library, illustrated in Chapter 3, will enable users to access the Metering Parameters required in Energy Measurement Application.

Note that this document will only focus on describing the Metrology Library. For other layers, please refer to alternative documents published alongside this Metrology User Manual.

1.4.1 Library structure

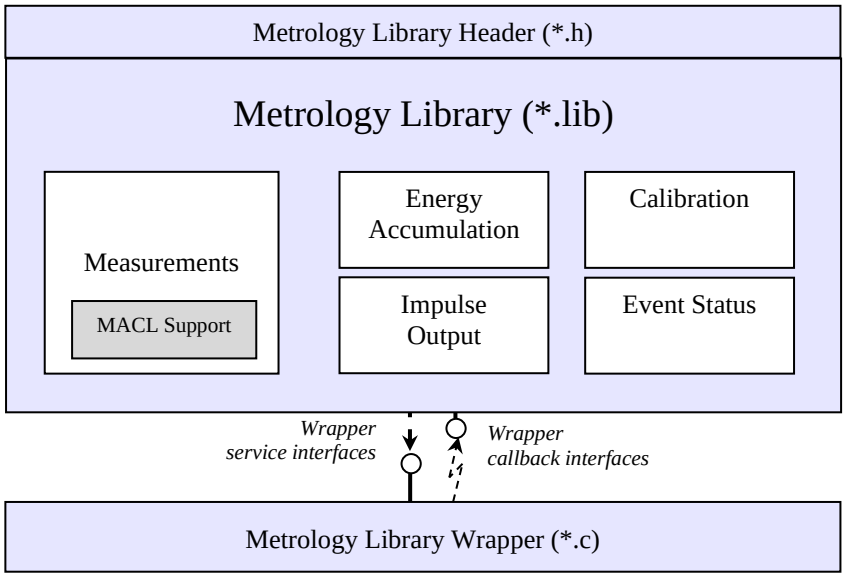


Figure 1-2 Metrology Library structure

The Metrology Library contains a few features, including Measurements, Calibration, Energy Accumulation, and Impulse Output.

Measurements: This is the most important part of this library, as it contains the methodology of all measurements related to energy meter. This component calls to service interfaces on the wrapper layer to get all required data (e.g., samples of V, I1, I2, etc) and control the operation of MCU peripherals. Besides the service interfaces, the measurement component also needs some acknowledgement signals (callbacks) from the wrapper layer for its operations, so that the library can operate in the background of the whole system. This component uses MACL of RL78/I1C series for arithmetic calculation of income signal.

Calibration: Provides the functions to calibrate the meter and output the parameters used by the metrology library for accurate measurements. This is optional for metrology library operation and can be substituted for external calibration methods instead. This component also uses the RTC interrupt of RL78/I1C series, a high accuracy clock source, for calibrating the sampling frequency on MCU

Energy Accumulation: This is the energy counter for 4 quadrant output.

Impulse Output: Co-related with energy counter increment to output an impulse based on a constant value setting.

1.4.2 Wrapper structure

Wrapper is used to link the library functions with required external functions. The library operates by calling the wrapper functions, instead of directly calling to functions on lower layers (except for MACL registers which is used directly under metrology). This characteristic makes the library independent from the lower layer. Thus, changes in the lower layers only require modification on wrapper to adapt to new functions, before proceeding with a re-build and run.

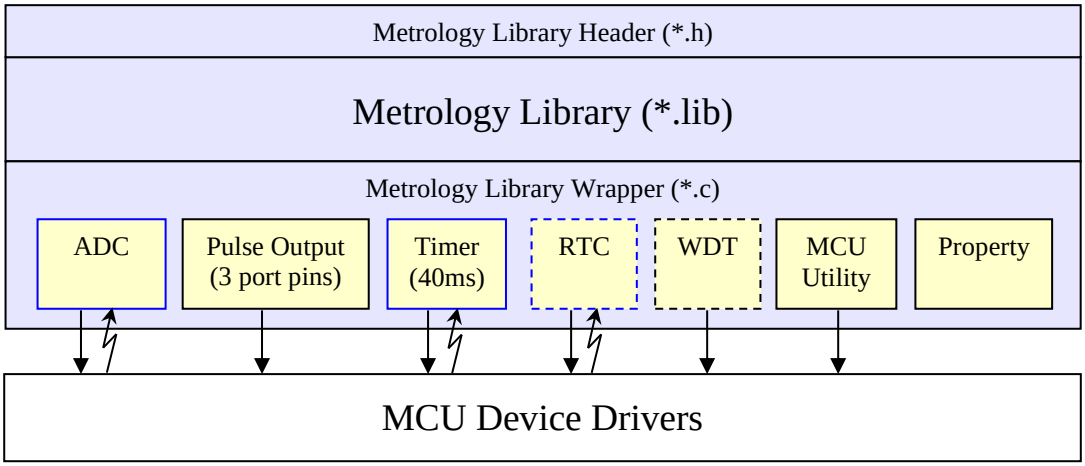


Figure 1-3 Metrology Library Wrapper structure

Shown in the figure above, ADC, Timer (40ms) and RTC have interrupt acknowledgement callbacks. Their interrupt priority setting are as follows:

Wrapper Module	Interrupt Priority Level
ADC	Level 0 (highest), acknowledgement of conversion end.
Timer	Level 3, interval callback, 40ms
RTC	Level 3, 0.5s or 1s constant interrupt callback (optional, for calibration)

Take note, please implement, or link all above wrapper modules to device driver layer, before using this library.

1.4.3 Directories / File structure

```
Library
|   typedef.h                           GSCE Standard Typedef
|
+---middleware
|   +---headers
|       em_calibration.h                Library - Calibration module header
|       em_constraint.h                 Library - Constraints module header
|       em_core.h                       Library - Library header collection
|       em_errcode.h                    Library - Error code definitions
|       em_measurement.h                Library - Measurement module header
|       em_operation.h                  Library - Operation module header
|       em_type.h                       Library - Type definitions
|       wrp_em_adc.h                    Wrapper - ADC module header
|       wrp_em_mcu.h                    Wrapper - MCU module header
|       wrp_em_pulse.h                  Wrapper - PULSE header
|       wrp_em_sw_property.h            Wrapper - PROPERTY header
|       wrp_em_timer.h                  Wrapper - TIMER module header
|       wrp_em_wdt.h                    Wrapper - WDT module header
|
\---metrology_wrapper
|   |
|       wrp_em_adc.c                    Wrapper - ADC module implementation
|       wrp_em_mcu.c                    Wrapper - Wrapper - MCU module implementation
|       wrp_em_pulse.c                  Wrapper - PULSE module implementation
|       wrp_em_sw_config.h              Wrapper - Wrapper Configuration Header file
|       wrp_em_sw_property.c            Wrapper - PROPERTY module implementation
|       wrp_em_timer.c                  Wrapper - TIMER module implementation
|       wrp_em_wdt.c                    Wrapper - WDT module implementation
```

Figure 1-4 Directories / File structure

1.5 Metrology Library Functions

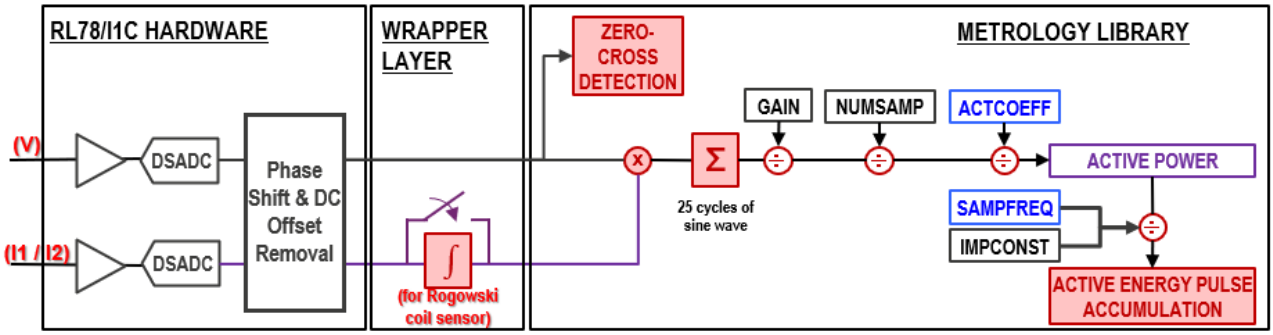
The following charts describe all measurement metrologies.

- Calibration variables are highlighted as blue rectangles.
- Measured parameters are highlighted as purple rectangles.

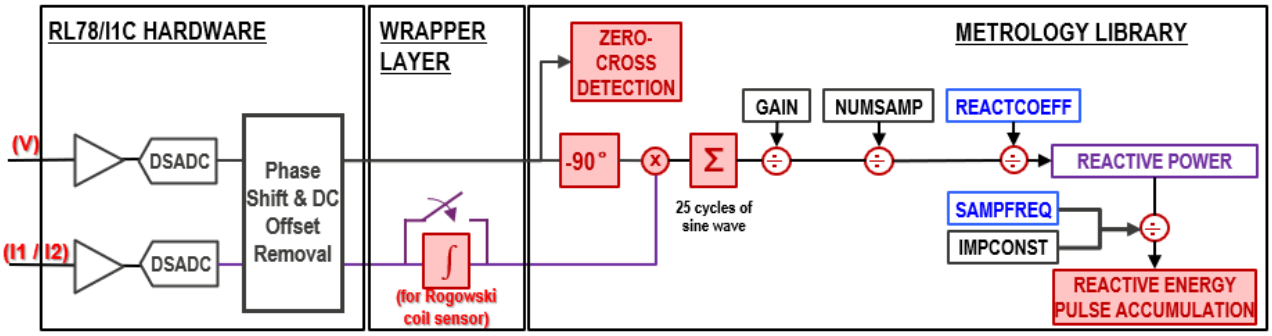
Note: For **versions 220915 and below**, the RMS accumulation is 25 cycles, while Power accumulation is 50 cycles. For the later version, both RMS and Power accumulation uses 25-line cycles accumulation length.

1.5.1 Measurement

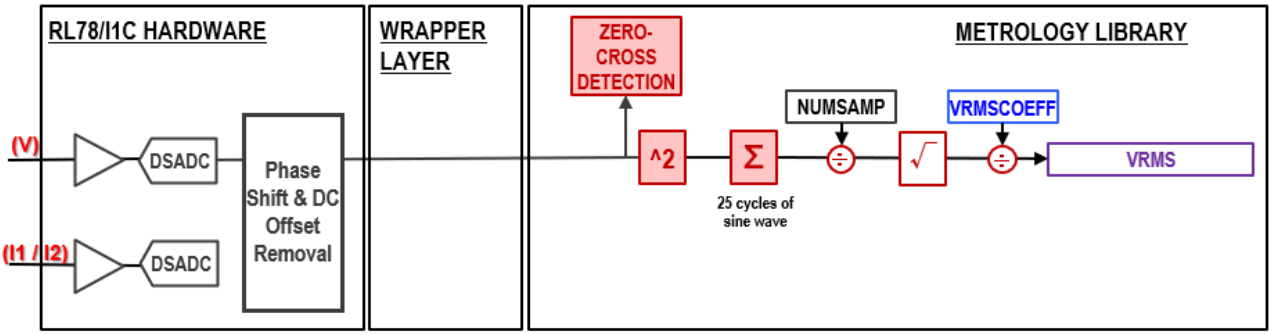
1.5.1.1 Active Power and Active Energy



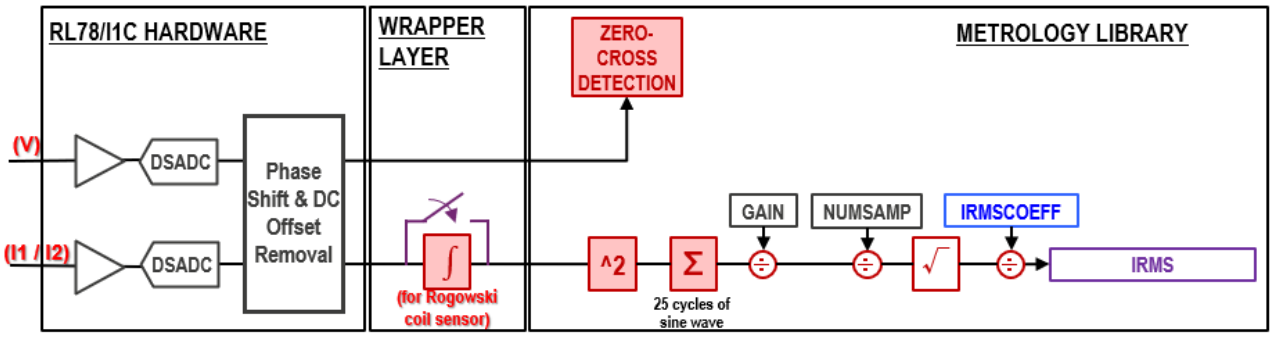
1.5.1.2 Reactive Power and Reactive Energy



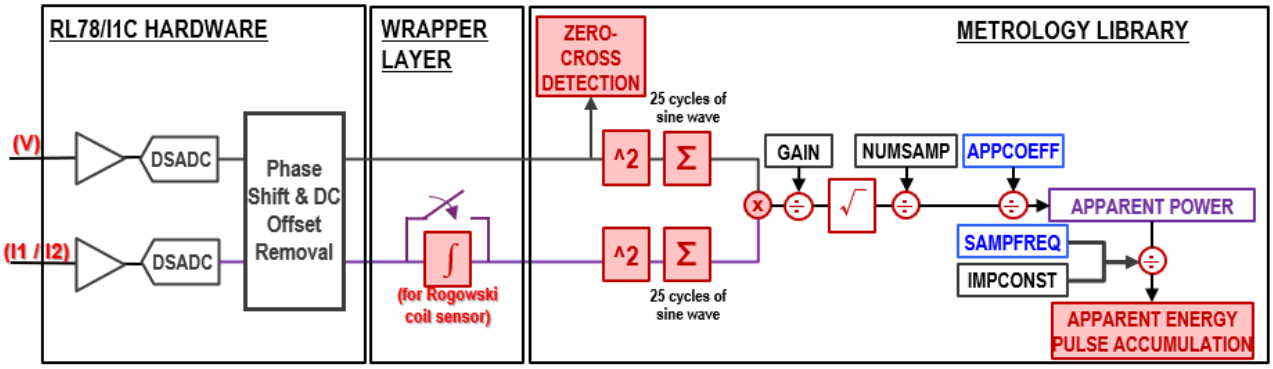
1.5.1.3 VRMS (true RMS)



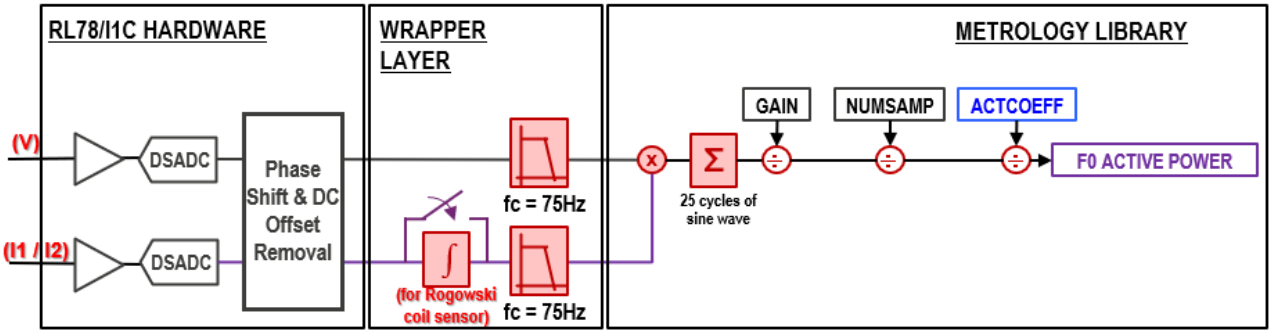
1.5.1.4 IRMS (true RMS, for both channel 1 and channel 2)



1.5.1.5 Apparent Power & Energy



1.5.1.6 Fundamental Active Power (F0 Active Power)



1.5.1.7 Power Factor (PF)

$$\text{Power Factor} = \frac{\text{Active Power}}{\text{Apparent Power}}$$

1.5.1.8 Line Frequency

$$\text{Line Frequency} = \frac{\text{Sampling Frequency} \times 25}{\text{Number of samples within 25-line cycles}}$$

1.5.2 Operation

1.5.2.1 Gain Switch

This feature is used to increase dynamic range of current meausrement (0.1A to over 100A) & increase accuracy at low current. To switch the gain, the library needs support from Wrapper layer to control the FPGA gain of MCU on I1 and I2 channels.

Following figures illustrate how the gain switch work inside library. This is just an example, the number of set (line cycles) actual used to do checking in library is not shown here. To configure the UPPER & LOWER threshold values, please refer to EM_SW_PROPERTY.

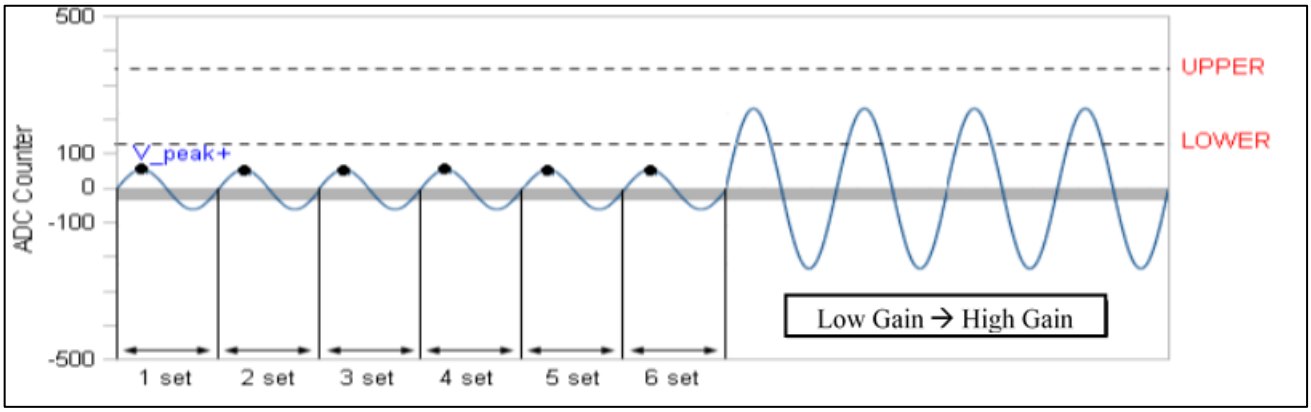


Figure 1-4 Low-High gain switching at low current sensing

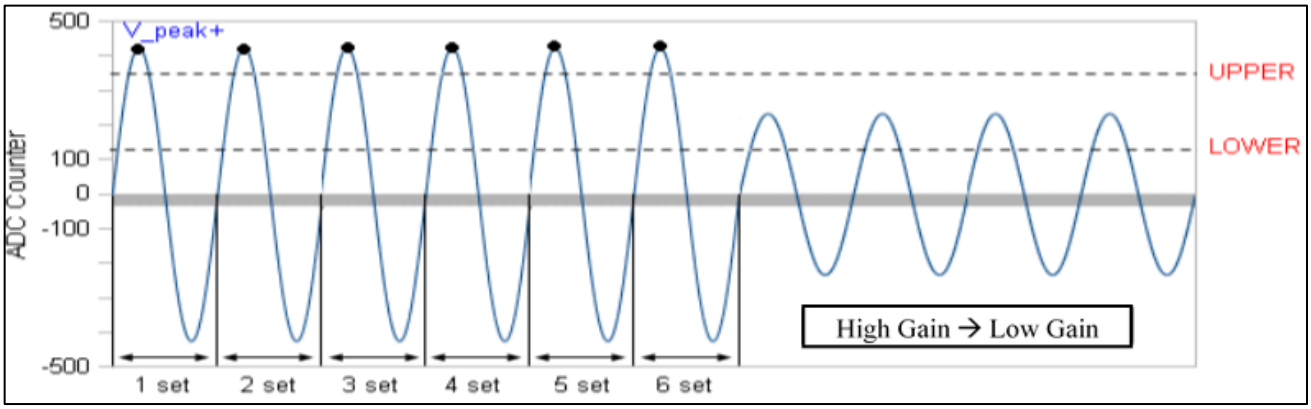


Figure 1-5 High-Low gain switching at high current sensing

1.5.2.2 Pulse and Energy Accumulation

Every meter has a list of meter constant parameters associated with it, usually expressed in imp/kWh or imp/kVarh

Based on the meter constant unit, each meter output pulse will indicate a set amount of energy consumed

In this metrology, with energy_pulse_ratio = 1 in EM_SW_PROPERTY, one pulse output will equal to one energy counter.

Calculated power will be used to calculate the accumulation step (amount of energy/pulse per DSAD interval). This is done by the metrology library in all energy accumulation mode except energy accumulation mode 0

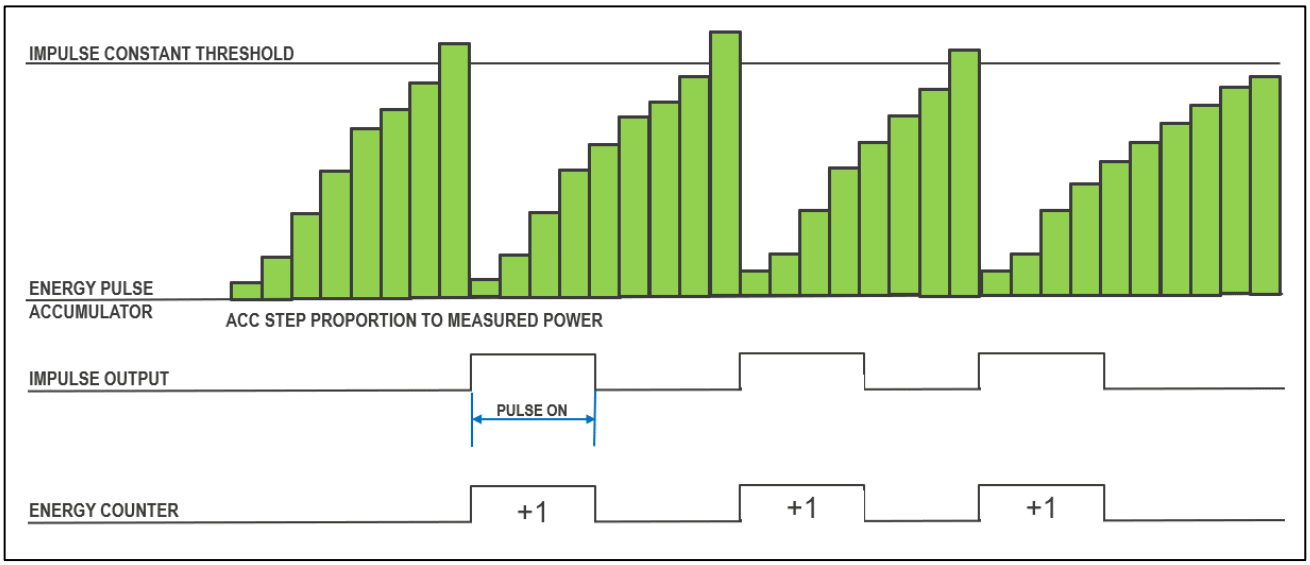


Figure 1-6 Illustration of energy and pulse accumulation

1.5.2.3 Energy accumulation mode

The table below shows the behavior of metrology during each Energy accumulation mode:

0	The library stopped updating step for energy accumulation. Users call EM_SetEnergyAccumulationPower to update
1	Library always use Phase channel power for energy accumulation
2	Library always use Neutral channel power for energy accumulation
3	Library selects between Phase and Neutral based on IRMS value (prioritize for Phase IRMS)

In mode 3, either Phase or Neutral power will be used for energy accumulation step update depending on configuration for Automatic line selection.

1.5.2.4 Automatic line selection

The metrology monitors the IRMS values of Phase and Neutral, and determines the line used for energy accumulation when in energy accumulation mode 3. The selection is done based on the following checks:

- I1 = 0 ; I2 = 0 : EM_LINE_PHASE
- I1 > 0 ; I2 = 0 : EM_LINE_PHASE
- I1 = 0 ; I2 > 0 : EM_LINE_NETRAL
- I1 > 0 ; I2 > 0: selection based on earth_diff_threshold in EM_SW_PROPERTY with following condition
  - ((I2 – I1) / I1 * 100) <= threshold: EM_LINE_PHASE
  - ((I2 – I1) / I1 * 100) > threshold: EM_LINE_NEUTRAL

1.5.2.5 Fixed sampling detection

The metrology library detects and uses the voltage signal zero crossing, for line cycles counting.

In normal condition (sine wave on voltage signal), accumulation will end upon reaching the 25th line cycles and the number of samples fall within the expected ranges, based on freq_high_threshold, freq_low_threshold configured in the EM_SW_PROPERTY.

Abnormal conditions where the zero cross of the signal seems out of range, detectable by the library, will cause the accumulation to end at a fixed rate rather than a fixed number of line cycles.

Some abnormal condition examples are:

- Completing a 25-line cycles detection, but the number of samples are equal to defined minimum number of samples. At no signal, ADC noises around zero are high-frequency signal, is likely an over-frequency condition.
- Reached maximum number of samples. With no ZX signal and a DC signal on voltage, is likely an under-frequency condition.

In fixed a sampling condition, the calculated line frequency will always be equal to the target_ac_source_frequency in EM_PLATFORM_PROPERTY.

These conditions likely indicate an abnormal signal on voltage line, but it is transparent when using library.

Table 1-6 - Conditions for fixed sampling

Number of line cycles	Number of samples				Expected Status
	samples < min	samples == min	min < samples < max	samples >= max	
>=25	x	o	x	x	Fixed - Over Fac
>=25	x	x	o	x	Normal
< 25	x	x	x	o	Fixed - Under Fac

x: means condition not met.

o: means condition met.

1.5.3 Event Status

The metrology library provides a few events status as seen in the EM_STATUS structure.

1.5.3.1 No-load event

To prevent energy accumulation of a no-load meter, the metrology checks the VRMS, IRMS and POWER accumulator to determine the no-load condition.

User configurable parameters for no-load detection are: irms_noload_threshold, power_noload_threshold and no_voltage_threshold in EM_SW_PROPERTY structure.

Tamper condition will still be supported under no-load condition, when there is current, but no voltage signal.

Table 1-7 - Effect of no-load conditions on reading calculated parameters:

Condition				**Expectation**					
VRMS >= VRMS Threshold	IRMS >= IRMS Threshold	Active >= Active Threshold	Reactive >= Reactive Threshold	VRMS	IRMS	Active	Reactive	Apparent	Fundamental active
0	0	x	x	0	0	0	0	0	0
0	1	x	x	0	Value	0	0	0	0
1	0	x	x	Value	0	0	0	0	0
1	1	0	0	Value	0	0	0	0	0
1	1	0	1	Value	Value	0	Value	Value	0
1	1	1	0	Value	Value	Value	0	Value	Value
1	1	1	1	Value	Value	Value	Value	Value	Value

x: don’t care condition.

0, 1 on Condition: means No or Yes

0, Value on Expectation: means reading masked 0.0f or actual calculated value

The no-load information will be available in the EM_STATUS structure for active power and reactive power.

There would be no energy registered in metrology if the no-load condition is present for both active and reactive in Phase and Neutral.

This no-load condition is updated in EM_TIMER_InterruptCallback function every 25-line cycle.

1.5.3.2 Sag and Swell event

Sag and swell detection are determined based on peak value of voltage signal.

The RMS threshold will be calculated based on the Sag and swell RMS threshold in EM_SW_PROPERTY structure, assuming a sine wave signal is provided.

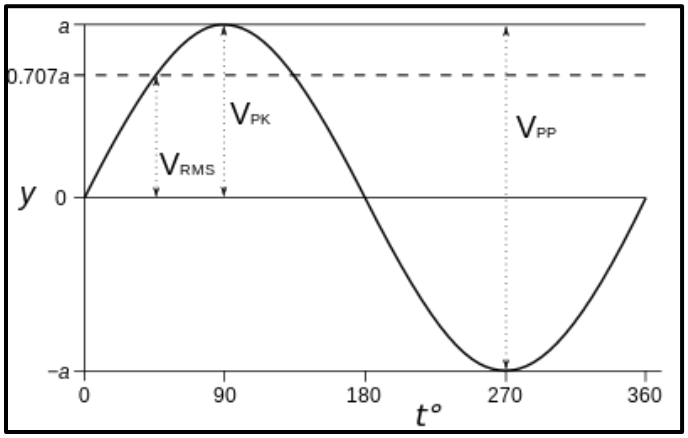


Figure 1-7 Sine wave RMS and peak

The V-peak would be compared with the threshold (only positive peak).

Versions below 221102:

- Peak checking per cycle
- One threshold for sag detection
- One threshold for swell detection
- One cycle count threshold for both sag and swell detection
- Edge detection: every cycle at ZX

After 3 continuous peaks detected below sag threshold in EM_ADC_IntervalProcessing function, a flag will be raised and checked in EM_TIMER_InterruptCallback function. The status would then be updated in timer processing as an event.
The event will only be released after 30 fixed & continuous checking in EM_TIMER without event occurrence.
Swell detection operates similarly, but instead checks the upper threshold.

Version 221102, and above:

- Peak checking per half-cycle
- Two thresholds for sag detection (hysteresis)
- Two thresholds for swell detection (hysteresis)
- Separate half-cycle count threshold for sag and swell detection.
- Edge detection:
    - Every half-cycle at ZX for sag rising, swell falling, swell rising detection.
    - Sample count for sag falling detection (sample count threshold calculated from configured half-cycle)

All checking and status are updated in EM_ADC_IntervalProcessing function, as Illustrated below:

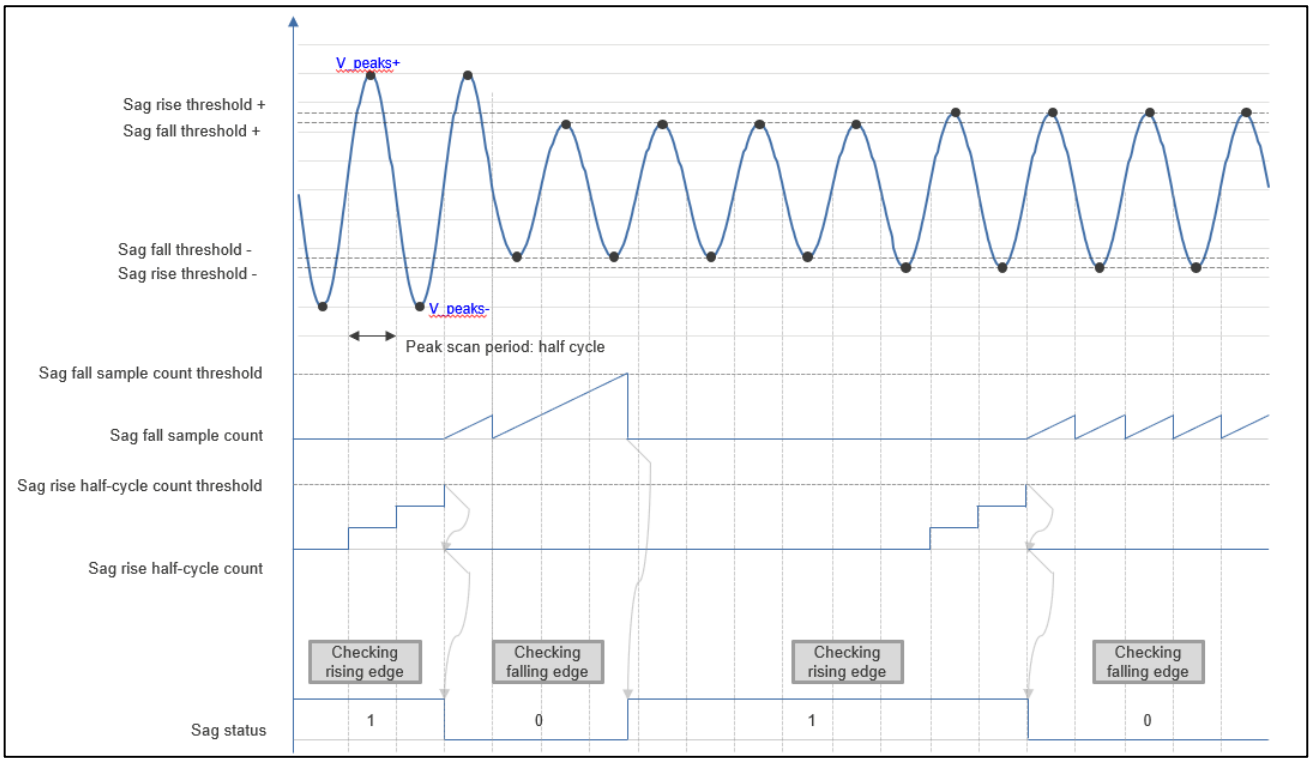


Figure 1-8 Sag falling and rising detection

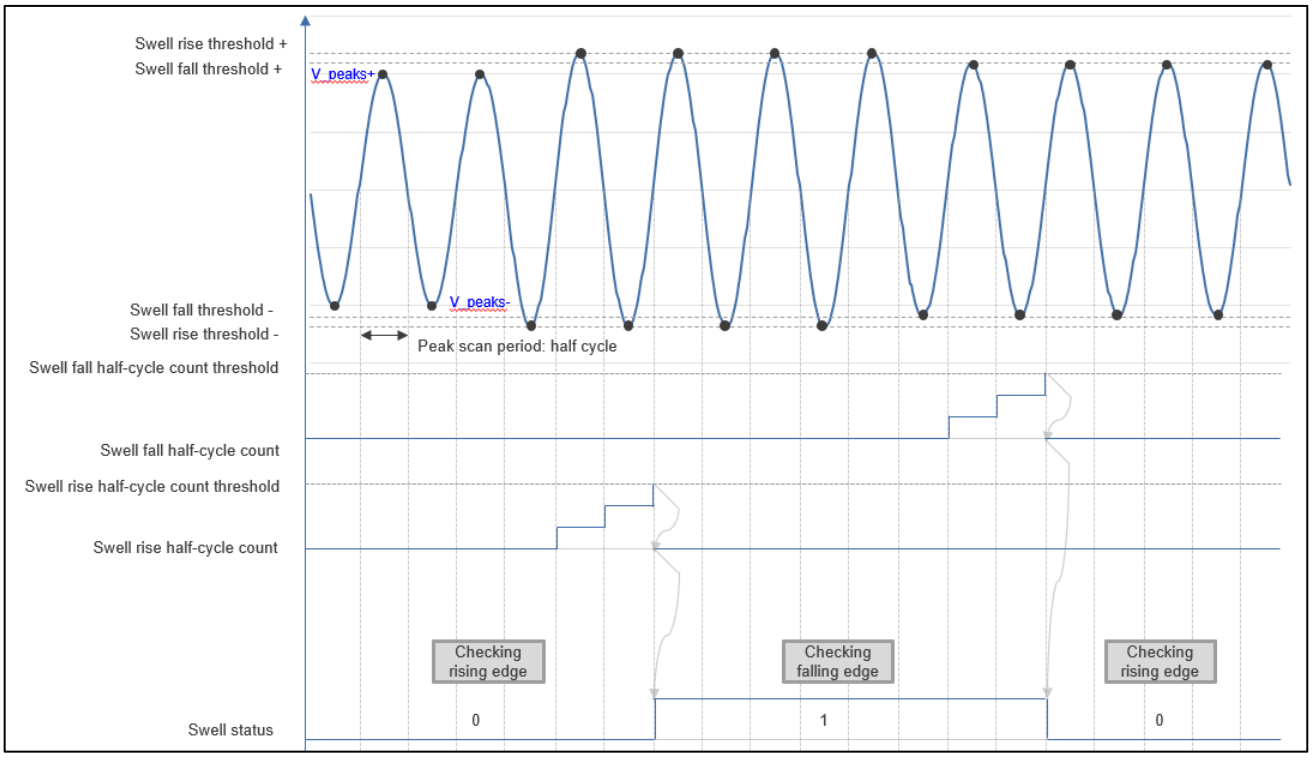


Figure 1-9 Swell falling and rising detection

1.6 Basic Operation Flow

1.6.1 Extraction of library information

The library is embedded with some constant variables containing its built information in ASCII string, ending with /0 terminated characters.

Normally these variables will be optimized by the compiler, if dead code optimization is enabled, and it's not used in the source code. During the initial phase, knowing the values of this information can be helpful to determine the correct type of library to use for development.

To use them in the source code, extern the following variables as shown below:

```
extern const uint8_t FAR_PTR g_em_lib_type[];

extern const uint8_t FAR_PTR g_em_lib_target_platform[];

extern const uint8_t FAR_PTR g_em_lib_compiler[];

extern const uint8_t FAR_PTR g_em_lib_git_revision[];

extern const uint8_t FAR_PTR g_em_lib_build_date[];    /* yyMMdd */
```

Another option is to force the compiler to keep the symbol through compiler options. The variables can then be watched on a watch window of the debugger. Please refer to the compiler user manual for more details on compiler options.

For CCRL, please check on the -SYmbol_forbid option.

CS+ IDE support for this option is in: Linker Options → Optimization → Symbols excluded from optimization of unreferenced symbol deletion.

1.6.2 Initialize drivers

The library requires hardware peripherals to be initialized. Please refer to Specification for more hardware details.

A skeleton code can be used as a starting point, but checks on the R_Systeminit function are advised to determine the code's peripheral initialization.

In the skeleton code, peripheral initialization is done similarly to the Code Generator:

```
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_port.h"
#include "r_cg_tau.h"
#include "r_cg_wdt.h"
#include "r_cg_dsadc.h"
#include "r_cg_mac32bit.h"
void R_Systeminit(void)
{
      R_PORT_Create();
      R_CGC_Create();
      R_TAU0_Create();
      R_WDT_Create();
      R_DSADC_Create();
      R_MAC32Bit_Create();
};
```

1.6.3 Wrapper Implementation

Essentially, the wrapper is used for API mapping and library configuration. All calls originate from the library to operate with the driver or read the settings.

A skeleton code can be used as a reference implementation of wrapper functions required by library. API details can be referred to in Function from wrapper.

The flowcharts below show the basic operation of the wrapper layer with the library (Initialization, Start, Stop, Interrupt).

1.6.3.1 Initialization

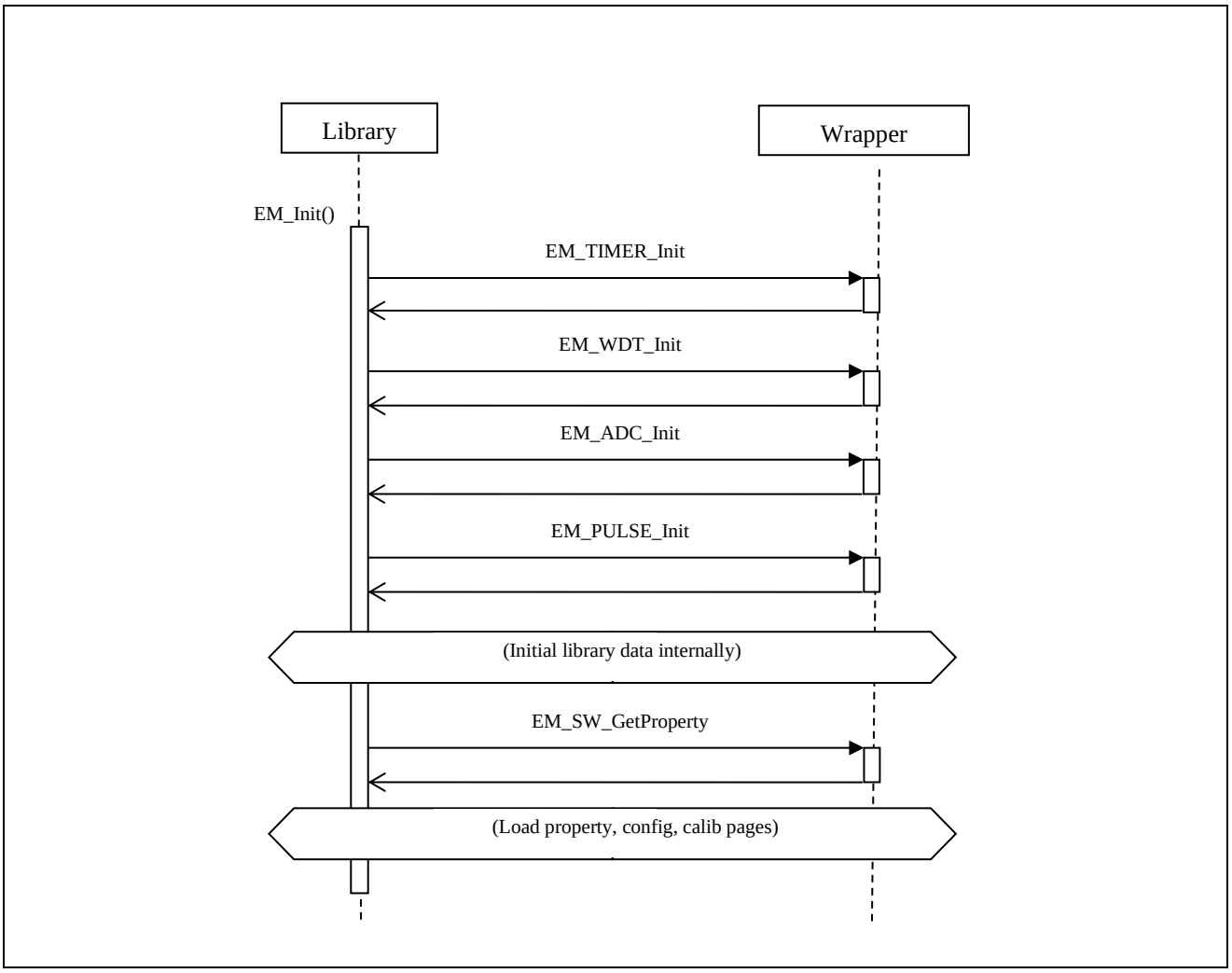


Figure 1-10 Wrapper initialization through library calls

1.6.3.2 Start

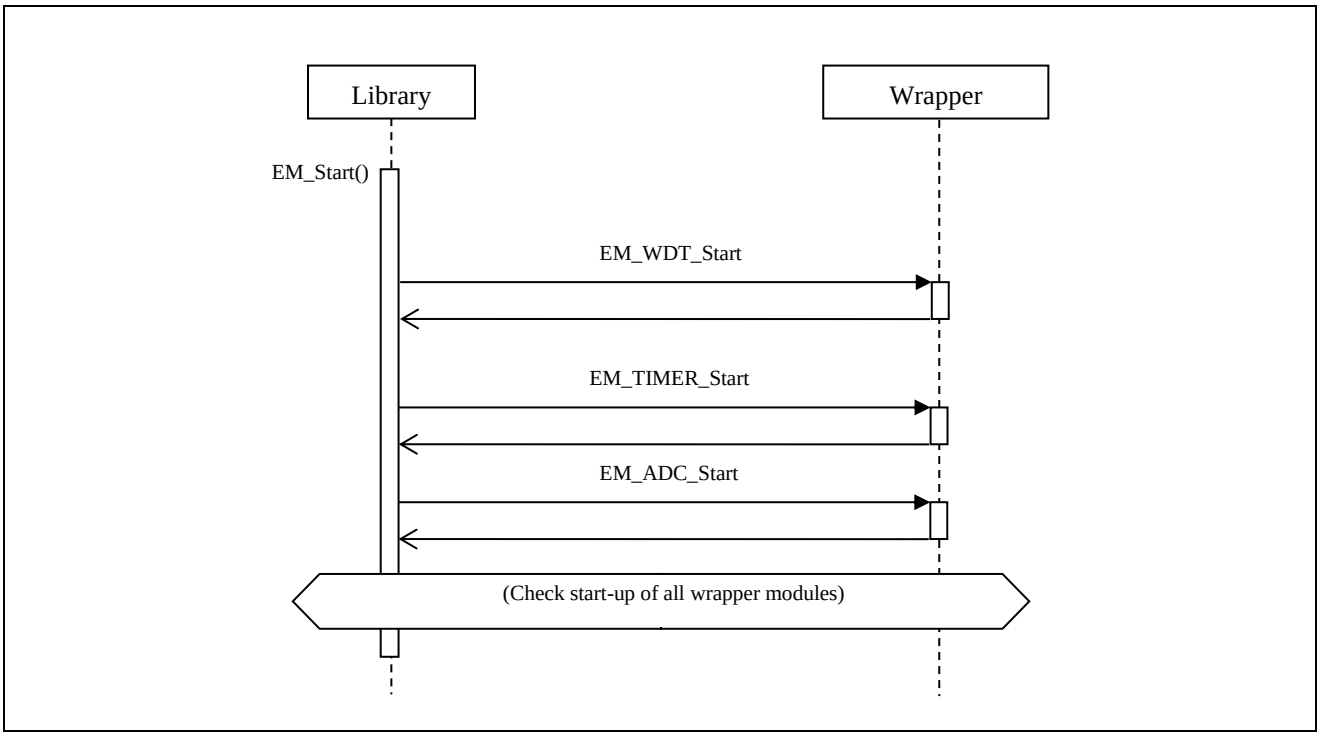


Figure 1-11 Wrapper module start-up through library calls

1.6.3.3 Stop

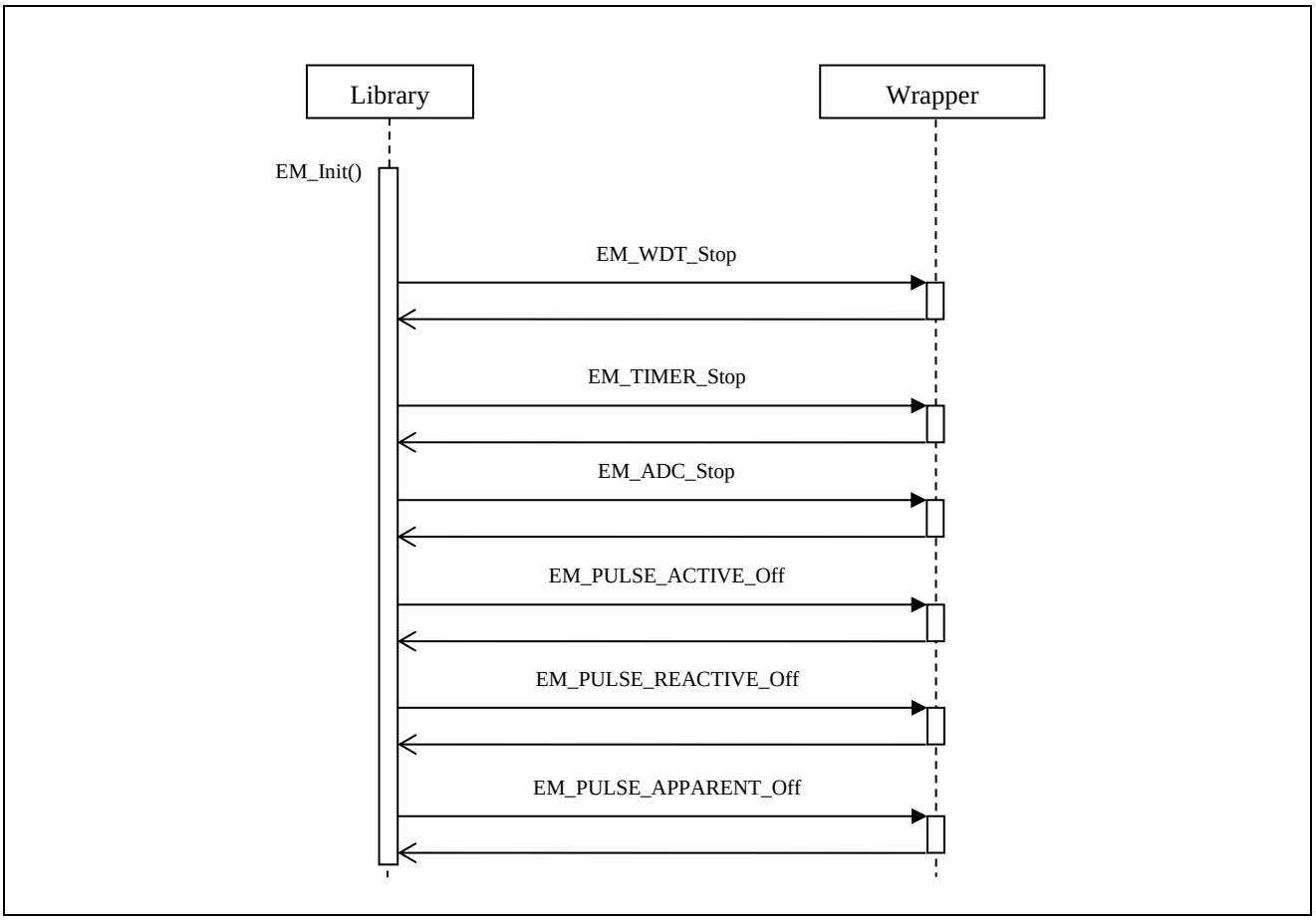


Figure 1-12 Wrapper modules stop through library calls

1.6.4 Register callback functions in Interrupts

a) ADC

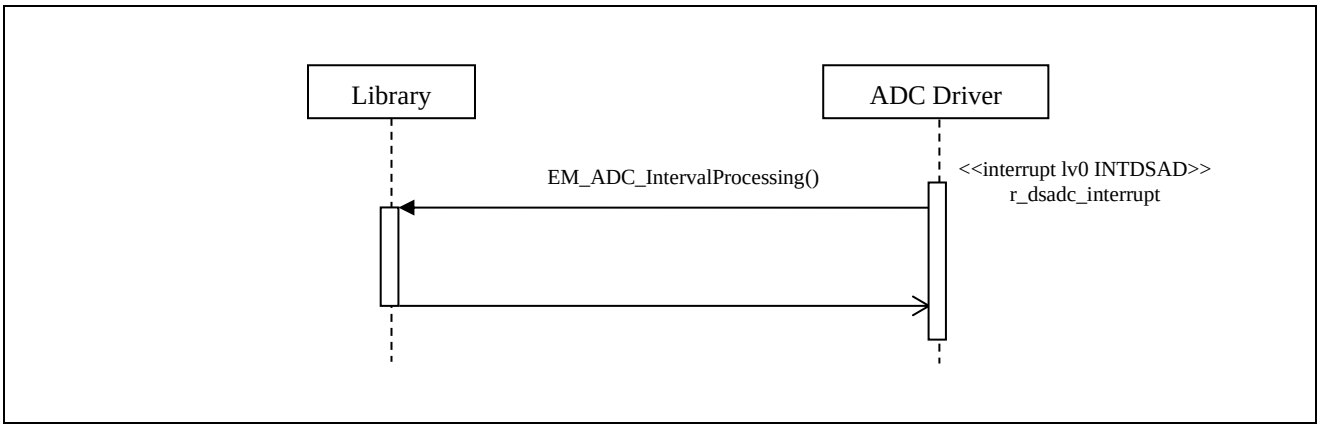


Figure 1-3 ADC Driver interrupt call to library

b) TIMER

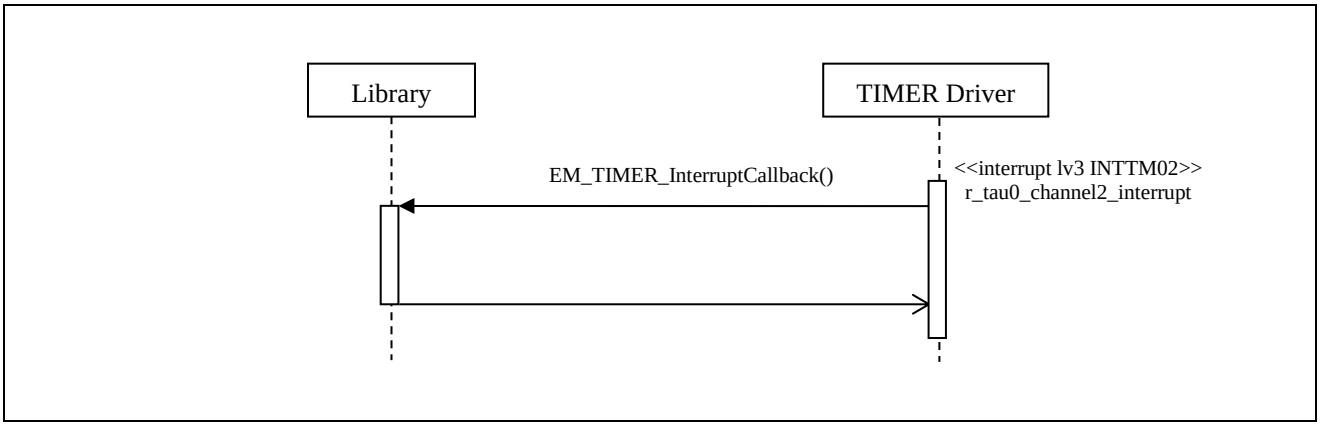


Figure 1-44 TIMER Driver interrupt call to library

1.6.5 Starting up metrology

After finishing all pre-requisite steps, the metrology can then be initialized and run by calling the following 2 functions: EM_Init and EM_Start

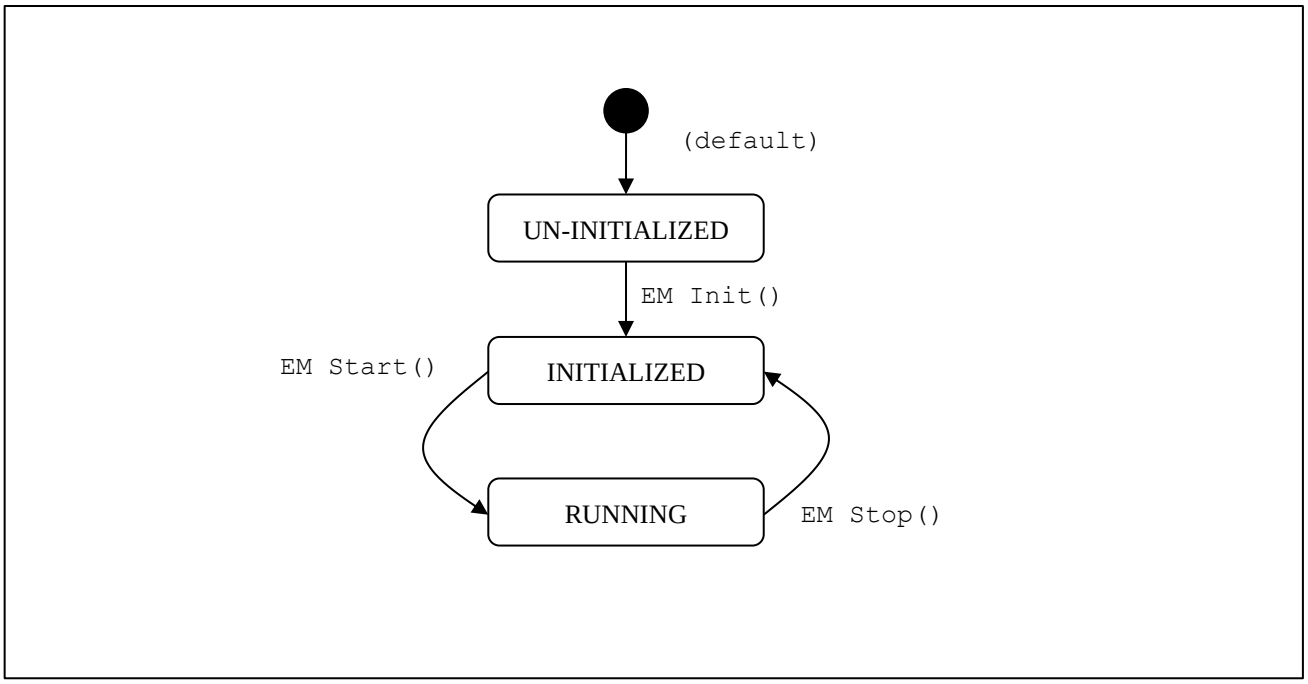


Figure 1-55 State of metrology library

2. Data types & Definitions

2.1 Data types

Table 2-1 Data types used in library

Data Type	Typedef
signed char	int8_t
unsigned char	uint8_t
signed short	int16_t
unsigned short	uint16_t
signed long	int32_t
unsigned long	uint32_t
float	float32_t
signed long long	int64_t
unsigned long long	uint64_t
double	double64_t
int32_t	EM_SW_SAMP_TYPE

2.2 Macro definitions

2.2.1 For EM_ERRCODE

Table 2-2 Macro used for return code of function:

Macro Name	Value	Explanation
EM_OK	0x00	OK
EM_CALIBRATING	0x02	Calibrating
EM_ERROR	0x80	General Error
EM_ERROR_PARAMS	0x81	Parameters error
EM_ERROR_NULL_PARAMS	0x82	Null Parameters inserted
EM_ERROR_NOT INITIALIZED	0x83	Metrology Initialization Error
EM_ERROR_NOT_RUNNING	0x84	Metrology Status not running
EM_ERROR_STILL_RUNNING	0x85	Metrology in Running State
EM_ERROR_STARTUP_ADC	0x8D	Fail to start DSAD peripheral
EM_ERROR_STARTUP_TIMER	0x8E	Fail to start TIMER peripheral
EM_ERROR_STARTUP_RTC	0x8F	Fail to start RTC peripheral
EM_ERROR_PLATFORM_PROPERTY_NULL	0x90	Property parameter error
EM_ERROR_PLATFORM_PROPERTY_TARGET_FREQ	0x91	Error for platform target frequency
EM_ERROR_SW_PROPERTY_NULL	0x92	Error for software property
EM_ERROR_SW_PROPERTY_ROUNDING	0x93	Error for software property rounding
EM_ERROR_SW_PROPERTY_GAIN	0x94	Error for software property gain
EM_ERROR_SW_PROPERTY_OPERATION	0x95	Error for software property operation
EM_ERROR_SW_PROPERTY_SAG_SWELL	0x96	Error for software property sag swell
EM_ERROR_CALIB_NULL	0xA0	Error for calibration
EM_ERROR_CALIB_PARAMS_COMMON	0xA1	Error for calibration parameters
EM_ERROR_CALIB_PARAMS_LINE1	0xA2	Error for calibration parameters Line1
EM_ERROR_CALIB_PARAMS_LINE2	0xA3	Error for calibration parameters Line2
EM_ERROR_CALIB_PARAMS_LINE3	0xA4	Error for calibration parameters Line3
EM_ERROR_CALIB_PARAMS_NEUTRAL	0xA5	Error for calibration parameters Neutral
EM_ERROR_CALIBRATING_NULL	0xF0	Error for calibrating
EM_ERROR_CALIBRATING_INVALID_LINE	0xF1	Error for calibrating invalid line
EM_ERROR_CALIBRATING_CYCLE	0xF2	Error for calibrating cycle
EM_ERROR_CALIBRATING_V_I	0xF3	Error for calibrating v & I
EM_ERROR_CALIBRATING_RTC_PERIOD	0xF4	Error for calibrating rtc period
EM_ERROR_CALIBRATING_IMAX_AND_NUM_OF_GAIN	0xF5	Error for calibrating imax and number of gains
EM_ERROR_CALIBRATING_MAX_GVALUE_SETTING	0xF6	Error for calibrating max gain value
EM_ERROR_CALIBRATING_NOT_STARTED	0XF9	Error for calibrating not started
EM_ERROR_CALIBRATING_FAILED_FS_OUT_RANGE	0XFA	Error for calibrating failed out of range
EM_ERROR_CALIBRATING_FAILED_IGAIN_OUT_RANGE	0XFB	Error for calibrating out of range
EM_ERROR_CALIBRATING_FAILED_OUT_ANGLE	0XFC	Error for calibrating out of angle
EM_ERROR_CALIBRATING_FAILED_REVERSE	0xFD	Error for calibrating reverse
EM_ERROR_CALIBRATING_FAILED_MAX_ANGLE	0xFE	Error for calibrating max angle
EM_ERROR_CALIBRATING_FAILED_V_LEAD_I	0xFF	Error for calibrating V lead I

2.2.2 For EM_CONSTRAINT

Table 2-3 Macro used to limit the setting values of `EM_CONSTRAINT` structure

Macro Name	Value	Explanation
EM_GAIN_PHASE_NUM_LEVEL_MIN	1	Gain phase number level min
EM_GAIN_PHASE_NUM_LEVEL_MAX	2	Gain phase number level max
EM_GAIN_NEUTRAL_NUM_LEVEL_MIN	1	Gain neutral number level min
EM_GAIN_NEUTRAL_NUM_LEVEL_MAX	2	Gain neutral number level max
EM_SAMPLING_FREQUENCY_MIN	1200	Sampling Frequency Min (Hz)
EM_SAMPLING_FREQUENCY_MAX	4000	Sampling Frequency Max (Hz)
EM_SAMPLING_FREQUENCY_CALIBRATION	3906	Sampling Frequency for calibration (Hz)
EM_TARGET_AC_SOURCE_FREQ_SELECTION0	50	Target AC Source Frequency Selection 0 (Hz)
EM_TARGET_AC_SOURCE_FREQ_SELECTION1	60	Target AC Source Frequency Selection 1 (Hz)
EM_MAX_ROUNDING_DIGIT	4	Maximum Round decimal
EM_VOL_CHANNEL_NUM	1	Voltage channel number
EM_CURRENT_CHANNEL_NUM	2	Current channel number
EM_VOL_LOW_MIN	10.0f	Voltage low min
EM_FREQ_LOW_MIN	40.0f	Frequency low min
EM_FREQ_HIGH_MAX	70.0f	Frequency low max
EM_EARTH_DIFF_THRES_MAX	50.0f	Earth diff threshold max
EM_PULSE_ON_TIME_MIN	10.0f	Pulse On Time Min
EM_VRMS_COEFF_MIN	0.1f	VRMS Co-efficient min
EM_I1RMS_COEFF_MIN	0.1f	I1RMS Co-efficient min
EM_I2RMS_COEFF_MIN	0.1f	I2RMS Co-efficient min
EM_ACT_POWER_COEFF_MIN	0.1f	Active Power Co-efficient min
EM_REA_POWER_COEFF_MIN	0.1f	Reactive Power Co-efficient min
EM_APP_POWER_COEFF_MIN	0.1f	Apparent Power Co-efficient min

2.3 Structure definitions

This section provides the details of the structures used in the library.

2.3.1 EM_STATUS

Explanation

The `EM_STATUS` structure holds the status of all measured parameters on the meter library. Parameters can be returned by calling to the `EM_GetStatus`. Each bit field `(uint16_t:1)` named below indicate a status of, 1 is occurred, 0 is recovered. The following table provides details on the members of the `EM_STATUS` structure.

Structure (2 bytes)

Datatype	Structure element	Explanation
uint16_t:1	noload_active	No load status of Active Power
uint16_t:1	noload_reactive	No load status of Reactive Power
uint16_t:1	noload_active2	No load status of Active Power 2
uint16_t:1	noload_reactive2	No load status of Reactive Power 2
uint16_t:1	is_voltage_sag	Voltage Sag
uint16_t:1	is_voltage_swell	Voltage Swell

2.3.2 EM_PLATFORM_PROPERTY

Explanation

The `EM_PLATFORM_PROPERTY` structure holds information required to configure the property of the platform.

Structure (2 bytes)

Datatype	Structure element	Explanation
uint8_t	target_ac_source_frequency	Target AC Source frequency (50Hz or 60Hz)
uint8_t	reserved	(Not use)

2.3.3 EM_CALIBRATION

Explanation

The `EM_CALIBRATION` structure holds the calibration information, used to configure the library using the `EM_Init` or `EM_SetCalibInfo` functions.

Structure (56 bytes)

Datatype	Structure element		Explanation
float32_t (4 bytes)	sampling_frequency		Actual sampling frequency of the meter
struct (36 bytes)	coeff		Specify co-efficient of input signal
	float32_t	vrms	VRMS Co-efficient
	float32_t	i1rms	I1RMS Co-efficient – Phase Current
	float32_t	i2rms	I2RMS Co-efficient – Neutral Current
	float32_t	active_power	Active power coefficient (I1)
	float32_t	reactive_power	Reactive power coefficient (I1)
	float32_t	apparent_power	Apparent power coefficient (I1)
	float32_t	active_power2	Active power coefficient (I2)
	float32_t	reactive_power2	Reactive power coefficient (I2)
	float32_t	apparent_power2	Apparent power coefficient (I2)
struct (8 bytes)	sw_phase_correction		Phase correction list (degree)
	float32_t FAR_PTR *	i1_phase_degrees	I1 Phase Angle Degree List
	float32_t FAR_PRT *	I2_phase_degrees	I2 Phase Angle Degree List
struct (8 bytes)	sw_gain		Gain value list
	float32_t FAR_PTR *	i1_gain_values	I1 Gain Value List
	float32_t FAR_PTR *	i2_gain_values	I2 Gain Value List

Structure Element Definitions

sampling_frequency

Set the calibrated sampling frequency of meter, in Hz.

coeff struct

Set vrms, i1rms, i2rms for RMS value calibration (VRMS, I1RMS, I2RMS).
Set active_power, reactive_power, apparent_power for Power & Energy calibration on I1 channel.
Set active_power2, reactive_power2, apparent_power2 for Power & Energy calibration on I2 channel.

The limit of set value is defined by macros as following:
`EM_VRMS_COEFF_MIN`
`EM_I1RMS_COEFF_MIN`
`EM_I2RMS_COEFF_MIN`
`EM_ACT_POWER_COEFF_MIN`
`EM_REA_POWER_COEFF_MIN`
`EM_APP_POWER_COEFF_MIN`

sw_phase_correction

Set phase correction between the Voltage and I1 channels, in degree, on `i1_phase_degrees` list.
Set phase correction between the Voltage and I2 channels, in degree, on `i2_phase_degrees` list.

- If the gain switch library version is used (WSUR, WQUR, WQFR), a max of 2 gain levels is supported. If only 1 level of gain is required, set the unused gain level to 0.
- If a non-gain switch library version is used, only set values to `i1_phase_degrees[0]` and `i2_phase_degrees[0],` keep others as 0.

sw_gain

Set gain value for I1 channels on `i1_gain_values` list.
Set gain value for I2 channels on `i2_gain_values` list.

A max of 2 levels of gain is support. If only 1 gain level is required, set the unuse gain level as 1.0f

- If the gain switch library version is used (WSUR, WQUR, WQFR), a max of 2 gain levels is supported. If only 1 level of gain is required, set the unused gain level to 0.
- If a non-gain switch library version is used, only set values to `i1_phase_degrees[0]` and `i2_phase_degrees[0],` keep others as 0.

2.3.4 EM_ENERGY_COUNTER

Explanation

The `EM_ENERGY_COUNTER` structure contains the Metrology Energy accumulation counter formatted to uint64_t:

Structure (64 bytes)

Datatype	Structure element	Explanation
uint64_t	active_imp	Active Import Energy accumulation counter
uint64_t	active_exp	Active Export Energy accumulation counter
uint64_t	reactive_ind_imp	Reactive Inductive Import Energy accumulator counter
uint64_t	reactive_ind_exp	Reactive Inductive Export Energy accumulator counter
uint64_t	reactive_cap_imp	Reactive Capacitive Import Energy accumulator counter
uint64_t	reactive_cap_exp	Reactive Capacitive Export Energy accumulator counter
uint64_t	apparent_imp	Apparent Import Energy accumulation counter
uint64_t	apparent_exp	Apparent Export Energy accumulation counter

2.3.5 EM_ENERGY_VALUE

Explanation

Library version 2209515 or below: The `EM_ENERGY_VALUE` structure contains the Metrology Energy accumulation counter formatted to float32_t:

Structure (32 bytes)

Datatype	Structure element	Explanation
float32_t	active_imp	Active Import Energy in Wh
float32_t	active_exp	Active Export Energy in Wh
float32_t	reactive_ind_imp	Reactive Inductive Import Energy in VArh
float32_t	reactive_ind_exp	Reactive Inductive Export Energy in VArh
float32_t	reactive_cap_imp	Reactive Capacitive Import Energy in VArh
float32_t	reactive_cap_exp	Reactive Capacitive Export Energy in VArh
float32_t	apparent_imp	Apparent Import Energy in VAh
float32_t	apparent_exp	Apparent Export Energy in VAh

Library version above 220915: The `EM_ENERGY_VALUE` structure contains the Metrology Energy accumulation counter formatted to uint64_t integer and float32_t decimal:

Structure (96 bytes)

Datatype	Structure element		Explanation
struct (64 bytes)	integer		Integer part of energy value
	uint64_t	active_imp	Active Import Energy in Wh
	uint64_t	active_exp	Active Export Energy in Wh
	uint64_t	reactive_ind_imp	Reactive Inductive Import Energy in VArh
	uint64_t	reactive_ind_exp	Reactive Inductive Export Energy in VArh
	uint64_t	reactive_cap_imp	Reactive Capacitive Import Energy in VArh
	uint64_t	reactive_cap_exp	Reactive Capacitive Export Energy in VArh
	uint64_t	apparent_imp	Apparent Import Energy in VAh
	uint64_t	apparent_exp	Apparent Export Energy in VAh
struct (32 bytes)	decimal		Decimal part of energy value
	float32_t	active_imp	Active Import Energy in Wh
	float32_t	active_exp	Active Export Energy in Wh
	float32_t	reactive_ind_imp	Reactive Inductive Import Energy in VArh
	float32_t	reactive_ind_exp	Reactive Inductive Export Energy in VArh
	float32_t	reactive_cap_imp	Reactive Capacitive Import Energy in VArh
	float32_t	reactive_cap_exp	Reactive Capacitive Export Energy in VArh
	float32_t	apparent_imp	Apparent Import Energy in VAh
	float32_t	apparent_exp	Apparent Export Energy in VAh

2.3.6 EM_CALIB_ARGS

Explanation

The `EM_CALIB_ARGS` structure is Metrology Calibration Arguments

Structure (22 bytes)

Datatype	Structure element	Explanation
uint16_t	rtc_period	1000 or 500 only (Corresponding to 1s or 0.5s period interrupt)
uint16_t	max_gvalue	Maximum gain value
uint8_t	stable_ndelay	Advance option: Internal delay between each phase and gain set
uint8_t	cycle	Number of line cycle used for fs, gain and coefficient calibration
uint8_t	cycle_angle	Number of line cycle used for signals angle calibration
float32_t	v	Voltage value on reference power supply
float32_t	i	Current value on reference power supply
float32_t	imax	Max measuring current for calibrating meter
EM_LINE	line_v	Voltage line selection
EM_LINE	line_i	Current line selection

2.3.7 EM_CALIB_OUTPUT

Explanation

The `EM_CALIB_OUTPUT` structure is Metrology Calibration Output

Structure (34 bytes)

Datatype	Structure element	Explanation
float32_t	fs	Sampling frequency
float32_t	gain	Current gain
float32_t	gain1	Current gain 1
float32_t	vcoeff	Voltage coefficient
float32_t	icoeff	Current coefficient
float32_t	pcoeff	Power coefficient, for active, reactive, and apparent
float32_t	angle_error	Angle error between current and voltage
float32_t	angle_error1	Angle error between current and voltage gain 1
EM_CALIB_STEP	step	Current step during calibration

2.3.8 EM_SAMPLES

Explanation

The `EM_SAMPLES` structure is parsing V & I samples to metrology for calculation.

Structure (24 bytes)

Datatype	Structure element	Explanation
EM_SW_SAMP_TYPE	v	Voltage
EM_SW_SAMP_TYPE	i1	Current phase
EM_SW_SAMP_TYPE	i2	Current neutral
EM_SW_SAMP_TYPE	v_fund	Filtered voltage for fundamental calculation
EM_SW_SAMP_TYPE	i1_fund	Filtered current phase for fundamental calculation
EM_SW_SAMP_TYPE	i2_fund	Filtered current neutral for fundamental calculation

2.3.9 EM_OPERATION_DATA

Explanation

The `EM_OPERATION_DATA` structure containing library running data: energy counter, energy/pulse accumulation remainder

Structure (100 bytes)

Datatype	Structure element		Explanation
EM_ENERGY_COUNTER	energy_counter		Energy counter
struct (36 bytes)	remainder		Remainder of energy counter and pulse
	uint32_t	active_imp	Active import
	uint32_t	active_exp	Active export
	uint32_t	reactive_ind_imp	Reactive inductive import
	uint32_t	reactive_ind_exp	Reactive inductive export
	uint32_t	reactive_cap_imp	Reactive capacitive import
	uint32_t	reactive_cap_exp	Reactive capacitive export
	uint32_t	apparent_imp	Apparent import
	uint32_t	apparent_exp	Apparent export
	uint8_t	pulse_active	Pulse active count
	uint8_t	pulse_reactive	Pulse reactive count
	uint8_t	pulse_apparent	Pulse apparent count
	uint8_t	pading	Padding

2.3.10 EM_SW_PROPERTY

Explanation

The `EM_SW_PROPERTY` structure holds information that is required to configure the property of wrapper layer.

Structure (84 bytes)

Datatype	Structure element		Explanation
struct (20 bytes)	adc		Gain switching function
	uint8_t	gain_phase_num_level	Number of gains for I1 channel
	uint32_t	gain_phase_upper_threshold	I1 Upper threshold to switch to higher gain
	uint32_t	gain_phase_lower_threshold	I1 Lower threshold to switch to lower gain
	uint8_t	gain_neutral_num_level	Number of gains for I2 channel
	uint32_t	gain_neutral_upper_threshold	I2 Upper threshold to switch to higher gain
	uint32_t	gain_neutral_lower_threshold	I2 Lower threshold to switch to lower gain
struct (36 bytes)	operation		Common operation
	float32_t	irms_noload_threshold	Set the threshold for IRMS No Load Detection (Ampere)
	float32_t	power_noload_threshold	Set the threshold for Power No Load Detection (Watt)
	float32_t	no_voltage_threshold	Voltage lowest RMS level (Volt)
	float32_t	freq_low_threshold	Lowest frequency (Hz)
	float32_t	freq_high_threshold	Highest frequency (Hz)
	float32_t	earth_diff_threshold	Different threshold for selecting Phase-Neutral (%)
	uint32_t	meter_constant	Meter constant (imp/KWh)
	float32_t	pulse_on_time	Pulse on time (ms)
	uint8_t	energy_pulse_ratio	Ratio of energy step vs pulse meter constant: 1-254
	uint8_t	pulse_export_direction	Option to output pulse for export direction: 0 (disable) or 1 (enable)
	uint8_t	enable_pulse_reactive	Option to enable reactive pulse output: 0 (disable) or 1 (enable)
	uint8_t	enable_pulse_apparent	Option to enable apparent pulse output: 0 (disable) or 1 (enable)
struct (4 bytes)	rounding		Rounding for Measured Parameters
	uint8_t	power	Rounding digits for power
	uint8_t	rms	Rounding digits for rms value
	uint8_t	freq	Rounding digits for frequency
	uint8_t	pf	Rounding digits for pf
struct (4 bytes)	samp		Sampling timer module
	float32_t	shifting90_interpolation_error	Reactive power amplitude compensation (%)
struct (20 bytes)	sag_swell		Sag and Swell detection
	float32_t	sag_rms_rise_threshold	The VRMS threshold of Sag rising edge
	float32_t	sag_rms_fall_threshold	The VRMS threshold of Sag falling edge
	float32_t	swell_rms_rise_threshold	The VRMS threshold of Swell rising edge
	float32_t	swell_rms_fall_threshold	The VRMS threshold of Swell rising edge
	uint16_t	sag_detection_half_cycle	Number of signal half cycle to detect Sag event, 0 means no detection
	uint16_t	swell_detection_half_cycle	Number of signal half cycle to detect Swell event, 0 means no detection

gain_phase_num_level

Specify how many gains used for Phase channel (I1). This setting is mandatory and will affect the EM_CALIBRATION struct, on `i1_phase_degree` and `i1_gain_values`.

Minimum: 1

Maximum: 2

Example,

- If 2 gains are used, the first 2 elements of the array `i1_phase_degree` must have phase error values, while others value can be kept as 0. Next, the first element on `i1_gain_values` must be 1.0f, and the next should have a specified gain value, e.g., 16.0f.
- If 1 is specified, means single gain.

gain_neutral_num_level

Specify how many gains used for Neutral channel (I2). This setting is mandatory and will affect the EM_CALIBRATION struct, on `i2_phase_degree` and `i2_gain_values`.

Minimum: 1

Maximum: 2

Example,

- If 2 gains are used, the first 2 elements of the array `i2_phase_degree` must have phase error values, while others value can be kept as 0. Next, the first element on `i2_gain_values` must be 1.0f, and the next should have a specified gain value, e.g., 16.0f.
- If 1 is specified, means single gain.

gain_phase_upper_threshold
gain_phase_lower_threshold
gain_neutral_upper_threshold
gain_neutral_lower_threshold

Specify the upper and lower thresholds (in DSAD steps) for the gain switching network on I1 and I2 channel.

The upper/lower ratio should be greater than the ratio of the second gain / first gain in `i1_gain_values` and `i2_gain_values` to keep the signal within range after switching gain.

irms_noload_threshold

Specify the amplitude threshold to mask the accumulated value of IRMS (in Ampere) during No-Load operation.

power_noload_threshold

Specify the amplitude threshold to mask accumulated value of power (used commonly for both active and reactive, unit can understand in Watt or Var) during No-Load operation.

no_voltage_threshold

Specify the amplitude threshold to mask accumulated value of VRMS (in Volt) Minimum: `EM_VOL_LOW_MIN`

freq_low_threshold

freq_high_threshold

Specify the frequency measurement range. The frequency low high threshold will be used to calculate number of samples in fixed sampling accumulation.

Minimum: `EM_FREQ_LOW_MIN`

Maximum: `EM_FREQ_HIGH_MAX`

earth_diff_threshold

Set ELT threshold (%) for the detection of Earth Load tamper.

Maximum: `EM_EARTH_DIFF_THRES_MAX`

meter_constant

Set the meter constant in imp/kWh for energy and pulse output.
This setting is not checked.

Pulse_on_time

Indicate the time of pulse on duration (in ms)

Minimum: `EM_PULSE_ON_TIME_MIN`

energy_pulse_ratio

Ratio of energy counter and number of pulse output. The setting is normally defined as 1.

By increasing the ratio, the energy resolution will also increase. But at the cost of increasing the size of the energy counter. Take note to ensure the meter constant and energy_pulse_ratio does not exceed the 48-bit energy counter.

Minimum: `1`

Maximum: `254`

pulse_export_direction

Setting to enable pulse for export direction. This setting is normally defined as 1, to enable pulse export direction.

If enabled, when current flowing is in the reverse direction (export), pulse output will occur. If disable, pulse will not occur.

Energy accumulation still occurs regardless of this option.

enable_pulse_reactive

enable_pulse_apparent

Setting to enable pulse output for reactive and apparent energy correspondingly.

rounding (power, rms, freq, pf)

Specify the number of digits for rounding before returning the calculated parameters from metrology

Samp (shifting90_interpolation_error)

Configure the amplitude compensation for reactive power (amplitude may be affected by voltage sample interpolation)

$$y = \frac{x}{1 + (shifting90\_interpolation\_error / 100)}$$

x: reactive accumulator just before the division of coefficient

y: compensated reactive accumulator.

sag_rms_rise_threshold

sag_rms_fall_threshold

Specify the RMS threshold for sag rising and falling edge detection (in Volt)

swell_rms_rise_threshold

swell_rms_fall_threshold

Specify the RMS threshold for swell rising and falling edge detection (in Volt)

sag_detection_half_cycle

Specify number of half cycle for sag detection

Note that for sag falling edge detection, the configured number of half cycle is translated to equivalent number of samples based on calibrated sampling frequency and line frequency used for detection.

swell_detection_half_cycle

Specify number of half cycle for swell detection

2.4 Enum definitions

This section provides the details of the enumerations used in the library.

2.4.1 EM_LINE

Explanation

The `EM_LINE` enumeration groups all selections of power measurements line. (Phase & Neutral).

Enum values (1 byte)

Enum Value	Significance
EM_LINE_PHASE = 0x00	Phase line
EM_LINE_NEUTRAL = 0x04	Neutral line

2.4.2 EM_CALIB_STEP

Explanation

The `EM_CALIB_STEP` enumeration groups the function and steps of the metrology calibration.

Enum values (1 byte)

Enum Value	Significance
EM_CALIB_STEP_NOT_INITIATED	Metrology calibration not initialed
EM_CALIB_STEP_FS	Metrology Calibration Frequency Sampling Step
EM_CALIB_STEP_IGAIN	Metrology Calibration Current Gain Step
EM_CALIB_STEP_SIGNALS	Metrology Calibration Signal Coefficient Step
EM_CALIB_STEP_ANGLE	Metrology Calibration Angle Adjustment Step

2.4.3 EM_PF_SIGN

Explanation

The `EM_PF_SIGN` enumeration groups all selections of power factor sign (Lead, Lag, Unity).

Enum values (1 byte)

Enum Value	Significance
PF_SIGN_LEAD_C = -1	Current (phase or neutral) lead Voltage
PF_SIGN_UNITY = 0	Current (phase or neutral) and Voltage are unity.
PF_SIGN_LAG_L = 1	Current (phase or neutral) lag Voltage

2.4.4 EM_SYSTEM_STATE

Explanation

The `EM_SYSTEM_STATE` enumeration groups all operation state of library.

Enum values (1 byte)

Enum Value	Significance
SYSTEM_STATE_UNINITIALIZED = 0	Library is not initialized (un-initialized)
SYSTEM_STATE_INITIALIZED = 1	Library is already initialized (configured)
SYSTEM_STATE_RUNNING = 2	Library is running

2.5 Function definitions

2.5.1 Provided functions

Table 2-4 Provided APIs list

	API Name	Description
Control Library Operation	EM_Init	Initial Metrology Library
	EM_Start	Start Metrology Library
	EM_Stop	Stop Metrology Library
	EM_GetSystemState	Get the System state of Metrology Library
	EM_GetStatus	Get the status of all measured parameters
	EM_GetLastSagDuration Note1	Get last sag event cycle duration
	EM_GetLastSwellDuration Note1	Get last swell event cycle duration
	EM_GetEnergyAccumulationMode	Get energy counter accumulation mode
	EM_SetEnergyAccumulationMode	Set energy counter accumulation mode
	EM_SetEnergyAccumulationPower	Set energy accumulation power to metrology(manual)
	EM_GetOperationData	Get Metrology internal data (backup purpose)
	EM_SetOperationData	Set Metrology internal data (restore purpose)
Calibration	EM_GetCalibInfo	Get the current calibration page on library
	EM_SetCalibInfo	Set calibration information of the library by calibration page
	EM_CalibInitiate	Initiate calibration
	EM_CalibRun	Run calibration for the step
	EM_RTC_CalibInterruptCallback	Register into 0.5s or 1s timer
Output Measurement	EM_GetActivePower	Read the measured active power (Watt)
	EM_GetFundamentalActivePower	Read the measured fundamental active Power (Watt)
	EM_GetReactivePower	Read the measured reactive power (VAr)
	EM_GetApparentPower	Read the measured apparent power (VA)
	EM_GetEnergyCounter	Read the accumulating energy counter
	EM_EnergyCounterToEnergyValue Note1	Convert energy counter to equivalent energy value
	EM_EnergyValueToEnergyCounter Note1	Convert energy value to equivalent energy counter
	EM_AddEnergyCounter Note1	Add to metrology energy counter
	EM_EnergyDataToEnergyValue Note2	Convert energy data in operation data structure to equivalent energy value with integer and decimal
	EM_EnergyValueToEnergyData Note2	Convert energy value with integer and decimal to equivalent energy data in operation data structure
	EM_AddEnergyData Note2	Add energy data in operation data structure to metrology energy counter + remainder
	EM_GetVoltageRMS	Read the True RMS voltage (Volt)
	EM_GetCurrentRMS	Read the True RMS Current (Ampere)
	EM_GetPowerFactor	Read the Power Factor (PF)
	EM_GetPowerFactorSign	Read the Power Factor Sign (Lead/Lag)
	EM_GetLineFrequency	Read the Line Frequency (Hz)
	EM_GetRMSLine	Get the RMS line selected by metrology

Notes 1: Available for **versions 220915 and below**.

Notes 2: Available in **versions above 220915**.

2.5.2 Wrapper functions

Table 2-5 Metrology library wrapper APIs list

Category	Definition Name	Description
ADC	EM_ADC_Init	Initialize ADC module that used for Metrology Library
	EM_ADC_Start	Start ADC module that used for Metrology Library
	EM_ADC_Stop	Stop ADC module that used for Metrology Library
	EM_ADC_GainReset	Reset phase or neutral gain to lowest
	EM_ADC_GainIncrease	Increase phase or neutral gain 1 level
	EM_ADC_GainDecrease	Decrease phase or gain 1 level
	EM_ADC_GainGetLevel	Get current phase or neutral gain level
	EM_ADC_SetGainValue	Set the phase or neutral gain level
	EM_ADC_SetPhaseCorrection	Adjust the phase or neutral angle of Voltage and Current channels
	EM_ADC_IntervalProcessing	This is a callback function. Acknowledgement of the sampling completion of ADC to Metrology Library
PULSE Output	EM_PULSE_Init	Initialize PULSE modules that used for Metrology Library
	EM_PULSE_ACTIVE_On	Turn ON for PULSE Active LED
	EM_PULSE_ACTIVE_Off	Turn OFF for PULSE Active LED
	EM_PULSE_REACTIVE_On	Turn ON for PULSE Reactive LED
	EM_PULSE_REACTIVE_Off	Turn OFF for PULSE Reactive LED
	EM_PULSE_APPARENT_On	Turn ON for PULSE Apparent LED
	EM_PULSE_APPARENT_Off	Turn OFF for PULSE Apparent LED
TIMER (40ms interval)	EM_TIMER_Init	Initialize a 40ms interval timer for Metrology Library
	EM_TIMER_Start	Start TIMER module as interval timer
	EM_TIMER_Stop	Stop the 40ms interval TIMER module
	EM_TIMER_InterruptCallback	This is a callback function. Acknowledgement of a 40ms interval time has been elapsed, to Metrology Library
WDT	EM_WDT_Init	Initialize WDT module
	EM_WDT_Start	Start WDT module
	EM_WDT_Stop	Stop WDT module
	EM_WDT_Restart	Restart (feed) WDT module
MCU Utility	EM_MCU_Delay	Delay the CPU processing a specified time (us)
	EM_MCU_MultipleInterruptEnable	Enable/Disable multiple interrupt servicing
Wrapper Property	EM_SW_GetProperty	Return the Wrapper Property page, include all settings on wrapper layer

3. Library functions

3.1 Common Functions

3.1.1 EM_Init

Prototype

```
uint8_t EM_Init(EM_PLATFORM_PROPERTY FAR_PTR *p_property, EM_CALIBRATION
FAR_PTR *p_calib);
```

Explanation

Initial Metrology Library.

This function initializes all required HW modules of Library through wrapper API: WDT, ADC, Timer.
And the internal data of Library.

Configurable values in p_property and p_calib are verified before their transfer to internal RAM data.

Wrapper function `EM_SW_GetProperty is` called to retrieve the user configuration and verify it, before transferring to internal RAM data.

If the execution is successful (`EM_OK`), system state of the library changes to `SYSTEM_STATE_INITIALIZED`. Otherwise, it stays in `SYSTEM_STATE_UINITIALIZED`.

Use `EM_GetSystemState()` to get the current state of library.

When the library is running (system state = `SYSTEM_STATE_RUNNING`), calling to this API (`EM_init`) will stop the operation of library, and re-initialize the library.

If the execution failed (return is not `EM_OK`), please check the setting of property, configuration, and calibration page again before re-calling to this function.

Argument(s)

Name	Data type	I/O	Description
p_property	EM_PLATFORM_PROPERTY FAR_PTR *	I	Platform property page
p_calib	EM_CALIBRATION FAR_PTR *	I	Platform calibration page

Return value

Execution status

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_PLATFORM_PROPERTY_NULL	p_property is NULL
EM_ERROR_CALIB_NULL	p_calib is NULL
EM_ERROR_PLATFORM_PROPERTY_TARGET_FREQ	p_property->target_ac_source_frequency not 50 or 60
EM_ERROR_SW_PROPERTY_NULL	EM_SW_GetProperty return NULL
EM_ERROR_SW_PROPERTY_GAIN	1 > sw property num of gain > 2 OR sw property num of gain = 2 and sw property gain upper threshold < sw property gain lower threshold
EM_ERROR_SW_PROPERTY_OPERATION	sw property no_voltage_threshold < EM_VOL_LOW_MIN sw property freq_low_threshold < EM_FREQ_LOW_MIN sw property freq_high_threshold > EM_FREQ_HIGH_MAX abs(sw property earth_diff_threshold) > EM_EARTH_DIFF_THRES_MAX sw property pulse_on_time < EM_PULSE_ON_TIME_MIN
EM_ERROR_SW_PROPERTY_ROUNDING	Rounding value > EM_MAX_ROUNDING_DIGIT
EM_ERROR_SW_PROPERTY_SAG_SWELL	If sw property sag swell detection_cycle > 0 and swell threshold < sw property no voltage threshold sag threshold < swell threshold

Restriction/Caution

Take care when configuring the parameter setting on the Platform property and Calibration page. Ensure that all settings are valid before calling to this API.

Take note on pointers with far attribute. Do not cast it into near attribute.

Sample Usage

The sample code below will initialize the library with following setting:

Setting	Value
AC Source System	50Hz
No voltage threshold	10V
Operation Current (Max)	60A
Operation Freq. Range	40-70Hz
ELT Threshold	12.5%
Phase Correction Angle	-2.804991 degree (negative value means I1 lead V)
Neutral Correction Angle	-2.826520 (negative value means I2 lead V)
Meter Constant	3200 imp/KWh
Energy to pulse ratio	1
Pulse On Time	30 ms
V-coefficient	13212.6113
I1-coefficient	79681.9298
I2-coefficient	62415.9454
Power coefficient (active, reactive, apparent)	1052806342.4862 (V-coefficient * I1-coefficient)
Power coefficient 2 (active, reactive, apparent)	824669696.7631 (V-coefficient * I2-coefficient)

Source code is as following:

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

/* Default platform property */
const EM_PLATFORM_PROPERTY FAR_PTR g_EM_DefaultProperty =
{
      50,    /* Target AC Source Frequency */
};


/* SW Phase Correction Angle List */
const float32_t FAR_PTR  g_EM_DefaultCalibPhaseAngleList[] =
{
    -2.804991f,    /* Phase Gain Level 0 Phase Shift Angle (in degree) */
    -2.805976f,    /* Phase Gain Level 1 Phase Shift Angle (in degree) */
};

/* SW Neutral Correction Angle List */
const float32_t FAR_PTR  g_EM_DefaultCalibNeutralAngleList[] =
{
    -2.826520f,    /* Neutral Gain Level 0 Phase Shift Angle (in degree) */
    -2.808475f,    /* Neutral Gain Level 1 Phase Shift Angle (in degree) */
};


/* SW Gain Value List (Phase Channel) */
const float32_t FAR_PTR    g_EM_DefaultCalibPhaseGainValueList[] =
{
   1.0f,           /* Phase Gain Level 0 Value (lowest, value is 1.0, fixed)*/
   2.0005384f,     /* Phase Gain Level 1 Value    |                        */
};
/* SW Gain Value List (Neutral Channel) */
const float32_t FAR_PTR    g_EM_DefaultCalibNeutralGainValueList[] =
{
   1.0f,           /* Neutral Gain Level 0 Value (lowest, value is 1.0, fixed)*/
   2.0037093f,     /* Neutral Gain Level 1 Value    |                        */
};

/* Platform default calibration */
const EM_CALIBRATION FAR_PTR g_EM_DefaultCalibration =
{
      3898.000000  /* Actual sampling frequency of the meter */

      {
            13212.6113,               /* VRMS Co-efficient */
            79681.9298,               /* I1RMS Co-efficient */
            62415.9454,               /* I2RMS Co-efficient */
            1052806342.4862,          /* Active Power Co-efficient */
            1052806342.4862,          /* Reactive Power Co-efficient */
            1052806342.4862,          /* Apparent Power Co-efficient */
            824669696.7631,           /* Active Power Co-efficient */
            824669696.7631,           /* Reactive Power Co-efficient */
            824669696.7631,           /* Apparent Power Co-efficient */
      },
```

```
    {
      (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseAngleList     ,
      (float32_t FAR_PTR *)g_EM_DefaultCalibNeutralAngleList   ,
    },

    {
      (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList  ,
      (float32_t FAR_PTR *)g_EM_DefaultCalibNeutralGainValueList,
    }
};

static FAR_PTR const EM_SW_PROPERTY em_sw_property =
{
    /* ADC */
    {
        1,
        1000000,
        500000,

        1,
        1000000,
        500000,
    },

    /* Operation */
    {
        0.01,
        0.01 * 180.0,
        10.0,
        40.0,
        70.0,
        12.5,
        3200,
        30
        1,
        1,
        1,
         0,
    },

    /* Rounding */
    {
        4,
        4,
        4,
        4,
    },

    /* Phase shift 90 Interpolation error */
    {
        0.0,
    },
    /* IIR Filter */
    {
        6,
        0,
        150,
    },
```

```
 /* Sag and Swell */
    {
        170,
        265,
        3,
    },
};

EM_SW_PROPERTY FAR_PTR * EM_SW_GetProperty(void)
{
    return (EM_SW_PROPERTY FAR_PTR *)&em_sw_property;
}

void init_library(void)
{
     uint8_t result;

     result = EM_Init(
            &g_EM_DefaultProperty,
            &g_EM_DefaultCalibration
    );
     if (result == EM_OK)
     {
            /* init library success */
     }
     else
     {
            /* Check on return value for diagnostic */
     }
};
```

3.1.2 EM_Start

Prototype

```
uint8_t EM_Start(void);
```

Explanation

Start Metrology Library Operation.

Only call to this API when library is already initialized (system state = `SYSTEM_STATE_INITIALIZED`)
Otherwise, `EM_ERROR` will be return.

If execution is successful (`EM_OK`), the system state changes to `SYSTEM_STATE_RUNNING`.

Use `EM_GetSystemState()` to get the current state of library.

If calling to this API returns a `EM_ERROR_STARTUP`, an error has occurred on the driver or wrapper layer. Check the driver or mapping of API on wrapper again before re-calling the function.

Argument(s)

None

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NOT_INITIALIZED	System is not initialized
EM_ERROR_STARTUP_TIMER	EM_TIMER_InterruptCallback not called after EM_Init
EM_ERROR_STARTUP_ADC	EM_ADC_IntervalProcessing not called OR MACEN is not 1

Restriction/Caution

This API should only be called when system state is `SYSTEM_STATE_INITIALIZED`.
Else use `EM_Init()` to initialize the library first.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */
uint8_t result;
result = EM_Start();
if (result == EM_OK)
{
     /* startup success */
}
else
{
     /* error happen, check result code for more detail of reason */
}
```

3.1.3 EM_Stop

Prototype

```
uint8_t EM_Stop(void);
```

Explanation

Stop Metrology Library Operation.

Only call to this API when the library is running (system state = SYSTEM_STATE_RUNNING).
Otherwise, an EM_ERROR will be returned.

If execution is successful (EM_OK), the system state will change to SYSTEM_STATE_INITIALIZED.

Use EM_GetSystemState() to get the current state of library.

Argument(s)

None

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NOT_RUNNING	System is not running

Restriction/Caution

Only call to this API when system state is SYSTEM_STATE_RUNNING.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

uint8_t result;

result = EM_Stop();
if (result == EM_OK)
{
      /* library operation stopped */
}
else if (result == EM_ERROR)
{
      /* error happen: system not running */
}
```

3.1.4 EM_GetCalibInfo

Prototype

```
EM_CALIBRATION EM_GetCalibInfo(void);
```

Explanation

EM Core Get Calibration Information.

Argument(s)

None

Return value

Calibration information structure. Refer to EM_CALIBRATION for more details and usage of this structure type.

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

EM_CALIBRATION calib;

calib = EM_GetCalibInfo();
```

3.1.5 EM_SetCalibInfo

Prototype

```
uint8_t EM_SetCalibInfo(EM_CALIBRATION FAR_PTR * p_calib);
```

Explanation

Configure calibration information of the library by changing calibration page.

Calling to this API while library is running (system state = `SYSTEM_STATE_RUNNING`) will cause an error (`EM_ERROR`) to be returned. The library should be stopped, using `EM_Stop()`, before using this API to configure the library.

If execution is successful (`EM_OK`), all settings on calibration page will be loaded into the library. Otherwise, setting on calibration page is ignored.

Argument(s)

Name	Data type	I/O	Description
p_calib	EM_CALIBRATION *	I	The pointer to calibration structure. Refer to EM_CALIBRATION for more details and its usage.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_STILL_RUNNING	The library is running. library must be stopped before changing settings
EM_ERROR_CALIB_NULL	Parameter is NULL
EM_ERROR_CALIB_PARAMS_COMMON	Sampling frequency out of EM_SAMPLING_FREQUENCY_MIN, EM_SAMPLING_FREQUENCY_MAX
EM_ERROR_CALIB_PARAMS_LINE1	VRMS coeff < EM_VRMS_COEFF_MIN OR I1RMS coeff < EM_I1RMS_COEFF_MIN OR Active power coeff < EM_ACT_POWER_COEFF_MIN OR Reactive power coeff < EM_REA_POWER_COEFF_MIN OR Apparent power coeff < EM_APP_POWER_COEFF_MIN OR i1_phase_degrees == NULL OR i1_gain_values == NULL
EM_ERROR_CALIB_PARAMS_NEUTRAL	I2RMS coeff < EM_I2RMS_COEFF_MIN OR Active power 2 coeff < EM_ACT_POWER_COEFF_MIN OR Reactive power 2 coeff < EM_REA_POWER_COEFF_MIN OR Apparent power 2 coeff < EM_APP_POWER_COEFF_MIN OR i2_phase_degrees == NULL OR I2_gain_values == NULL

Restriction/Caution

This API should only be called when the system is `SYSTEM_STATE_INITIALIZED`.
Use this API only to change the library setting. For library initialization, use `EM_Init()` instead.

Please take care when configuring the parameter settings of the calibration page, ensure that all of them are valid before initiating the settings to library.

Take note about pointers with far attribute. Do not cast it into near attribute.

Sample Usage

Below is an example that has implemented a function to change V-coeff = 3900.0, I1-coeff = 4200.0 , I2-coeff = 5200.0 while library is running.

```
void change_library_calib(void)
{
	uint8_t result;
	EM_SYSTEM_STATE last_state;
	EM_CALIBRATION calib;

	/* Stop library if running */
	last_state = EM_GetSystemState();
	if (last_state == SYSTEM_STATE_RUNNING)
	{
		EM_Stop();
	}
	/* Get current configuration page from library */
	calib = EM_GetCalibInfo();

	/* Change V-coeff, I1-coeff, I2-coeff */
	calib.coeff.vrms = 3900.0f;
	calib.coeff.i1rms = 4200.0f;
	calib.coeff.i2rms = 5200.0f;
	calib.coeff.active_power =
	calib.coeff.reactive_power =
	calib.coeff.apparent_power = (calib.coeff.vrms * calib.coeff.i1rms);

	/* Load configuration page to library again */
	result = EM_SetCalibInfo(&calib);
	if (result == EM_OK)
	{
		/* set config success */
	}
	else if (result == EM_ERROR_PARAMS)
	{
		/* parameter is NULL or setting on calib page is invalid */
	}
	else if (result == EM_ERROR)
	{
		/* library is running! */
	}

	/* Start library again (if stopped before) */
	if (last_state == SYSTEM_STATE_RUNNING &&
	    EM_GetSystemState() == SYSTEM_STATE_INITIALIZED)
	{
		EM_Start();
	}
}
```

3.1.6 EM_GetSystemState

Prototype

```
EM_SYSTEM_STATE EM_GetSystemState(void);
```

Explanation

Get the System state of Metrology Library.

Argument(s)

None

Return value

System state of library. An enumeration type, EM_SYSTEM_STATE.

Enumeration Name	Explanation
SYSTEM_STATE_UNINITIALIZED	Library is not initialized
SYSTEM_STATE_INITIALIZED	Library is already initialized
SYSTEM_STATE_RUNNING	Library is running

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

EM_SYSTEM_STATE result;

result = EM_GetSystemState();
```

3.1.7 EM_GetStatus

Prototype

```
EM_STATUS EM_GetStatus(void);
```

Explanation

Get the status of all measured parameters of Metrology Library.

Use this API to indicate the status of some internal measurement under metrology.

Argument(s)

None

Return value

Metrology library event status. A structure type, `EM_STATUS`.

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

EM_STATUS status;

status = EM_GetStatus();
```

3.1.8 EM_GetLastSagDuration

Prototype

```
uint16_t EM_GetLastSagDuration(void);
```

Explanation

EM User API. Get last sag event cycle duration.

Argument(s)

None

Return value

uint16_t cycle duration of sag event

Restriction/Caution

This API will only be taking effect if sag swell detection cycle > 0

Available for **versions below 221102**

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

/* Get the cycle duration of sag event */
uint16_t v_sag_cycle_duration_acc = EM_GetLastSagDuration();
```

3.1.9 EM_GetLastSwellDuration

Prototype

```
uint16_t EM_GetLastSwellDuration(void);
```

Explanation

EM User API. Get last swell event cycle duration.

Argument(s)

None

Return value

uint16_t cycle duration of swell event

Restriction/Caution

This API will only be taking effect if sag swell detection cycle > 0

Available in **versions below 221102**

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

/* Get the cycle duration of swell event */
uint32_t v_swell_cycle_duration_acc = EM_GetLastSwellDuration();
```

3.1.10 EM_GetEnergyAccumulationMode

Prototype

```
uint8_t EM_GetEnergyAccumulationMode(void);
```

Explanation

EM User API. Get energy counter accumulation mode.

Argument(s)

None

Return value

uint8_t energy accumulation mode

0: Library stop updating power value for energy accumulation. Users call EM_SetEnergyAccumulationPower to update.

1: Library always use Phase channel power for energy accumulation

2: Library always use Neutral channel power for energy accumulation

3: Library select between Phase and Neutral based on IRMS value (prioritize for Phase IRMS)

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

/* Get current energy accumulation mode */
uint8_t accumulation_mode = EM_GetEnergyAccumulationMode();
```

3.1.11 EM_SetEnergyAccumulationMode

Prototype

```
void EM_SetEnergyAccumulationMode(uint8_t mode);
```

Explanation

EM User API. Set energy counter accumulation mode.

Note: In mode 0, after switching to mode 0, EM will use last updated value until user call EM_SetEnergyAccumulationPower to set a custom power value

Argument(s)

uint8_t mode: energy accumulation mode

0: EM stop updating power value for energy accumulation. Users call EM_SetEnergyAccumulationPower to update.

1: EM always use Phase channel power for energy accumulation

2: EM always use Neutral channel power for energy accumulation

3: EM select between Phase and Neutral based on IRMS value (prioritize for Phase IRMS)

For auto-selection between Phase and Neutral, please check chapter Automatic line selection.

Return value

None

Restriction/Caution

Input value larger than 3 will be set to 3.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

/* Set the energy accumulation mode */
EM_SetEnergyAccumulationMode(3);
```

3.1.12 EM_SetEnergyAccumulationPower

Prototype

```
void EM_SetEnergyAccumulationPower(float32_t active, float32_t reactive,
float32_t apparent);
```

Explanation

Set custom energy accumulation power (in mode 0 only)

Apparent sign will be ignored.

Active and reactive sign will determine the quadrant of energy accumulation:

QI : active > 0, reactive > 0: import active, import inductive reactive, import apparent

QII : active < 0, reactive > 0: export active, import capacitive reactive, export apparent

QIII: active < 0, reactive < 0: export active, export inductive reactive, export apparent

QIV : active > 0, reactive < 0: import active, export capacitive reactive, import apparent

Argument(s)

float32 active: active power (in Watt)

float32 reactive: reactive power (in Var)

float32 active: apparent power (in VA)

Return value

None

Restriction/Caution

Used when energy accumulation mode is set to mode 0.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */


if (EM_GetEnergyAccumulationMode() == 0)
{
      EM_SetEnergyAccumulationPower(1100.0000, 0.0, 1100.0000)
}
```

3.1.13 EM_GetOperationData

Prototype

```
uint8_t EM_GetOperationData(EM_OPERATION_DATA * p_operation_data);
```

Explanation

Get metrology operation internal data.

Argument(s)

Name	Data type	I/O	Description
p_operation_data	EM_OPERATION_DATA *	I	Metrology Operation Data.

Return value

Execution status

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is NULL

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

EM_OPERATION_DATA em_data;
uint8_t status;

status = EM_GetOperationData(&em_data);
```

3.1.14 EM_SetOperationData

Prototype

```
uint8_t EM_SetOperationData(EM_OPERATION_DATA * p_operation_data);
```

Explanation

Set metrology operation internal data.

Argument(s)

Name	Data type	I/O	Description
p_operation_data	EM_OPERATION_DATA *	I	Metrology Operation Data

Return value

Execution status

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is NULL

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */

EM_OPERATION_DATA em_data;
uint8_t status;

status = EM_SetOperationData(&em_data);
```

3.2 Functions for Calibration

3.2.1 EM_CalibInitiate

Prototype

```
uint8_t EM_CalibInitiate(EM_CALIB_ARGS FAR_PTR * p_calib_args, EM_CALB_WORK *
p_calib_work, EM_CALIB_OUTPUT * p_calib_output);
```

Explanation

EM Core Calibration initiation function

Need to register EM_RTC_CalibInterruptCallback to either a 0.5s or 1s interval. A high accuracy timer interrupt source is required for the sampling frequency calibration.

After successful called (return value EM_OK), EM will be in calibration state and needs to call EM_CalibRun for multiple instances, till EM_CalibRun does not return an EM_CALIBRATING.

Argument(s)

Name	Data type	I/O	Description
p_calib_args	EM_CALIB_ARGS *	I	The pointer to calibration arguments. Refer to 2.4.8 EM_CALIB_ARGS for more details and its usage.
p_calib_work	EM_CALIB_WORK *	I	The pointer to working area. 256 bytes even alignment working area for calibration.
p_calib_output	EM_CALIB_OUTPUT *	I	The pointer to output area. Refer to 2.4.9 EM_CALIB_OUTPUT for more details and its usage.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NOT_RUNNING	The library is running. Must stop the library before change setting
EM_ERROR_CALIBRATING_NULL	Parameters is NULL
EM_ERROR_CALIBRATING_INVALID_LINE	line_v is not EM_LINE_PHASE OR line_i is not EM_LINE PHASE OR line_i is not EM_LINE_NEUTRAL
EM_ERROR_CALIBRATING_CYCLE	cycle < 1 OR cycle_angle < 1
EM_ERROR_CALIBRATING_V_I	v < 1 OR i < 1 OR imax < 0
EM_ERROR_CALIBRATING_RTC_PERIOD	rtc_period != 1000 AND rtc_period != 500
EM_ERROR_CALIBRATING_IMAX_AND_NUM_OF_GAIN	Imax is 0 while sw property number of gain is 1
EM_ERROR_CALIBRATING_MAX_GVALUE_SETTING	max_gvalue is 0
EM_ERROR_STARTUP_RTC	EM_RTC_CalibInterruptCallback is not called for calibration.

Restriction/Caution

Should call this API when system is `SYSTEM_STATE_RUNNING`.
Take note on pointers with far attribute. Do not cast it into near attribute.

Sample Usage

Please refer to EM_CalibRun example

3.2.2 EM_CalibRun

Prototype

```
uint8_t EM_CalibRun(void);
```

Explanation

EM Core Calibration execution function

This function will calibrate step by step in each call and return the calibration status.

After completing all calibration steps (Not returning EM_CALIBRATING), calibration data will be available in EM_CALIB_OUTPUT structure address and input to the EM_CalibInitiate function. In case of error, this can give additional information on erroneous calibrated value.

Currently the calibration only supports:

- 3906Hz sampling frequency
- Reference power supply need to be in UPF (PF = 1.0)
- Both I signal should lead V signal

Argument(s)

None

Return value

Macro Value Name	Explanation
EM_OK	Execute successfully, calibration finished
EM_ERROR_CALIBRATING_NOT_STARTED	EM_CalibInitiate not called or EM_CalibInitiate not returning EM_OK
EM_ERROR_CALIBRATING_FAILED_FS_OUT_RANGE	Calibrated sampling frequency is out of 20% of ideal sampling frequency (check in calib output structure)
EM_ERROR_CALIBRATING_FAILED_IGAIN_OUT_RANGE	Calibrated current gain (when sw number of gains is 1) > maxgvalue setting
EM_ERROR_CALIBRATING_FAILED_MAX_ANGLE	Calibrated angle is out of range, special value: 90: could be reactive load, supply need to be UPF 180: V and I reversed Others: exceed max DSADPHCR range (for either 50Hz or 60Hz)
EM_ERROR_CALIBRATING_FAILED_V_LEAD_I	V is leading I

Restriction/Caution

None

Sample Usage

Below is an example to do calibration for: 100-line cycles for coefficient accumulation, 100-line cycles for angle error accumulation, reference voltage 220V, reference current 5A, max current 60A, single gain for EM_LINE_PHASE

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */
#include "em_calibration.h"     /* EM/Library Manual Calibration */

void calibrate_phase(void)
{
    EM_CALIB_ARGS calib_args;
    EM_CALIB_OUTPUT calib_output;
    EM_CALIB_WORK calib_work;
    uint8_t rlt;
    calib_args.rtc_period = 500;
    calib_args.max_gvalue = 64;
    calib_args.stable_ndelay = 10;
    /* Settable parameter */
    calib_args.cycle = 100;
    calib_args.cycle_angle = 100;
    calib_args.imax = 60.0f;
    calib_args.v = 220.0f;
    calib_args.i = 5.0f;
    calib_args.line_v = EM_LINE_PHASE;
    calib_args.line_i = EM_LINE_PHASE;
    rlt = EM_CalibInitiate(&calib_args, &calib_work, &calib_output);
    if (rlt != EM_OK)
    {
        /* Check return value */
    }
    else
    {
        /* EM Calibrate initiated OK */
        while (1)
        {
            rlt = EM_CalibRun();
            if (rlt == EM_CALIBRATING || rlt == EM_OK)
            {
                /* Continue */
            }
            else
            {
                /* Calibration error, check return value and calib output */
            }

            if (rlt != EM_CALIBRATING)
            {
                break;
            }
        }
    }
}
```

3.2.3 EM_RTC_CalibInterruptCallback

Prototype

```
void EM_RTC_CalibInterruptCallback(void);
```

Explanation

Register into 0.5s or 1s interval timer for calibration only.

If external calibration is used, don’t need to register this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_operation.h"       /* EM/Library Operation */


EM_RTC_CalibInterruptCallback();
```

3.3 Functions for Measurement Output

3.3.1 EM_GetActivePower

Prototype

```
float32_t EM_GetActivePower(EM_LINE line);
```

Explanation

Get the measured active power (Watt) from library.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

Floating-point, single-precision value of Active Power (Watt).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t active;

/* Get Active Power from Phase Line */

active = EM_GetActivePower(EM_LINE_PHASE);

/* Get Active Power from Neutral Line */

active = EM_GetActivePower(EM_LINE_NEUTRAL);
```

3.3.2 EM_GetFundamentalActivePower

Prototype

```
float32_t EM_GetFundamentalActivePower(EM_LINE line);
```

Explanation

Get the measured fundamental active power (Watt) from library.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement for line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

Floating-point, single-precision value of Fundamental Active Power (Watt).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t fundamental_active;

/* Get Fundamental Active Power from Phase Line */

fundamental_active = EM_GetFundamentalActivePower(EM_LINE_PHASE);


/* Get Fundamental Active Power from Neutral Line */

fundamental_active = EM_GetFundamentalActivePower(EM_LINE_NEUTRAL);
```

3.3.3 EM_GetReactivePower

Prototype

```
float32_t EM_GetReactivePower(EM_LINE line);
```

Explanation

Get the measured reactive power (VAr) from the library.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement for line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

Floating-point, single-precision value of Reactive Power (VAr).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t reactive;

/* Get Reactive Power from Phase Line */

reactive = EM_GetReactivePower(EM_LINE_PHASE);


/* Get Reactive Power from Neutral Line */

reactive = EM_GetReactivePower(EM_LINE_NEUTRAL);
```

3.3.4 EM_GetApparentPower

Prototype

```
float32_t EM_GetApparentPower(EM_LINE line);
```

Explanation

Get the measured apparent power (VA) from library.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement for line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

Floating-point, single-precision value of Apparent Power (VA).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t apparent_power;

/* Get Apparent Power from Phase Line */

apparent_power = EM_GetApparentPower(EM_LINE_PHASE);


/* Get Apparent Power from Neutral Line */

apparent_power = EM_GetApparentPower(EM_LINE_NEUTRAL);
```

3.3.5 EM_GetEnergyCounter

Prototype

```
uint8_t EM_GetEnergyCounter(EM_ENERGY_COUNTER *p_counter);
```

Explanation

Get the accumulating energy counter.

Argument(s)

Name	Data type	I/O	Description
p_energy	EM_ENERGY_COUNTER	I	Measurement for line selection. Refer to EM_ENERGY_COUNTER for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions 220915 and below**.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_ENERGY_COUNTER counter;

uint8_t rlt;


/* Get Energy Counter */

rlt = EM_GetEnergyCounter(&counter);
```

3.3.6 EM_EnergyCounterToEnergyValue

Prototype

```
uint8_t EM_EnergyCounterToEnergyValue(EM_ENERGY_COUNTER *p_counter,
EM_ENERGY_VALUE *p_value);
```

Explanation

Convert energy counter to single precision energy value.

Argument(s)

Name	Data type	I/O	Description
p_counter	EM_ENERGY_COUNTER	I	Refer to EM_ENERGY_COUNTER for more details & usage for this structure type.
p_value	EM_ENERGY_VALUE	I	Refer to EM_ENERGY_VALUE for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions 220915 and below**.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_ENERGY_COUNTER em_energy_counter;

EM_ENERGY_VALUE em_energy_value;

/* Convert Energy Counter to Energy Value */

EM_EnergyCounterToEnergyValue(&em_energy_counter, &em_energy_value);
```

3.3.7 EM_EnergyValueToEnergyCounter

Prototype

```
uint8_t EM_EnergyCounterToEnergyValue(EM_ENERGY_COUNTER *p_counter, NEAR_PTR
EM_ENERGY_VALUE *p_value);
```

Explanation

Convert energy value to energy counter.

Argument(s)

Name	Data type	I/O	Description
p_counter	EM_ENERGY_COUNTER	I	Refer to EM_ENERGY_COUNTER for more details & usage for this structure type.
p_value	EM_ENERGY_VALUE	I	Refer to EM_ENERGY_VALUE for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions 220915 and below**.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_ENERGY_COUNTER em_energy_counter;

EM_ENERGY_VALUE em_energy_value;

/* Convert Energy Value to Energy Counter */

EM_EnergyValueToEnergyCounter (&em_energy_counter, &em_energy_value);
```

3.3.8 EM_AddEnergyCounter

Prototype

```
uint8_t EM_AddEnergyCounter(EM_ENERGY_COUNTER *p_counter);
```

Explanation

Add to metrology Energy Counter.

Argument(s)

Name	Data type	I/O	Description
p_counter	EM_ENERGY_COUNTER	I	Refer to EM_ENERGY_COUNTER for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions 220915 and below**.

The addition is done directly to internal metrology energy counter, which updates regularly in DSAD interrupt (read/write). So, the metrology should be stopped before calling this API.

Take note that as the amount of counter added does not affect the pulse output, alignment between number of pulse output and energy counter it would break.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_ENERGY_COUNTER em_energy_counter;


/* Add Energy Counter */

EM_AddEnergyCounter(&em_energy_counter);
```

3.3.9 EM_EnergyDataToEnergyValue

Prototype

```
uint8_t EM_EnergyDataToEnergyValue(EM_OPERATION_DATA *p_data, EM_ENERGY_VALUE
*p_value);
```

Explanation

Convert energy data to energy value with integer and decimal part.

Argument(s)

Name	Data type	I/O	Description
p_data	EM_OPERATION_DATA	I	Refer to EM_OPERATION_DATA for more details & usage for this structure type.
p_value	EM_ENERGY_VALUE	I	Refer to EM_ENERGY_VALUE for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions above 220915**.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_OPERATION_DATA em_data;

EM_ENERGY_VALUE em_energy_value;

/* Convert Energy Data to Energy Value */

EM_EnergyDataToEnergyValue(&em_data, &em_energy_value);
```

3.3.10 EM_EnergyValueToEnergyData

Prototype

```
uint8_t EM_EnergyCounterToEnergyValue(EM_OPERATION_DATA *p_counter,
EM_ENERGY_VALUE *p_value);
```

Explanation

Convert energy value (integer and decimal part) to energy data.

Argument(s)

Name	Data type	I/O	Description
p_data	EM_OPERATION_DATA	I	Refer to EM_OPERATION_DATA for more details & usage for this structure type.
p_value	EM_ENERGY_VALUE	I	Refer to EM_ENERGY_VALUE for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions above 220915**.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_OPERATION_DATA em_data;

EM_ENERGY_VALUE em_energy_value;

/* Convert Energy Value to Energy Data */

EM_EnergyValueToEnergyData(&em_data, &em_energy_value);
```

3.3.11 EM_AddEnergyData

Prototype

```
uint8_t EM_AddEnergyData(EM_OPERATION_DATA *p_data);
```

Explanation

Add energy data information in operation data structure to metrology energy counter + remainder.

Argument(s)

Name	Data type	I/O	Description
p_data	EM_OPERATION_DATA	I	Refer to for more details & usage for this structure type.

Return value

Execution status.

Macro Value Name	Explanation
EM_OK	Execute successfully
EM_ERROR_NULL_PARAMS	Parameter is invalid (NULL)

Restriction/Caution

Available in **versions above 220915**.

The addition is done directly to internal metrology energy counter, which updates regularly in DSAD interrupt (read/write). So, the metrology should be stopped before calling this API.

Take note that as the amount of counter added does not affect the pulse output, alignment between number of pulse output and energy counter it would break.

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


EM_OPERATION_DATA em_data;


/* Add Energy Data to metrology operation data */

EM_AddEnergyData(&em_data);
```

3.3.12 EM_GetVoltageRMS

Prototype

```
float32_t EM_GetVoltageRMS(void);
```

Explanation

Get the voltage RMS value (Volt)

Argument(s)

None

Return value

Floating-point, single precision value of True RMS Voltage (Volt).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t voltageRMS;

/* Get Voltage RMS value */

voltageRMS = EM_GetVoltageRMS();
```

3.3.13 EM_GetCurrentRMS

Prototype

```
float32_t EM_GetCurrentRMS(EM_LINE line);
```

Explanation

Get the current RMS value (Ampere).

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to 2.4.1 EM_LINE for more details & usage of this structure type.

Return value

Floating-point, single precision value of True RMS Current (Ampere).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */


float32_t currentRMS, currentRMS;

/* Get Current RMS value from Phase Line */

currentRMS = EM_GetCurrentRMS(EM_LINE_PHASE);


/* Get Current RMS value from Neutral Line */

currentRMS = EM_GetCurrentRMS(EM_LINE_NEUTRAL);
```

3.3.14 EM_GetPowerFactor

Prototype

```
float32_t EM_GetPowerFactor(EM_LINE channel);
```

Explanation

Get the Power factor value.

Argument(s)

Name	Data type	I/O	Description
channel	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage of this structure type.

Return value

Floating-point, single precision value of Power Factor.

The sign of power factor indicates the sign of active power.

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t powerfactor;

/* Get Power Factor from Phase Line */

powerfactor = EM_GetPowerFactor(EM_LINE_PHASE);


/* Get Power Factor from Neutral Line */

powerfactor = EM_GetPowerFactor(EM_LINE_NEUTRAL);
```

3.3.15 EM_GetPowerFactorSign

Prototype

```
EM_PF_SIGN EM_GetPowerFactorSign(EM_LINE line);
```

Explanation

Get the sign of Power Factor (Lead, Lag or Unity).

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage of this structure type.

Return value

Enumeration, EM_PF_SIGN

Restriction/Caution

EM_GetPowerFactor() should be called after this API.

Sample Usage

```
#include "typedef.h"              /* GSCE Standard Typedef */
#include "em_type.h"              /* EM/Library Typedef */
#include "em_measurement.h"       /* EM/Library Measurement Output */

EM_PF_SIGN sign;

/* Get Power Factor sign from Phase Line */
sign = EM_GetPowerFactorSign (EM_LINE_PHASE);
If (sign == PF_SGIN_LEAD_C)
{
      …
}

/* Get Power Factor sign from Neutral Line */
sign = EM_GetPowerFactorSign (EM_LINE_NEUTRAL);
If (sign == PF_SGIN_LEAD_C)
{
      …
}
```

3.3.16 EM_GetLineFrequency

Prototype

```
float32_t EM_GetLineFrequency(void);
```

Explanation

Get the Line Frequency (Hz).

Argument(s)

None

Return value

Floating-point, single precision value of Line Frequency (Hz).

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

float32_t freq;

/* Get Line Frequency */
freq = EM_GetLineFrequency();
```

3.3.17 EM_GetRMSLine

Prototype

```
EM_LINE EM_GetRMSLine(void);
```

Explanation

Select the best value of IRMS based on threshold.

Internally, EM will regularly update the selected RMS line as return value for this function. For more info, refer to Automatic line selection.

Argument(s)

None

Return value

Selected current line with enum EM_LINE.

Restriction/Caution

None

Sample Usage

```
#include "typedef.h"            /* GSCE Standard Typedef */
#include "em_type.h"            /* EM/Library Typedef */
#include "em_measurement.h"     /* EM/Library Measurement Output */

EM_LINE line;

/* Get the best value of IRMS */
line = EM_GetRMSLine();
```

4. Function from wrapper

4.1 Wrapper function for ADC

This component is used to link the ADC module to the library and consist of the following APIs.

4.1.1 EM_ADC_Init

Prototype

```
void EM_ADC_Init(void);
```

Explanation

Initializes the ADC module used in the Metrology Library.

This function **MUST** initialize the ADC successfully, else, the library will experience unexpected errors in run-time.

This function will be called once the library is initialized by EM_Init().

Do the calling to Driver API of ADC Device Driver inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_ADC_Init(void)
{
      /* Init by ADC Device Driver */
     /* Do the calling to Device Driver Layer here */
}
```

4.1.2 EM_ADC_Start

Prototype

```
void EM_ADC_Start(void);
```

Explanation

Start the ADC module used in the Metrology Library.

This function **MUST** start the ADC successfully, else, the library will experience unexpected errors in run-time.

Do the calling for the ADC Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_ADC_Start(void)
{
      /* Start by ADC Device Driver */
      /* Do the calling to Device Driver Layer here */
}
```

4.1.3 EM_ADC_Stop

Prototype

```
void EM_ADC_Stop(void);
```

Explanation

Stop the ADC module used in the Metrology Library.

This function **MUST** stop the ADC successfully, else, the library will experience unexpected error in run-time.

Do the calling of the ADC Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_ADC_Stop(void)
{
      /* Stop by ADC Device Driver */
      /* Do the calling to Device Driver Layer here */
}
```

4.1.4 EM_ADC_GainReset

Prototype

```
void EM_ADC_GainReset(EM_LINE line);
```

Explanation

Reset phase gain to lowest level (level 0).

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_ADC_GainReset(void)
{
      /* Reset phase gain to lowest level here */
}
```

4.1.5 EM_ADC_GainIncrease

Prototype

```
void EM_ADC_GainIncrease(EM_LINE line);
```

Explanation

Increase gain of Phase channel one level. Example, level 0 → level 1.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

None

Restriction/Caution

Only need to be implemented on Wide Range library versions (WSUR, WQUR, WQFR).

Sample Implementation

```
/***********************************************************************
* Function Name    : void EM_ADC_GainIncrease(EM_LINE line)
* Description      : Increase gain of Phase channel by 1 level
* Arguments        : None
* Functions Called : None
* Return Value     : None
***********************************************************************/
void EM_ADC_GainIncrease(EL_LINE line)
{
     /* Increase FPGA Gain of Phase channel by 1 level here */
}
```

4.1.6 EM_ADC_GainDecrease

Prototype

```
void EM_ADC_GainDecrease(EM_LINE line);
```

Explanation

Decrease gain of Phase channel one level. Example, level 1 → level 0.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

None

Restriction/Caution

Only need to be implemented on Wide Range library versions (WSUR, WQUR, WQFR).

Sample Implementation

```
void EM_ADC_GainDecrease(EM_LINE line)
{
      /* Decrease FPGA Gain of Phase channel by 1 level here */
}
```

4.1.7 EM_ADC_GainGetLevel

Prototype

```
uint8_t EM_ADC_GainGetLevel(EM_LINE line);
```

Explanation

Get the current gain level of channel. Example, if level 0 is current level, 0 is returned.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection. Refer to EM_LINE for more details & usage for this structure type.

Return value

None

Restriction/Caution

Only need to be implemented on Wide Range library versions (WSUR, WQUR, WQFR).

Sample Implementation

```
uint8_t EM_ADC_GainGetLevel(EM_LINE line)
{
      /* Dummy return 0, please return the actual gain level here */
      return 0;
}
```

4.1.8 EM_ADC_SetGainValue

Prototype

```
void EM_ADC_SetGainValue(EM_LINE line, uint8_t gain);
```

Explanation

Set the gain level of channel. Example, read gain value first and use this API to set the channel gain level.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection.

Name	Data type	I/O	Description
gain	uint8_t *	I	Gain level of channel

Return value

None

Restriction/Caution

Need to get the gain first by using R_DSADC_GetGainEnumValue(gain) first.

Sample Implementation

```
void EM_ADC_SetGainValue(EM_LINE line, uint8_t gain)
{
      dsad_gain_t dsad_gain = R_DSADC_GetGainEnumValue(gain);

}
```

4.1.9 EM_ADC_SetPhaseCorrection

Prototype

```
void EM_ADC_SetPhaseCorrection(EM_LINE line, float32_t degree);
```

Explanation

Adjust the phase angle of Voltage and Current channels.

Phase correction is done on current channels, relative to voltage signal (voltage channel sets the phase shift control register to 0).

Degree should be in the negative, indicating Current leading Voltage.

This function **MUST** be successfully with the phase adjustment, else, the library will experience unexpected error during run-time.

On the library, when changing the settings of phase correction on `EM_CALIBRATION` structure through the calling of `EM_Init()` or `EM_SetCalibInfo()`, this function will be called to adjust the phase angle between V and I1.

Do the calling of the ADC Device Driver APIs inside this function.

Argument(s)

Name	Data type	I/O	Description
line	EM_LINE	I	Measurement with line selection.

Name	Data type	I/O	Description
degree	float32_t	I	Phase shifting sign and amount in degree

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_ADC_SetPhaseCorrection(EM_LINE line, float32_t degree)
{

	/* negative ? the current lead voltage */
	if (degree < 0)
	{
		/* delay Current1 channel here */
	}
	else
	{

	}
}
```

4.1.10 EM_ADC_IntervalProcessing

Prototype

```
void EM_ADC_IntervalProcessing(EM_SAMPLES * p_samples);
```

Explanation

This is a callback function that acknowledges the sampling completion of ADC to Metrology Library.

This function MUST be linked to ADC Device Driver Interrupt Callback (ISR) successfully, else, the library will not proceed pass the start-up call of `EM_Start()`, and an `EM_ERROR_STARTUP` will be returned. In this case, before starting the library again, please re-check the registration of this API to the driver carefully.

Link this function to ADC Device Driver Interrupt Callback (ISR).

IMPORTANT. Set up the ADC ISR at a HIGH priority level.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

Assume the ADC Device Driver has its ISR, located at vector INTDSAD and named as `r_adc_interrupt`, as following:

```
EM_SAMPLES g_wrp_adc_samples;
#pragma interrupt INTDSAD r_adc_interrupt

static void r_adc_interrupt(void)
{
     /* Start user code. Do not edit comment generated here */
     /* Read samples and preprocess the signal to g_wrp_adc_samples */
     /* This is the linking of ADC Wrapper Interrupt Callback to driver */
     EM_ADC_IntervalProcessing(&g_wrp_adc_samples);
     /* End user code. Do not edit comment generated here */
}
```

4.2 Wrapper function for Pulse Output

This component is used to link the PULSE modules (3 PORT pins) to Library, consisting of all following APIs.

4.2.1 EM_PULSE_Init

Prototype

```
void EM_PULSE_Init(void);
```

Explanation

Initialize PULSE modules used for the Metrology Library.
This function **MUST** initialize the PULSE (Port pins) in output mode.

This function will be called once the library is initialized by EM_Init().

Do the calling of the PULSE (Port pin) Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_Init(void)
{
      /* Call to PULSE Driver Initialization here */
}
```

4.2.2 EM_PULSE_ACTIVE_On

Prototype

```
void EM_PULSE_ACTIVE_On(void);
```

Explanation

Turn ON for PULSE Active LED.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_ACTIVE_On(void)
{
      /* Turn ON PULSE Active LED here */
}
```

4.2.3 EM_PULSE_ACTIVE_Off

Prototype

```
void EM_PULSE_ACTIVE_Off(void);
```

Explanation

Turn OFF PULSE Active LED.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_ACTIVE_Off(void)
{
      /* Turn OFF PULSE Active LED here */
}
```

4.2.4 EM_PULSE_REACTIVE_On

Prototype

```
void EM_PULSE_REACTIVE_On(void);
```

Explanation

Turn ON PULSE Reactive LED.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_REACTIVE_On(void)
{
      /* Turn ON PULSE Reactive LED here */
}
```

4.2.5 EM_PULSE_REACTIVE_Off

Prototype

```
void EM_PULSE_REACTIVE_Off(void);
```

Explanation

Turn OFF PULSE Reactive LED.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_REACTIVE_Off(void)
{
      /* Turn OFF PULSE Reactive LED here */
}
```

4.2.6 EM_PULSE_APPARENT_On

Prototype

```
void EM_PULSE_APPARENT_On(void);
```

Explanation

Turn ON PULSE Apparent LED.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_APPARENT_On(void)
{
      /* Turn ON PULSE Apparent LED here */
}
```

4.2.7 EM_PULSE_APPARENT_Off

Prototype

```
void EM_PULSE_APPARENT_Off(void);
```

Explanation

Turn OFF PULSE Apparent LED.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_PULSE_APPARENT_Off(void)
{
      /* Turn OFF PULSE Apparent LED here */
}
```

4.3 Wrapper functions for Timer

This component is used to link a TIMER to the Library. Please set up a 40ms interval timer for this module. The library will use the interrupt callback for checking of event and update energy counter.

4.3.1 EM_TIMER_Init

Prototype

```
void EM_TIMER_Init(void);
```

Explanation

Initialize a 40ms interval timer used by the Metrology Library.
This function MUST initialize the timer successfully, else, the library will experience un-expected error in run-time.

This function will be called once the library is initialized by `EM_Init()`.

Do the calling of Timer Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_TIMER_Init(void)
{
      /* Call to TIMER Driver Initialization here */
}
```

4.3.2 EM_TIMER_Start

Prototype

```
void EM_TIMER_Start(void);
```

Explanation

Start TIMER module as an interval timer. The interrupt must be generated after starting up the timer successfully, else, unexpected errors maybe occur when starting up the library by `EM_Start()`.

Do the calling of Timer Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_TIMER_Start(void)
{
      /* Start Timer by Device Driver API */
      /* Do the calling to Device Driver Layer here */
}
```

4.3.3 EM_TIMER_Stop

Prototype

```
void EM_TIMER_Stop(void);
```

Explanation

Stop the 40ms interval TIMER module.

Do the calling of Timer Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_TIMER_Stop(void)
{
      /* Stop Timer by Device Driver API */
      /* Do the calling to Device Driver Layer here */
}
```

4.3.4 EM_TIMER_InterruptCallback

Prototype

```
void EM_TIMER_InterruptCallback(void);
```

Explanation

This is a callback function that acknowledges a 40ms interval timer has been elapsed to the Metrology Library.

This function MUST be linked to the Timer Device Driver Interrupt Callback (ISR) successfully, else, the library will not pass the starting up call of the `EM_Start()`, and `EM_ERROR_STARTUP` will be returned. In this case, before starting up the library again, please re-check the registration of this API to the driver carefully.

Link this function to Timer Device Driver Interrupt Callback (ISR).

IMPORTANT. Set up the Timer ISR at priority level that is just lower than ADC ISR
(ADC ISR > TIMER ISR)

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

Assume that Timer channel 2 has been set-up to use the library for TIMER module, Timer Channel2 ISR, located at vector INTTM02 and named as `r_tau0_channel2_Interrupt`, as following:

```
#pragma interrupt INTTM02 r_tau0_channel2_interrupt

static void r_tau0_channel2_interrupt(void)
{
     /* Start user code. Do not edit comment generated here */

     /* This is the linking of Timer Wrapper Interrupt Callback to driver */
     EI();
     EM_TIMER_InterruptCallback();

     /* End user code. Do not edit comment generated here */
}
```

4.4 Wrapper functions for Watch Dog Timer (WDT)

This component is optional. If the system uses the WDT to ensure the meter operation, please link the following APIs to driver layer. The library will call to feed the WDT when long/heavy jobs are involved

4.4.1 EM_WDT_Init

Prototype

```
void EM_WDT_Init(void);
```

Explanation

Initialize WDT module. This function will be called once the library is initialized by `EM_Init()`.

Do the calling of the WDT Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_WDT_Init(void)
{
      /* Call to WDT Driver Initialization here */
}
```

4.4.2 EM_WDT_Start

Prototype

```
void EM_WDT_Start(void);
```

Explanation

Start WDT module.

Do the calling of the WDT Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_WDT_Start(void)
{
      /* Start WDT by Device Driver API */
      /* Do the calling to Device Driver Layer here */
}
```

4.4.3 EM_WDT_Stop

Prototype

```
void EM_WDT_Stop(void);
```

Explanation

Stop WDT module.

Do the calling of the WDT Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_WDT_Stop(void)
{
      /* Stop WDT by Device Driver API */
      /* Do the calling to Device Driver Layer here */
}
```

4.4.4 EM_WDT_Restart

Prototype

```
void EM_WDT_Restart(void);
```

Explanation

Restart (feed) WDT module.

Do the calling of the WDT Device Driver APIs inside this function.

Argument(s)

None

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_WDT_Restart(void)
{
      /* Restart WDT by Device Driver API */
      /* Do the calling to Device Driver Layer here */
}
```

4.5 Wrapper function for MCU Utility

Setup utility function of the MCU to use in the library.

4.5.1 EM_MCU_Delay

Prototype

```
void EM_MCU_Delay(uint16_t us);
```

Explanation

Delay the processing by a specified time (us).

Argument(s)

Name	Data type	I/O	Description
us	uint16_t	I	Specify the time to do delay (us)

Return value

None

Restriction/Caution

None

Sample Implementation

Below is an example of implementation for delay function where each loop elapses 1us. Change the implementation when there is a change in the MCU or fCPU.

```
void EM_MCU_Delay(uint16_t us)
{
      /* Implementation the delay here, below is just an example... */
      while (us)    /* Each loop must elapse 1us */
      {
            NOP();  /*    07     */
            NOP();  /*    08     */
            NOP();  /*    09     */
            NOP();  /*    10     */
            NOP();  /*    11     */
            NOP();  /*    12     */
            NOP();  /*    13     */
            NOP();  /*    14     */
            NOP();  /*    15     */
            NOP();  /*    16     */

            us--;  /* count down number of us */
      }
}
```

4.5.2 EM_MCU_MultipleInterruptEnable

Prototype

```
void EM_MCU_MultipleInterruptEnable(uint8_t enable);
```

Explanation

Enable/Disable multiple interrupt servicing.

If the MCU support instruction to enable/disable the multiple interrupt function, link them to this API using the implementation below. Else, this function can be skipped.

Argument(s)

Name	Data type	I/O	Description
enable	uint8_t	I	0 is Disable, not 0 is enable

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_MCU_MultipleInterruptEnable(uint8_t enable)
{
      if (enable)
      {
            /* Enable multiple interrupt, e.g. by EI() */;
      }
      else
      {
            /* Disable multiple interrupt, e.g. by DI() */;
      }
}
```

4.6 Wrapper function to provide Software Property

4.6.1 EM_SW_GetProperty

Prototype

```
EM_SW_PROPERTY FAR_PTR * EM_SW_GetProperty(void);
```

Explanation

Return the Wrapper Property page, include all settings on wrapper layer.
The information, on return value will provide the information for library initialization inside the `EM_SW_PROPERTY` structure.

Argument(s)

None

Return value

Property setting of Wrapper layer. Refer to EM_SW_PROPERTY for more details.

Restriction/Caution

Take note on pointers with far attribute. Do not cast it into near attribute.

Sample Implementation

```
sw_property = EM_SW_GetProperty();
```

Revision History	RL78/I1C Group User's Manual: Software

Rev.	Date	Description	
		Page	Summary
1.00	Nov 18th, 2016	All	Initial Edition issued
2.00	Jun 13th, 2022	All	Update metrology structure
3.00	Sep 30th, 2022	All	Additional updates on metrology calculation
4.00	Dec 5th, 2022	All	Adjust header and description for APIs to get/set energy accumulation mode
5.00	Feb 16th, 2023	All	Improving document explanations and format
5.01	April 20th.2023	All	Improving document explanations and format

RL78/I1C Group User's Manual: Software

Publication Date: Rev5.01 April 20, 2023

Published by: Renesas Electronics Corporation

RL78/I1C Group

Renesas Electronics Corporation