Kamstrup OmniPower wm-bus metering

Release development

Team 3, E5PRO5 2020, Aarhus Uni., School of Engineering

Nov 03, 2020

TABLE OF CONTENTS

This documentation covers the system to read the Kamstrup OmniPower 1-phase meter over wm-bus using a iM871-A transceiver.

CHAPTER

ONE

IMPLEMENTATION OF IM871-A DRIVER

1.1 Generic driver class for WM-Bus USB-dongle IM871A

Platform Python 3.5.10 on Linux

Synopsis This module implements a class for communication with IM871A module.

Authors Steffen Breinbjerg, Thomas Serup

Date 21 October 2020

1.1.1 Link Modes

IM871A is able to run in different modes. Default mode is S2.

Mode	Argument	Description
S1	s1	Stationary, one way communication
S1-m	s1m	S1 with shorter header
S2	s2	Stationary, bidirectional communication
T1	t1	Frequent transmit, one way communication
T2	t2	Frequent transmit, bidirectional communication
C1, Telegram Format A	c1a	Compact, one way communication. No fixed length
C1, Telegram Format B	c1b	Compact, bidirectional communication. Fixed length
C2, Telegram Format A	c2a	Compact, one way communication. No fixed length
C2, Telegram Format B	c2b	Compact, bidirectional communication. Fixed length

class driver.DriverClass.**IM871A**(*Port*)

Implementation of a driver class for IM871A USB-dongle. Takes path to IM871A as argument, e.g. '/dev/ttyUSB1'.

close()

Close the connection to IM871A

open() → bool

Opens the port if port has been closed. It opens with the path given when instantiating the class.

ping() → bool

Ping the WM-Bus module to check if it's alive.

read_data() → bool

Read single dataframe from meters sending with the specified link mode. Function is blocking until data arrives. Send data into 'named pipe' (USBx_pipe). Removes the WM-Bus frame before sending data to pipe.

reset_module() → bool
Reset the WM-Bus module. The reset will be performed after approx. 500ms.

setup_linkmode(*mode: str*) → bool
Setup link mode for communication with meter. Takes the link mode as argument. If no Link Mode is set, default is 'S2'

CHAPTER

TWO

OMNIPOWER IMPLEMENTATION

2.1 Parse Kamstrup OmniPower wm-bus telegrams

platform Python 3.5.10 on Linux, OS X

synopsis Implements parsing functionality for C1 telegrams and log handling for data series

author Janus Bo Andersen

date 28 October 2020

2.1.1 Version history

- Ver 1.0: Set up parser and decryption. Janus.
- Ver 2.0: Implement CRC16, timezone. Janus.
- Ver 2.1: More robust exception handling, parse ELL-SN. Janus

2.1.2 Overview

- This module implements parsing for the Kamstrup OmniPower meter, single-phase.
- The meter sends wm-bus C1 (compact one-way) telegrams.
- Telegrams on wm-bus are little-endian, i.e. LSB first.
- The meter sends 1 long and 7 short telegrams, and then repeats.
- Long telegrams include data record headers (DRH) and data, that is DIF/VIF codes + data.
- Short telegrams only include data.

2.1.3 Telegram fields

In a telegram C1 telegram, the data fields are:

#	Byte#	Bytes	M-bus field	Description	Expected value (little-endian)
0	0	1	L	Telegram length	0x27 (39 bytes, short frame), or 0x2D (45 bytes, long frame)
1	1	1	C	Control field (type and purpose of message)	0x44 (SND_NR)
2	2-3	2	M	Manufacturer ID (official ID code)	0x2D2C (KAM)
3	4-7	4	A	Address (meter serial number)	0x57686632 (big-endian:32666857)
4	8	1	Ver.	Version number of the wm-bus firmware	0x30
5	9	1	Medium	Type / medium of meter	0x02 (Electricity)
6	10	1	CI	Control Information	0x8D (Extended Link Layer 2)
7	11	1	CC	Communication Control	0x20 (Slow response sync.)
8	12	1	ACC	Access field	Varies
9	13-16	4	AES CTR / SN	AES counter (Session number)	Varies, see below
10	17-39 17-45	23 29	Data	Contains AES-encrypted data frame, varying for short and long frames	Encrypted data
11		2	CRC16	CRC16 check	

The fields 0-9 of the telegram can be unpacked using the little-endian format <*BBHIBBBBB*, where

- < marks little-endian,
- *B* is an unsigned 1 byte (char),
- *H* is an unsigned 2 byte (short),
- *I* is an unsigned 4 byte (int)

2.1.4 AES counter / Session number

The AES_CTR / Extended Link Layer SN field (ELL-SN) is is structured as per EN 13757-4, p. 54.

The example shows ELL-SN value of 0x01870320 (little-endian) -> 0x20038701 (big-endian). Bit readout:

Byte:	3	2	1	0
Hex:	0x20	0x03	0x87	0x01
Binary:	0010 0000	0000 0011	1000 0111	0000 0001

Should give the following slicing and interpretation:

Bits:	31-29	28-04	03-00
Field:	ENC	Time	Session
Example:	001	0 0000 0000 0011 1000 0111 0000	0001
Hex:	0x1	0x3870 (14448)	0x1

- ENC (Encryption): 0 -> No encryption, 1 -> AES-CTR mode, higher -> reserved.
- Time: Minute counter since 01/01/2013 (?), or since meter started, requires RTC set on meter. So time measurement was taken about 10 days after the meter was started.
- Session: Incremented by meter for each transmission, unless using partial/fractured frames.

Parsing. The whole ELL-SN field is read out and masks are used to extract fields.

2.1.5 Telegram examples

Encrypted short telegrams:

L	C	M	A	Ver	Med	CI	CC	ACC	AES CTR	Encrypted payload	CRC 16
27	44	2D 2C	5768 6632	30	02	8D	20	2E	2187 0320	D3A4F149 B1B8F578 3DF7434B 8A66A557 86499ABE 7BAB59	xxxx
27	44	2d 2c	5768 6632	30	02	8d	20	63	60dd 0320	c42b87f4 6fc048d4 2498b44b 5e34f083 e93e6af1 617631	3d9c
27	44	2d 2c	5768 6632	30	02	8d	20	8e	11de 0320	188851bd c4b72dd3 c2954a34 1be369e9 089b4eb3 858169	494e

Encrypted long telegrams:

L	C	M	A	Ver	Med	CI	CC	ACC	AES CTR	Encrypted payload	CRC 16
2D	44	2D 2C	5768 6632	30	02	8D	20	64	61DD 0320	38931d14 b405536e 0250592f 8b908138 d58602ec a676ff79 e0caf0b1 4d	0e7d

2.1.6 Decryption

- The wireless m-bus on OmniPower uses AES-128 Mode CTR (if enabled, otherwise no encryption).
- See EN 13757-4:2019, p. 54, as ELL (Ext. Link-Layer) with ENC = 0x1 => AES-CTR.
- A decryption prefix (initial counter block) is built from some of the fields.
- See table 54 on p. 55 of EN 13757-4:2019.

It can be packed using the format *<HIBBBIB*.

M	A	Ver	Med	CC	AES_CTR	FN	BC
2D2C	57686632	30	02	20	21870320	0000	00

- AES Prefix (initialization vector): Fields M, . . . , AES_CTR, FN.
- FN: frame number (frame # sent by meter within same session number, in case of multi-frame transmissions).
- AES Counter: BC
- BC: Block counter (encryption block number, counts up for each 16 byte block decrypted within the telegram).

2.1.7 Decrypted payload examples

The interpretation of the fields in the OmniPower is

Field	Kamstrup name	Data fmt (DIF)	Value type (VIF/E)	VIF/E meaning	DIF VIF/E
Data 1	A+	32-bit uint	Energy, 10^1 Wh	Consumption from grid, accum.	04 04
Data 2	A-	32-bit uint	Energy, 10^1 Wh	Production to grid, accum.	04 84 3C
Data 3	P+	32-bit uint	Power, 10^0 W	Consumption from grid, instantan.	04 2B
Data 4	P-	32-bit uint	Power, 10^0 W	Production to grid, instantan.	04 AB 3C

Transport layer control information fields (TPL-CI), ref. EN 13757-7:2018, p. 17, introduce Application Layer (APL) as:

- 0x78 with full frames (Response from device, full M-Bus frame)
- 0x79 with compact frames (Response from device, M-Bus compact frame)

Data integrity check (CRC16)

The first 2 bytes (16 bits) of a payload is always the CRC16 value of the sent message. This value must be checked versus CRC16 calculated on the received payload.

Decrypted short telegram payload

CRC16	TPL-CI	Data fmt. sign.	CRC16 data	Data 1	Data 2	Data 3	Data 4
1170	79	138C	4491	CE000000	00000000	03000000	00000000

Measurement data starts at byte 7, and can easily be extracted using *<IIII* little-endian format.

In this example, 206 10^1 Wh (2.06 kWh) have been consumed, and the current power draw is 3 10^0 W (0.003 kW).

Decrypted long telegram payload

In this kind of telegram, the DRHs are included.

CRC16	TPL-CI	DIF/VIF 1	Data 1	DIF/VIF/VIFE 2	Data 2	DIF/VIF 3	Data 3	DIF/VIF 4	Data 4
9831	78	04 04	D7000000	04 84 3C	00000000	04 2B	03000000	04 AB 3C	00000000

Extraction is slightly more complex, requiring either a longer parsing pattern or perhaps a regex.

In this example, 215 10^1 Wh (2.15 kWh) have been consumed, and the current power draw is 3 10^0 W (0.003 kW).

2.2 The C1 Telegram class

class meter.OmniPower.**C1Telegram**(*telegram: bytes*)

Implements capture of data fields for a C1 telegram from OmniPower

decrypt_using(*meter:* meter.OmniPower.OmniPower) → bool

Decrypts a telegram and inserts it into telegram (self) Uses a meter object and its key to perform decryption. Requires instantiated OmniPower meter with valid AES-key.

2.3 The OmniPower class

class meter.OmniPower.**OmniPower**(*name: str = 'Kamstrup OmniPower one-phase', meter_id: str = '32666857', manufacturer_id: str = '2C2D', medium: str = '02', version: str = '30', aes_key: str = '9A25139E3244CC2E391A8EF6B915B697'*)

Implementation of our OmniPower single-phase meter Passed values are hex encoded as string, e.g. '2C2D' for value 0x2C2D.

add_measurement_to_log(*measurement:* meter.MeterMeasurement.MeterMeasurement) → bool

Pushes a new measurement to the tail end of the log

decrypt(*telegram:* meter.OmniPower.C1Telegram) → bytes

Decrypt a telegram. Returns decrypted bytes. Raises CrcCheckException if CRCs do not match after decryption.

Requires:

- the prefix from the telegram (telegram.prefix), and
- the encryption key stored in the meter object.

Decrypts the data stored in field telegram.encrypted

dump_log_to_json() → str

Returns a JSON string of all measurement frames in log, with an incremented number for each observation.

extract_measurement_frame(*telegram:* meter.OmniPower.C1Telegram) → *meter.MeterMeasurement.MeterMeasurement*

Requires that the telegram is already decrypted, otherwise returns empty measurement

is_this_my(*telegram:* meter.OmniPower.C1Telegram) → bool

Check whether a given telegram is from this meter by comparing meter setting to telegram

process_telegram(*telegram:* meter.OmniPower.C1Telegram) → bool

Does entire processing chain for a telegram, including adding to log. Returns True if processing is OK and added to log OK. Otherwise False.

classmethod unpack_long_telegram_data(*data: bytes*) → Tuple[int, ...]

Long C1 telegrams contain DIF/VIF information and field data values

classmethod unpack_short_telegram_data(*data: bytes*) → Tuple[int, ...]

Short C1 telegrams only contain field data values, no information about DIF/VIF

2.4 Exception classes

class meter.OmniPower.**TelegramParseException**(*exception_message: str*)
Use this to raise an exception if a bytestream telegram fails to parse into C1 format.

class meter.OmniPower.**AesKeyException**(*exception_message: str*)
Use this to raise an exception when an AES key is missing or wrong length.

CHAPTER

THREE

IMPLEMENTATION OF GENERIC MEASUREMENTS

3.1 Generic class for measurements and measurement frames

platform Python 3.5.10 on Linux, OS X

synopsis This module implements classes for generic measurements taken from a meter.

authors Janus Bo Andersen, Jakob Aaboe Vestergaard

date 13 October 2020

3.2 The Measurement class

class meter.MeterMeasurement.**Measurement**(*value: float*, *unit: str*)

Single physical measurement. A single measurement of a physical quantity pair, consisting of a value and a unit.

3.3 The MeterMeasurement class

class meter.MeterMeasurement.**MeterMeasurement**(*meter_id: str*, *timestamp: datetime.datetime*)

A single measurement collection based on one frame from the meter. Will contain multiple measurements of physical quantities taken at the same time.

add_measurement(*name: str*, *measurement:* meter.MeterMeasurement.Measurement) → None

Store a new measurement in the collection.

as_dict() → dict

Serializes and outputs the Measurement frame as a structured dict.

json_dump() → str

Returns a JSON formatted string of all data in frame.

CHAPTER

FOUR

UTILS: CRC16 FOR WM-BUS

4.1 CRC16 for wm-bus

synopsis CRC16 calculator for EN 13757

author Janus Bo Andersen

date October 2020

4.1.1 Overview:

- This function performs the CRC16 algorithm.
- The result can be used to confirm data integrity of received payload in a wm-bus message.
- Wm-bus follows the CRC16 standard outlined in EN 13757.

A *CrcCheckException* class is also implemented, which is used to raise exceptions if a CRC check fails.

The IM871-A transceiver removes the outer CRC16 (last two bytes of a message) and replaces it with its own, which follows another standard, CRC16-CCITT. So the outer CRC16 can not be checked with the function in this module.

4.1.2 CRC16 EN 13757:

- CRC16 uses a generator polynomial, g(x), described in EN 13757-4.
- See p. 42 for data-link layer CRC, and an example with a C1 telegram on p. 84.
- See p. 58 for transport layer CRC polynomial.

g(x) = x^16 + x^13 + x^12 + x^11 + x^10 + x^8 + x^6 + x^5 + x^2 + 1

In binary (excluding x^16 as it is shifted out anyway), this g(x) is represented as

Byte 1	Byte 2	Hex value
0011 1101	0110 0101	0x3D65
MSbit = x^15	LSbit = x^0 = 1	

See EN 13757-4, table 43, p. 50 for expected structure of ELL for a CI=0x8D telegram. PayloadCRC is included in the encrypted part of telegram.

4.1.3 Algorithm rules:

- Treats data most-significant bit first
- Final CRC shall be complemented
- Multi-byte data is transmitted LSB first
- CRC is transmitted MSB first

4.1.4 Math background:

- CRC uses a finite field F=[0, 1], so we do subtraction using XOR.
- CRC is the final remainder from repeated long division of message by polynomial,when no further division is possible.
- The output CRC is complemented by XOR with 0xFFFF.

4.1.5 Algorithm implementation comments:

The implemented algorithm uses Python's ability for 'infinite' width of integers. That is slightly inefficient, and can't be ported to C code on an embedded device. But it is significantly easier to debug and understand than byte-wise algorithms or lookup tables.

4.2 CRC16 functions

utils.crc16_wmbus.**crc16_wmbus**(*message: bytes*) → bytes

Takes a bytes object with a message (ascii encoded hex values). Returns the CRC16 value for the message encoded in a bytes object.

Example: f(b'79138C4491CE00000000000000300000000000000') -> b'1170'.

utils.crc16_wmbus.**crc16_check**(*payload: bytes*) → bool

Takes a payload and splits into CRC16-field and message. Computes CRC16 on the message and compares to CRC16-field. Return True if match. Raises CrcCheckException if no match.

4.3 CRC check exception

class utils.crc16_wmbus.**CrcCheckException**(*crc_recv: bytes*, *crc_calc: bytes*, *exception_message: str*)

Use this to raise an exception when a CRC16 check has failed.

CHAPTER

FIVE

UTILS: CRC16 FOR IM871-A

5.1 Implementation of CRC16 for IM871-A (CCITT)

synopsis CRC16 CCITT implementation based on Steffen's C implementation.

authors Steffen and Janus.

date 29 Oct 2020.

- See IMST's WMBUS_HCL_Spec_V1.6.pdf.
- CRC computation starts from the Control Field and ends with the last octet of the Payload Field.
- IM871A uses CRC16-CCITT Polynomial G(x) = 1 + x^5 + x^12 + x^16.

5.2 CRC16 function

utils.crc16_im871a.**crc16_im871a_check**(*m: bytes*) → bool

Confirm CRC16 integrity of a full bytestring received from IM871-A. Argument must be the entire message from IM871-A. Function returns TRUE when the check sum matches the expected CRC16 value.

utils.crc16_im871a.**crc16_im871a_calc**(*m: bytes*) → bytes

Compute CRC16 (CCITT) for a message received serially from IM871-A. Argument must:

- NOT contain the first field, e.g. 0xA5.
- Start and contain the control field, e.g. 0x8203.
- Contain full payload, e.g. 0x2744. . . 2637.
- NOT contain the trailing 2 bytes of expected CRC16 value, e.g. 0xC1AB.

Full message example: a5 8203 27442d2c5768663230028d20cd12340720519df247ff65e751662a300bc4e5c67da86477f0182637 c1ab.

CHAPTER

SIX

UTILS: TIMEZONE HANDLING

6.1 Zulu timezone handling

synopsis Stamp measurements with Zulu time (UTC), and easy future-proof management of timestamps as required by ReMoni.

author Janus Bo Andersen.

date October 2020.

description Output dates in ISO 8601 format, i.e. output 2020-10-25T10:08:00Z for 25th October 2020 at 10:08:00 (HH:MM:SS) in UTC time.

Note that ISO 8601 allows using +00:00 instead of Z. ReMoni prefers Z for this implementation.

- UTC class implementation based on: https://docs.python.org/3.5/library/datetime.html
- Zulu time definition based on: https://en.wikipedia.org/wiki/ISO_8601

6.2 ZuluTime class

class utils.timezone.**ZuluTime**

Very explicit Zulu-time class to implement Zulu time as UTC time. Implements required methods on abstract base class *tzinfo*.

dst(*dt*)

No Daylight savings time.

tzname(*dt*)

Name of timezone.

utcoffset(*dt*)

UTC is zero time offset from UTC, of course.

6.3 Print datetime object with ISO 8601 format

utils.timezone.**zulu_time_str**(*timestamp: datetime.datetime*) → str
Print a timestamp with ISO format like 2020-10-25T10:08:00Z

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

u

INDEX

Z