User's Manual

16

RL78/I1C Group

User's Manual: 3PH4W Metrology

RENESAS MCU
RL78 Family / I1C Series

— Preliminary —

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev5.03 November 1, 2023

Table of Contents

1. Overview ... 5
1.1 Introduction ... 5
1.2 Specification ... 7
1.3 Occupied RAM, ROM, CPU Load ... 9
1.4 Meter firmware structure ... 10
1.4.1 Library structure ... 11
1.4.2 Wrapper structure ... 12
1.4.3 Directories / File structure ... 13
1.4.4 Metrology Operation Overview ... 13
1.5 Metrology Library Functions ... 14
1.5.1 Measurement ... 14
1.5.2 Operation ... 18
1.5.3 Event Status ... 21
1.6 Basic Operation Flow ... 24
1.6.1 Extraction of library information ... 24
1.6.2 Initialize drivers ... 25
1.6.3 Wrapper Implementation ... 26
1.6.4 Register callback functions in Interrupts ... 29
1.6.5 Starting up metrology ... 30

2. Data types & Definitions ... 31
2.1 Data types ... 31
2.2 Macro definitions ... 32
2.2.1 For EM_ERRCODE ... 32
2.2.2 For EM_CONSTRAINT ... 33
2.3 Structure definitions ... 34
2.3.1 EM_STATUS ... 34
2.3.2 EM_PLATFORM_PROPERTY ... 34
2.3.3 EM_CALIBRATION ... 35
2.3.4 EM_ENERGY_COUNTER ... 36
2.3.5 EM_ENERGY_VALUE ... 37
2.3.6 EM_SAMPLES ... 38
2.3.7 EM_OPERATION_DATA ... 39
2.3.8 EM_SW_PROPERTY ... 40
2.3.9 EM_ACCMODE_SAMPLES ... 45
2.3.10 EM_ACCMODE_ACCUMULATOR ... 45
2.4 Enum definitions ... 46
2.4.1 EM_LINE ... 46
2.4.2 EM_FUND_SEQUENCE ... 46
2.4.3 EM_PF_SIGN ... 47
2.4.4 EM_SYSTEM_STATE ... 47
2.5 Function definitions ... 48
2.5.1 Provided functions ... 48
2.5.2 Wrapper functions ... 51

3. Library functions ... 52
3.1 Common Functions ... 52
3.1.1 EM_Init ... 52
3.1.2 EM_Start ... 58
3.1.3 EM_Stop ... 59

3.1.4 EM_GetCalibInfo 60
3.1.5 EM_SetCalibInfo 61
3.1.6 EM_GetSystemState 64
3.1.7 EM_GetStatus 65
3.1.8 EM_GetEnergyAccumulationMode 66
3.1.9 EM_SetEnergyAccumulationMode 67
3.1.10 EM_SetEnergyAccumulationPower 68
3.1.11 EM_GetOperationData 69
3.1.12 EM_SetOperationData 70
3.2 Functions for Sample Accumulation 71
3.2.1 EM_AccMode_Run 71
3.2.2 EM_AccMode_CheckStatus 72
3.2.3 EM_AccMode_GetAccumulator 73
3.2.4 EM_ADC_AccMode_IntervalProcessing 74
3.3 Functions for Measurement Output 75
3.3.1 EM_GetActivePower 75
3.3.2 EM_GetFundamentalActivePower 76
3.3.3 EM_GetReactivePower 77
3.3.4 EM_GetApparentPower 78
3.3.5 EM_GetEnergyCounter 79
3.3.6 EM_EnergyCounterToEnergyValue 80
3.3.7 EM_EnergyValueToEnergyCounter 81
3.3.8 EM_AddEnergyCounter 82
3.3.9 EM_EnergyDataToEnergyValue 83
3.3.10 EM_EnergyValueToEnergyData 84
3.3.11 EM_AddEnergyData 85
3.3.12 EM_GetVoltageRMS 86
3.3.13 EM_GetCurrentRMS 87
3.3.14 EM_GetFundamentalVoltageRMS 88
3.3.15 EM_GetFundamentalCurrentRMS 89
3.3.16 EM_GetPowerFactor 90
3.3.17 EM_GetPowerFactorSign 91
3.3.18 EM_GetLineFrequency 92
3.3.19 EM_GetPhaseAngleRtoY 93
3.3.20 EM_GetPhaseAngleYtoB 94
3.3.21 EM_GetPhaseAngleBtoR 95
3.3.22 EM_GetCurrentPhaseAngle 96
3.3.23 EM_GetVoltageTHD 97
3.3.24 EM_GetCurrentTHD 98
3.3.25 EM_GetActivePowerTHD 99

4. Function from wrapper 100
4.1 Wrapper function for ADC 100
4.1.1 EM_ADC_Init 100
4.1.2 EM_ADC_Start 101
4.1.3 EM_ADC_Stop 102
4.1.4 EM_ADC_GainPhaseReset 103
4.1.5 EM_ADC_GainPhaseIncrease 104
4.1.6 EM_ADC_GainPhaseDecrease 105
4.1.7 EM_ADC_GainPhaseGetLevel 106
4.1.8 EM_ADC_SetPhaseCorrection 107
4.1.9 EM_ADC_SAMP_UpdateStep 108
4.1.10 EM_ADC_IntervalProcessing 109

4.2 Wrapper function for Pulse Output ... 110
4.2.1 EM_PULSE_Init ... 110
4.2.2 EM_PULSE_ACTIVE_On ... 111
4.2.3 EM_PULSE_ACTIVE_Off ... 112
4.2.4 EM_PULSE_REACTIVE_On ... 113
4.2.5 EM_PULSE_REACTIVE_Off ... 114
4.2.6 EM_PULSE_APPARENT_On ... 115
4.2.7 EM_PULSE_APPARENT_Off ... 116
4.3 Wrapper functions for Timer ... 117
4.3.1 EM_TIMER_Init ... 117
4.3.2 EM_TIMER_Start ... 118
4.3.3 EM_TIMER_Stop ... 119
4.3.4 EM_TIMER_InterruptCallback ... 120
4.4 Wrapper functions for Watch Dog Timer (WDT) ... 121
4.4.1 EM_WDT_Init ... 121
4.4.2 EM_WDT_Start ... 122
4.4.3 EM_WDT_Stop ... 123
4.4.4 EM_WDT_Restart ... 124
4.5 Wrapper function for MCU Utility ... 125
4.5.1 EM_MCU_Delay ... 125
4.5.2 EM_MCU_MultipleInterruptEnable ... 126
4.6 Wrapper function to provide Software Property ... 127
4.6.1 EM_SW_GetProperty ... 127

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.

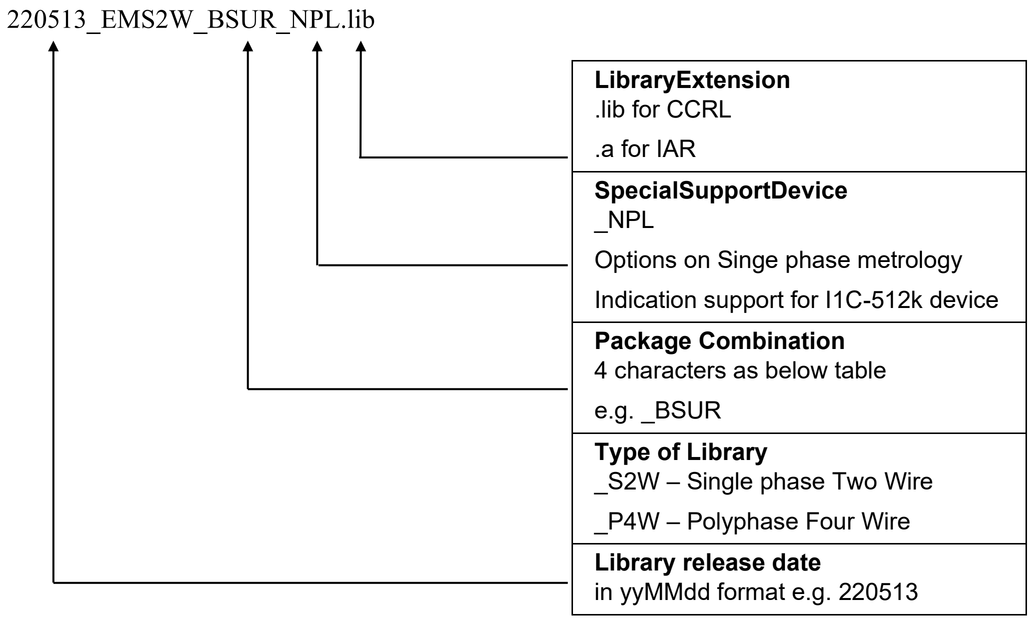


Table 1-1 Package Combination

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_EMP4W_BSUR.lib is the three-phase four-wire metrology library for CCRL compiler, without reactive measurement for common I1C device.

1.2 Specification

The following tables list specifications of library:

Table 1-2 Basic specification

Item	Specification
Support microcontroller	RL78 / I1C Group
Data-endian	Little-endian
MCU Frequency	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	Three Phase Four Wires (3P4W) No. of channels: • 3 Voltage Channel: V_{RN}, V_{YN}, V_{BN} • 4 Current Channels: I_R, I_Y, I_B, I_N (Support Shunt, CT and Rogowski coil sensor)

Library required modules	• Hardware acceleration of arithmetic calculation (MACL)
	• Interval timer (40ms) with interrupt.
	• DSAD continuous conversion with interrupt (highest priority)
	• ADC with double sampling trigger for voltage and 90degree voltage sample [Note1]
	  ○ I1C: 10bit ADC
	  ○ I1C 512k: 12bit ADC
	  ○ ELC, DTC
	• PORT, 3 I/O pins for pulse output
	• *WDT (optional).*

Note1: This document focuses on information related to metrology library and wrapper, driver setup for sampling chain would be available in separate documentation.

Table 1-3 Library features

Item	Specification
Gain Switch	Support to sense the sampling counter; switch to higher amplifier gain on current signal 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	Support nominal 3906Hz sampling frequency
Pulse output	3 channels, IEC standard, configurable pulse on duration in ms
Sampling resolution	Max 24 bits counter
Measurement output	• VRMS (Per phase and Total) • Fundamental VRMS (Per phase and Total) • IRMS (Per phase, Neutral and Total) • Fundamental IRMS (Per phase and Total) • Power: Active, Fundamental Active, Reactive, Apparent (Per phase and Total) [Note] • Energy Accumulation: Active, Reactive, Apparent (Total) • Power Factor (Per phase and Total) • Power Factor sign (Lead, Lag, Unity) (Per phase and Total) • Line Frequency (Per phase and Total) • Voltage phase to phase angle. • Current phase to phase angle, phase to neutral angle. • Vector sum current. • Total harmonic distortion (VRMS, IRMS, Active Power) (Per phase and Total) • Meter calibration information • Event Status Bit: Meter no-load status, voltage sag & swell

Note: Reactive and Fundamental Active are only supported on certain library versions.

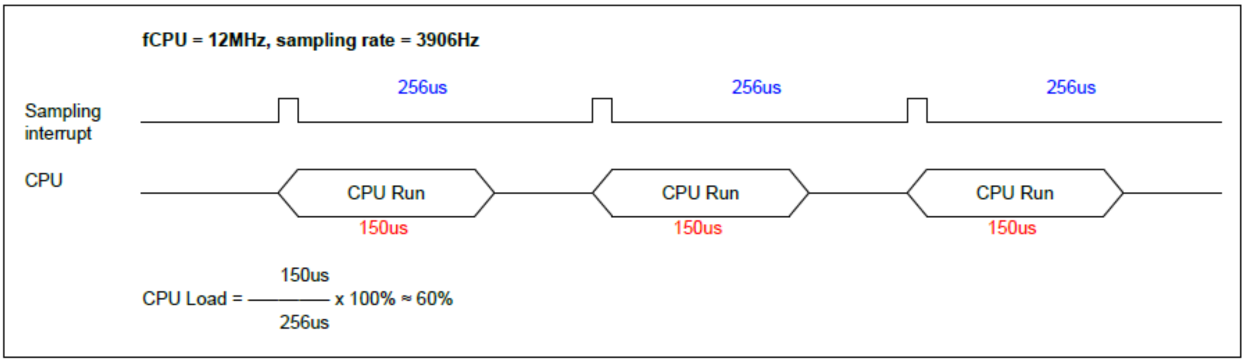
1.3 Occupied RAM, ROM, CPU Load

How is the CPU load of the 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), allowing all CPU load of the library to be estimable by measuring based on DSAD interrupt, with how much time spend for processing of 1 sampling set (R, Y, B).

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



Below is a summary for **version 230529**.

Table 1-4 ROM, RAM, and CPU load of library+wrapper CCRL, sampling frequency 3906Hz, CPU @ 24MHz

Suffix	Library with included modules	ROM (in bytes)	RAM (in bytes)	% Max CPU Load
BSUR	EM Basic	16253	1984	28.499
BQUR	EM Basic + Reactive	17681	2212	34.098
BQFR	EM Basic + Reactive + Fundamental	21311	2632	73.584
WSUR	EM Basic + Gain Switch	18042	1984	29.818
WQUR	EM Basic + Gain Switch + Reactive	19832	2212	35.612
WQFR	EM Basic + Gain Switch + Reactive + Fundamental	24512	2632	75.439

Table 1-5 ROM, RAM, and CPU load of library+wrapper of NPL CCRL, sampling frequency 3906Hz, CPU @ 24MHz

Suffix	Library with included modules	ROM (in bytes)	RAM (in bytes)	% Max CPU Load
BSUR	EM Basic	16247	1984	28.125
BQUR	EM Basic + Reactive	17652	2212	34.310
BQFR	EM Basic + Reactive + Fundamental	21282	2632	73.796
WSUR	EM Basic + Gain Switch	18036	1984	29.443
WQUR	EM Basic + Gain Switch + Reactive	19803	2212	35.824
WQFR	EM Basic + Gain Switch + Reactive + Fundamental	24483	2632	75.651

1.4 Meter firmware structure

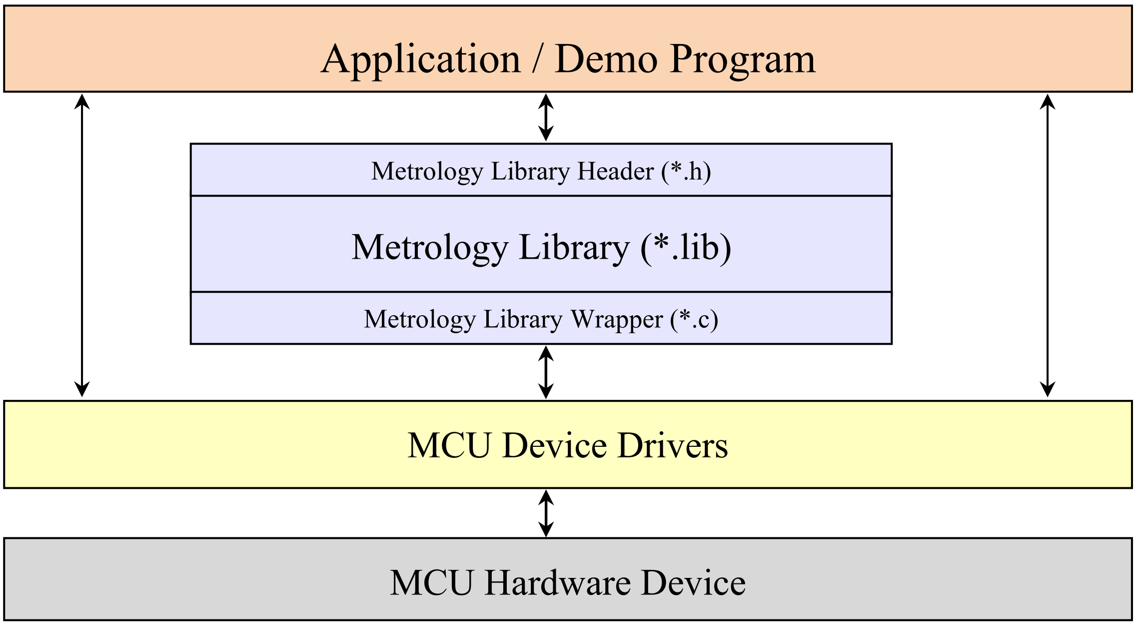


Figure 1-1 Meter firmware structure

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 flags. 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

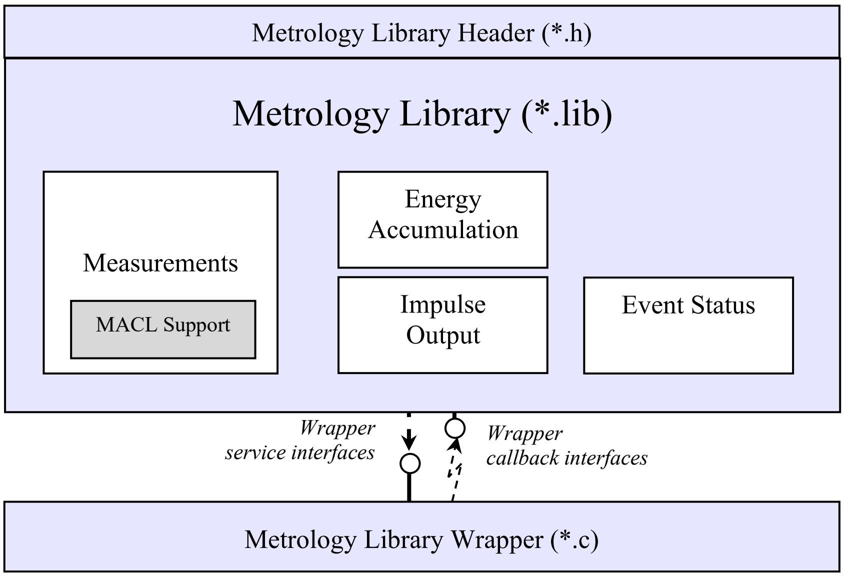


Figure 1-2 Metrology Library structure

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 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.

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

The 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.

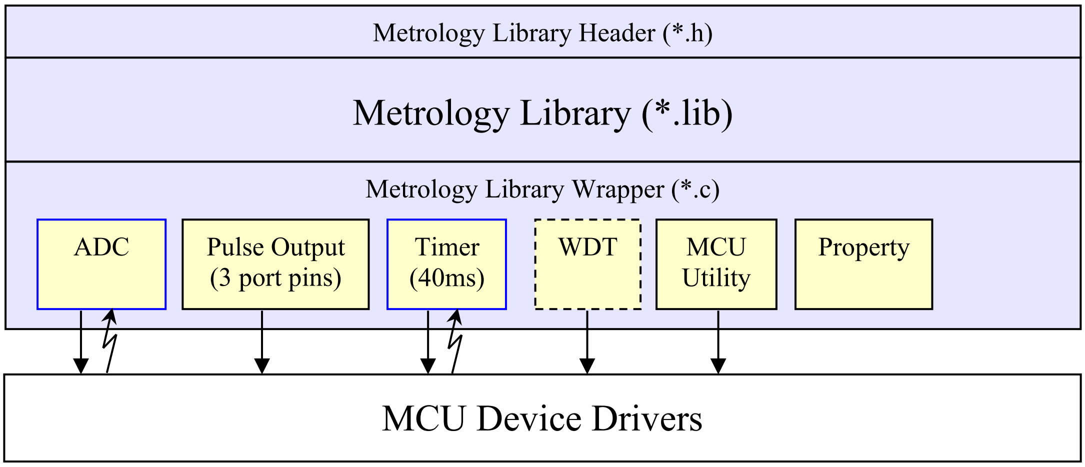


Figure 1-3 Metrology Library Wrapper structure

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 Note1	Level 0 (highest), acknowledgement of conversion end.
Timer	Level 3, interval callback, 40ms

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

Note1: ADC wrapper for metrology seen as 1 but for 3Ph4W this included sampling chain drivers for DSAD, 10bitADC (12bit ADC in 512k device), ELC and DTC

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

1.4.4 Metrology Operation Overview

An overview of the metrology operation timings of the ADC, Timer (40ms) and RTC interrupts can be reviewed in the figure below.

*Note: Total parameter calculation of R,Y, B will only occur in Timer INT after all accumulation of the phases are processed in buffer.

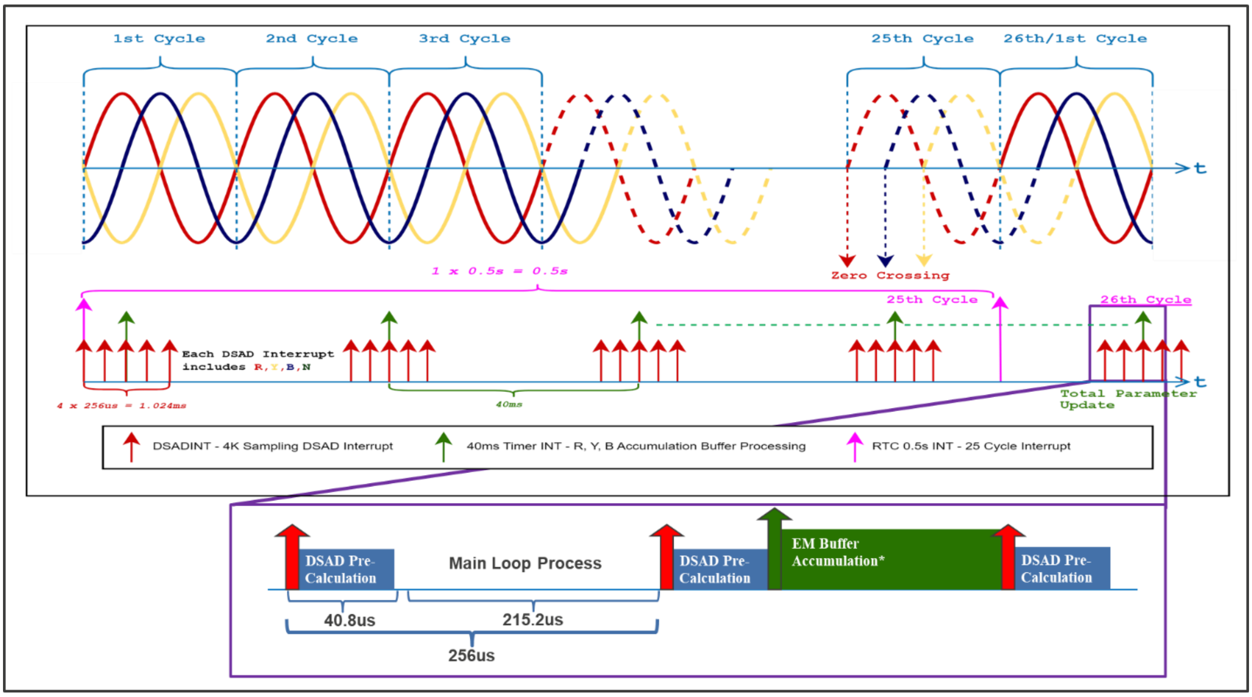


Figure 1-5 Timing Diagram on 24MHz

1.5 Metrology Library Functions

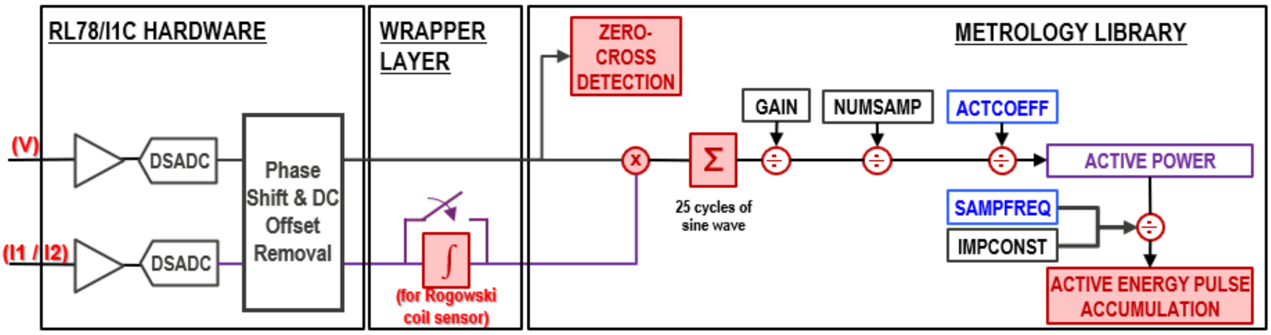
The following charts describe all measurement metrologies.

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

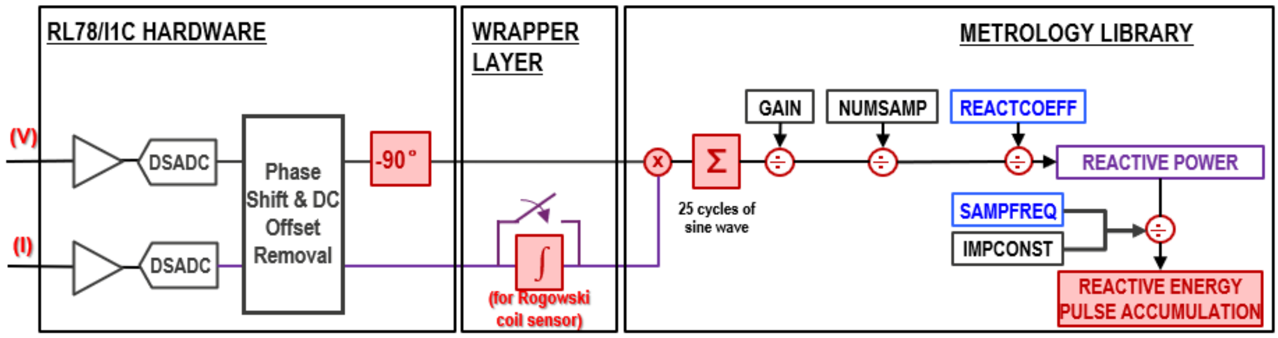
Note: For **versions 220915 and below**, the RMS accumulation is 25 cycles, while Power accumulation is 50 cycles. For the later version, both RMS and Power accumulation uses 25-line cycles accumulation length.

1.5.1 Measurement

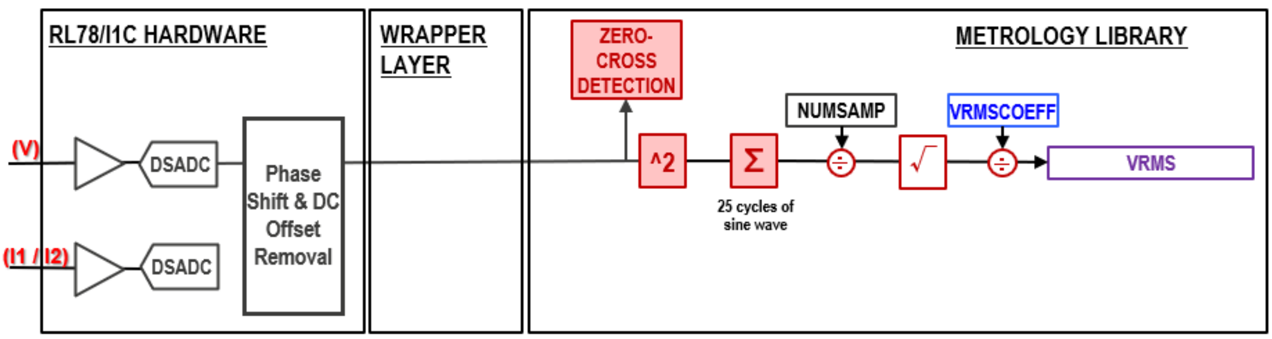
1.5.1.1 Active Power and Active Energy



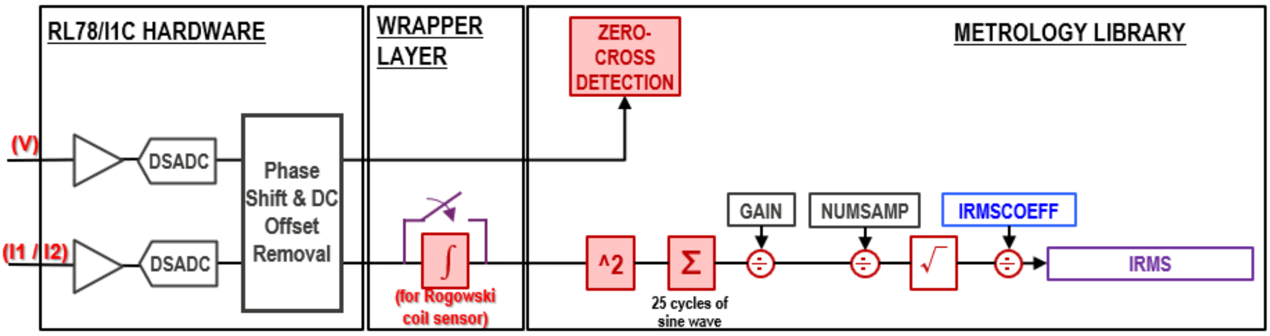
1.5.1.2 Reactive Power and Reactive Energy



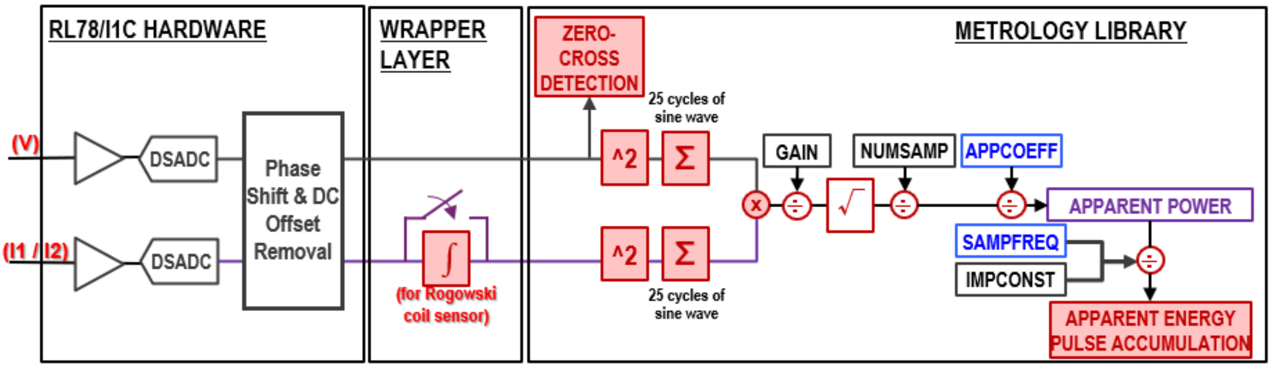
1.5.1.3 VRMS (true RMS)



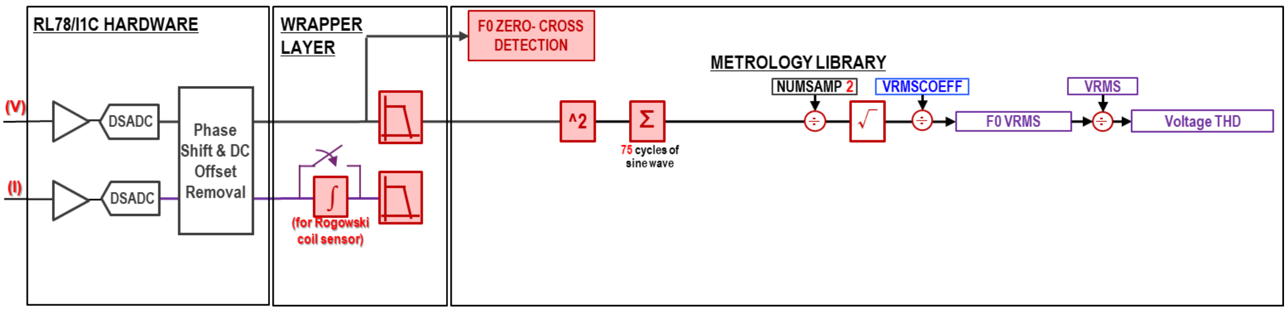
1.5.1.4 IRMS (true RMS)



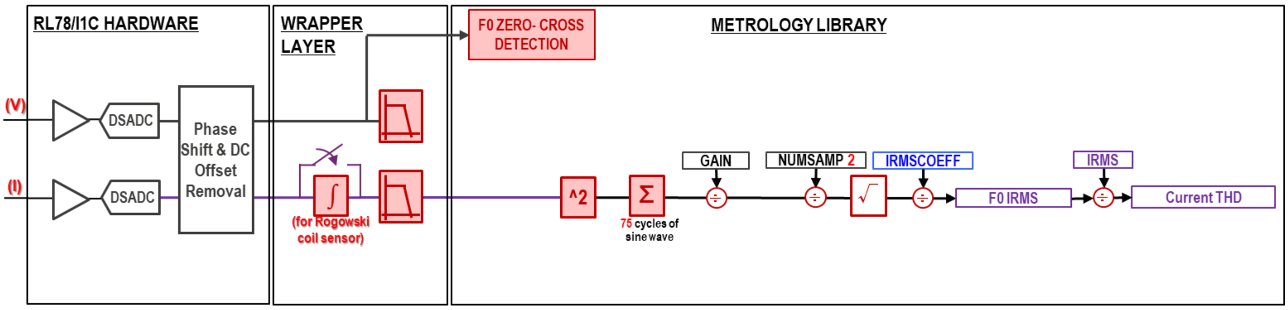
1.5.1.5 Apparent Power & Energy



1.5.1.6 Fundamental VRMS (true RMS)

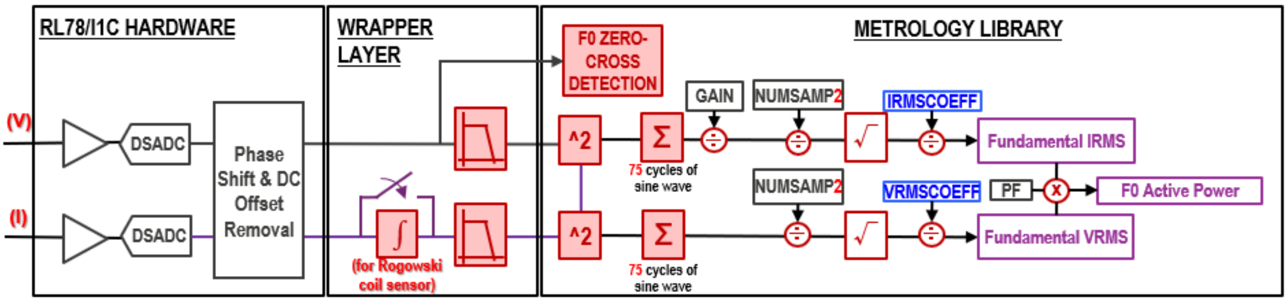


1.5.1.7 Fundamental IRMS (true RMS)

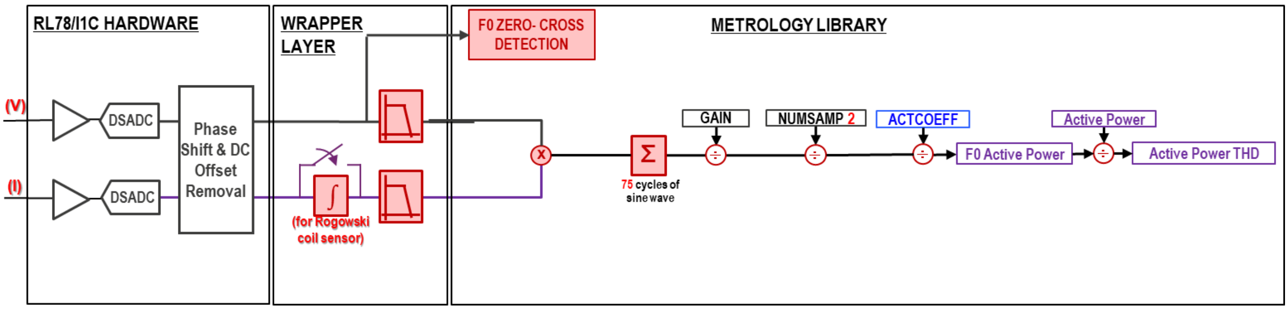


1.5.1.8 Fundamental Active Power (F0 Active Power)

For **versions below 230529**



For **versions 230529 and above**



1.5.1.9 Power Factor (PF)

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

1.5.1.10 Line Frequency

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

1.5.1.11 Total Harmonic Distortion

$$\text{THD (RMS)} = \frac{\text{SQRT(ABS(RMS\^{}2 – Fundamental\_RMS\^{}2))}}{\text{Fundamental\_RMS}}$$

$$\text{THD (Power)} = \frac{\text{ABS (ABS(Power) – ABS(Fundamental\_Power))}}{\text{Fundamental\_Power}}$$

THD reading is in ratio, not in percentage

THD RMS <= 0.04 will be masked when read out

THD Power <= 0.0016 will be masked when read out

1.5.1.12 Line to Line Angle Measurement

- Voltage phase to phase angle measurement.
- Current phase to phase angle measurement.
- Current phase to neutral angle measurement.

Angle measurement based on number of samples elapsed between signals zero cross in 25-line cycles.

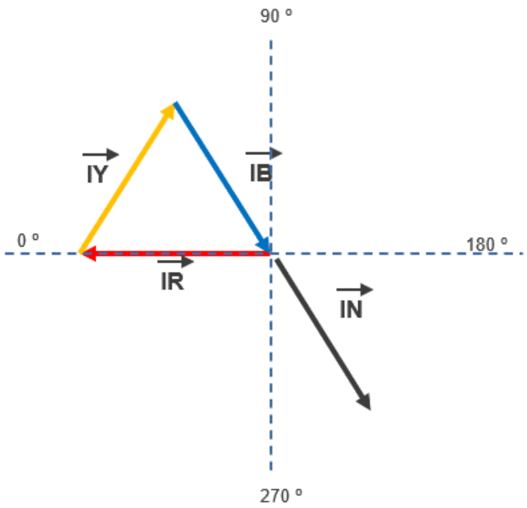
$$\text{Angle (Degree)} = \frac{\text{Number of samples elapsed between zero cross within 25-line cycles} \times 360}{\text{Number of samples within 25-line cycles}}$$

1.5.1.13 Vector Sum Current

Vector sum current is sum of all current vector with 2 information: IRMS, phase angle between current signals.

- IRMS_R, IRMS_Y, IRMS_B, IRMS_N
- Current angle of: RtoY, RtoB, RtoN, YtoR, YtoB, YtoN, BtoR, BtoY, BtoN

$$\overrightarrow{I_{vector}} = \overrightarrow{I_R} + \overrightarrow{I_Y} + \overrightarrow{I_B} + \overrightarrow{I_N}$$



VECTOR_SUM (RMS) = SQRT(IRMS_HorizonalSum + IRMS_VerticalSum)

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 PGA gain of MCU on current channels (except for neutral).

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

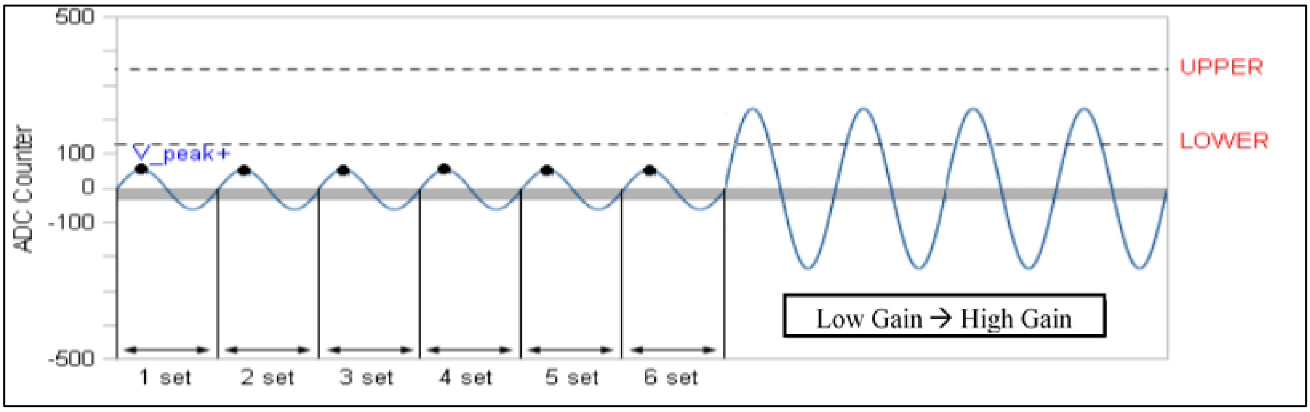


Figure 1-6 Low-High gain switching at low current sensing

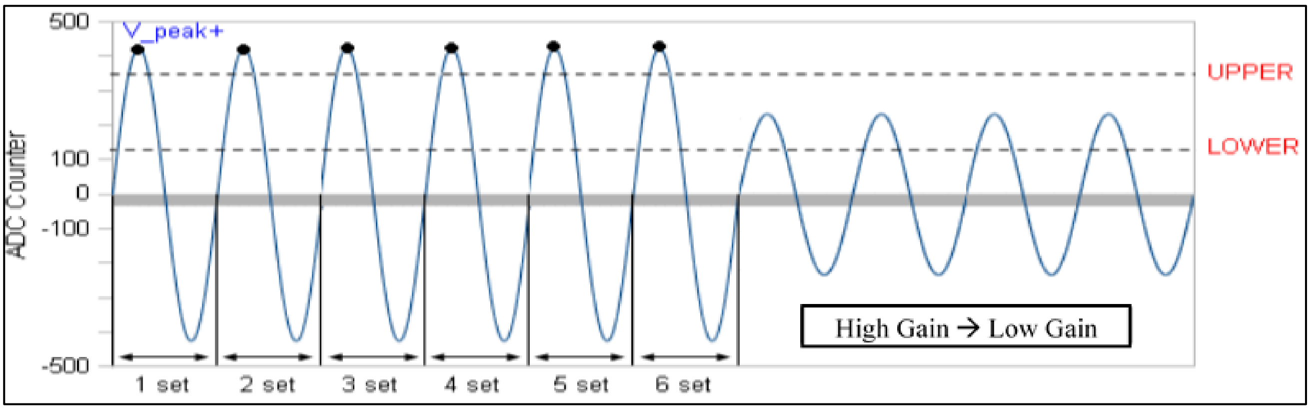


Figure 1-7 High-Low gain switching at high current sensing.

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.

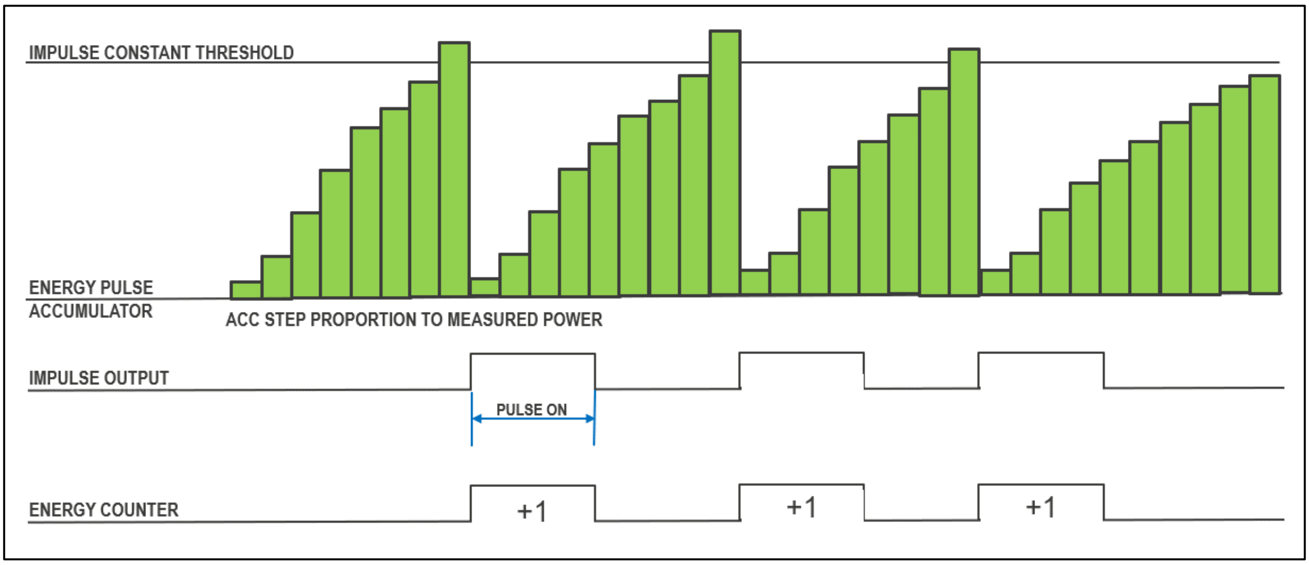


Figure 1-8 Illustration of energy and pulse accumulation

1.5.2.3 Energy accumulation mode

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

0	The library stops updating step for energy accumulation. Users call EM_SetEnergyAccumulationPower to update
1	Library always use Phase channel power for energy accumulation
2	Library use phase channel absolute power for energy accumulation (forward only active energy)

On updating, measured power of all phases is summed together and used for updating the energy accumulation step.

Difference between mode 1 and mode 2 in summing active power:

In mode 1: active energy accumulation step by: active_power_R + active_power _Y + active_power _B

In mode 2: active energy accumulation step by: abs(active_power _R) + abs(active_power _Y) + abs(active_power _B)

1.5.2.4 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 is processed internally by the 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.

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

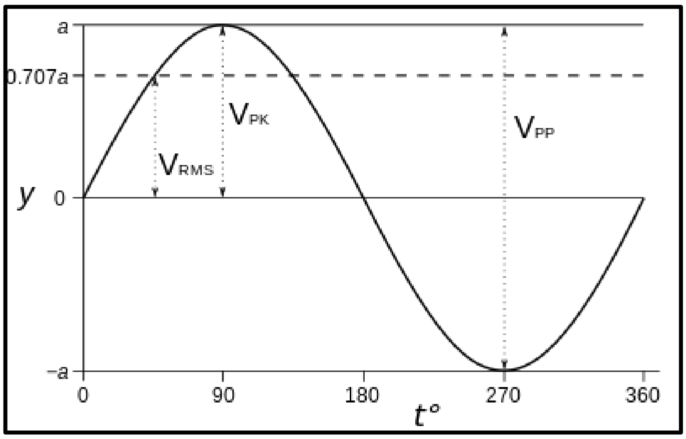


Figure 1-9 Sine wave RMS and peak

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:

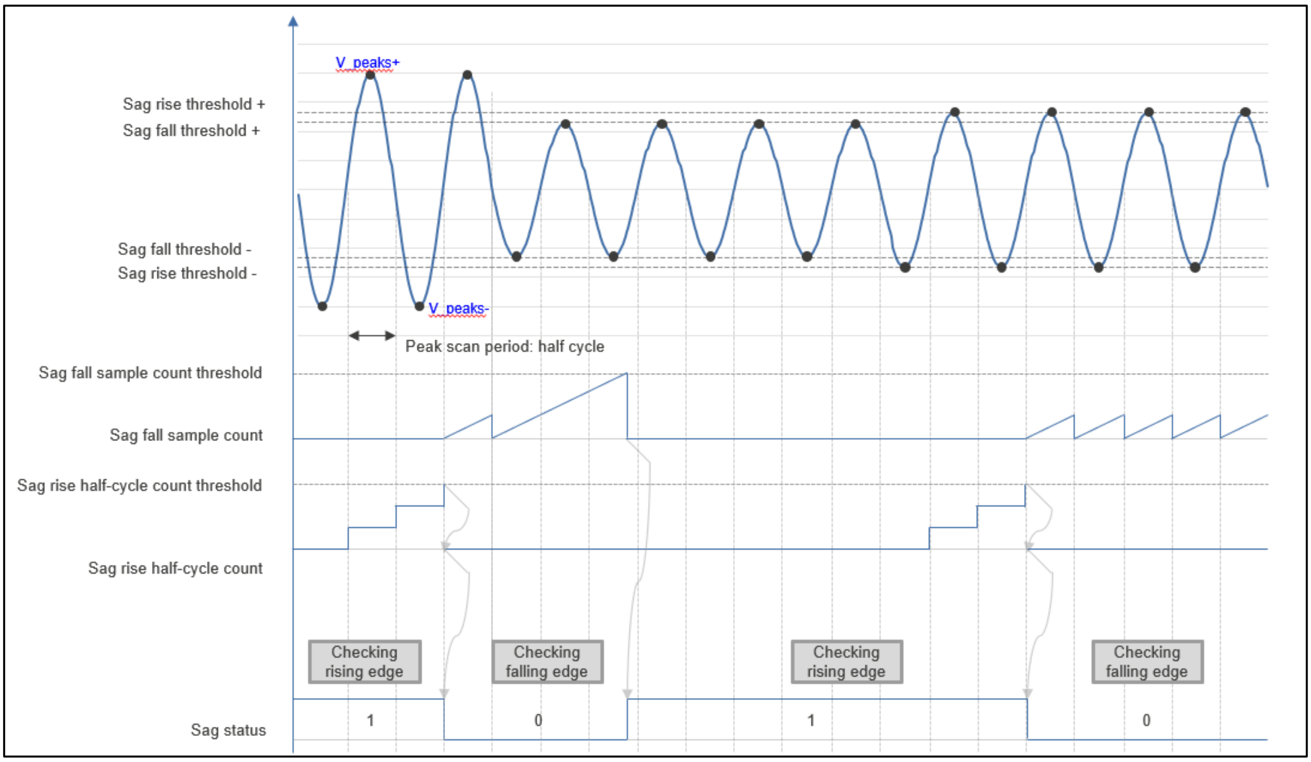


Figure 1-10 Sag falling and rising detection.

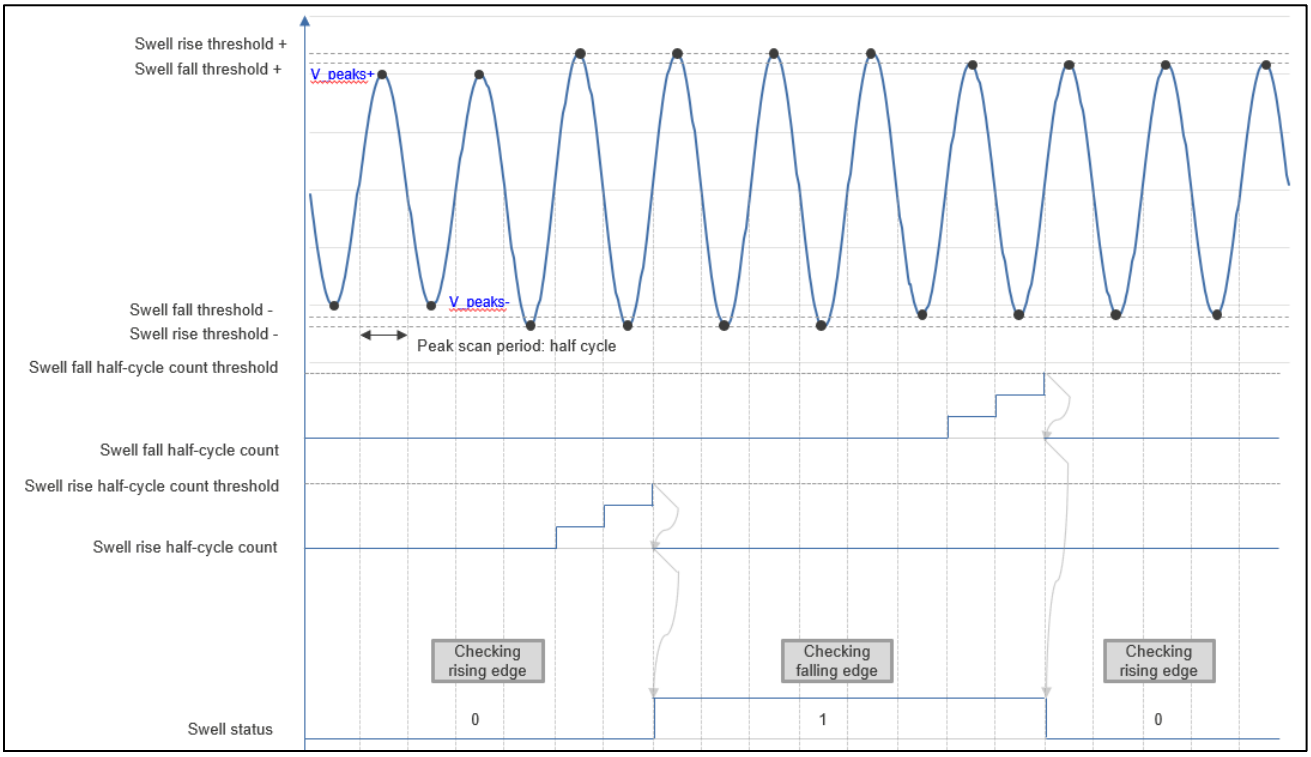


Figure 1-11 Swell falling and rising detection

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:

e.g. sample peripheral initialization on I1C device

```
#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"
#include "r_cg_adc.h"
#include "r_cg_dtc.h"
#include "r_cg_elc.h"
void R_Systeminit(void)
{
        R_PORT_Create();
        R_CGC_Create();
        R_TAU0_Create();
        R_WDT_Create();
        R_DSADC_Create();
        R_MAC32Bit_Create();
        R_ADC_Create();
        R_DTC_Create();
        R_ELC_Create();
};
```

e.g. sample peripheral initialization on I1C 512k device

```
#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"
#include "r_cg_12adc.h"
#include "r_cg_dtc.h"
#include "r_cg_elc.h"
void R_Systeminit(void)
{
        R_PORT_Create();
        R_CGC_Create();
        R_TAU0_Create();
        R_WDT_Create();
        R_DSADC_Create();
        R_MAC32Bit_Create();
        R_12ADC_Create();
        R_DTC_Create();
        R_ELC_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

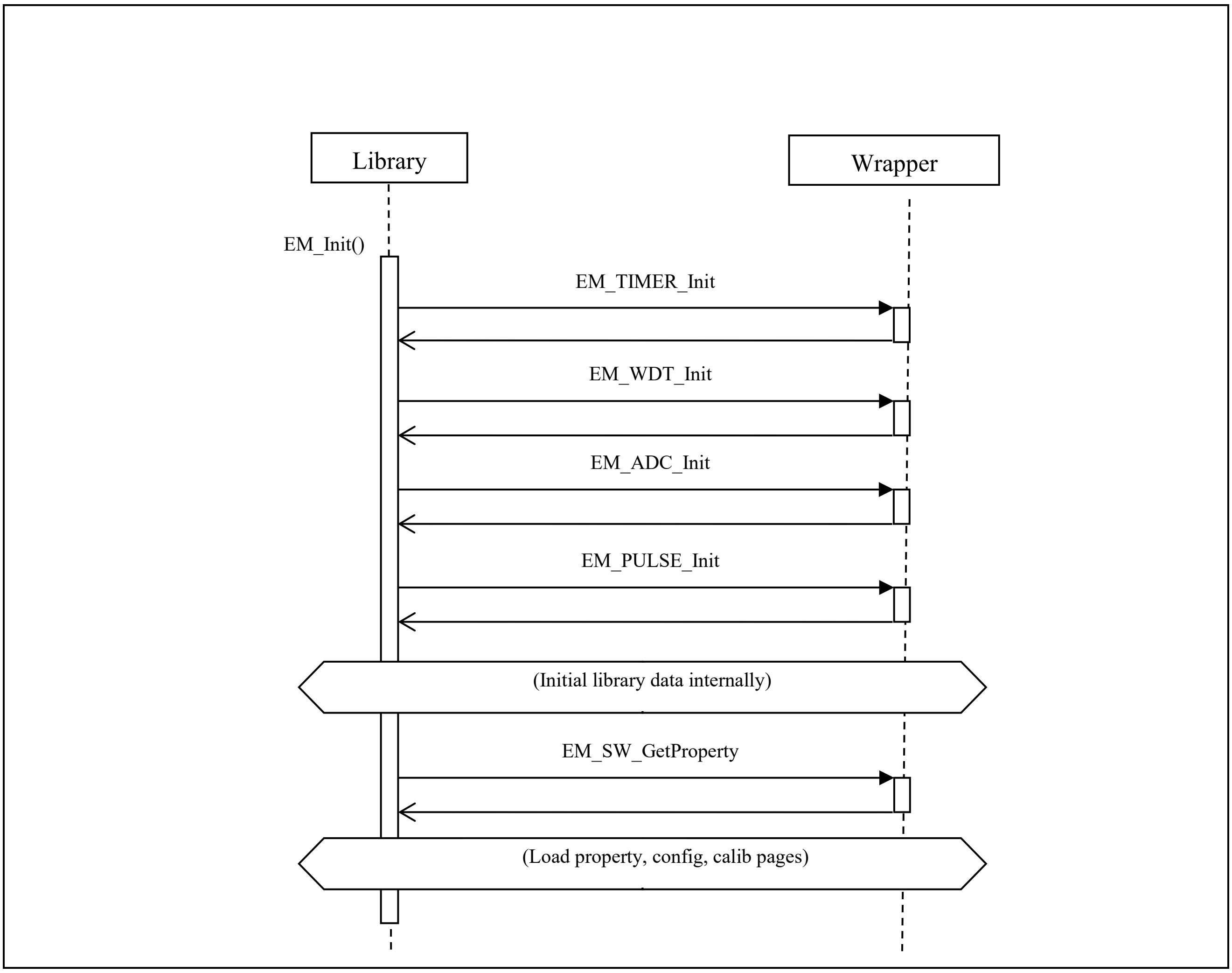


Figure 1-12 Wrapper initialization through library calls

1.6.3.2 Start

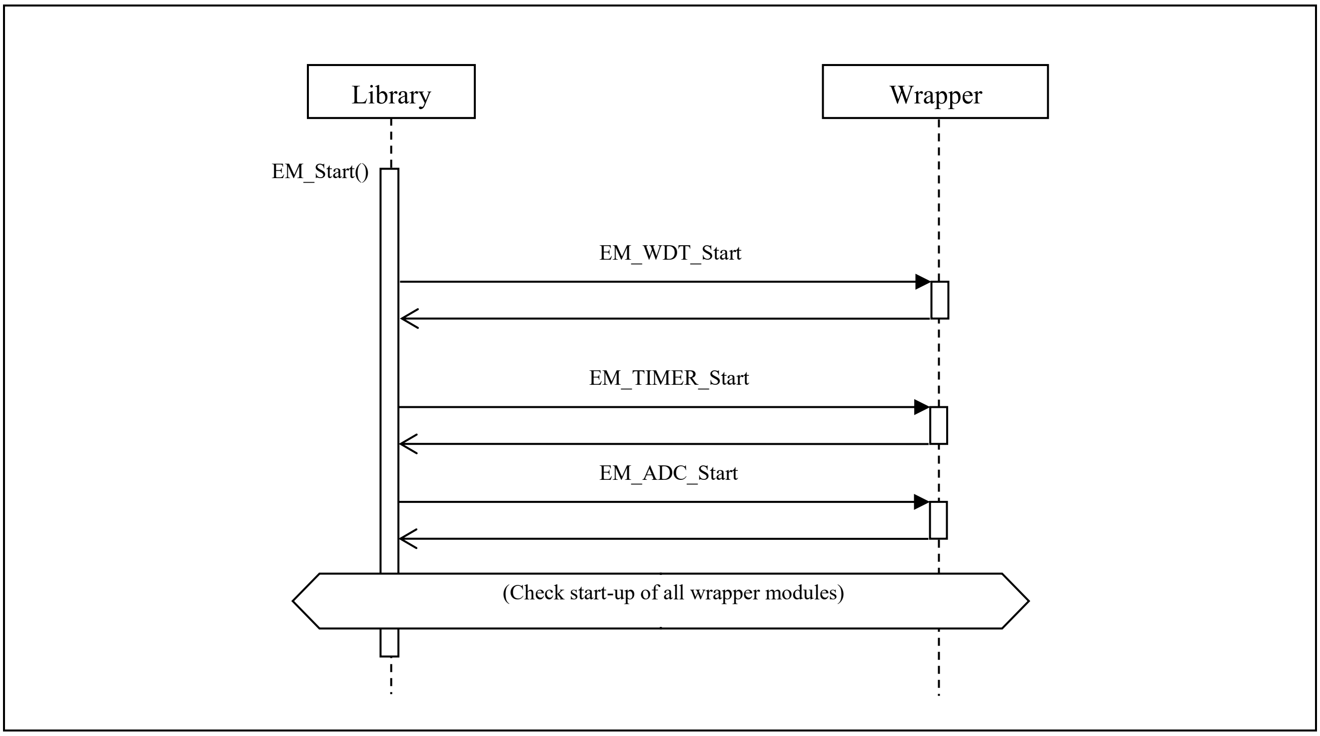


Figure 1-13 Wrapper module start-up through library calls

1.6.3.3 Stop

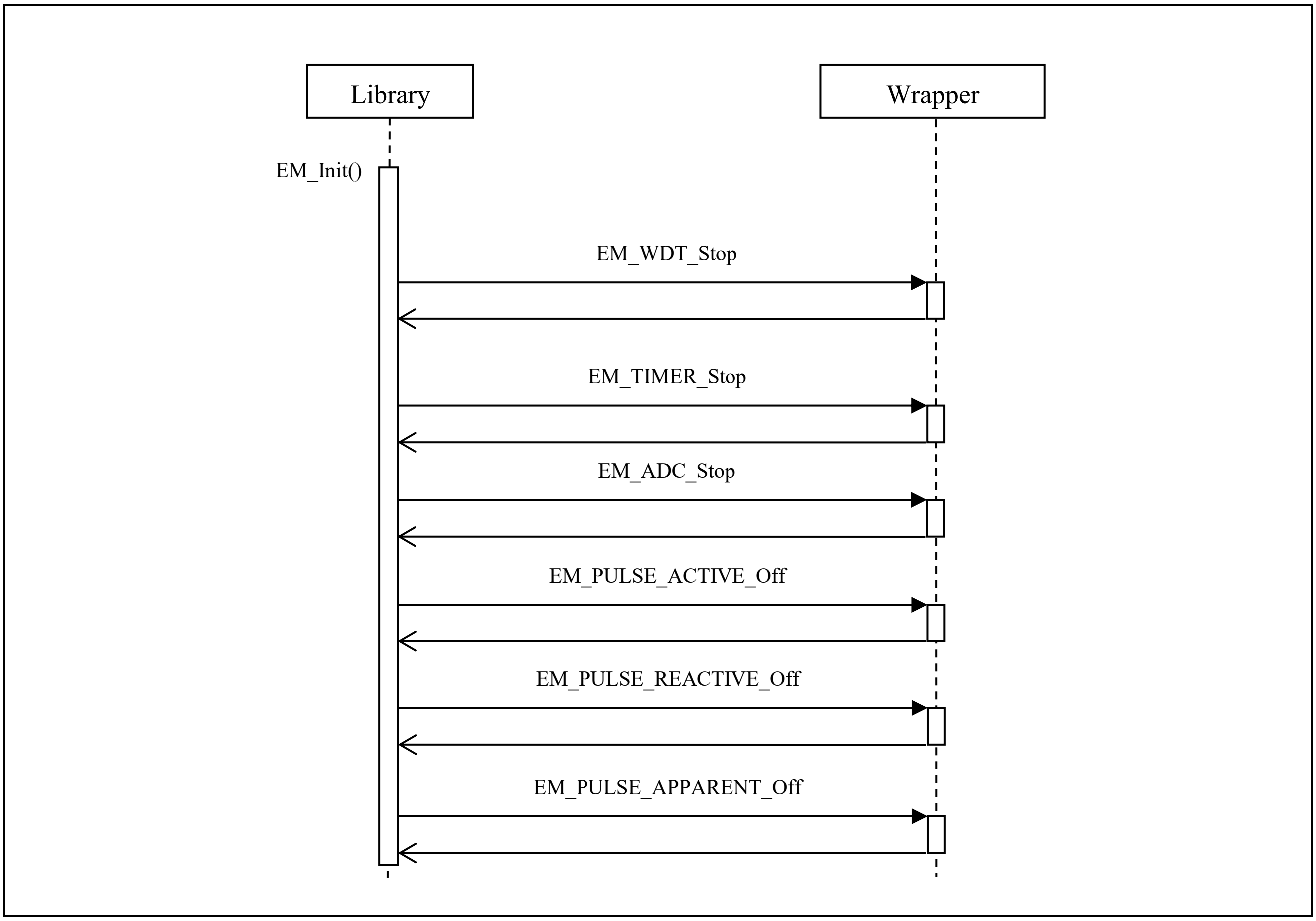


Figure 1-14 Wrapper modules stop through library calls.

1.6.4 Register callback functions in Interrupts

a) ADC

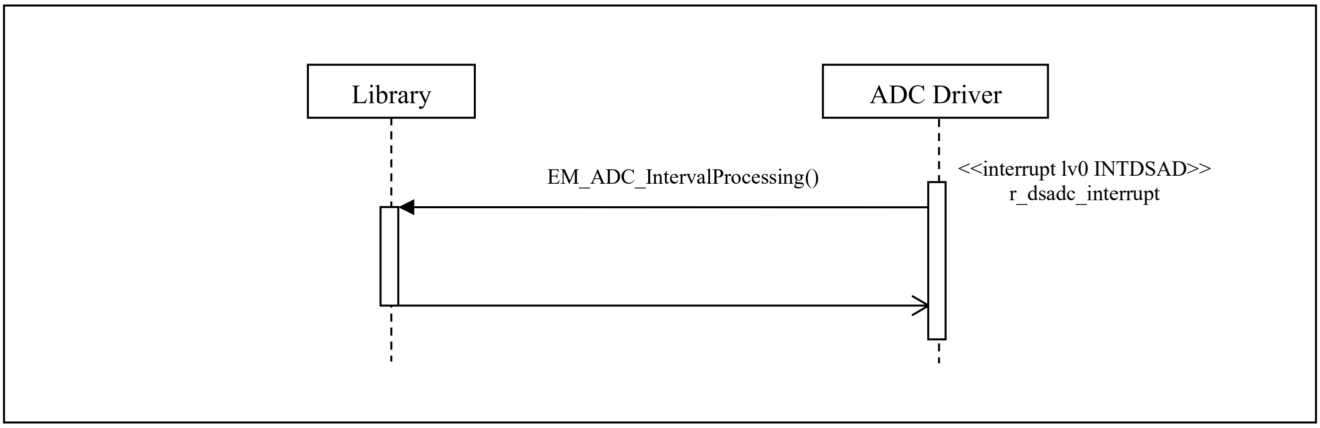


Figure 1-15 ADC Driver interrupt call to library

b) TIMER

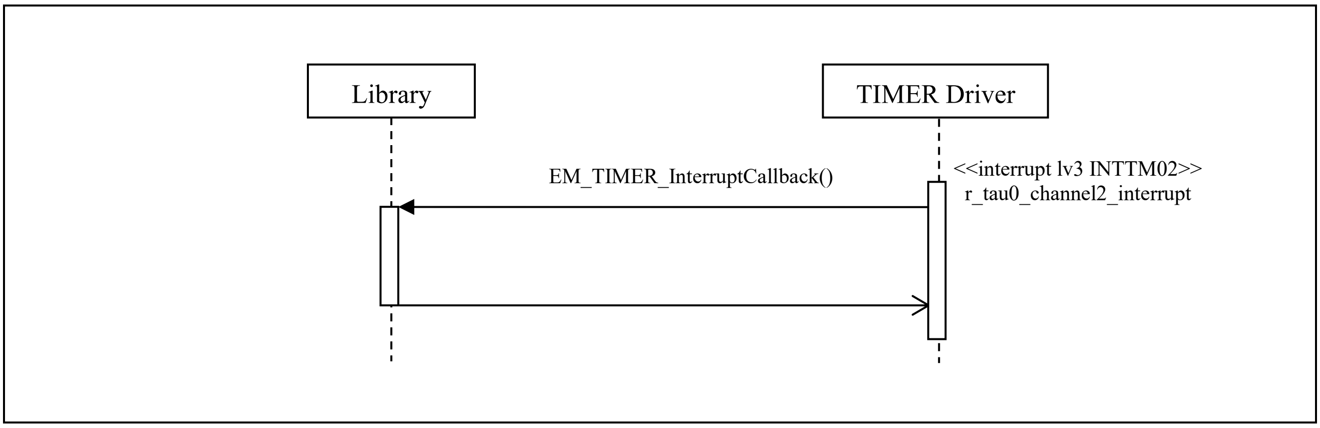


Figure 1-16 TIMER Driver interrupt call to library

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

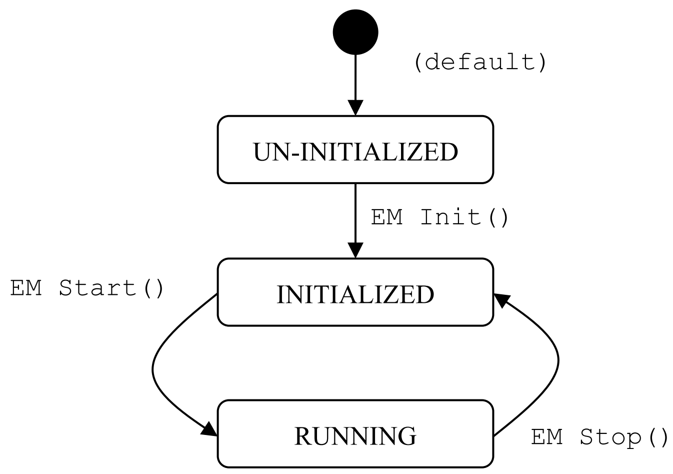


Figure 1-17 State of metrology library

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

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_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	3	Voltage channel number
EM_CURRENT_CHANNEL_NUM	4	Current channel number
EM_CALC_NUM_OF_LINE	5	Number of line (3 phase, neutral, total)
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_PULSE_ON_TIME_MIN	10.0f	Pulse On Time Min
EM_VRMS_COEFF_MIN	0.1f	VRMS Co-efficient min
EM_IRMS_COEFF_MIN	0.1f	IRMS 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 indicates 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_R	No load status of Active Power Phase R
uint16_t:1	noload_reactive_R	No load status of Active Power Phase Y
uint16_t:1	noload_active_Y	No load status of Active Power Phase B
uint16_t:1	noload_reactive_Y	No load status of Reactive Power Phase R
uint16_t:1	noload_active_B	No load status of Reactive Power Phase Y
uint16_t:1	noload_reactive_B	No load status of Reactive Power Phase B
uint16_t:1	voltage_sag_R	Voltage Sag Phase R
uint16_t:1	voltage_sag_Y	Voltage Sag Phase Y
uint16_t:1	voltage_sag_B	Voltage Sag Phase B
uint16_t:1	voltage_swell_R	Voltage Swell Phase R
uint16_t:1	voltage_swell_Y	Voltage Swell Phase Y
uint16_t:1	voltage_swell_B	Voltage Swell Phase B

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 (92 bytes)

Datatype	Structure element			Explanation
float32_t (4 bytes)	sampling_frequency			Actual sampling frequency of meter
Struct (64 bytes)	coeff			Specify co-efficient of input signal
	struct	phase_r		Phase R coefficients
		float32_t	vrms	VRMS Co-efficient
		float32_t	irms	IRMS Co-efficient
		float32_t	active_power	Active power coefficient
		float32_t	reactive_power	Reactive power coefficient
		float32_t	apparent_power	Apparent power coefficient
	struct	phase_y		Phase Y coefficients
		float32_t	vrms	VRMS Co-efficient
		float32_t	irms	IRMS Co-efficient
		float32_t	active_power	Active power coefficient
		float32_t	reactive_power	Reactive power coefficient
		float32_t	apparent_power	Apparent power coefficient
	struct	phase_b		Phase B coefficients
		float32_t	vrms	VRMS Co-efficient
		float32_t	irms	IRMS Co-efficient
		float32_t	active_power	Active power coefficient
		float32_t	reactive_power	Reactive power coefficient
		float32_t	apparent_power	Apparent power coefficient
	struct	neutral		Neutral coefficients
		float32_t	irms	IRMS Co-efficient
Struct (12 bytes)	sw_phase_correction			Phase correction list (degree)
	struct	phase_r		Phase correction Phase R
		float32_t FAR_PTR *	i_phase_degrees	Phase Angle Degree List
	struct	phase_y		Phase correction Phase Y
		float32_t FAR_PTR *	i_phase_degrees	Phase Angle Degree List
	struct	phase_b		Phase correction Phase B
		float32_t FAR_PTR *	i_phase_degrees	Phase Angle Degree List
Struct (12 bytes)	sw_gain			Gain value list
	struct	phase_r		Gain Phase R
		float32_t FAR_PTR *	i_gain_values	Gain Value List
	struct	phase_y		Gain Phase Y
		float32_t FAR_PTR *	i_gain_values	Gain Value List
	struct	phase_b		Gain Phase B
		float32_t FAR_PTR *	i_gain_values	Gain Value List

Structure Element Definitions

sampling_frequency

Set the calibrated sampling frequency of meter, in Hz.

coeff struct

Set vrms, irms for RMS value calibration on each phase and neutral (irms only)

Set active_power, reactive_power, apparent_power for corresponding power value calibration on each phase

The limit of set value is defined by macros as following:

```
EM_VRMS_COEFF_MIN
EM_IRMS_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 Current channels, in degree, on `i_phase_degrees` list. of each Set phase

- 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 `i_phase_degrees[0],` keep others as 0.

sw_gain

Set gain value for Current channels on `i_gain_values` list of each phase.

Max of 2 level 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 value to `i_gain_values[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_SAMPLES

Explanation

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

Structure (66 bytes)

Datatype		Structure element	Explanation
struct (20 bytes)	phase_r		Phase_R sample
	EM_SW_SAMP_TYPE	i	Current
	EM_SW_SAMP_TYPE	v	Voltage
	EM_SW_SAMP_TYPE	v90	Voltage 90 degree phase shift
	EM_SW_SAMP_TYPE	i_fund	Filtered current for fundamental calculation
	EM_SW_SAMP_TYPE	v_fund	Filtered voltage for fundamental calculation
struct (20 bytes)	phase_y		Phase_Y sample
	EM_SW_SAMP_TYPE	i	Current
	EM_SW_SAMP_TYPE	v	Voltage
	EM_SW_SAMP_TYPE	v90	Voltage 90 degree phase shift
	EM_SW_SAMP_TYPE	i_fund	Filtered current for fundamental calculation
	EM_SW_SAMP_TYPE	v_fund	Filtered voltage for fundamental calculation
struct (20 bytes)	phase_b		Phase_B sample
	EM_SW_SAMP_TYPE	i	Current
	EM_SW_SAMP_TYPE	v	Voltage
	EM_SW_SAMP_TYPE	v90	Voltage 90 degree phase shift
	EM_SW_SAMP_TYPE	i_fund	Filtered current for fundamental calculation
	EM_SW_SAMP_TYPE	v_fund	Filtered voltage for fundamental calculation
struct (4 bytes)	neutral		Neutral sample
	EM_SW_SAMP_TYPE	i	Current
struct (2 bytes)	status		Sample status
	EM_FUND_SEQUENCE	fund_sequence	Fund voltage sample indicator
	Uint8_t	reserved	

2.3.7 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

2.3.8 EM_SW_PROPERTY

Explanation

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

Structure (86 bytes)

Datatype	Structure element			Explanation
struct (30 bytes)	adc			Gain switching function
	struct	phase_r		Phase_R gain switch configuration
		uint8_t	gain_phase_num_level	Number of gains for current channel
		uint32_t	gain_upper_threshold	Upper threshold to switch to lower gain
		uint32_t	gain_lower_threshold	Lower threshold to switch to higher gain
	struct	phase_y		Phase_Y gain switch configuration
		uint8_t	gain_phase_num_level	Number of gains for current channel
		uint32_t	gain_upper_threshold	Upper threshold to switch to lower gain
		uint32_t	gain_lower_threshold	Lower threshold to switch to higher gain
	struct	phase_b		Phase_B gain switch configuration
		uint8_t	gain_phase_num_level	Number of gains for current channel
		uint32_t	gain_upper_threshold	Upper threshold to switch to lower gain
		uint32_t	gain_lower_threshold	Lower threshold to switch to higher gain
struct (32 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)
	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 (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_num_level

Specify how many gains used for the phase current. This setting is mandatory and will affect the `EM_CALIBRATION` struct, on `i_phase_degree, i_gain_values`.

Minimum: 1

Maximum: 2

Example,

- If 2 gains are used, the first 2 elements of the array `i_phase_degree` must have phase error values, while others value can be kept as 0. Next, the first element on `i_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_upper_threshold
gain_lower_threshold

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

The upper/lower ratio should be greater than the ratio of the second gain / first gain in `i_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`

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.

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

2.3.9 EM_ACCMODE_SAMPLES

Explanation

The `EM_ACCMODE_SAMPLES` structure is metrology sample accumulation input signal buffer with type of EM_SW_SAMP_TYPE to be input for sample accumulation function.

Structure (24 bytes)

Datatype	Structure element	Explanation
EM_SW_SAMP_TYPE	signal0	Signal0 for acc0
EM_SW_SAMP_TYPE	signal1	Signal1 for acc0
EM_SW_SAMP_TYPE	signal2	Signal2 for acc1
EM_SW_SAMP_TYPE	signal3	Signal3 for acc1
EM_SW_SAMP_TYPE	signal4	Signal4 for acc2
EM_SW_SAMP_TYPE	signal5	Signal5 for acc2

2.3.10 EM_ACCMODE_ACCUMULATOR

Explanation

The `EM_ACCMODE_ACCUMULATOR` structure is metrology sample accumulation accumulated buffer with type of float32_t to be read out as result of accumulation. 3 accumulators according to pair of input signals in EM_ACCMODE_SAMPLES structure

Structure (12 bytes)

Datatype	Structure element	Explanation
float32_t	acc0	Accumulator for signal0 and signal1
float32_t	acc1	Accumulator for signal2 and signal3
float32_t	acc2	Accumulator for signal4 and signal5

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

Enum values (1 byte)

Enum Value	Significance
LINE_PHASE_R = 0x00	Phase R line is selected
LINE_PHASE_Y = 0x01	Phase Y line is selected
LINE_PHASE_B = 0x02	Phase B line is selected
LINE_NEUTRAL = 0x03	Neutral line is selected
LINE_TOTAL = 0x04	Total sum of all phase lines

2.4.2 EM_FUND_SEQUENCE

Explanation

The `EM_FUND_SEQUENCE` enumeration controls the sequence order of fundamental calculation.

This sequence provides information about fundamental filtered current sample:

- Either EM_SW_FUND_SEQUENCE_PHASE_R, EM_SW_FUND_SEQUENCE_PHASE_Y, EM_SW_FUND_SEQUENCE_PHASE_B specified, effective sampling rate on current sample dropped to 1/3 and its value indicating which filtered current sample currently in EM_SAMPLES structure.
- If EM_SW_FUND_SEQUENCE_PHASE_ALL specified, means all filtered current sample in EM_SAMPLES structure for R,Y,B is valid, and effective sampling rate is same as filtered voltage sample.

Enum values (1 byte)

Enum Value	Significance
EM_SW_FUND_SEQUENCE_PHASE_R	Sample set contain Phase R filtered voltage
EM_SW_FUND_SEQUENCE_PHASE_Y	Sample set contain Phase Y filtered voltage
EM_SW_FUND_SEQUENCE_PHASE_B	Sample set contain Phase B filtered voltage
EM_SW_FUND_SEQUENCE_PHASE_ALL	Sample set contain all phase filtered voltage

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_GetEnergyAccumulationMode	Get energy counter accumulation mode
	EM_SetEnergyAccumulationMode	Set energy counter accumulation mode
	EM_SetEnergyAccumulationPower	Set energy accumulation power to metrology
	EM_GetOperationData	Get Metrology internal data
	EM_SetOperationData	Set Metrology internal data
Calibration	EM_GetCalibInfo	Get the current calibration page on library
	EM_SetCalibInfo	Set calibration information of the library by calibration page
Sample Accumulation	EM_AccMode_Run	Run sample accumulation
	EM_AccMode_CheckStatus	Check status of sample accumulation
	EM_AccMode_GetAccumulator	Get accumulator buffer
	EM_ADC_AccMode_IntervalProcessing	Callback function for sample accumulation
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_GetFundamentalVoltageRMS	Read the Fundamental True RMS voltage (Volt)
	EM_GetFundamentalCurrentRMS	Read the Fundamental 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_GetPhaseAngleRtoY	Get voltage phase angle from R to Y
	EM_GetPhaseAngleYtoB	Get voltage phase angle from Y to B
	EM_GetPhaseAngleBtoR	Get voltage phase angle from B to R
	EM_GetCurrentPhaseAngle	Get current phase angle from R to Y, Y to B, B to R or either R,Y,B to neutral
	EM_GetVoltageTHD	Get total harmonic distortion for voltage
	EM_GetCurrentTHD	Get total harmonic distortion for current
	EM_GetActivePowerTHD	Get total harmonic distortion for active power

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_GainPhaseReset	Reset phase or neutral gain to lowest
	EM_ADC_GainPhaseIncrease	Increase phase or neutral gain 1 level
	EM_ADC_GainPhaseDecrease	Decrease phase or gain 1 level
	EM_ADC_GainPhaseGetLevel	Get current phase or neutral gain level
	EM_ADC_SetPhaseCorrection	Set the phase or neutral angle of Voltage and Current channels
	EM_ADC_SAMP_UpdateStep	Adjust hardware and software for v90 samples
	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
Phase_R Correction Angle	-2.153270 degree (negative value means I lead V)
Phase_Y Correction Angle	-2.150831 degree (negative value means I lead V)
Phase_B Correction Angle	-2.088668 degree (negative value means I lead V)
Meter Constant	3200 imp/KWh
Energy to pulse ratio	1
Pulse On Time	10 ms
Phase R V-coefficient	37.22082
Phase R I-coefficient	79681.9298
Phase R Power coefficient (active, reactive, apparent)	(V-coefficient * I-coefficient)
Phase Y V-coefficient	37.19622
Phase Y I-coefficient	83655.52
Phase Y Power coefficient (active, reactive, apparent)	(V-coefficient * I-coefficient)
Phase B V-coefficient	37.20874
Phase B I-coefficient	84222.86
Phase B Power coefficient (active, reactive, apparent)	(V-coefficient * I-coefficient)
Neutral I-coefficient	83878.65

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 PhaseR Correction Angle List */
const float32_t FAR_PTR   g_EM_DefaultCalibPhaseAngleList_Phase_R[] =
{
    -2.115543f,    /* PhaseR Gain Level 0 Phase Shift Angle (in degree) */
    -2.104473f,    /* PhaseR Gain Level 1 Phase Shift Angle (in degree) */
};

/* SW PhaseY Correction Angle List */
const float32_t FAR_PTR   g_EM_DefaultCalibPhaseAngleList_Phase_Y[] =
{
    -2.048136f,    /* PhaseY Gain Level 0 Phase Shift Angle (in degree) */
    -2.037581f,    /* PhaseY Gain Level 1 Phase Shift Angle (in degree) */
```

```
 };

 /* SW PhaseB Correction Angle List */
 const float32_t FAR_PTR   g_EM_DefaultCalibPhaseAngleList_Phase_B[] =
 {
     -2.048136f,    /* PhaseY Gain Level 0 Phase Shift Angle (in degree) */
     -2.037581f,    /* PhaseY Gain Level 1 Phase Shift Angle (in degree) */
 };

 /* SW Gain Value List (Phase Channel) */
const float32_t FAR_PTR    g_EM_DefaultCalibPhaseGainValueList_Phase_R[] =
{
   1.0f,        /* Phase Gain Level 0 Value (lowest, value is 1.0, fixed)*/
   4.005f,      /* Phase Gain Level 1 Value    |                       */
 };

const float32_t FAR_PTR    g_EM_DefaultCalibPhaseGainValueList_Phase_Y[] =
{
   1.0f,        /* Phase Gain Level 0 Value (lowest, value is 1.0, fixed)*/
   3.997f,      /* Phase Gain Level 1 Value    |                       */
 };

const float32_t FAR_PTR    g_EM_DefaultCalibPhaseGainValueList_Phase_B[] =
{
   1.0f,        /* Phase Gain Level 0 Value (lowest, value is 1.0, fixed)*/
   4.003f,      /* Phase Gain Level 1 Value    |                       */
 };


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

         {
             37.22082,
             79681.9298,
              37.22082 * 79681.9298,
              37.22082 * 79681.9298,
              37.22082 * 79681.9298,

          },
         {
             37.19622,
             83655.52,
              37.19622 * 83655.52,
              37.19622 * 83655.52,
              37.19622 * 83655.52,

          },
         {
             37.20874,
             84222.86,
              37.20874 * 84222.86,
              37.20874 * 84222.86,
              37.20874 * 84222.86,

          },
         {
             83878.65,
          },
```

```
{
      {
         (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList_Phase_R     ,
      },
      {
         (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList_Phase_Y     ,
      },
      {
         (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList_Phase_B     ,
      },
    },

   {
      {
         (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList_Phase_R     ,
      },
      {
         (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList_Phase_Y     ,
      },
      {
         (float32_t FAR_PTR *)g_EM_DefaultCalibPhaseGainValueList_Phase_B     ,
      },
   }
};

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

    },

    /* Operation */
    {
        0.01,
        0.01 * 180.0,
        10.0,
        40.0,
        70.0,
        3200,
        10
        1,
        1,
        1,
        0,
    },
```

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

    /* Sag and Swell */
    {
        145,
        127,
        260,
        242,
        3,
        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	Phase_R VRMS coeff < EM_VRMS_COEFF_MIN OR Phase_R IRMS coeff < EM_IRMS_COEFF_MIN OR Phase_R Active power coeff < EM_ACT_POWER_COEFF_MIN OR Phase_R Reactive power coeff < EM_REA_POWER_COEFF_MIN OR Phase_R Apparent power coeff < EM_APP_POWER_COEFF_MIN OR Phase_R i_phase_degrees == NULL OR Phase_R i_gain_values == NULL
EM_ERROR_CALIB_PARAMS_LINE2	Similar check of Phase_R for Phase_Y
EM_ERROR_CALIB_PARAMS_LINE3	Similar check of Phase_R for Phase_B
EM_ERROR_CALIB_PARAMS_NEUTRAL	Neutral IRMS coeff < EM_IRMS_COEFF_MIN

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 attributes. Do not cast it into near attribute.

Sample Usage

Below is an example that has implemented a function to change Phase_R V-coeff = 3900.0, I-coeff = 4200.0 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 Phase_R V-coeff, I-coeff */
       calib.coeff.phase_r.vrms = 3900.0f;
       calib.coeff.phase_r.irms = 4200.0f;
       calib.coeff.active_power =
       calib.coeff.reactive_power =
       calib.coeff.apparent_power = (calib.coeff.phase_r.vrms *
 calib.coeff.phase_r.irms);

       /* 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_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: EM stop updating power value for energy accumulation. Users call EM_SetEnergyAccumulationPower to update.

1: EM uses sum of phase active, reactive, apparent power for energy accumulation.

2: EM uses the sum of absolute phase active power and sum of phase reactive, apparent power for energy accumulation.

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();
```

3.1.9 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: 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 use phase channel absolute power for energy accumulation (forward only active energy).

Return value

None

Restriction/Caution

If the input value larger than 2, 2 will be set

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(2);
```

3.1.10 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 signs 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)
}
```

3.1.11 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);
```

3.1.12 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 Sample Accumulation

3.2.1 EM_AccMode_Run

Prototype

```
uint8_t EM_AccMode_Run(uint16_t number_of_cycles);
```

Explanation

Trigger and run sample accumulation with specified number of cycles.

Clear number of accumulated samples and cycles to 0

Clear accumulator acc0, acc1, acc2 to 0

Set metrology sample accumulation mode to 1

Argument(s)

Name	Data type	I/O	Description
number_of_cycles	uint16_t	I	Requested number of cycles for accumulation. Need to take care number of sample overflow (e.g. around 78 samples/cycle at 50Hz, 3906Hz sampling rate)

Return value

Execution status

Macro Value Name	Explanation
EM_ERROR_NOT_RUNNING	Metrology not started
EM_ERROR_PARAMS	number_of_cycles is 0
EM_ACCUMULATING	Successful triggered, metrology sample accumulation mode 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 */

uint8_t status;

status = EM_AccMode_Run (7800);
```

3.2.2 EM_AccMode_CheckStatus

Prototype

```
uint8_t EM_AccMode_CheckStatus (uint16_t * p_sample_count, uint16_t *
p_cycle_count);
```

Explanation

Check metrology sample accumulation status with additional output information.

The current number of accumulated samples and cycles will be returned through input pointers. If NULL specified, function do not set to address in pointer.

Argument(s)

Name	Data type	I/O	Description
p_sample_count	uint16_t	O	Returned current number of samples accumulated
p_cycle_count	uint16_t	O	Returned current number of cycles accumulated

Return value

Execution status

Macro Value Name	Explanation
EM_ERROR_NOT_RUNNING	Sample accumulation mode is not running or already finished running
EM_ACCUMULATING	Sample accumulation mode 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 */

uint8_t status;
uint16_t sample_count;
uint16_t cycle_count;

status = EM_AccMode_CheckStatus (&sample_count, &cycle_count);
```

3.2.3 EM_AccMode_GetAccumulator

Prototype

```
void EM_AccMode_GetAccumulator (EM_ACCMODE_ACCUMULATOR * p_accumulator);
```

Explanation

Get the sample accumulator in float representation (conversion from int64_t to float32_t). If NULL specified, function do not set to address in pointer.

Argument(s)

Name	Data type	I/O	Description
p_accumulator	EM_ACCMODE_ACCUMULATOR	O	Returned accumulated buffer

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_ACCMODE_ACCUMULATOR acc;

EM_AccMode_GetAccumulator (&acc);
```

3.2.4 EM_ADC_AccMode_IntervalProcessing

Prototype

```
void EM_ADC_AccMode_IntervalProcessing(EM_ACCMODE_SAMPLES * p_samples);
```

Explanation

DSAD interrupt handler for sample accumulation Accumulate input signal to accumulator

acc0 = acc0 + ((int64_t)signal0 * signal1)

acc1 = acc1 + ((int64_t)signal2 * signal3)

acc2 = acc2 + ((int64_t)signal4 * signal5)

Count the number of sample and number of cycles based on zero cross.

When number of cycle accumulated reach set number of cycle in function EM_AccMode_run, the function clear metrology sample accumulation mode flag (metrology continue normal accumulation)

Argument(s)

Name	Data type	I/O	Description
p_samples	EM_ACCMODE_SAMPLES	O	Sample buffer

Return value

None

Restriction/Caution

Run this in DSAD interrupt handler to accumulate input signals

Sample Usage

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

EM_ACCMODE_SAMPLES samples;
samples.signal0 = 1;
samples.signal1 = 2;
samples.signal2 = 3;
samples.signal3 = 4;
samples.signal4 = 5;
samples.signal5 = 6;


EM_AccMode_IntervalProcessing (&samples);
```

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) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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 act;

/* Get Phase Active Power  */

act = EM_GetActivePower(LINE_PHASE_R);

act = EM_GetActivePower(LINE_PHASE_Y);

act = EM_GetActivePower(LINE_PHASE_B);

/* Get Total Active Power */

act = EM_GetActivePower(LINE_TOTAL);
```

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) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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_power;

/* Get Phase Fundamental Active Power */

fundamental_active_power = EM_GetFundamentalActivePower(LINE_PHASE_R);

fundamental_active_power = EM_GetFundamentalActivePower(LINE_PHASE_Y);

fundamental_active_power = EM_GetFundamentalActivePower(LINE_PHASE_B);

/* Get Total Fundamental Active Power */

fundamental_active_power = EM_GetFundamentalActivePower(LINE_TOTAL);
```

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 Active Power (VAr) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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 Phase Reactive Power */

reactive = EM_GetReactivePower(LINE_PHASE_R);

reactive = EM_GetReactivePower(LINE_PHASE_Y);

reactive = EM_GetReactivePower(LINE_PHASE_B);

/* Get Total Reactive Power */

reactive = EM_GetReactivePower(LINE_TOTAL);
```

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 Reactive Active Power (VA) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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 Phase Apparent Power */

apparent_power = EM_GetApparentPower(LINE_PHASE_R);

apparent_power = EM_GetApparentPower(LINE_PHASE_Y);

apparent_power = EM_GetApparentPower(LINE_PHASE_B);

/* Get Total Apparent Power */

apparent_power = EM_GetApparentPower(LINE_TOTAL);
```

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 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(EM_LINE line);
```

Explanation

Get the voltage RMS value (Volt)

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 True RMS Voltage (Volt) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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 Phase Voltage RMS value */

voltageRMS = EM_GetVoltageRMS(LINE_PHASE_R);

voltageRMS = EM_GetVoltageRMS(LINE_PHASE_Y);

voltageRMS = EM_GetVoltageRMS(LINE_PHASE_B);

/* Get Total Voltage RMS value */

voltageRMS = EM_GetVoltageRMS(LINE_TOTAL);
```

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) for each phase, neutral and total.

LINE_TOTAL return summing of all phase value

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;

/* Get Phase Current RMS */

currentRMS = EM_GetCurrentRMS(LINE_PHASE_R);

currentRMS = EM_GetCurrentRMS(LINE_PHASE_Y);

currentRMS = EM_GetCurrentRMS(LINE_PHASE_B);

/* Get Neutral Current RMS */

currentRMS = EM_GetCurrentRMS(LINE_NEUTRAL);

/* Get Total Current RMS */

currentRMS = EM_GetCurrentRMS(LINE_TOTAL);
```

3.3.14 EM_GetFundamentalVoltageRMS

Prototype

```
float32_t EM_GetFundamentalVoltageRMS (EM_LINE line);
```

Explanation

Get the fundamental voltage RMS value (Volt)

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, of fundamental True RMS Voltage (Volt) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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_vrms;

/* Get Phase Fundamental VRMS */

fundamental_vrms = EM_GetFundamentalVoltageRMS(LINE_PHASE_R);

fundamental_vrms = EM_GetFundamentalVoltageRMS(LINE_PHASE_Y);

fundamental_vrms = EM_GetFundamentalVoltageRMS(LINE_PHASE_B);

/* Get Total Fundamental VRMS */

fundamental_vrms = EM_GetFundamentalVoltageRMS(LINE_TOTAL);
```

3.3.15 EM_GetFundamentalCurrentRMS

Prototype

```
float32_t EM_GetFundamentalCurrentRMS(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) for each phase and total

LINE_NEUTRAL will always return 0

LINE_TOTAL return summing of all phase value

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;

/* Get Phase Fundamental Current RMS */

currentRMS = EM_GetFundamentalCurrentRMS(LINE_PHASE_R);

currentRMS = EM_GetFundamentalCurrentRMS(LINE_PHASE_Y);

currentRMS = EM_GetFundamentalCurrentRMS(LINE_PHASE_B);

/* Get Total Fundamental Current RMS */

currentRMS = EM_GetFundamentalCurrentRMS(LINE_TOTAL);
```

3.3.16 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 for each phase and total.

LINE_NEUTRAL will always return 1.0

LINE_TOTAL return power factor results from total active power and total apparent power

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 Phase Power Factor */

powerfactor = EM_GetPowerFactor(LINE_PHASE_R);

powerfactor = EM_GetPowerFactor(LINE_PHASE_Y);

powerfactor = EM_GetPowerFactor(LINE_PHASE_B);

/* Get Total Power Factor */

powerfactor = EM_GetPowerFactor(LINE_TOTAL);
```

3.3.17 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, indicating Lead, Lag status of phase and total
LINE_NEUTRAL will always return PF_SIGN_UNITY

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 Phase Power Factor sign */
sign = EM_GetPowerFactorSign (LINE_PHASE_R);
sign = EM_GetPowerFactorSign (LINE_PHASE_Y);

sign = EM_GetPowerFactorSign (LINE_PHASE_B);

/* Get Total Power Factor sign */
sign = EM_GetPowerFactorSign (LINE_TOTAL);
```

3.3.18 EM_GetLineFrequency

Prototype

```
float32_t EM_GetLineFrequency(EM_LINE line);
```

Explanation

Get the Line Frequency (Hz).

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, present for Line Frequency (Hz) for each phase and total.

LINE_NEUTRAL will return line frequency on current channel.

LINE_TOTAL return max value of frequency between phase frequency

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 Phase Line Frequency */
freq = EM_GetLineFrequency(LINE_PHASE_R);

freq = EM_GetLineFrequency(LINE_PHASE_Y);

freq = EM_GetLineFrequency(LINE_PHASE_B);

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

3.3.19 EM_GetPhaseAngleRtoY

Prototype

```
float32_t EM_GetPhaseAngleRtoY(void);
```

Explanation

Get voltage phase angle from R to Y

Argument(s)

None

Return value

Angle in degree

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 phase_angle;

/* Get phase angle */
phase_angle = EM_GetPhaseAngleRtoY();
```

3.3.20 EM_GetPhaseAngleYtoB

Prototype

```
float32_t EM_GetPhaseAngleYtoB(void);
```

Explanation

Get voltage phase angle from Y to B

Argument(s)

None

Return value

Angle in degree

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 phase_angle;

/* Get phase angle */
phase_angle = EM_GetPhaseAngleYtoB();
```

3.3.21 EM_GetPhaseAngleBtoR

Prototype

```
float32_t EM_GetPhaseAngleBtoR(void);
```

Explanation

Get voltage phase angle from B to R

Argument(s)

None

Return value

Angle in degree

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 phase_angle;

/* Get phase angle */
phase_angle = EM_GetPhaseAngleBtoR();
```

3.3.22 EM_GetCurrentPhaseAngle

Prototype

```
float32_t EM_GetCurrentPhaseAngle(EM_LINE base_line, EM_LINE relative_line);
```

Explanation

Get current phase angle from base_line to relative_line

Argument(s)

Name	Data type	I/O	Description
base_line	EM_LINE	I	Base line of angle measurement. Refer to EM_LINE for more details & usage of this structure type.
relative_line	EM_LINE	I	Relative line of angle measurement. Refer to EM_LINE for more details & usage of this structure type.

Return value

Angle in degree

Some possible combination between base_line to relative_line

- RtoY, YtoB, BtoR
- RtoN, YtoN, BtoN
- The reset will return zero

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 phase_angle;

/* Get phase angle */
phase_angle = EM_GetCurrentPhaseAngle(LINE_PHASE_R, LINE_PHASE_Y);

phase_angle = EM_GetCurrentPhaseAngle(LINE_PHASE_Y, LINE_PHASE_B);

phase_angle = EM_GetCurrentPhaseAngle(LINE_PHASE_B, LINE_PHASE_R);

phase_angle = EM_GetCurrentPhaseAngle(LINE_PHASE_R, LINE_NEUTRAL);

phase_angle = EM_GetCurrentPhaseAngle(LINE_PHASE_Y, LINE_NEUTRAL);

phase_angle = EM_GetCurrentPhaseAngle(LINE_PHASE_B, LINE_NEUTRAL);
```

3.3.23 EM_GetVoltageTHD

Prototype

```
float32_t EM_GetVoltageTHD(EM_LINE line);
```

Explanation

Get the total harmonic distortion for voltage RMS

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 THD True RMS Voltage (ratio) for each phase and total

LINE_NEUTRAL is not supported

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 thd;

/* Get Phase Voltage THD */

thd = EM_GetVoltageTHD(LINE_PHASE_R);

thd = EM_GetVoltageTHD(LINE_PHASE_Y);

thd = EM_GetVoltageTHD(LINE_PHASE_B);

/* Get Total Voltage THD */

thd = EM_GetVoltageTHD(LINE_TOTAL);
```

3.3.24 EM_GetCurrentTHD

Prototype

```
float32_t EM_GetCurrentTHD(EM_LINE line);
```

Explanation

Get the total harmonic distortion for current RMS

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 THD True RMS Current (ratio) for each phase and total

LINE_NEUTRAL is not supported

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 thd;

/* Get Phase Current THD */

thd = EM_GetCurrentTHD(LINE_PHASE_R);

thd = EM_GetCurrentTHD(LINE_PHASE_Y);

thd = EM_GetCurrentTHD(LINE_PHASE_B);

/* Get Total Current THD */

thd = EM_GetCurrentTHD(LINE_TOTAL);
```

3.3.25 EM_GetActivePowerTHD

Prototype

```
float32_t EM_GetActivePowerTHD(EM_LINE line);
```

Explanation

Get the total harmonic distortion for active power.

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 THD Active Power (ratio) for each phase and total

LINE_NEUTRAL is not supported.

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 thd;

/* Get Phase Active Power THD */

thd = EM_GetActivePowerTHD(LINE_PHASE_R);

thd = EM_GetActivePowerTHD(LINE_PHASE_Y);

thd = EM_GetActivePowerTHD(LINE_PHASE_B);

/* Get Total Active Power THD */

thd = EM_GetActivePowerTHD(LINE_TOTAL);
```

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_GainPhaseReset

Prototype

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

4.1.5 EM_ADC_GainPhaseIncrease

Prototype

```
void EM_ADC_GainPhaseIncrease(EM_LINE line);
```

Explanation

Increase gain of Phase channel one level. Example, level0 → level1

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

Sample Implementation

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

4.1.6 EM_ADC_GainPhaseDecrease

Prototype

```
void EM_ADC_GainPhaseDecrease(EM_LINE line);
```

Explanation

Decrease gain of Phase channel one level. Example, level1 → level0

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

Sample Implementation

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

4.1.7 EM_ADC_GainPhaseGetLevel

Prototype

```
uint8_t EM_ADC_GainPhaseGetLevel(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 needs to be implemented on Wide Range library versions (WSUR, WQUR, WQFR).

Sample Implementation

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

4.1.8 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 errors 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
      {

      }
}
```

4.1.9 EM_ADC_SAMP_UpdateStep

Prototype

```
void EM_ADC_SAMP_UpdateStep(EM_LINE line, float32_t fac);
```

Explanation

Adjust hardware + software for v90 sampling
Fine tune according to measured line frequency from metrology

Argument(s)

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

Name	Data type	I/O	Description
fac	float32_t	I	Measured line frequency

Return value

None

Restriction/Caution

None

Sample Implementation

```
void EM_ADC_SAMP_UpdateStep (EM_LINE line, float32_t fac)
{

      /* Calculate software delay and TAU01 value for v90 based on fac */
}
```

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 events and update the 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 supports 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	Improve document explanation and format
5.02	Jun 15th, 2023	All	Update metrology code size, CPU load and fundamental power calculation
5.03	Nov 1st, 2023	All	Added Timing Diagram and changed FPGA to PGA Gain

RL78/I1C Group User’s Manual: Software

Publication Date: Rev5.03 November 1, 2023

Published by: Renesas Electronics Corporation

RL78/I1C Group

Renesas Electronics Corporation