UM1948
User manual

EVALSTPM3x-3PH evaluation board firmware

Introduction

The document describes the software implementation of the EVALSTPM3X-3PH power meter based on the STM8S903K3 microcontroller and the STPM33 and the STPM34 metrology devices, with a firmware within 8 kilobytes.

Besides 3-phase power meter can also be used as a reference for 1-phase power meter.

The evaluation board can be supposed as a whole “chip” (virtual 3-phase IC) application, to be connected with an STM32 family microcontroller by SPI or UART.

Setting the STM8S903K3 microcontroller dummy, the STPM34 or the STPM33 can be controlled by the external microcontroller independently via SPI interface. In this case, the firmware in the STM8S903K3 can be ported into external microcontroller such as an STM32 family device.

This manual includes the software structure introduction, the used microcontroller resources, the main function interfaces and the program flowchart.

Regarding to the STM8S903K3, refer to datasheet "STM8S903K3 STM8S903F3" and RM0016 reference manual for more details.

Contents

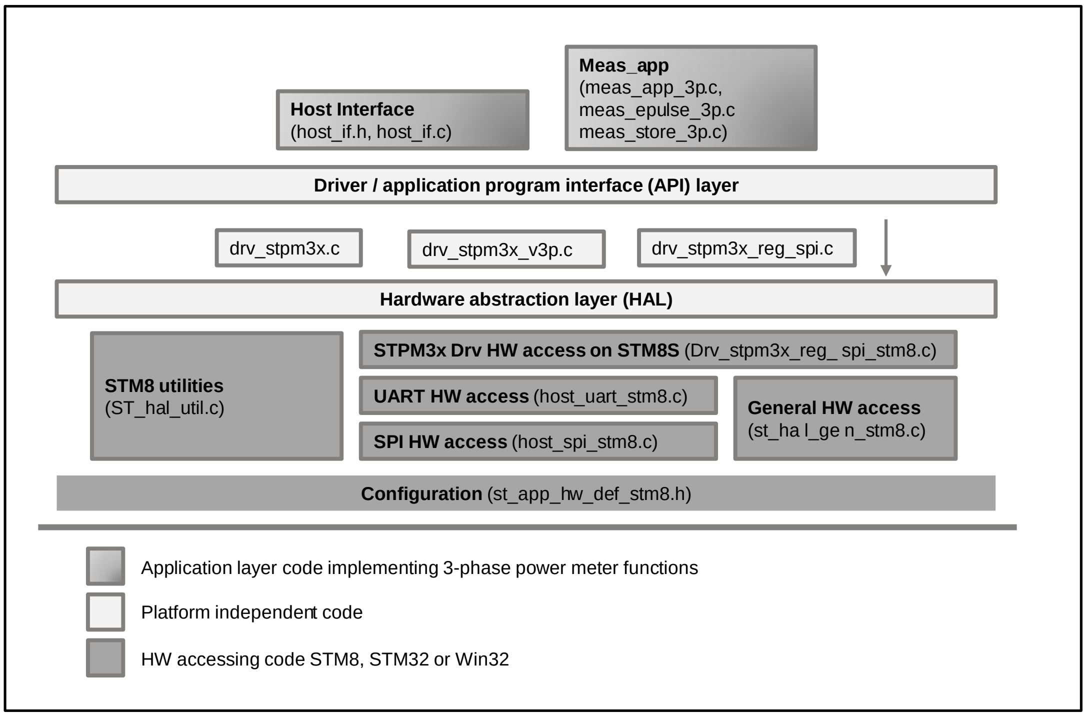
1 Overview

1.1 System architecture

The software is designed by following a platform independent approach. The driver and the application layer can be ported to the other part number in the STM8, STM32 or even Win32 platform.

The below figure shows the system architecture. It could be divided into 3 layers.

Figure 1: Project organization


Meas_app
(meas_app_3p.c,
meas_epulse_3p.c
meas_store_3p.c)
Host Interface
(host_if.h, host_if.c)
Driver / application program interface (API) layer
drv_stpm3x.c
drv_stpm3x_v3p.c
drv_stpm3x_reg_spi.c
Hardware abstraction layer (HAL)
STM8 utilities
(ST_hal_util.c)
STPM3x Drv HW access on STM8S (Drv_stpm3x_reg_ spi_stm8.c)
UART HW access (host_uart_stm8.c)
SPI HW access (host_spi_stm8.c)
General HW access
(st_ha l_ge n_stm8.c)
Configuration (st_app_hw_def_stm8.h)
Application layer code implementing 3-phase power meter functions
Platform independent code
HW accessing code STM8, STM32 or Win32

The entities below the hardware abstraction layer (HAL) belong to the hardware (HW) access layer.

It implements the clock initialization, UART initialization, GPIOs initialization, HW connection initialization between either the STPM34 or the STPM33 and the microcontroller, software timer implementation, interrupt setting and so on.

The whole system can work on many target controllers such as the STM32, STM8 or even Win32, if corresponding HW access layer is implemented. In the STM8 implementation, all the HW information is defined in st_app_hw_def_stm8.h of each project directory. User could simply modify this file and related source files to port software on different STM8 and even STM32 microcontrollers.

The 2nd layer, hardware independent, is the driver layer/application program interface layer. This layer supplies all interface functions between the STPM3x and the microcontroller (STM8), and some special function routines, which implement the controlling operation to the STPM3x. Above this layer, the application layer develops all functions by just calling API layer functions, without need to focus on the STPM3x details and the bottom HW implementation.

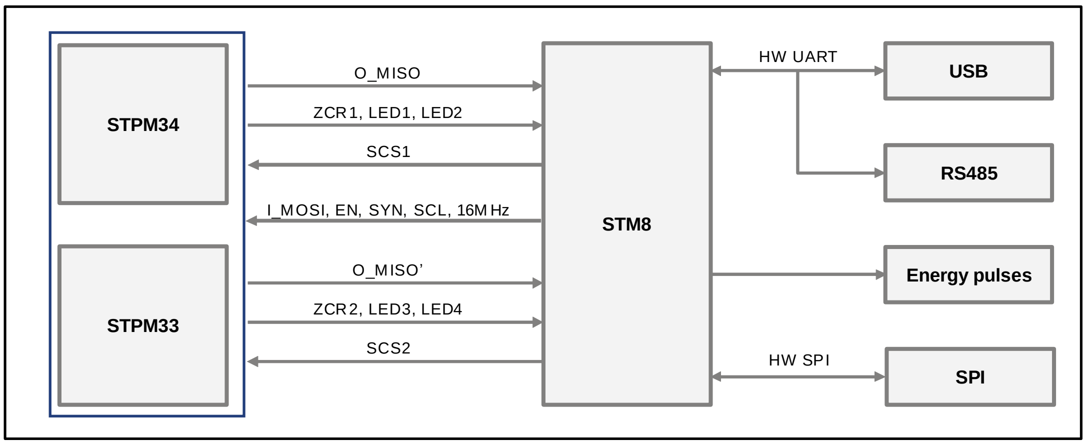
The application layer is the final goal of the entire application design. In this software, it includes the energy pulse generation, energy data storing in EEPROM, energy related data collecting and implementation of UART commands from the EVALSTPM3x-3PH GUI, which is human-machine interface. Among the energy related data, energy values only are read from the STPM3x registers (by API layer), power values are calculated by the STM8 (not by API layer). Further necessary and important information is obtained on GUI by accessing the STPM3x registers (by API layer) via UART commands. At the same time, calibration information is also transferred by GUI to microcontroller, then to STMP3x registers (by API layer).

1.2 Used resources

1.2.1 Hardware connections

The block diagram of the entire application is shown below:

Figure 2: Application block diagram



The below table describes the pin assignment of the STM8 on HW PCB V1.1.

Table 1: STM8S903K3 pin assignment

Pin	Net name	Description	Remark
1	NRST	Reset pin	
2	OSCIN	Resonator/ crystal in	
3	OSCOUT	Resonator/ crystal out	
4	VSS	GND	
5	VCAP	1.8 V regulator capacitor	
6	VDD	Digital power supply	
21	PC4	I_CLK_MTR	Supply 16 MHz to the STPM34 and the STPM33
8	PF4	O_Q_LED	
9	PB7	O_P_LED	
10	PB6	I_SCS1	Pin selection for the STPM34
29	PD4	I_SCS2	Pin selection for the STPM33
11	PB5	I_SYN	

Pin	Net name	Description	Remark
13	PB3	I_EN	
14	PB2	O_MISO	Software SPI, MISO is just for the STPM34
15	PB1	I_MOSI	
16	PB0	I_SCL	
25	PD0	O_MISO'	MISO is just for the STPM33
19	PC2	O_ZCR1	Timer1_channel2
20	PC3	O_ZCR2	Timer1_channel3
32	PD7	O_RTS_STM8	
27	PD2	I_CTS_STM8	
30	PD5	O_TX_STM8	UART interface
31	PD6	I_RX_STM8	
22	PC5	I_SCK_STM8	Hardware SPI (PC5,PC6,PC7)
23	PC6	I_MOSI_STM8	
24	PC7	O_MISO_STM8	
18	PC1	O_LED1	STPM34 LED pulse output
7	PA3	O_LED2	
17	PE5	O_LED3	STPM33 LED pulse output
28	PD3	O_LED4	
12	PB4	O_INTCB	
26	PD1	IO_SWIM	

1.2.2 Software configuration

The latest HW PCB version is V1.1, for which “HW_STPM3X_3P_V2” macro must be defined in file st_app_hw_def.h. To be flexible and save memory, many macro definitions could be found in file stpm3x_app_config.h. Using IAR EWSTM8 1.41.1 compiler and the below configuration, at least 7862 bytes Flash and at least 570 bytes RAM are required under none optimization condition.

Table 2: Macro definition

Macro	Configuration	Remark
STPM3x_CFG_USE_NAMED_REG	Off	Used as a convenient access (registers) method, mask the detailed info of registers. Refer to and to *Section 2.1.10: "TBool stpm3x_RegWrite(TU8 nReg, TU32 nVal)"*
STPM3x_CFG_USE_V3P_REG	Off	Used as a convenient access (registers) method, in virtual 3-phase measurement chip. The STPM34 and STPM33 are in a unique virtual address, not split chips
STPM3x_CFG_USE_SW_EPULSE	On	Used to select different pulse output implementation method. Refer to *Section 3.1: "Pulse output generation"*

Macro	Configuration	Remark
STPM3x_CFG_USE_HOST_UART	On	Used to select different interfaces to GUI. In the software, UART is selected
STPM3x_CFG_USE_HOST_SPI	Off	Used to select different interfaces to GUI. If on, select SPI interface
STPM3x_CFG_USE_PACK_INIT	On	Used to select different initialization methods. If on, all registers are initialized by 32-bit register writing method once; If off, all registers are initialized by register bit writing method one by one
STPM3x_CFG_CALC_ACT_POWER	On	Used for power (active, reactive) calculations. It is always on
STPM3x_CFG_POLY_PHASE	On	Used to select different application purposes. If on, 3-phase power meter is available; if off, 1-phase power meter is available
STPM3x_CFG_ENERGY_ACCU	On	Used for energy accumulation calculations. It is always on

1.2.3 MCU peripheral use

All used peripherals on the STM8S903K3 are described in the below table.

Table 3: Application peripheral usage

Peripherals	Application	Interrupt function	Interrupt priority
TIMER 1 (16-bit)	STPM34 ZCR output of voltage channel 1 or voltage channel 2	TIM1_CH2 input capture interrupt	2
	STPM33 ZCR output of voltage channel	TIM1_CH3 input capture interrupt	2
TIMER 5 (16-bit)	When STPM3x_CFG_USE_SW_EPULSE is configured as on, the timer generates 64 ms time base to update energy and power	Update interrupt	2
TIMER 6 (8-bit)	0.25 ms time base: • Multiplied by 4 to get 1 ms as system tick • When STPM3x_CFG_USE_SW_EPULSE is configured as on, it accumulates energy fragment by 0.25 ms time slot to decide if the generated energy pulses • When STPM3x_CFG_USE_SW_EPULSE is configured as off, it detects 80 ms time out for pulse width control	Update interrupt	3 (STPM3x_CFG_USE_SW_EPULSE on); 2 (STPM3x_CFG_USE_SW_EPULSE off)

Peripherals	Application	Interrupt function	Interrupt priority
EXTI	PC1 rising edge to indicate P pulse from the STPM34 (used when STPM3x_CFG_USE_SW_EPULSE is configured as off)	EXTI	3
	PA3 rising edge to indicate Q pulse from the STPM34 (used when STPM3x_CFG_USE_SW_EPULSE is configured as off)	EXTI	3
	PE5 rising edge to indicate P pulse from the STPM33 (used when STPM3x_CFG_USE_SW_EPULSE is configured as off)	EXTI	3
	PD3 rising edge is to indicate Q pulse from the STPM33 (used when STPM3x_CFG_USE_SW_EPULSE is configured as off)	EXTI	3
	PB4 falling edge to indicate the STPM3x event (the STM8S should read related registers to inquire which event in which chip)	EXTI	2
UART	UART interface to GUI	UART Rx interrupt	1 (the lowest priority)
SPI	SPI interface to GUI	SPI Rx and Tx interrupt	1 (the lowest priority)
EEPROM	Stores energy data to EEPROM inside the STM8	No	
GPIO	• GPIO simulation of SPI to communicate with the STPM3x • CCO to provide 16 MHz to the STPM3x • GPIO level polling to detect CTS from GUI • GPIO level control to generate P and Q energy pulses respectively	No	

1.2.4 Firmware layer description

The below table lists the main files of different firmware layers.

Table 4: Main files

Layer	File name	Brief
Hardware abstraction layer	Host_spi_stm8.c	STM8 SPI based communication with GUI
	Host_uart_stm8.c	STM8 UART based communication with GUI
	St_hal_gen_stm8.c	HAL implementation on the STM8
	Drv_stpm3x_reg_spi_stm8.c	STPM3x SPI initialization for the STM8
	St_hal_util.c	Utility function implementation to software timer and other utilities
Driver layer	Drv_stpm3x.c	Definition of the STPM3x driver
	Drv_stpm3x_reg_spi.c	STPM3x register SPI access implementation
	Drv_stpm3x_v3p.c	Virtual 3-ph metering IC based on the STPM3x
Application layer	Host_if.c	HOST communication demo with private protocol
	Meas_app_3p.c	3-ph meter measurement based on the STPM3x
	Meas_epulse_3p.c	3-ph energy pulse generation operations based on the STPM3x
	Meas_store_3p.c	3-ph meter storage definition for the STPM3x base meter
Interrupt	Stm8s_it.c	Main interrupt service routines
.h files	Stpm3x_app_config.h	Configuration file of the STPM3x application
	St_app_hw_def.h	Hardware definition on the STM8
	Drv_stpm3x.h	Definition of the STPM3x driver
	Drv_stpm3x_reg.h	Definition of the STPM3x register operations
	Drv_stpm3x_v3p.h	Virtual 3-ph metering IC based on the STPM3x

Stpm3x_app_config.h is the basic system function configuration file. Different options may change the software implementation methods, or select different applications, or mask some unimportant functions to save memory according to applications. Please refer to *Table 2: "Macro definition"* for detailed description.

St_app_hw_def.h is the hardware definition file related to connection. It is necessary to change it accordingly if some pin connection has changed (and possibly, it is also necessary to change some routines in Stm8s_it.c).

Drv_stpm3x.h is the STPM3x driver prototype definition file. In the file, all interface functions between the STPM3x and MCU are declared. Drv_stpm3x_v3p.h is the driver prototype definition file (based on Drv_stpm3x.h) when all the STPM3x chips are looked as a whole, that is, a virtual 3-ph metering chip.

2 Main interfaces

2.1 Interfaces of the STPM3x driver

The interface to application layer is mainly driver layer since it involves the communication between the STPM3x and microcontroller. It can also be used by other applications based on different platforms. Besides, some initialization functions related to platform are also introduced here because of their importance.

2.1.1 void stpm3x_HwInit(TBool bUseSPI)

Description: hardware initialization of pins connected to the STPM3x and setting of the STPM3x communication interface.

Parameters:

- bUseSPI
  - True, sets SPI interface
  - False, sets UART interface

Return value: none.

Remarks: the timing sequence is strict, so control if different system clocks or platforms are used.

2.1.2 TBool stpm3x_DataIO(TU16 nRdWtAddr,TU8*pTxDat,TBool bUseCRC, TU8 *pRxDat)

Description: GPIO simulation of SPI communication (4-line) with the STPM3x. If CRC is used, 5-byte frame is sent and 5-byte frame is received.

Parameters:

- nRdWrAddr: register address for read or write
  - High byte stores write address
  - Low byte stores read address
  - When read, it sets high byte to 0xFF
  - When write, it sets low byte to 0xFF
- *pTxDat: stores the 5-byte frame to be sent
- *pRxDat: stores the 5-byte frame to be received
- bUseCRC
  - True, the frame is 5 bytes
  - False, the frame is 4 bytes

Return value: true, CRC of received frame is right; false, CRC of received frame is wrong.

Remarks: SCL simulation should be 50% duty.

2.1.3 void stpm3x_IO_EnableLED(TBool bEnable)

Description: Enable EXTI for LED pins. EXTI can only be enabled after the STPM3x register initialization.

Parameters:

- bEnable
  - True, enable EXTI
  - False, disable EXTI

Return value: none.

2.1.4 TBool stpm3x_ReadReg32(TU8 nAddr, TU8 *pnVal)

Description: Read 32-bit data of some registers.

Parameters:

- nAdd: register address (even number), listed in the register map of datasheet
- *pnVal: 32-bit data read from the register

Return value: true, success; false, fail.

2.1.5 TBool stpm3x_WriteReg32(TU8 nAddr, TU8 *pnVal)

Description: write 32-bit data to some registers.

Parameters:

- nAddr: register address (even number), listed in register map of datasheet
- *pnVal: 32-bit data to be written to the register

Return value: true, success; false, fail.

2.1.6 TBool stpm3x_ReadTarReg32(TU8 nTarRow, TU8 *pnVal)

Description: read 32-bit data of some registers from the STPM34 or STPM33.

Parameters:

- nTarRow: MSB indicates which chip to choose, other bits are register’s row address, listed in the register map of datasheet
- *pnVal: 32-bit data read from the register

Return value: true, success; false, fail.

2.1.7 TBool stpm3x_ReadRegBits(TU8 nAddr, TU8 nOffset, TU8 nBitNum, TU32 * pnVal)

Description: some continuous bits of the register 32-bit data are read.

Parameters:

- nAddr: register address (even number), listed in the register map of datasheet
- nOffset: offset of the 32-bit data, the value can be 0 to 31
- nBitNum: bit number beginning from the offset of 32-bit data, the value can be 1 to 32
- *pnVal: data read from the register with the offset and bit number setting

Return value: true-success; false-fail.

2.1.8 TBool stpm3x_WriteRegBits(TU8 nAddr, TU8 nOffset, TU8 nBitNum, TU32 nVal)

Description: some continuous bits are written to the register position beginning from offset.

Parameters:

- nAddr: register address (even number), listed in register map of datasheet
- nOffset: offset of the 32-bit data, the value can be 0 to 31
- nBitNum: bit number beginning from the offset of 32-bit data, the value can be 1 to 32
- nVal: data to be written to the register with the offset and bit number setting

Return value: true-success; false-fail.

2.1.9 TBool stpm3x_RegRead(TU8 nReg, TU32 *pnVal)

Description: some continuous bits of the register predefined by the array and index are read. The detailed information (register address, offset and bit number) is encapsulated inside the array.

Parameters:

- nReg: element index of the predefined array. The value is related to some enum definition, showing the register meaning
- *pnVal: data readout

Return value: true-success; false-fail.

2.1.10 TBool stpm3x_RegWrite(TU8 nReg, TU32 nVal)

Description: some continuous bits are written to the register predefined by the array and index. The detailed information (register address, offset and bit number) is encapsulated inside the array.

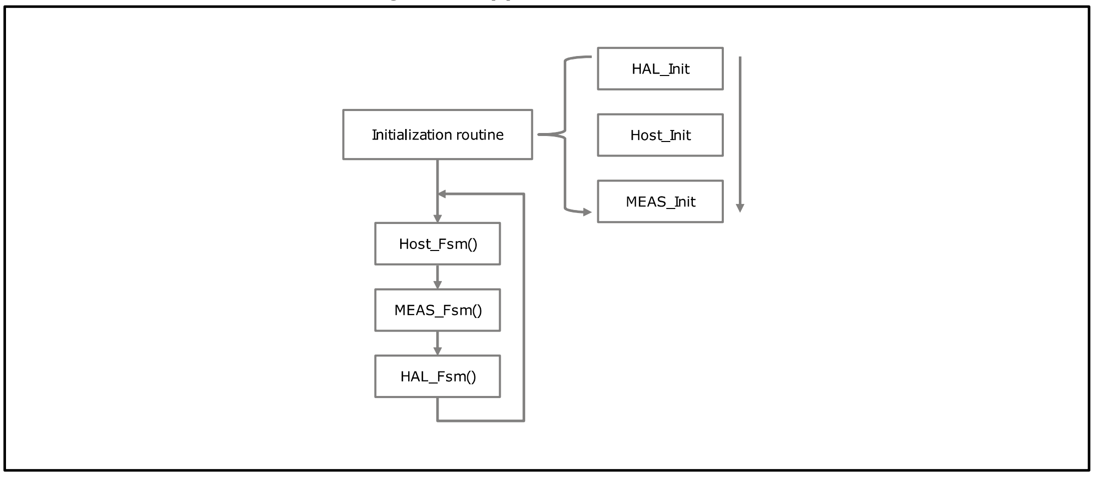
Parameters:

- nReg: element index of the predefined array. The value is related to some enum definition, showing the register meaning
- nVal: data to be written

Return value: true-success; false-fail.

2.2 Flowchart

Figure 3: Application flowchart



2.3 GUI communication command

GUI command frame format:

```
TX: { STX | Target(4b):Func(4b) | Addr | Val[2] (LSBF) | Addr2 | 00 | CS8 }
RX: { STX | Target(4b):Func(4b) | ErrCode | Val[4] (LSBF) | CS8 }
```

STX: 0x68

Target: 01/02/03/08, to the STPM34/STPM33/both/virtual 3-PH, @refer ESTPM3xTar

Func: 00/01/02/03/04, /Reset/Latch/Direct IO/Read/Write, @refer ESTPM3xFunc

Addr: 255 x 4B STPM3x RAM/virtual 3-PH RAM (needs to be even number to read)

Val[2]: the value to be written, LSBF

Addr2: only available in direct IO mode, when Addr is ReadAddr and Addr2 is WriteAddr

Val[4]: the value read, LSBF

CS8: 8-bit check sum of the frame

3 Software design calibration

3.1 Pulse output generation

Pulse output generation is a key point of the application, since accuracy tests are carried out using the pulses/kWh produced by the power meter.

There are two methods to generate the energy pulses.

- Software pulse method (STPM3x_CFG_USE_SW_EPULSE is configured as on)

The pulse is generated by adding up the energy (DeltaE) every time slot (0.25 ms) when the current time slot elapses one by one. Once the accumulated energy exceeds the threshold, which represents the energy of just one pulse, MCU outputs 80 ms-width pulse to light P LED or Q LED. Obviously, the shorter time slot, the more accurate the measurement results. However, a tradeoff between the time slot and MCU efficiency is necessary, since too short time slot leads to a heavy MCU workload.

- Hardware pulse method (STPM3x_CFG_USE_SW_EPULSE is configured as off)

The pulse is generated by external interrupts from 4 LED pins of the STPM34 and STPM33.

In the interrupt service routine, when the number of active energy LED pulses exceeds the threshold, MCU outputs 80 ms-width pulse to P LED (and the same for reactive energy to Q LED).

The number of pulses/kWh from all LED pin STPM3x chips is the threshold for the 3-ph power meter to output just one active energy pulse on LED P. This threshold depends on the application design and particularly from analog font end component choice. Microcontroller continuously counts interrupt occurrences to light P or Q LED at proper time.

Note:

1. The threshold is clearly different in two methods. In software pulse method, it indicates a fixed energy value. In hardware pulse method, it indicates a fixed number derived from pulse constant (imp/kwh). However, the threshold is both determined by hardware parameters (R_1, R_2, k_S) and software configuration (LEDPWM in DSP_CR1 and DSP_CR2), just different in unit (ws or impulse number).
2. To obtain better accuracy, the operation of ending P and Q pulse after 80 ms is also in 0.25 ms interrupt service routine, since both of edges need to be accurate.
3. As explained in the application peripheral usage table, the used MCU resources are different between the two methods. In the hardware pulse method, the pulse output depends on 4-pin interrupt number, so 4 EXTI are necessary. However, in the software method, the energy is calculated by adding up the “energy fragment” after each time slot, so the 4-pin interrupt is not needed. But a proper time slot (0.25 ms) is necessary; moreover, the “energy fragment” must be updated in a short interval if the current power changes fast. It is 64 ms (2^6) generated from hardware timer so that the interrupt and processing timing is in good condition and the power of two is convenient for multiplication and division on the STM8 platform. It is less than the shortest interval (about 124 ms) at maximum power load (3*100A, 110%Un).
4. Only 0.25 ms and 64 ms timings are coming from MCU timer directly, while all other timers (second, minute, hour, day, week, month and year) are software timers, derived indirectly from 0.25 ms.
5. In the above 2 methods, the accuracy influence factors are different:

- In the hardware pulse method, setting the highest EXTI priority, and the interval time (longer or shorter) of energy register reading is an important factor for the accuracy of energy displayed on GUI.
- In software pulse method, the fixed time slot interrupt and the time interval accuracy of energy register reading (update energy fragment) have the highest impact on accuracy.

3.2 No-load condition judgement

When STPM3x_CFG_CALC_ACT_POWER is configured as on, if active power is less than 2.2 W (PowerThreshold*Plsb) on A, B, C single phase or 3-ph totally, a no-load flag is set to 1.

When STPM3x_CFG_CALC_ACT_POWER is configured as off, no-load condition is determined through apparent power, Vrms*Irms (please note Vlsb*Ilsb=Plsb>>2.2 W).

3.3 STM8 limitation

Some application attention points are reported below:

1. The STM8 is MSBF, but the STPM3x is LSBF, by default, this has to be taken into account in data conversion between the STM8 and the STPM3x.
2. Considering that the STM8 is 8-bit microcontroller, single byte variable is proper in most cases.
3. The STM8 computation performance is poor, especially the float number multiplication and division. Only the power calculation is finished by STM8, which is well done by left and right shift according to proper time slot (2^N). Other calculations, for example, frequencies, phase angle, power factor are calculated on GUI.
4. In hardware pulse method, O_LED1, O_LED2, O_LED3 and O_LED4 are located in different ports (port A, port C, port D, port E respectively). This is necessary because pins of the same port share the same interrupt by logical OR, so in some cases it is impossible to distinguish them correctly, if they are very close to each other.

4 Accuracy results

With the above firmware implementation and after the device calibration, the following accuracy curves, for active power, have been taken on P LEDs.

Testing conditions are:

V_N = 230 V

I_N = 5 A

f_{line} = 50 Hz

CP = 400 pulses/kWh

In figures below, accuracy results are benchmarked with IEC 62053-22 normative requirement for class 0.2 meters.

Table 5: Active power error overcurrent

Applied current [A]	Full scale percentage [%]	Percentage error [%]		
		cosphi = 1	cosphi = 0.5 ind	cosphi = 0.5 cap
10	200	0,067	0,046	0,124
5	100	0,068	0,028	0,128
2.5	50	0,078	0,016	0,088
1	20	0,066	0,038	0,060
0.5	10	0,091	-0,300	-0,250
0.25	5	0,065	0,046	0,124
0.1	2	-0,097	0.028	0,128
0.05	1	-0,271		

Figure 4: Active power accuracy at cosphi = 1

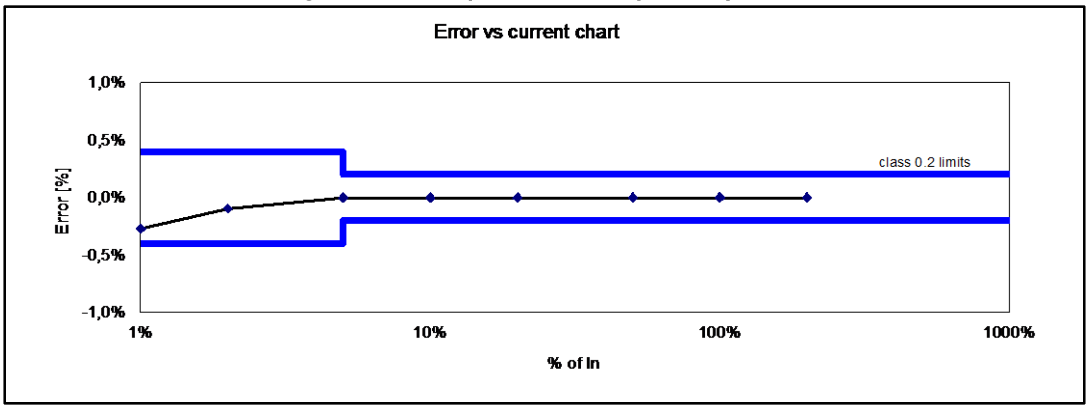
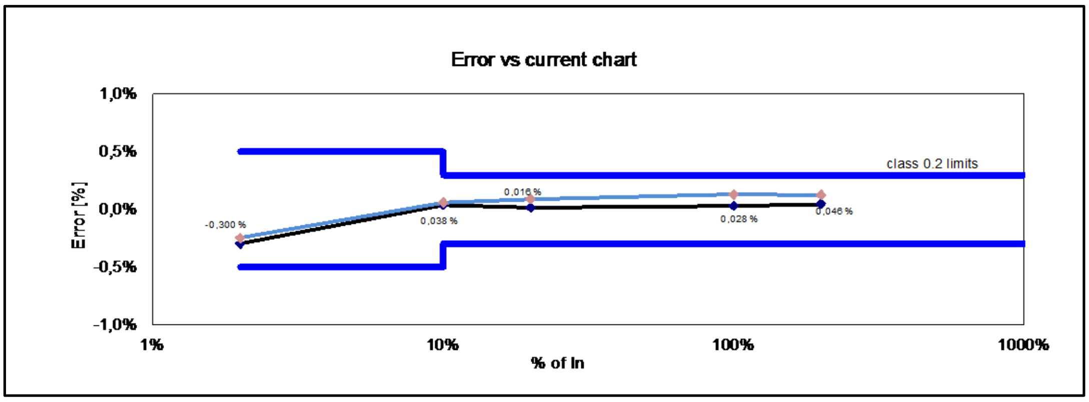


Figure 5: Active power accuracy at cosphi = 0.5 inductive and capacitive



5 Revision history

Table 6: Document revision history

Date	Revision	Changes
09-Dec-2015	1	Initial version.