Business *ENERGY*

Services Offer Creation

SENSEOR THERMAL SENSOR

MODBUS MODBUS SPECIFICATION OF SENSEOR GATEWAY

Revision R1

Summary
Résumé

Author(s)
Auteur(s)

Michel Cercueil

Document-reference / Référence du document : XX

Rev.	Date : D/M/Y	Author(s)	Modifications
1	22/10/2013	M. Cercueil	Creation

CONTENT

1. GENERAL

1.1. Goal of the document

The goal of this document is to specify all the Modbus communication of the gateway that will read the temperature from wireless SAW thermal sensors.

1.2. General instruction

All the settings and operations of the gateway must be accessible using the Modbus link. This includes:

- Configuration of the gateway,
- Configuration of the sensors,
- Reading of values,

1.3. Factory reset

There should be a mean to set the communication settings or restore the default communication settings in the case where we don't know the current configuration.

2. RS485 SPECIFICATION

2.1. Physical requirements

Modbus connection is realized using an RJ45 connector.

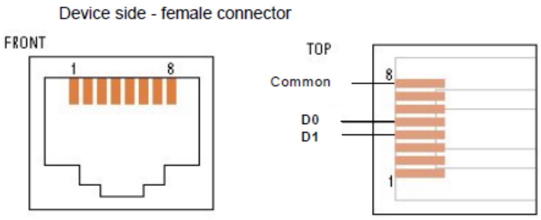

Figure 1 2W- MODBUS on RJ45 connector (required pin-out)

Pin on RJ45	IDv	EIA/TIA-485 name	Description for IDv
4	D1	D1	**Transceiver terminal 1, V1 Voltage** (V1 > V0 for binary 1 [OFF] state)
5	D0	D0	**Transceiver terminal 0, V0 Voltage** (V0 > V1 for binary 0 [ON] state)
8	Common	Common	**Signal and Power Supply Common**

Table 1 Table 7 RJ45 Connector pinout

2.2. Automatic Baud and parity recognition

If the functionality is activated in the device settings, the Modbus communication settings (Baud rate and parity) must be set automatically based on the recognition of the current configuration on the Modbus network.
Refer to document *"TI 074 - Specification on Autoadaptation Modbus SL V0.9"* for a complete description of the method.
The activation of the Auto-Baud function can be performed either by the HMI or through a Modbus setting register.
This mode is **activated by default**.

3. MODBUS COMMUNICATION

3.1. Addressing rules

The slave address must be configurable between 1 and 247.

The device must accept **broadcast requests at @0**.

The device must always answer to requests at **@248** dedicated to point to point connection.

3.2. Transmission mode

The Modbus communication implements the **RTU mode**. ASCII mode is not supported.

All data exchanged hereafter are in the form of 16-bit words (also called registers). Each piece of information has a 16-bit address.

3.3. Modbus Functions supported

Function code	Detail
01	Read coils (see §9.2)
03	Read registers (see §9.3)
16	Writing of n words. (see §0)
43	Sub-function 14: reading of identification. (see §6)

Table 2 Supported Modbus function codes

3.4. Invalid access

The device must always answer to Modbus read requests even if the device can’t complete all the information requested.

3.4.1. Access to reserved registers

Modbus server must answer for reserved registers with invalid data corresponding to the data type of the extended data according to Invalid data value section.

3.4.2. Access to unused registers

Modbus server must answer for unused registers with invalid data corresponding to Int16 data type according to §4.3.

3.4.3. Access to part of consistent data

If a Modbus client accesses only a part of a data (for example only one word of a Float32 or Uint32 register), the server must return an illegal data value: Modbus exception (code 03).

If a Modbus client accesses several data and if at least one of the data is not read completely, server must return illegal data value: Modbus exception (code 03).

4. DATA ADDRESSES AND CODING

4.1. Endianess

All the Modbus communication uses the **Big-endian** convention.

- *Exception*

  The CRC of the modbus frame is encoded using the Little-endian convention (first byte contains the low order bits and the second contains the high order bits).

4.2. Register Number, Bit and Word addressing rule

The Register Address in this document is in *hexadecimal.*

The Register Number in this document is Register Address + 1 in *decimal*.

Some registers can be accessed either through bit or word addressing.

Addresses for booleans are in the 0…0x0FFF word address range. Theses bit addresses are computed as follows:

```
Bit Address = Word Address x 16 + Bit rank in the word
```

- *Example*

  Bit 0 of Word 0x0FFF is addressed 0xFFF0.

4.3. Invalid data value

Data type	Value
String	0x00
Int16	0x8000
Uint16	0xFFFF
Uint32	0xFFFF FFFF
Float32	0xFFC00 0000
Date/time (encoded in 4 words)	0xFFFF FFFF FFFF FFFF

Table 3 Invalid data values for each data type

For 16 bits bit-field data, a 16-bits validity word is used. Each bit of this validity word indicates the validity of the corresponding bit in the data word.

The validity word is always located at Data address – 1.

If data validated by this validity word can't be invalidated nor validated, the validity word is set to *Valid Value*.

5. MODBUS MAPPING

Register	Address	Nb. Of registers	Access	Type	Unit	Name	Description
System information							
51	0x32	1	R	Uint16	-	Number of sensors	Number of thermal sensors managed by the gateway (=15 currently)
52	0x33	1	R	Uint16	-	Product identifier	TBD
53	0x34	4	R/W	Int16	-	Test zone	Freely accessible registers to test the communication. Initialized to 0.
Temperature sensors calibration							
101	0x64	12	RW	-	-	Sensor 1 calibration structure	See Table 5 Sensor calibration structure
113	0x70	12	RW	-	-	Sensor 2 calibration structure	
125	0x7C	12	RW	-	-	Sensor 3 calibration structure	
137	0x88	12	RW	-	-	Sensor 4 calibration structure	
149	0x94	12	RW	-	-	Sensor 5 calibration structure	
161	0xA0	12	RW	-	-	Sensor 6 calibration structure	
173	0xAC	12	RW	-	-	Sensor 7 calibration structure	
185	0xB8	12	RW	-	-	Sensor 8 calibration structure	
197	0xC4	12	RW	-	-	Sensor 9 calibration structure	
209	0xD0	12	RW	-	-	Sensor 10 calibration structure	
221	0xDC	12	RW	-	-	Sensor 11 calibration structure	
233	0xE8	12	RW	-	-	Sensor 12 calibration structure	
245	0xF4	12	RW	-	-	Sensor 13 calibration structure	
267	0x100	12	RW	-	-	Sensor 14 calibration structure	
279	0x10C	12	RW	-	-	Sensor 15 calibration structure	

Commentaire [MC1]: A compléter

Register	Address	Nb. Of registers	Access	Type	Unit	Name	Description
Communication settings							
401	0x190	1	RW	Uint16	-	Slave address	Modbus slave address of the device. Must be between 1 and 247.
402	0x191	1	R	Uint16	-	Auto-Baud mode	Set to 1 if automatic communication recognition is activated (Baud rate and parity). Default=1.
403	0x192	1	R	Uint16	-	Baud rate	0=9600 Baud (default), 1=19200 Baud, 2=38400, 3=57600, 4=115200
404	0x193	1	R	Uint16	-	Parity	0=none, 1=even (default), 2=odd
Temperature measurements							
1001	0x3E8	2	R	Float32	°C	Sensor 1 temperature	If the sensor is defective or the measurement could not be performed for any reason, the value in this register should be set to NaN (See §4.3)
1003	0x3EA	2	R	Float32	°C	Sensor 2 temperature	
1005	0x3EC	2	R	Float32	°C	Sensor 3 temperature	
1007	0x3EE	2	R	Float32	°C	Sensor 4 temperature	
1009	0x3F0	2	R	Float32	°C	Sensor 5 temperature	
1011	0x3F2	2	R	Float32	°C	Sensor 6 temperature	
1013	0x3F4	2	R	Float32	°C	Sensor 7 temperature	
1015	0x3F6	2	R	Float32	°C	Sensor 8 temperature	
1017	0x3F8	2	R	Float32	°C	Sensor 9 temperature	
1019	0x3FA	2	R	Float32	°C	Sensor 9 temperature	
1021	0x3FC	2	R	Float32	°C	Sensor 9 temperature	
1023	0x3FE	2	R	Float32	°C	Sensor 10 temperature	
1025	0x400	2	R	Float32	°C	Sensor 11 temperature	
1027	0x402	2	R	Float32	°C	Sensor 12 temperature	
1029	0x404	2	R	Float32	°C	Sensor 13 temperature	
1031	0x406	2	R	Float32	°C	Sensor 14 temperature	
1033	0x408	2	R	Float32	°C	Sensor 15 temperature	

Register	Address	Nb. Of registers	Access	Type	Unit	Name	Description
Sensors details							
	0x	1	R	Uint16	-	Sensor 1 details	First byte : Signal strength Second byte : Sensor status (0=OK, 1=No answer)
	0x	1	R	Uint16	-	Sensor 2 details	
	0x	1	R	Uint16	-	Sensor 3 details	
	0x	1	R	Uint16	-	Sensor 4 details	
	0x	1	R	Uint16	-	Sensor 5 details	
	0x	1	R	Uint16	-	Sensor 6 details	
	0x	1	R	Uint16	-	Sensor 7 details	
	0x	1	R	Uint16	-	Sensor 8 details	
	0x	1	R	Uint16	-	Sensor 9 details	
	0x	1	R	Uint16	-	Sensor 10 details	
	0x	1	R	Uint16	-	Sensor 11 details	
	0x	1	R	Uint16	-	Sensor 12 details	
	0x	1	R	Uint16	-	Sensor 13 details	
	0x	1	R	Uint16	-	Sensor 14 details	
	0x	1	R	Uint16	-	Sensor 15 details	

Table 4 Gateway registers

- *Sensors calibration*

Each SAW sensor contains 2 resonators which need 3 calibration parameters: F0, CTF1 and CTF2. These values will be provided along with each sensor. The calibration structure for each sensor will look as follow:

Address	Nb. Of registers	Type	Name
@	2	Float32	F0 resonator 1
@+2	2	Float32	CTF1 resonator 1
@+4	2	Float32	CTF2 resonator 1
@+6	2	Float32	F0 resonator 2
@+8	2	Float32	CTF1 resonator 2
@+10	2	Float32	CTF2 resonator 2

Table 5 Sensor calibration structure

6. IDENTIFICATION

Refer to "Support Read Device Identification Requests Over Modbus" file for more details on identifications.

The Device must support **regular identification data requests**.

If the device is asked for a description level higher than its conformity level, it must respond in accordance with its actual conformity level.

6.1. Read device Identification function

The MODBUS function 43 (0x2B) performs the *Read Device Identification* service. This specialized interface is identified by a MEI type (Modbus Encapsulated Interface) which is a Modbus sub-function 14 (0x0E) assigned to access to the Device Description objects and Read Identification Services.

Here is an example of an Identification Request and its response:

Request		**Response**	
Field Name	***Value***	***Field Name***	***Value***
Address	0xAA	Address	0xAA
Function	0x2B	Function	0x2B
MEI Type	0x0E	MEI Type	0x0E
ReadDevId code	0x02	ReadDev Id	0x02
Object Id	0x00	Conformity Level	0x02
Error Check Lo	0x8F	More Follows	0x00
Error Check Hi	0x58	Next Objet Id	0x00
		Nuber of Objects	0x06
		Object0.Id	0x00
		Object0.Length	0x12
		Objet0.Value	"Schneider Electric"
		Object1.Id	0x01
		Object1.Length	0x0D
		Object1.Value	"Pxxxxxxxxxxxx"
		Object2.Id	0x02
		Object2.Length	0x0B
		Object2.Value	"001.000.000"
		Object3.Id	0x04
		Object3.Length	0x12
		Object3.Value	"ProRelayMonitoring"
		Object4.Id	0x05
		Object4.Length	0x01
		Object4.Value	NULL
		Object5.Id	0x06
		Object5.Length	0x01
		Object5.Value	NULL
		Error Check. Lo	0xXX
		Error Check. Hi	0xXX

Table 6 Modbus Serial Line PDU

Read DevID:	specifies the access (01 for basic identification, 02 for regular identification, 03 for extended and 04 for specific object access)
Conformity Level:	depends of the level of the identification request.
More Follows:	set to 0x00 if no more objects available, set to 0xFF if further Modbus transaction required.
Next Object ID:	set to 0x00 if *More Follows* is set to 0x00, identification of the next object otherwise

6.2. Minimal identification data

Minimal identification data are specified in the Device Description Guide.

Id	**Name / Description**	**Type *(1)***
0x00	VendorName	ASCII String
0x01	ProductCode	ASCII String
0x02	MajorMinorRevision	2 ASCII String
0x04	Product Name	ASCII String
0x05	Model Name	ASCII String
0x06	User App Name	ASCII String

Table 7 Minimal identification data

6.2.1. Major Minor Revision Number

The Major and Minor Revision number is represented by two “xxx.yyy.zzz” strings of 11 chars for firmware and hardware revision.

- “xxx” corresponds to major revision number [0 … 127]
- “yyy” corresponds to minor revision number [0 … 255]
- “zzz” corresponds to patch revision number [0 … 255]

These values must always be incremented by one and respect the following rules:

- Increment “xxx” forces “yyy” and “zzz” reset to 0
- Increment “yyy” forces “zzz” reset to 0

➢ <u>*Rules*</u>

- Major revision number must be reserved for developments not released to customer and when a compatibility break is introduced.
- Minor revision number must be reserved for adding or removing a feature.
- Patch revision number must be reserved for fixing bugs.

As hardware bug fixing can be done without firmware upgrade, patch version for hardware must be forced to 0 when the hardware revision is embedded in the firmware.

The User App Name is pre defined to Product Model value. It is re initialized to Product Model Value when the device is reset.

6.3. Default Device identification Response

If the device is asked for a description level higher than its conformity level, it must respond in accordance with its actual conformity level.

7. EXCEPTION RESPONSE

For each frame which cannot be treated by the Modbus device must answer with an exception code in the form: `function code + 0x80` with the exception code corresponding to this error as described below.

7.1. Exceptions codes

- 01: Illegal function code
- 02: Illegal data address
- 03: Invalid data value
- 04: Unrecoverable error

7.2. Example of Exception Response

Here is an example of Exception response: Identification function not supported

Request		**Response**	
Field Name	*Value*	*Field Name*	*Value*
Address	0xAA	Address	0xAA
Function	0x2B	Function	0xAB
MEI Type	0x0E	MEI Type	0x0E
ReadDevId code	0x01	Exception code	0x01
Object Id	Obj.ID	Error Check Lo	0x68
Error Check Lo	CRC Lo	Error Check Hi	0xEC
Error Check Hi	CRC Hi		

Table 8 Modbus Serial Line PDU

- *Query:*

  Function code: Extension function code 43 (decimal) 0x2B (hex): MODBUS Generic service access
  MEI Type: 14 (0x0E): MODBUS Encapsulated Interface assigned number for Device Ident. Interface
  ReadDevId code: 01: request for basic device identification (stream access)
  02, 03 and 04: regular, extended and specific identification
  Object Id: Identification of the first object to obtain

- *Response:*

  Function code: Exception response code 171 (decimal) 0xAB (hex) (function code 0x2B + 0x80)
  MEI Type: 14 (0x0E) MEI Type assigned number for Device Identification Interface
  Exception code: 01 = illegal function
  Error Checking: 2 bytes for CRC checksum

8. REFERENCES

TI081-V2	Modbus Date-Time Implementation guide (Format & access)
TI 24 and TI102	Support of read device identification requests over Modbus
TI45	MBSL 248-Address-rev05b
TI087	Modbus Read Device ID SPECIFICATION
	Modbus Over Serial Line V1.0
TI074	Specification on Autoadaptation Modbus SL V0.9

9. APPENDICES

9.1. Frame Description

Refer to "Modbus Over Serial Line V1.0" document

The MODBUS application protocol defines a MODBUS Protocol Data Unit (PDU) that comprises two fields: the function code field and the Data field.

To initiate a MODBUS transmission, the MODBUS PDU has to be completed by the client who sends the frame. The client builds the MODBUS PDU and then adds fields to build the MODBUS Serial Line PDU.

From the PDU, two fields are added: the address field that contains the slave address (as described above in §3.1) and the CRC field that contains a 16-bit value to perform CRC error checking.

Address Field	Function code	Data	CRC
1 byte	1 byte	0 to 252 bytes	2 bytes

Table 9 Modbus Serial Line PDU

9.2. Function code Read Coils (0x01)

Refer to "Modbus Application Protocol" document

This function code is used to read from 1 to 2000 contiguous status of coils of the device. The request PDU specifies the starting address of the first coil to read and the number of coils.In the PDU coils are addressed starting at 0.

Example of a request to read discrete outputs 20 to 38:

- *Note*

  This example doesn't comprise the Modbus slave address and the CRC checking bytes which complete each frame.

Request		Response	
Field Name	*Value*	*Field Name*	*Value*
Function	0x01	Function	0x01
Starting Address Hi	0x00	Byte Count	0x03
Starting Address Lo	0x13	Outputs status 27-20	0xCD
Quantity of Reg. Hi	0x00	Outputs status 35-28	0x6B
Quantity of Reg. Lo	0x13	Outputs status 38-36	0x05

Table 10 Example of Read Coils request (0x01)

The status of outputs 27-20 is shown as the byte value 0xCD, or binary 1100 1101. Output 27 is the MSB of this byte and output 20 is the LSB.

Note that the five remaining bits after 38 are zero filled.

9.3. Function code Read Holding Register (0x03)

Refer to the "Modbus Application Protocol" document

This function code is used to read the contents of a contiguous block of holding registers of the device. The request PDU specifies the starting register address and the number of registers to read. In the PDU Registers are addressed starting at 0.

Example of a request to read register 108 – 110

- *Note*

  The table below does not comprise Address field, Modbus slave address and Error Check field

Request		Response	
Field Name	Value	Field Name	Value
Function	0x03	Function	0x03
Starting Address Hi	0x00	Byte Count	0x06
Starting Address Lo	0x6B	Reg. Hi (108)	0x02
Quantity of Reg. Hi	0x00	Reg. Lo (108)	0x2B
Quantity of Reg. Lo	0x03	Reg. Hi (109)	0x00
		Reg. Lo (109)	0x00
		Reg. Hi (110)	0x00
		Reg. Lo (110)	0x64

Table 11 Example of Read Holding Register request (0x03)

9.4. Function code Read Register (0x04)

Report to the "Modbus Application Protocol" file

This function code is used to read from 1 to 125 contiguous registers of the device. The Request PDU specifies the Starting Register and the number of registers to read. In the PDU, registers are addressed starting at 0.

The Register Data in the response message is divided in 2 bytes per register. First byte contains the High order bits and the second contains the Low order bits.

Example of a request to read register 09 that contains value 0x0A which is 10 decimal:

- *Note*

  The table below does not comprise *Address* field, *Modbus slave* address and *Error Check* field

Request		Response	
Field Name	Value	Field Name	Value
Function	0x04	Function	0x04
Starting Address Hi	0x00	Byte Count	0x02
Starting Address Lo	0x08	Reg. Hi	0x00
Quantity of Reg. Hi	0x00	Reg. Lo	0x0A
Quantity of Reg. Lo	0x01		

Table 12 Example of Read Register request (0x04)

9.5. Function code Write multiple Registers (0x10)

Refer to the "Modbus Application Protocol" document

This function code is used to write a block of contiguous registers (1 to 123 registers) in the device.

The requested written values are specified in the request data field. Data is packed as two bytes per register.

The normal response returns the function code, starting address, and quantity of registers written.

Example of a request to write registers 1 and 2:

- *Note*

  The table below does not comprise *Address* field, *Modbus slave address* and *Error Check* field

Request		Response	
Field Name	Value	Field Name	Value
Function	0x10	Function	0x10
Starting Address Hi	0x00	Starting Address Hi	0x00
Starting Address Lo	0x01	Starting Address Lo	0x01
Quantity of Reg. Hi	0x00	Quantity of Registers Hi	0x00
Quantity of Reg. Lo	0x02	Quantity of Registers Lo	0x02
Byte Count	0x04		
Registers Value. Hi	0x00		
Registers Value. Lo	0x0A		
Registers Value. Hi	0x01		
Registers Value. Lo	0x02		

Table 13 Example of Write Multiple Registers request (0x10)