AN3254
Application note

SPI protocol for the STPMC1 metering device

Introduction

The STPMC1 device is an ASSP designed for effective measurement in power line systems utilizing the Rogowski coil, current transformer, and shunt or Hall current sensors. Used in combination with one or more STPMSx ICs, it implements all the functions needed in a 1, 2, or 3-phase energy meter. It can be coupled with a microprocessor for multifunction energy meter or it can directly drive a stepper motor for a simple active energy meter.

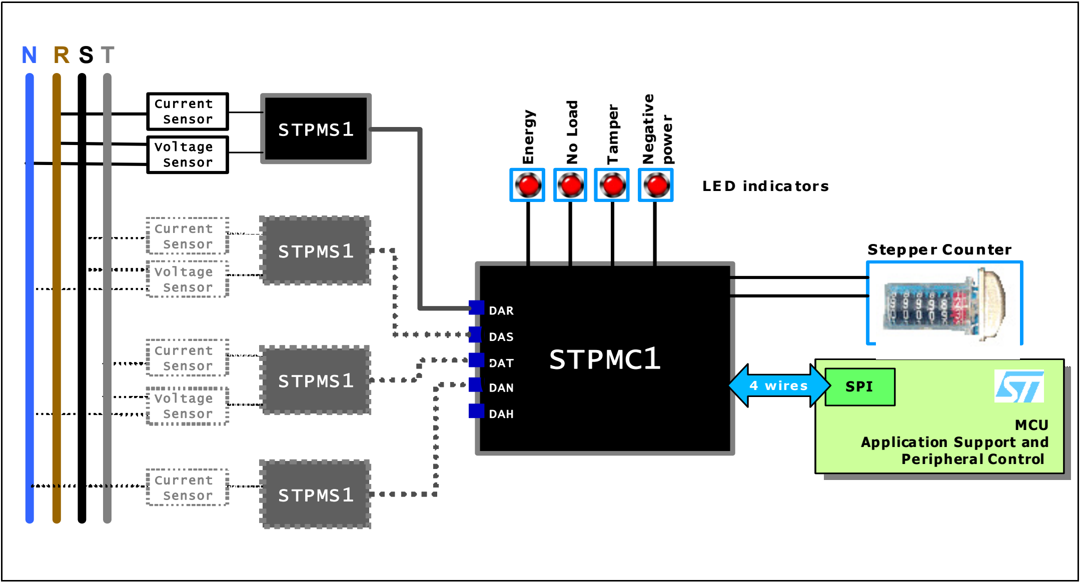
All the data measured by the STPMC1 are accessible through the SPI port, which is also used to configure and calibrate the device. The configuration and calibration data are retained in a 112-bit OTP block; in any case, these data can be dynamically changed in microprocessor based meters.

Measured data (like active and reactive energy, total and per phase, phase V_{RMS}, I_{RMS} and instantaneous voltage and current, line frequency, phase status, etc.) should be read by the microcontroller at a fixed time interval to be further processed.

This application note describes the SPI protocol to read measured data from the STPMC1 in a multiphase energy meter and how these readings should be processed by the application.

For more details on the device please refer to the STPMC1; *Programmable poly-phase energy calculator IC*, datasheet.

Figure 1. STPMC1 based application block diagram



Contents

1 SPI module description

The STPMC1 SPI interface supports a simple serial protocol, which is implemented in order to enable a communication between a host system (microcontroller or PC) and the device.

With this interface it is possible to perform the following tasks:

- remote reset of the device
- temporary and permanent programming of internal configuration/calibration data and system signals
- reading of internal data registers (shown in *Figure 5*).

Four pins of the device are dedicated to this purpose: SCS, SYN, SCL, and SDA.

When the STPMC1 is in standalone mode, SYN, SCL, and SDA can provide information on the meter status (see the STPMC1 datasheet for more information) and are not used for SPI communication. In this document, the SYN, SCL, and SDA operation as part of the SPI interface is described.

SCS, SYN, and SCL are all input pins while SDA can be input or output depending on whether the SPI is in write or read mode.

The internal registers are not directly accessible, rather, a 32-bit of transmission latches are used to pre-load the data before being read or written to the internal registers.

The condition in which SCS, SYN, and SCL inputs are set to a high level determines the idle state of the SPI interface and no data transfer occurs. Any SPI operation should start from this idle state. The exception to this rule is when the STPMC1 has been put into standalone application mode. In such mode it is possible that pin states of the SCL, SDA, and SYN are not high due to the states of corresponding internal status bits.

- SCS: enables SPI operation when low, both in standalone and in peripheral operating mode. This means that the master can abort any task in any phase by deactivation of SCS. In standalone mode SCS high enables SYN, SCL, and SDA to output the meter status.
- SYN: when SCS is low, the SYN pin status selects if the SPI is in read (SYN=1) or write mode (SYN=0). When SCS is high and SYN is also high, the results of the input or output data are transferred to the transmission latches.
- SCL: is the clock pin of the SPI interface. This pin function is also controlled by the SCS status. If SCS is low, SCL is the input of the serial bit synchronization clock signal. When SCS is high, SCL is also high, determining the idle state of the SPI. Configuration bit SCLP controls the polarity of the clock. SCLP=0 sets the clock idle state SCL=1, while SCLP=1 sets the clock idle state SCL=0.
- SDA: is the data pin. If SCS is low, the operation of SDA is dependent on the status of the SYN pin. If SYN is high, SDA is the output of serial bit data (read mode). If SYN is low, SDA is the input of serial bit data signal (write mode). If SCS is high, SDA is idle.

  When SCS is active (low), the signal SDA should change its state at the trailing edge of signal SCL and the signal SDA should be stable at the next leading edge of signal SCL. The first valid bit of SDA is always started with activation of signal SCL. This is valid if SCLP=0, otherwise the polarity of the clock is inverted.

A high level signal for these pins means a voltage level higher than 0.75 x V_{CC}, while a low level signal means a voltage value lower than 0.25 x V_{CC}.

1.1 Connection to microcontroller

The SPI master should be implemented by a host system, a PC, or a microcontroller.

The microcontroller's SPI bus is usually a 4-wire bus with full duplex functionality, whose signals are usually named as:

- SCLK: serial clock (output from master)
- MOSI: master output, slave input (output from master)
- MISO: master input, slave output (output from slave)
- SS: slave select (active low, output from master)

The best way to connect this standard SPI port to the STPMC1 SPI is to have SCS and SYN driven from some general purpose I/O port and SCL and SDA driven from SPI pins.

The suggested connection between the microcontroller and the STPMC1 is the following:

- MISO connected to SDA
- MOSI not connected
- SCLK connected to SCL
- SS connected to SCS
- a general purpose I/O pin connected to SYN.

In this way, the SPI peripheral unit of the microprocessor should operate as 2-wire (simplex synchronous transfers) SPI.

The microprocessor SPI peripheral can be used during STPMC1 device reading, while during the writing process it is possible to implement the SPI protocol via firmware.

In fact, in real applications the meter is calibrated and configured during meter production, so the main microcontroller task is to read from the device and, more rarely, to reset the device.

Moreover the reading time is crucial for a correct evaluation of the device data, it is advisable to emulate the writing procedure by firmware and to read using the SPI peripheral functionality, therefore exploiting all the port performances to reach very fast reading.

2 SPI interface timings

Table 1. SPI interface timings

Symbol	Parameter	Min.	Typ.	Max.	Unit
F_{SCLKr}	Data read speed			32	MHz
F_{SCLKw}	Data write speed			100	kHz
t_{DS}	Data setup time		20		ns
t_{DH}	Data hold time		0		ns
t_{ON}	Data driver on time		20		ns
t_{OFF}	Data driver off time		20		ns
t_{SYN}	SYN active width	$2/f_{XTAL1}$			s

In *Table 1* above, f_{XTAL1} is the oscillator clock frequency (see the STPMC1 datasheet for details).

3 SPI operations

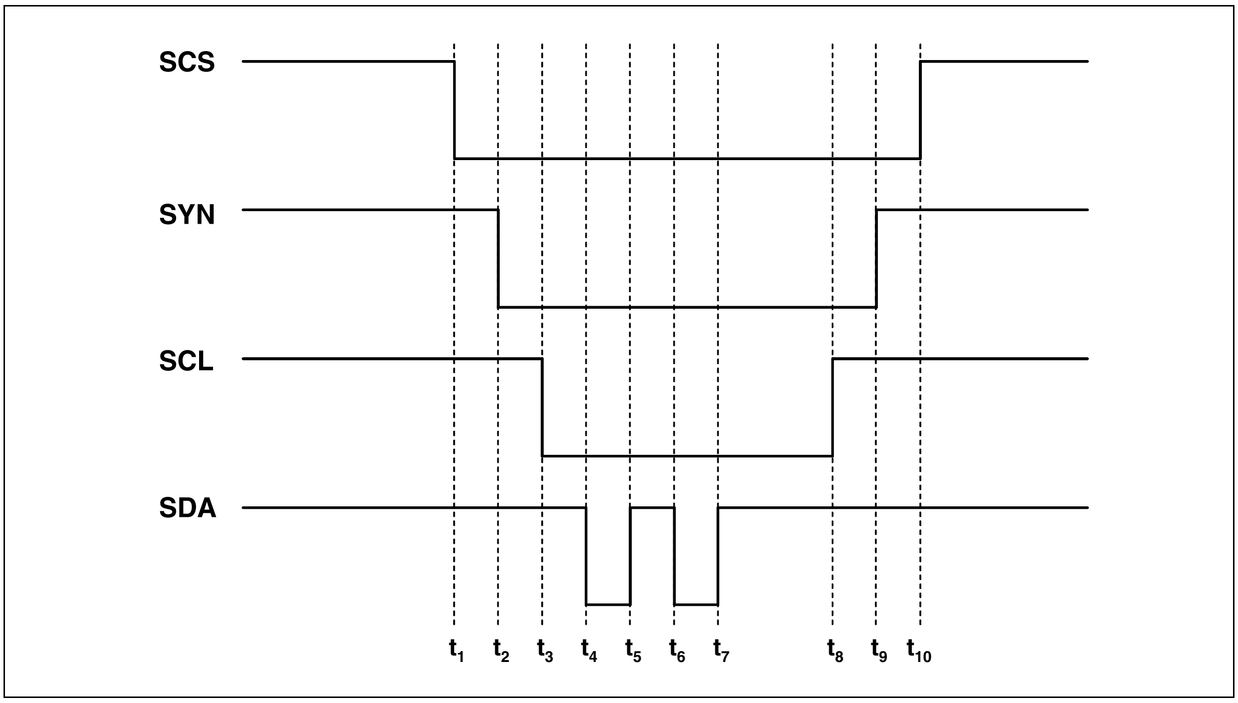
3.1 Remote reset request

The STPMC1 has no reset pin. It is automatically reset by the power on reset (POR) circuit when the V_{CC} crosses the 2.5 V value but it can be reset also through the SPI interface giving a dedicated command, the timing diagram is shown in *Figure 2*.

The reset through SPI (remote reset request - RRR) is sent from the onboard microprocessor when a malfunction of the metering device has been detected.

Unlike the POR, the RRR signal does not cause the 30 ms retarded restart of the analog module and the 120 ms retarded restart of the digital module. This reset does not clear the mode signals.

Figure 2. Remote reset request timing



Note: All the time intervals must be longer than 30 ns. $t_7 \rightarrow t_8$ is the reset time, this interval must also be longer than 30 ns.

3.2 Data registers writing

Each writable bit (configuration and mode signals bits) of the STPMC1 has its own 7-bit absolute address (see the STPMC1 datasheet for configuration bits map).

In order to change the state of some pins, a byte of data via the SPI must be sent to the device. This byte consists of 1-bit data to be written (MSB), followed by a 7-bit address of the destination bit, which makes a command byte.

For example, to set the STPMC1 configuration bit 47 (part of the R-phase current channel calibrator) to 1, the decimal 47 must be converted to its 7-bit binary value: 0101111. The byte command is then composed as:

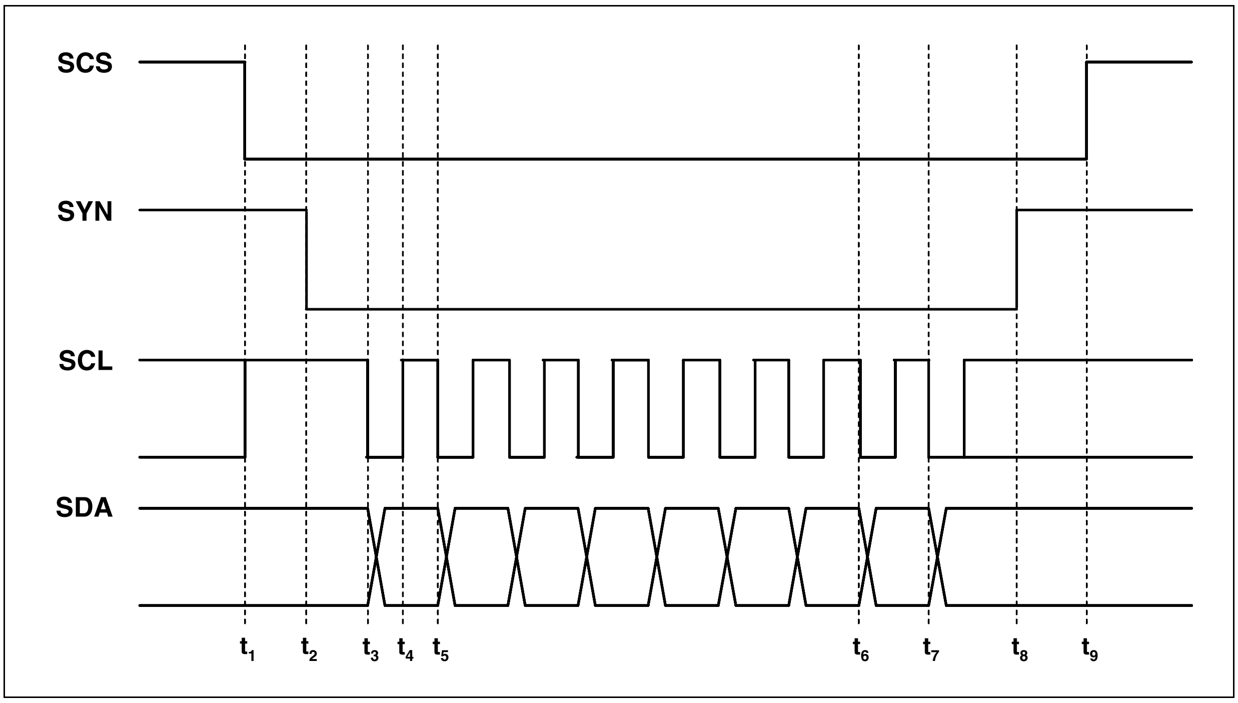
1 bit DATA value+7 bits address = 10101111 (0xAF)

The same procedure should be applied for the mode signals, which also have their specific address.

The LSB of the command is also called EXE bit because, instead of a data bit value, the corresponding serial clock pulse is used to generate the necessary latching signal. In this way, the writing mechanism does not need the measurement clock in order to operate, which makes the operation of the SPI module of the STPMC1 completely independent from the rest of the device logic, except from the signal POR.

The writing procedure timing is shown in *Figure 3*.

Figure 3. Timing for writing configuration bits and mode signals



$t_1 \rightarrow t_2$ (> 30 ns): SPI out of idle state

$t_2 \rightarrow t_3$ (> 30 ns): SPI enabled for write operation

t_3: data value is placed in SDA

t_4: SDA value is stable and shifted into the device

$t_3 \rightarrow t_5$ (> 10 µs): writing clock period

$t_3 \rightarrow t_5$: 1-bit data value

$t_5 \rightarrow t_6$: 6-bit address of the destination latch

$t_6 \rightarrow t_7$: 1-bit EXE command

t_8: end of SPI writing

t_9: SPI enters idle state

Commands for changing configuration bits and system signals should be sent during active signals SCS and SYN, as it is shown in *Figure 3*.

The SYN must be put low in order to disable the SDA output driver of the device and to make the SDA an input pin. A string of commands can be sent within one period of active signals SCS and SYN or a command can be followed by reading the data record but, in this case, the SYN should be deactivated in order to enable the SDA output driver and a SYN pulse should be applied before activation of the SCS in order to latch the data.

Given the connection between the STPMC1 and a microcontroller, as shown in the previous paragraph, it is possible to implement the writing procedure in the firmware through the following steps:

1. disable the SPI peripheral
2. set MISO, SCLK, and SS to be output
3. set the pin which is connected to SYN to be output high
4. activate SCS first and then SYN
5. activate SCL
6. apply a bit value to SDA and deactivate SCL
7. repeat the last two steps seven times to complete one byte transfer
8. repeat the last three steps for any remaining byte transfer
9. deactivate SYN and the SCS
10. enable again the SPI module.

To temporarily set any bit, it is necessary to set the RD system signal before any other bit. This bit determines the device functioning from OTP shadow latches and not from OTP memory. The procedure to set this signal is that shown above.

In the case of a precharge command (0xFF), emulation of the above is not necessary, it can be sent before any reading command. In fact, due to the pull up device on the SDA pin the processor needs to perform the following steps:

1. activate SYN first in order to latch the results
2. after at least 1 µs activate SCS
3. write one byte to the transmitter of SPI this produces 8 pulses on the SCL with SDA=1
4. deactivate SYN
5. read the data records as shown in *Section 3.4* (the sequence of reading is altered)
6. deactivate SCS.

3.3 Data registers permanent writing

In order to make a permanent set in OTP memory of some configuration bits, the following procedure should be conducted:

1. collect all addresses of bits to be permanently set into some list
2. clear all OTP shadow latches
3. set the system signal RD
4. connect a current source of at least +14 V, 1 mA to 3 mA to the V_{OTP} pin
5. wait until V_{OTP} voltage is stable
6. write one of the bits from the list (as the RD signal is set, the bit is written in the corresponding OTP shadow latch)
7. set the system signal WE
8. wait for 300 µs
9. clear the system signal WE
10. clear the OTP shadow latch which was set in step 6
11. until all wanted bits are permanently set, repeat steps 5 to 10
12. disconnect the current source
13. wait until V_{OTP} voltage is less than 3 V
14. clear the system signal RD
15. read all data records, in the last two there is read back of all configuration bits
16. if verification of CFG bits fails and there is still a chance to pass, repeat steps 1 to 16.

For the steps above which ask of set or clear, apply the timing shown in *Figure 3* with proper data on the SDA.

For step 15 apply the timing shown in *Figure 4*.

For a permanent set of the TSTD bit, which locks the device, the procedure above must be conducted in such a way that steps 6 to 13 are performed in series during a single period of active SCS because the idle state of SCS would make the signal TSTD immediately effective.

This would abort the procedure, and it would possibly destroy the device.

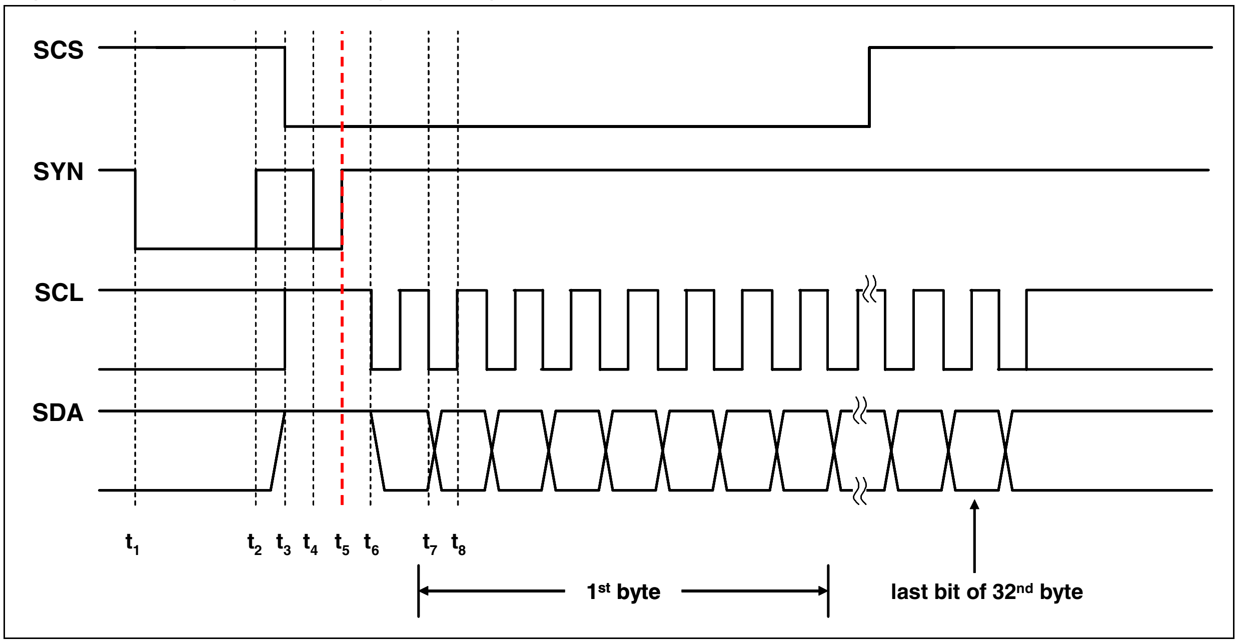
In fact the clearing of system signal RD would connect all gates of 3 V NMOS sense amplifiers of already permanently set bits to the V_{OTP} source.

3.4 Reading data registers

There are two phases of reading, called latching and shifting.

- Latching is used to sample results into transmission latches. This is done with the active pulse on SYN when SCS is idle. The length of pulse on SYN must be longer than 2 periods of the measurement clock, i.e. more than 500 ns.
- Shifting starts when SCS becomes active. In the beginning of this phase another, but much shorter pulse (30 ns) on SYN should be applied. An alternative way is to extend the pulse on SYN into the second phase of reading. Latching and shifting finish at the dotted line in the timing diagram shown in *Figure 4*.

Figure 4. Timing for reading data registers



$t_1 \rightarrow t_2$: latching phase. Interval value > 2 / f_{XTAL1}

$t_2 \rightarrow t_3$: data latched, SPI idle. Interval value > 30 ns

$t_3 \rightarrow t_4$: enable SPI for read operation. Interval value > 30 ns

$t_4 \rightarrow t_5$: serial clock counter is reset. Interval value > 30 ns

$t_5 \rightarrow t_6$: SPI reset and enabled for read operation. Interval value > 30 ns

t_7: internal data transferred to SDA

t_8: SDA data is stable and can be read

After the shifting phase, it is possible to read data, applying 32 serial clocks per data record. Up to 28 data records can be read this way.

There are seven groups of four data records available, each consisting of a parity nibble (see *Section 3.2*) and a 28-bit data field. *Figure 5* and *Figure 6* show the records structure and the information they hold in the default sequence of reading.

The system which reads the data record from the STPMC1 should check the integrity of each data record. If the check fails, the reading should be repeated, but this time only the shifting phase should be applied otherwise a new data would be latched into the transmission latches and the previous reading would be incorrectly lost.

Most of the registers contain the values of the electrical parameters and the status of the signals, except the registers CF0, CF1, CF2, and CF3 which represent the configuration bitmap.

The data records have a fixed position of reading. This means that no addressing of records is necessary.

The sequence of data records during the reading operation is fixed. However, an application may apply a precharge command prior to the reading phase. This command increases the group pointer forcing the device to respond with the next group data records sequence. In this way, a faster access to later groups is possible.

Figure 5. STPMC1 data registers

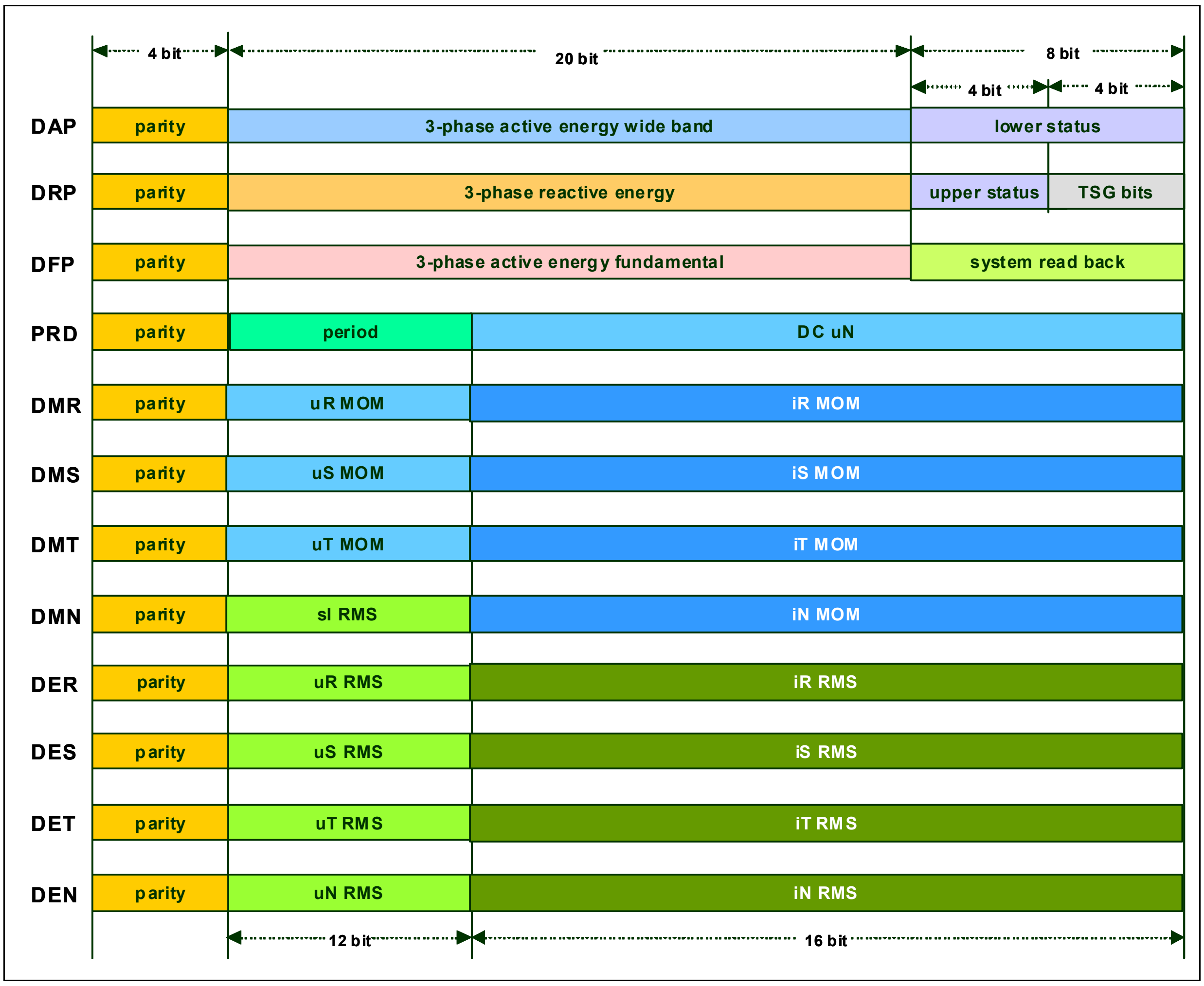
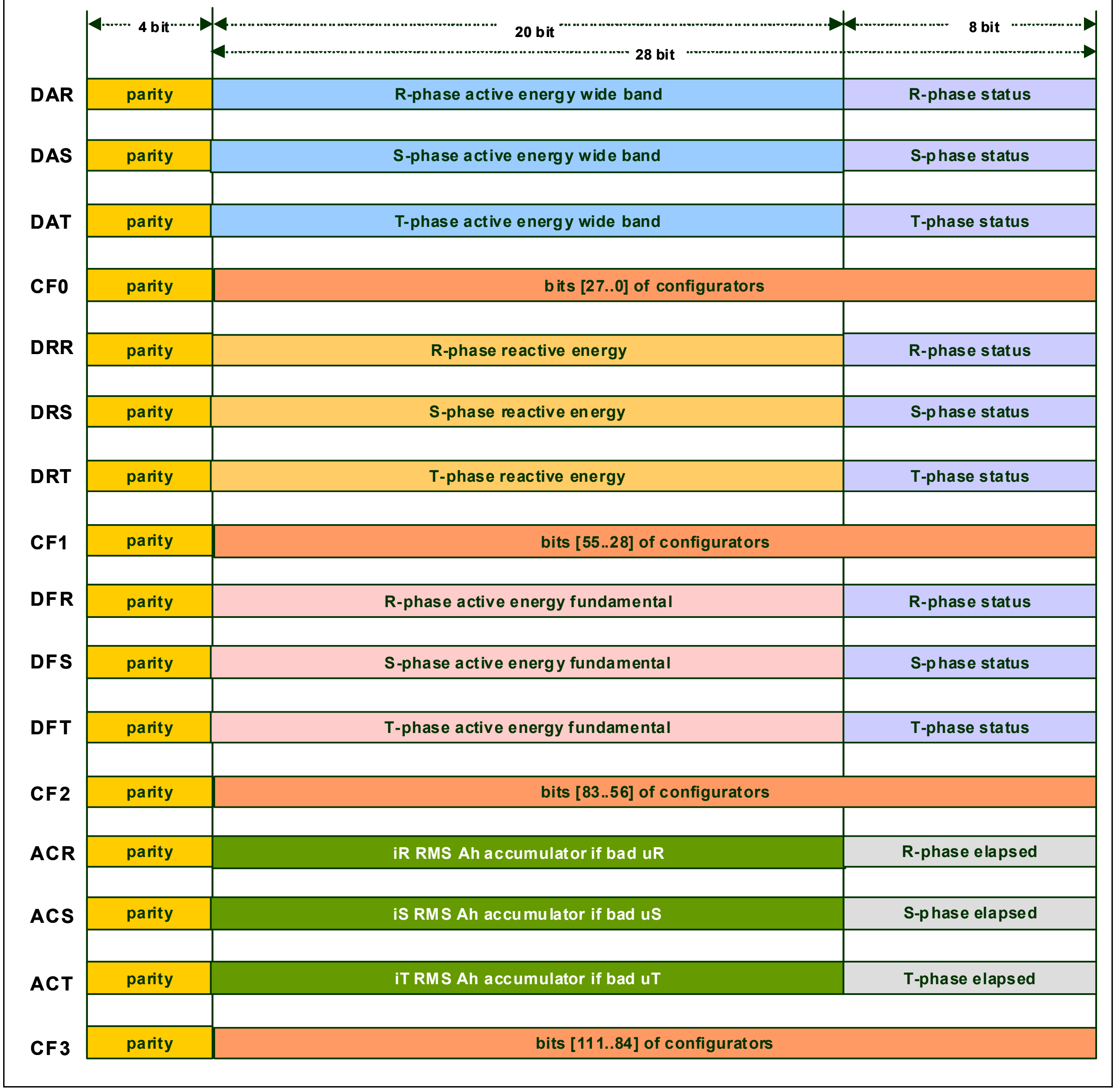


Figure 6. STPMC1 data registers



4 Data processing

4.1 Reading process

As previously mentioned, to start an SPI communication with the STPMC1 to read new values of registers, it is necessary to apply a latching phase first. Then a shifting phase starts, as described in *Figure 4*.

After that, 32 pulses of serial clocks need to be applied to pin SCL in order to read the DAP register. If an additional 32 pulses are applied to pin SCL, the DRP register is read.

Reading can be continued by applying 32 clocks per register until all registers of interest are read or a precharge command is applied first (8 pulses to pin SCLNLC while SYN=0 and SDA=1) which moves ahead the internal group pointer to group 1 (register DMR) which effectively skips DFP and PRD registers, and then reading may be continued.

The internal group pointer is incremented by 1 after each precharge command sent. If, for example, in the previous case two precharge commands are sent, the pointer will be set to group 2 (register DER).

It is up to an application to decide how many records should be read out from the device.

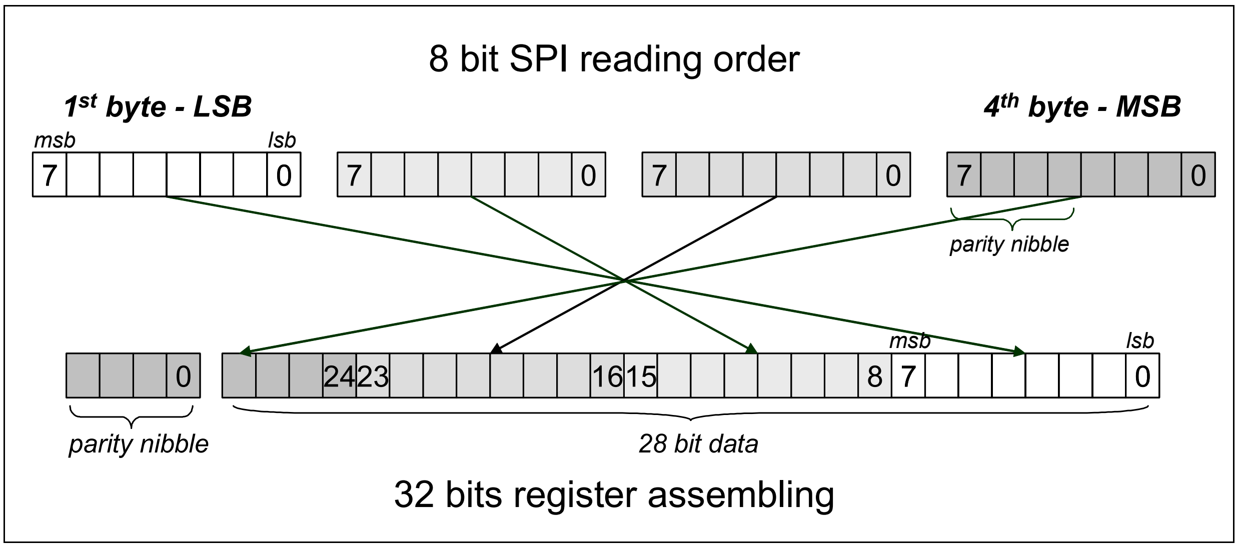
After all the registers are read, SCS can be returned to idle state which ends the shifting phase.

The shifting phase can be repeated and it should read the same values. This repetition is used to improve the reliability of successful reading in a strong EMI environment.

Every register is packed into 4 bytes where the most significant nibble (4 bits) is reserved for parity code and the rest of the 28 bits are used for data. This means that every register is protected by its own parity bit.

As shown in *Figure 7*, the first read out byte of the data record is the least significant byte (LSB) of the data value and the fourth is the most significant byte (MSB) of the data value, then it is necessary to re-order the four bytes after reading.

Figure 7. STPMC1 data register assembling



Normally, each byte is read out as the most significant bit (MSB) first. But this can be changed by setting the MSBF configuration bit. If this is done, each byte is read out as the least significant bit (LSB) first.

4.1.1 Data register assembling example

The following is an example of the reading and re-arranging of STPMC1 registers.

On the left there are the eight data records as they are read, represented as hexadecimal bytes while MSBF was cleared, on the right is the corresponding register.

1.	65 7A 7C 82	DAP = 82 7C 7A 65
2.	00 7A 0C E0	DRP = E0 0C 7A 00
3.	00 00 8C 92	DFP = 92 8C 0 00
4.	00 06 6E 22	PRD = 22 6E 06 00
5.	…	

4.2 Parity check

Each bit of parity nibble is defined as odd parity of all seven corresponding bits of data nibbles. In order to check the data record integrity, the application should execute something similar to the following C code, given as an example:

```
int BadParity (unsigned char *bp)
{
  register unsigned char prty = grp;
                                   /* temporary register set to group #
(0..6)*/
  prty  = *bp,                      /* take the 1st byte of data */
  prty ^= *(bp+1),                  /* XOR it with the 2nd byte */
  prty ^= *(bp+2),                  /* and with the 3rd byte */
  prty ^= *(bp+3),                  /* and with the 4th byte */
  prty ^= prty<<4, prty &= 0xF0;/* combine and remove the lower
nibble */
  return (prty != 0xF0);            /* returns 1, if bad parity */
}
```

If the parity nibble check fails, the reading task should be repeated, but this time without the request of latching, otherwise a new data would be latched and the previous reading would be incorrectly lost.

In a very harsh EMI environment, it would be a good practice to read the data records twice and then compare both readings. In this way, the probability of detecting bad readings would be significantly improved. However, a single piece of bad data can be discarded because no meaningful information is lost as long as the reading frequency is about 30 ms.

4.2.1 Parity check example

Let us calculate the parity of DMR, the first register of the second group:

```
   DMR:  02    80    00    C8
prty = grp = 1                      /* prty set to 1 - group #*/
  prty  = *bp = 3                   /* xor it with 1st byte of data 02 */
  prty ^= *(bp+1) = 83              /* xor it with the 2nd byte 80*/
  prty ^= *(bp+2) = 83              /* and with the 3rd byte 00 */
  prty ^= *(bp+3) = 4B              /* and with the 4th byte C8 */
  prty ^= prty<<4 = FB              /* and with B0 */
  prty &= 0xF0 = F0                 /* parity is ok */
```

4.3 Unpacking data

After reading (and the following re-ordering of bytes read), each register should be unpacked in order to obtain all individual values.

For this purpose it is necessary to mask the 28 bits according to the register map shown in *Figure 5* and *Figure 6*.

For example, the DAP register is unpacked into an 8-bit value of status (least significant byte) and a 20-bit value of the 3-phase active energy counter (the remaining upper 3 bytes with parity code masked out).

5 Converting readings into measured values

5.1 Energies

The first three registers contain 20-bit values of the internal 3-phase energy up/down counters, and in the following groups there are the registers containing each phase energy counter.

The value of the least significant bit of every energy counter is related to power meter constant P, which is the number of pulses per kWh that the meter, through calibration, is configured to provide to the LED pin.

This means that this value changes with the application and relative calibration.

Given P, the number of pulses per kWh provided, the energy registers LSB value is indicated in *Table 2* below:

Table 2. Energy registers LSB value

Register	$\overline{\text{SYS}}$ = 0,1,2,4,5,6,7	$\overline{\text{SYS}}$ = 3
3-ph Active Energy Wide Band (P)	$K_P = \frac{1000}{P \cdot 2^{10}}[Wh]$	$K_P = \frac{1000}{P \cdot 2^{10}}[Wh]$
3-ph Reactive Energy Wide Band (Q)	$K_Q = \frac{1000}{P \cdot 2^{10}}[VArh]$	$K_Q = \frac{1000}{P \cdot 2^{9}}[VArh]$
3-ph Active Energy Fundamental (F)	$K_F = \frac{1000}{P \cdot 2^{10}}[Wh]$	$K_F = \frac{1000}{P \cdot 2^{9}}[Wh]$
3-ph Reactive Energy Fundamental (R)	$K_R = \frac{1000}{P \cdot 2^{10}}[VArh]$	$K_R = \frac{1000}{P \cdot 2^{9}}[VArh]$

For example, the energy registers LSB values for $\overline{\text{SYS}}$ = 0, 1, 2, 4, 5, 6, 7 when P = 64000 pulses/kWh = 17.7 Hz*kW are:

$K_P = K_F = 15.258 * 10^{-6}$ Wh

$K_Q = K_R = 15.258 * 10^{-6}$ VArh

The energy registers LSB values for $\overline{\text{SYS}}$ = 3 and the same P are:

$K_P = 15.258 * 10^{-6}$ Wh

$K_F = 30.517 * 10^{-6}$ Wh

$K_Q = K_R = 30.517 * 10^{-6}$ VArh

This also means that the STPMC1 energy counters hold a very small energy value (in the example above, when LSB represents 15.258 µWh, the whole register stores 16 Wh), and further energy integration must be performed inside the application.

To accomplish this task, the procedure below should be followed.

Because all energy counters rollover in approximately 1 s when they are integrating maximal power, the reading must be done frequently enough. It is suggested to read the registers at least 32 times per second.

For each energy type, a variable ***e*** should be allocated, having the following structure (below is the variable definition for an ST7 microcontroller):

```
typedef struct energ {
   unsigned long old;      /* previous energy value - 32 bits */
   unsigned int quot;      /* quant/16 - 16 bits */
   signed int quant;      /* new - old, measure of power - 16 bits */
   signed long frac;       /* fractional part of energy integrator -
32 bits */
   signed long integ;      /* integer part of energy integrator - 32
bits */
} ENERG;
```

The application should keep the previous value of each energy counter in order to evaluate the difference of readings, from which also a direction of energy flow can be obtained. This value should be stored in ***e→ old*** before a reading. After the reading, the new energy register reading should be stored in ***e→ new***.

To calculate consummated energy the software should implement a 32-bit integrator. The suggested integrator is two stages, with ***e→ frac*** and ***e→ integ*** 32-bit signed integer variables. Into ***e→ frac*** is added the value ***e→ quant***, obtained as the difference between ***e → old*** and ***e→ new*** energy values; then the ***e→ old*** value should be rewritten with the ***e→ new*** value in order to enable a correct ***e→ quant*** computation next time.

When ***e→ frac*** collects a certain amount of energy, let's say 10 Wh for active energy (corresponding to a certain threshold value according to K_P), ***e→ integ*** should change for 1 bit and the ***e→ frac*** should change by the threshold value.

In this way, ***e→ frac*** stores 0.01 KWh, after which ***e→ integ*** is increased by one, and the ***e→ integ*** variable holds accumulated energy of which the least significant bit represents 10 Wh.

Considering an active energy meter where P = 64000 imp/kWh, for a step of 0.01 KWh = 10 Wh, as each bit of ***e→ quant*** represents K_P Wh (it is the same resolution of the internal energy counter, because ***e→ quant*** is calculated as a difference of two energy counter values), the threshold value is $10/K_P = 10/15.258*2^{06}$ = 0xA0021.

In a microcontroller based application, a high priority timer interrupt should be set to perform measuring tasks every 1/512 s. Within this interrupt service 16 different subtasks could be established in order to break the whole meter task into 16 shorter consecutive subtasks (reading of device register, checking the data read, and, if ok, computing the value of ***e→ quant***). In this way, the main program and other interrupt services are not blocked for more than a few 100 us every 2 ms, and the meter task is completed in 16 steps - that is in 1/32 s.

The interrupt service should do the following:

- update ***e→ frac*** and ***e→ integ*** of the energy variable using ***e→ quot*** = ***e→ quant*** / 16
- generate output pulses (if needed) from ***e→ frac***
- call the next subtask
- perform other tasks (if needed)

In this way the addition of ***e→ quant*** is split in 16 times. This generates a microcontroller output pulse that has a 16 times better accuracy of position in time, which would reduce the jitter of an eventual LED output.

5.2 Other values

5.2.1 Voltage, current, and frequency calculation

The ratio between the register value and the actual voltage, current, or frequency value is a function of the voltage and current sensor sensitivity and of the device internal parameters, like analog amplification, reference voltage, measurement frequency, calibrator, attenuation of each stage of decimation filter, and power meter constant.

Formulas to convert the readings into meaningful values are reported below.

For details on the device configuration bits mentioned below, please refer to the STPMC1 datasheet.

In any case, as the internal parameter values, here given as constants, are subject to process drift, and the sensors sensitivity are subject to tolerance, even if these fluctuations are compensated by the calibrators, the best way to obtain the proper parameters is to measure known signals and calculate the ratio between the register value and actual value. The device linearity ensures that the ratio remains constant.

Figure 8, *9*, and *10* below show the signal processing chains for each phase current and voltage.

Figure 8. Voltage signal path

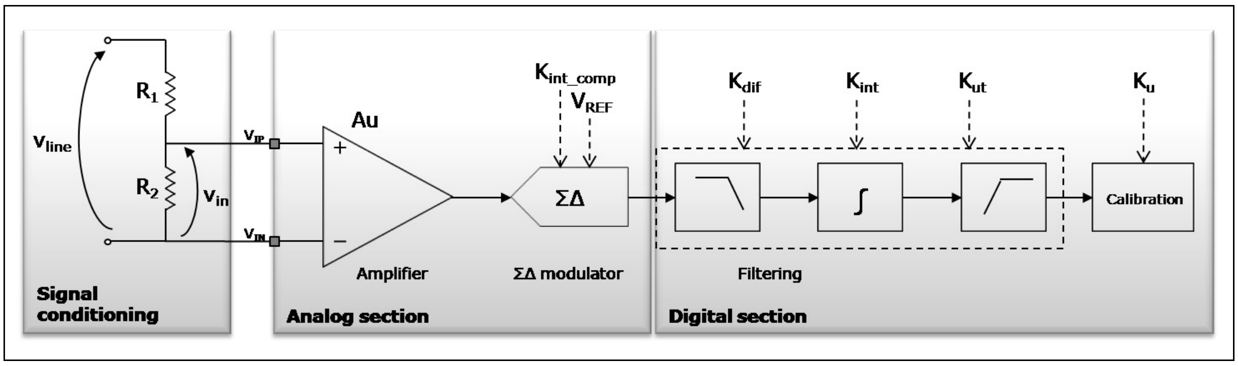


Figure 9. Current conditioning

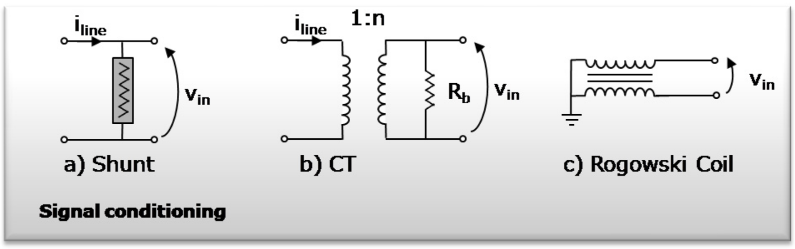
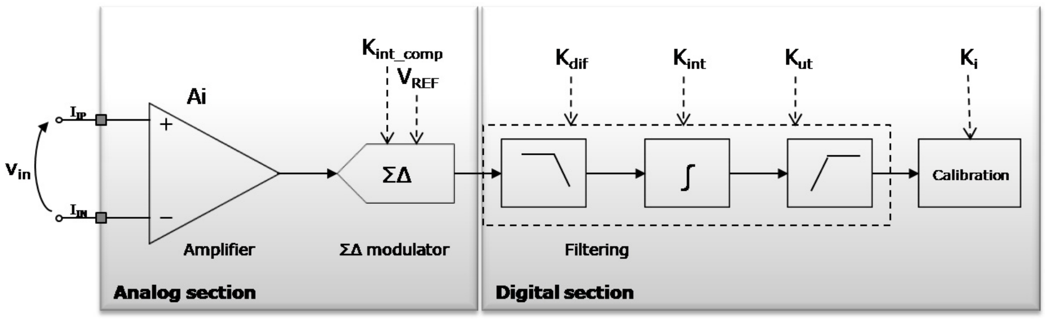


Figure 10. Current signal path



Each block of the path contributes to the signal processing with the parameters shown below.

Table 3 shows the variable parameters that must be considered as inputs for the following calculation, while *Table 4* shows the device internal constant parameters.

Table 3. Input parameters

Parameter	Meaning
R1, R2	Voltage Divider Resistors Value [Ohm]
Ks	Current Sensor Sensitivity Value [V/A]
x_i	Current Register Value expressed as decimal
x_u	Voltage Register Value expressed as decimal
x_period	Period Register Value expressed as decimal
Ai	Current Channel Gain

x_i can be one of the current readings in the registers from DMR to DEN (iR MOM, iS MOM, etc., sI RMS, iR RMS, iS RMS, etc.), and x_u can be one of the voltage readings in the registers from PRD to DEN (DC uN, uR MOM, uS MOM, etc., uR RMS, uS RMS, etc.).

x_period is the 12 bits value in the PRD register.

Table 4. STPMC1 internal parameters value

Parameter	Value	Meaning
Ku	0,875	Voltage calibrator ideal value if PM = 0 (1)
	0,9375	Voltage calibrator ideal value if PM = 1
Ki	0,875	Current calibrator ideal value if PM = 0
	0,9375	Current calibrator ideal value if PM = 1
Kisum	0,875	Calibrator ideal value for sI RMS (sum of currents)(2)
Au	4	Voltage channel gain
len_i	2^{16}	Current register length
len_u	2^{12}	Voltage register length
len_isum	2^{12}	sI RMS (sum of currents) register length
Kint_comp	1,004	Gain of decimation filter
π	3.14159	
Fm	$4 * 10^6$	If oscillator frequency is 4.000 or 8.000 MHz
	2^{22}	If oscillator frequency is 4.194 or 8.388 MHz
	4915200	If oscillator frequency is 4.915 or 9.830 MHz
Kut	2	For CT/Shunt
	1	For Rogowski coil
Vref	1.23	Internal voltage reference

1. PM is CFG 21, it sets the meter precision (Class 1 or Class 0.1)
2. 12 bits in DMN register

Line frequency is calculated as follows:

$$frequency = Fm /(x\_period * 2^5)$$

The integrator and differentiator gains are calculated as follows:

$$Kint = 2 * x\_period * 2^5/ (p * 2^{16})$$

$$Kdif = 1/ (2 * Kint)$$

Typical values for these gains are:

Table 5. Kint and Kdif typical values

Kdif	0,6135	Gain of differentiator @ line frequency = 50 Hz
	0.7359	Gain of differentiator @ line frequency = 60 Hz
Kint	0,815	Gain of integrator @ line frequency = 50 Hz
	0.679	Gain of integrator @ line frequency = 60 Hz

From these values the following scaling parameters are calculated for Rogowski Coil sensor:

```
Kf = frequency / 50

Kdspu = 1

Kdspi = Kf
```

and for CT or Shunt current sensors:

```
Kdspu = Kdif * Kint

Kdspi = Kdif
```

The uN RMS value (12 bits in DEN register) conversion formula is:

```
u = x_u * 2 / len_u
```

For the sum of RMS currents sI RMS (12 bits in DMN register) the value is obtained as follows:

```
si = x_i * Vref/(Ks * Kisum * Ai * len_isum * Kint * Kint_comp * Kdspi)
```

For the data from the momentary registers, as the related momentary voltage and current parameters are signed (they can be positive or negative), it is necessary to evaluate the sign with the following task. This does not apply to the RMS values:

```
if (x_i & (len_i>>1)) // positive current
  x_i = x_i & ((len_i>>1)-1);
else                  // negative current
{
  x_i = (len_i>>1) - x_i;
  x_i = x_i * (-1);
}

if (x_u & (len_u>>1)) // positive voltage
  x_u = x_u & ((len_u>>1)-1);
else                  // negative voltage
{
  x_u = (len_u>>1) - x_u;
  x_u = x_u * (-1);
}
```

Both RMS and momentary current and voltage conversion formulas then are:

```
u = (1+R1/R2)* x_u * Vref /(Kut * Ku * Au * len_u * kint_comp* Kdspu)

i = x_i * Vref/(Ks * Ki * Ai * len_i * Kint * Kint_comp * Kdspi)
```

For SYS configuration bits 2 or 3 (no neutral wire) the voltage value should be further multiplied by 2:

```
u = u * 2
```

5.2.2 Other values

The ACR, ACS, and ACT registers hold the information needed for this calculation.

Concatenating ACT[7:0], ACS[7:0], and ACR[7:0] bytes, two 12-bit vectors, defined below, are obtained:

ACT[7:0], ACS[7:0], ACR[7:0] = Asr[11, 10:0], Art[11, 10:0]

The delay times are calculated with the following formulas:

$$time_{Asr} = (Asr[10;0] - (2^{11})^{Asr[11]} + 1) \cdot \frac{8}{f_{MCLK}}$$

$$time_{Art} = (Art[10;0] - (2^{11})^{Asr[11]} + 1) \cdot \frac{8}{f_{MCLK}}$$

where f_{MCLK} depends upon the oscillator value according to *Table 6*:

Table 6. f_{MCLK} value

f_{XTAL1}	MDIV	f_{MCLK}
4.000 MHz	0	8.000 MHz
4.194 MHz	0	8.192 MHz
4.915 MHz	0	9.830 MHz
8.000 MHz	1	8.000 MHz
8.192 MHz	1	8.192 MHz
9.830 MHz	1	9.830 MHz

The phase delays in degrees can be calculated as follows:

$$\varphi° = \frac{time}{f_{line}} \cdot 360°$$

6 Revision history

Table 7. Document revision history

Date	Version	Description
19-Nov-2010	1	First release