

I. MODBUS PROTOCOL AND ITS IMPLEMENTATION IN SVAN 95X DEVICES

I.1 Introduction

Modbus is an application layer communication protocol located in layer 7 of OSI model. It allows an efficient communication between client and server, regardless of bus types or networking.

Used since 1979, Modbus allowed communication of millions of automated devices and its simplified and elegant structure contributed greatly to its renown.

In a nutshell, Modbus is a communications protocol based on request-response system by transmitting standardized messages that are operating on predefined data structures.

I.2 Modbus protocol

As it was mentioned in the Introduction, Modbus protocol is based on request-response system. In typical configuration of the protocol, one master device sends request messages in defined structure to a slave device, which reacts with properly designed response message.

Request message consists of two fields – function code and data:

Function code (1 byte)	Data (n bytes)
------------------------	----------------

- Function code is a singly byte ranging from 0x01 to 0x7F (1-127 in decimal). Those functions are defined by the protocol and slave device responds to them in a manner also defined by the protocol. Function codes from 0x01 to 0x40 (1-64 in decimal), from 0x49 to 0x63 (73-99 in decimal) and from 0x6F to 0x7F (111-127 in decimal) are public functions, commonly known Modbus functions (the protocol recognizes only a few dozen of them, the rest of the numbers are reserved for future development). The remainder, as in ranges from 0x41 to 0x48 (65-72 in decimal) and from 0x64 to 0x6E (100-110 in decimal) are reserved as private functions, which may be used for personal customization.
- Data contains a certain amount of data that hold information required for proper operating of the function codified in the first field. It can be addresses or data bytes to write. If the function operates on an undefined amount of data bytes, a single bytes contains information about their amount.

Maximum message length – 253 bytes = 1 byte of function code + 252 data bytes.

Upon receiving a request message, the slave device may respond in one of two ways:

- Sending a Response
- Sending an Exception Response

Response is sent when processing the request has conducted without errors – both the function code and data configuring it are correct. The structure of a Response message is as follows:

Function code (1 byte)	Data (n bytes)
------------------------	----------------

- Function code – single byte – an echo of the request function code.
- Data – data of the Response – a number of bytes generated by the slave device as a response to the request. It can be values (if, for instance, the Request demanded access to certain data structures), or echoing part or entire Request message as an acknowledgement of a successful operation (if, for instance, the Request aimed to write values in data structure). If a Response contains a number of bytes unspecified by the function protocol, one of them usually contains an

information on how many of those bytes will be in the message.

Exception Response is generated by the slave device when the processing of a Request message has concluded with an error – the cause might be incomplete or incorrect data, such as out of range addresses. The structure of an Exception Response is as follows:

Exception function code (1 byte)	Exception code (1 byte)
----------------------------------	-------------------------

- Exception function code – single byte – echoed Request function code + 0x80 (ie. for function 0x01 has exception function code equaling 0x81).
- Exception code – single byte – code that describes the reason of occurring exception.

Modbus protocol uses “Big-endian” coding, meaning that when numeric value exceeds one byte (forming a word message), Most Significant Byte is sent first.

Data structure used by Modbus protocol is as follows:

Name	Size	Access
Discrete Input	1 bit	Read-only for application, write by device
Coil	1 bit	Read/Write by application
Input Register	16-bit word	Read-only for application, write by device
Holding Register	16-bit word	Read/Write by application

For each of the four data types, the protocol supports 65536 units (so that maximum address can be written on two bytes). Addressing for those units ranges from 0 to 65535.

Generally, the data structures can be perceived as a certain input buffer (Holding Registers) with a set of switches (Coils) and a certain output buffer (Input Registers) with a set of switches (Discretes Inputs). The “switches”, addressed accordingly to reflect the registers would, in such perception, denote whether the registers hold valid information or not.

There are six basic functions of Modbus protocol, allowing effective exchange of data between the structures.

- Read Coils (function code: 0x01) – reads a certain amount of Coils from a slave device's data structure. .
- Read Discrete Inputs (function code: 0x02) – reads a certain amount of Discrete Inputs from a slave device's data structure.
- Read Holding Registers (function code: 0x03) – reads a certain amount of Holding Registers from a slave device's data structure.
- Read Input Registers (function code: 0x04) – reads a certain amount of Input Registers from a slave device's data structure.
- Write Single Coil (function code: 0x05) – writes a single Coil into a slave device's data structure.
- Write Single Register (function code: 0x06) – writes a single Holding Register into a slave device's data structure.

Additionally, it is often to use two supporting functions:

- Write Multiple Coils (function code: 0x0F) – writes a certain number of Coils into a slave device's data structure.
- Write Multiple Registers (function code: 0x10) – writes a certain number of Holding Registers into a slave device's data structure.

I.3 Modbus over serial line.

Modbus protocol used in serial transmission is bound to a few requirements:

- Maximum length of a message is 256 bytes – protocol's maximum length is completed by additional three bytes.
- Each slave device has to have a unique address, which describes it as a unique device in Modbus network. Such address is denoted as a single byte and ranges from 1 to 247 in decimal values.

Modbus message structure in serial transmission is slightly modified. To a standard Protocol Unit part (called PU) are added three more bytes, making the whole structure appear as follows:

Device address (1 byte)	Function code (1 byte)	Data (n bytes)	CRC/LRC (2 bytes)
-------------------------	------------------------	----------------	-------------------

- Device address – a single byte denoting the device to which the message is being send to. 0 denotes broadcast (where the message is accepted by all connected devices), 1-247 denote individual devices, 248-255 are reserved.
- Function code – single byte denoting a Modbus function.
- Data – a certain amount of data bytes containing function parameters.
- CRC/LRC – 2 bytes of error check. Depend on used transmission mode.

In SVAN 95x devices, Modbus is used with RTU (Remote Terminal Unit) transmission mode. Its structure is as follows:

Message structure:

Address	Function	Data	CRC
8 bits	8 bits	N x 8 bits	2 x 8 bits

In this mode, each message byte is being transmitted as two 4-bit hexadecimal characters, offering greater density. The error check bytes are generated using CRC algorithm (Cyclical Redundancy Checking). In this algorithm, the sender calculates the values of these two bytes and sends them along with the original message. The receiver calculates his own values based on the original message and compares with the values from the sender. If those two pair of values are not the same, an error occurs.

I.4 Implementation of Modbus protocol in SVAN 95x devices

The implementation supports only RTU transmission mode. It should be noted that SVAN devices require that DTR and RTS Controls are enabled in serial communication.

At the moment, implemented Modbus protocol supports 1024 units of the four data types described in earlier chapters of this document.

SVAN 95x devices offer a simple system of configuring Modbus protocol's operating. Since the protocol is one of the device's communication modes, it first requires to be activated. It can be done by accessing NETWORK tab (path: MENU/ SETUP / WIRELESS TRANSFER / NETWORK) and selecting MODBUS option. While active, the device will process the protocol's commands and will offer a new set of options from . It contains the following options:

- **MODBUS ADDR.** option – device's Modbus address, ranging from 1 to 247.
- **MODBUS # CMD.** – selecting this option, enables processing of Svantek # functions via Modbus protocol. Upon entering a # command into Modbus structures, it will be processed and any results will be placed in Input Registers (and Discrete Inputs) structure. **It is not recommended to use this option alongside with MB-H.REG.MAP option.**
- **MB-H.REG. MAP** – selecting this option enables result mapping. During measurement,

ongoing results will be placed in Holding Registers starting from appointed address. **It is not recommended to use this option alongside with MODBUS # CMD. Option.**

- **MB-MAP INDEX** – denotes address from which result map will be placed during measurements.

All Modbus structures will be saved into FLASH memory upon modification.

General algorithm of processing SVAN 945a commands using Modbus protocol can be found below. This option is not recommended to use alongside with Modbus result mapping.

- The sender writes bytes of SVAN command using functions (0x06 - Write Single Register or 0x10 – Write Multiple Registers) writing into Holding Registers.
- The sender writes Coils using proper functions (0x5 – Write Single Coil or 0x0F – Write Multiple Coils) in addresses resembling previously used Holding Registers, setting them to “ON” (1) status.
- Each time a Coil is written, the device checks the contents of Holding Registers which addresses resemble those of consecutively written Coils. If there's a number of consecutive Coils in the data structure that are set to 1 and Holding Registers of the same addresses contain a string beginning with '#' and ending with ';', the string is processed as SVAN command.
- The command is processed and the results are placed in Input Registers (starting at 0 address).
- Once the results are written in Input Registers, Discrete Inputs are set to 1s at addresses resembling the addresses of the previously written Input Registers. An Input succeeding the last Input written in previous step is set to 0, thus defining the addresses where the results are being held.
- The effect is putting results in Input Registers (as an output buffer) and setting Discrete Inputs of the same addresses to 1 (as a set of switches denoting which registers of the output buffer contain the response).
- The sender may now read Discrete Inputs (0x02 – Read Discrete Inputs) to determine in which addresses Input Registers hold the response and then read Input Registers (0x04 – Read Input Registers) to retrieve the result.
- Some Svan commands (such as #4 – files) return response of length greater than the capacity of the Input Register data structure. In such situation, the device will place as many bytes of response in Input Registers as possible without interfering with the response structure. Once these bytes are read by the server, the device will proceed to place the next portion of the command response in the Input Registers for further reading, repeating the process until all response bytes have been successfully transferred. If the server won't read all available data bytes within 5 seconds of placing them in the registers, the procedure will terminate with Input Registers containing the last placed portion of the command response.

Example 1:

We wish to check if the measurement is in progress, using #1 command.

The string of the command is: “#1,S?;”

- We write six bytes of the string, beginning from address 0 (see Write Multiple Registers function in Chapter 5) into Holding Registers of the slave device. Registers of addresses from 0 to 2 are written (each register holds two bytes).
- We write three Coils (one for each Holding Register) from address 0, setting them to “ON” – 1 (see Write Multiple Coils function in Chapter 5).
- The device notices that Coils have been written and searches Coils data structure as long as either the structure addresses end or it finds a set of consecutive Coils (all set to “ON” - 1), which addresses denote Holding Registers that hold a string of characters beginning with '#' and ending with ';'. In this case, it finds “#1,S?;” string written in the first step.
- The command is processed and, assuming that the measurement is not in progress, the result

- (string "#1,S0;") is placed in Input Registers beginning at address 0. Input Registers addressed from 0 to 2 now hold the response.
- Discrete Inputs are set to 1s to denote the Input Registers that hold the response. Discrete Inputs addressed from 0 to 2 are set to 1, while at address 3 it is set to 0.
 - We can now read Discrete Inputs of the slave device (see Read Discrete Inputs function in Chapter 5). Beginning from 0 address, finding a set of consecutive Discrete Inputs being set to 1 denotes how many Input Registers we need to read to get the result. In this instance, reading 8 Discrete Inputs (1 byte) will give us a response of 11100000 as values of consecutive Discrete Inputs beginning at address 0 and ending at address 7.
 - Based on information we obtained from previous step, we can now read three Input Registers starting at 0 address and the bytes that are returned in Response are the result string "#1,S0;".

Example 2:

We wish to retrieve a file named "01JAN" that total length is 2500 bytes.

- Following the procedure similar as presented in Example 1, we transfer command "#4,1,01JAN;" to the device using Modbus protocol.
- The device places 2048 bytes in Input Registers, filling addresses from 1 to 1024 in Input Register and Inputs data structures.
- We read data from Inputs to determine which registers to read (1 to 1024), then proceed to read all available bytes from Input Registers.
- The device places the remaining 452 bytes in Input Registers, filling addresses from 1 to 226 in Input Register and Inputs data structures.
- We read data from Inputs to determine which registers to read (1 to 452), then proceed to read the available bytes from Input Registers.

Should we not read the first 2048 bytes of response within 5 seconds since placing them in the registers by the device, the device would simply terminate the procedure, leaving this part of the command response in the registers.

I.5 Modbus functions recognized by SVAN 95x devices.

Modbus protocol in SVAN devices recognizes basic commands of the protocol - enough ensure successful communication. Their description can be found below:

0x01 – Read Coils

Reads up to 2000 consecutive Coils. Results are transmitted as bytes (each bytes hold 8 consecutive Coils, beginning with least significant bit).

Request message structure:

Function	Start address	Amount of Coils
1 byte	2 bytes	2 bytes
0x01	0x0000 - 0xFFFF	1 - 2000 (0x7D0)

Response message structure:

Function	Byte amount	Coils' data
1 bajt	1 byte	N bytes
0x01	N	values

N = amount of units/8; if division results in a rest, N = N+1
Excessive bits are filled with zeros

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x81	0x01 or 0x02 or 0x03 or 0x04

- 0x02 – Read Discrete Inputs

Reads up to 2000 consecutive Discrete Inputs. Results are sent as bytes (each byte holds 8 consecutive Discrete Inputs, beginning with least significant bit).

Request structure:

Function	Starting address	Amount of inputs
1 byte	2 byte	2 byte
0x02	0x0000 - 0xFFFF	1 - 2000 (0x07D0)

Response structure:

Function	Byte amount	Inputs' data
1 byte	1 byte	N byte
0x02	N	values

N = amount of units/8; if division results in a rest, N = N+1
Excessive bits are filled with zeros

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x82	0x01 or 0x02 or 0x03 or 0x04

- 0x03 – Read Holding Registers

Reads up to 125 consecutive Holding Registers. Results are sent as bytes – each register is denoted by two bytes according to Big-Endian coding.

Request structure:

Function	Starting address	Amount of registers
1 byte	2 bytes	2 bytes
0x03	0x0000 - 0xFFFF	1 - 125 (0x007D)

Response structure:

Function	Byte amount	Registers' data
1 byte	1 byte	N bytes

Function	Byte amount	Registers' data
0x03	N	bvalues

N = 2*register amount.

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x83	0x01 or 0x02 or 0x03 or 0x04

– 0x04 – Read Input Registers

Reads up to 125 consecutive Input Registers. Results are sent as bytes – each register is denoted by two bytes – transmitted in Big-Endian coding.

Request structure:

Function	Starting address	Amount of registers
1 byte	2 bytes	2 bytes
0x04	0x0000 - 0xFFFF	1 - 125 (0x007D)

Response structure:

Function	Byte amount	Registers' data
1 byte	1 byte	N bytes
0x04	N	bvalues

N = 2*register amount.

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x84	0x01 or 0x02 or 0x03 or 0x04

– 0x05 – Write Single Coil

Writes a single Coil. Value to be written is a 16-bit word. Word FF 00 (hex) denotes 1, while 00 00 (hex) is 0. Response is simply an echo of the original request message.

Request structure:

Function	Coil address	Value to write
1 byte	2 bytes	2 bytes
0x05	0x0000 - 0xFFFF	0x0000 (0) or 0xFF00 (1)

Response structure (echo of the request message):

Function	Address	Value to write
1 byte	2 bytes	2 bytes

Function	Address	Value to write
0x05	0x0000 - 0xFFFF	0x0000 (0) or 0xFF00 (1)

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x85	0x01 or 0x02 or 0x03 or 0x04

– 0x06 – Write Single Register

Writes a single Holding Register. Value to write is defined as a 16-bit word – consecutive bytes of the register in Big-Endian coding. Response is simply an echo of the request.

Request structure:

Function	Register address	Value to write
1 byte	2 bytes	2 bytes
0x06	0x0000 - 0xFFFF	0x0000 - 0xFFFF

Response structure (echo of the request message):

Function	Register address	Value to write
1 byte	2 bytes	2 bytes
0x06	0x0000 - 0xFFFF	0x0000 - 0xFFFF

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x86	0x01 or 0x02 or 0x03 or 0x04

– 0x0F – Write Multiple Coils

Writes up to 2000 consecutive Coils. Values to be written are bytes containing 8 Coil values, beginning with least significant bit. The device responds with partial echo of the Request.

Request structure:

Function	Starting address	Amount of Coils	Byte count	Values
1 byte	2 bytes	2 bytes	1 byte	N bytes
0x0F	0x0000 - 0xFFFF	0x0000 - 0x07B0	N	values

N = amount of units/8; if division results in a rest, N = N+1
Excessive bits are filled with zeros

Response structure:

Function	Starting address	Amount of Coils
1 byte	2 bytes	2 bytes
0x0F	0x0000 - 0xFFFF	0x0000 - 0x07B0

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x8F	0x01 or 0x02 or 0x03 or 0x04

– 0x10 – Write Multiple Registers

Writes up to 123 Holding Registers. The values are resembled by 16-bit words in Big-Endian coding. The device responds with partial echo of the Request.

Request structure:

Function	Starting address	Amount of Registers	Byte count	Values
1 byte	2 bytes	2 bytes	1 byte	N bytes
0x10	0x0000 - 0xFFFF	0x0000 - 0x007B	N	values

$$N = 2 * \text{amount of registers}$$

Response structure:

Function	Starting address	Amount of Registers
1 byte	2 bytes	2 bytes
0x10	0x0000 - 0xFFFF	0x0000 - 0x007B

Exception Response structure:

Function	Exception code
1 byte	1 byte
0x90	0x01 or 0x02 or 0x03 or 0x04

10. 0x46 – SVAN Configure Modbus Settings

The function permits changing Modbus protocol settings in SVANTEK device. It consists of sub-function code and parameter data. Response simply echoes the Request.

Request structure:

Function	Sub-function	Data
1 byte	2 bytes	2 bytes
0x08	0x0000 to 0x0003	0x0000 to 0xFFFF

Response structure:

Function	Sub-function	Data
1 byte	2 bytes	2 bytes
0x08	0x0000 to 0x0003	0x0000 to 0xFFFF

Exception Response structure:

Function	Exception code
1 byte	1 byte
0xC6	0x01 or 0x02 or 0x03 or 0x04

Input data depends on sub-function used. The following sub-functions are available within this function:

1. 0x00 – Set Modbus Mode

Changes Modbus Mode setting.

Sub-function	Input data
1 byte	2 bytes
0x00	0x0000 active, 0xFF00 inactive

2. 0x01 – Set # Command Processing

Changes whether SVAN # commands are to be processed by inputting Modbus data or not.

Sub-function	Input data
1 byte	2 bytes
0x01	0x0000 active, 0xFF00 inactive

3. 0x02 – Use Result Mapping

Toggles use of result mapping.

Sub-function	Input data
1 byte	2 bytes
0x02	0x0000 active, 0xFF00 inactive

4. 0x03 – Mapping Index

Changes index of Holding Register at which results are being written during result mapping.

Sub-function	Input data
1 byte	2 bytes
0x03	0x0000 active, 0xFF00 inactive

I.6 Result mapping mode

Using menu options, it is possible to set the device into result mapping mode via Modbus protocol. Selecting MB-H.REG.MAP option in MODBUS MODE section, enables this mode, in which the device places results from the ongoing measurement within Holding Registers, starting from address specified by MB-MAP INDEX option. This operation is conducted at the end of each measurement cycle. **It is not recommended to use this option alongside with processing # commands option.** The results are placed in accordance to the following table:

Main Results

Cell index	Result	Cell index	Result
x+0, x+1	Measure Time	x+26, x+27	Zero
x+2	Peak in Profile 1	x+28	Peak in Profile 3
x+3	P-P in Profile 1	x+29	P-P in Profile 3
x+4	MAX in Profile 1	x+30	MAX in Profile 3
x+5	MIN in Profile 1	x+31	MIN in Profile 3
x+6	SPL in Profile 1	x+32	SPL in Profile 3
x+7	RMS in Profile 1	x+33	RMS in Profile 3
x+8	Lden in Profile 1	x+34	Lden in Profile 3
x+9	Ltm3 in Profile 1	x+35	Ltm3 in Profile 3
x+10	Ltm5 in Profile 1	x+36	Ltm5 in Profile 3
x+11	Lav in Profile 1	x+37	Lav in Profile 3
x+12	TLav in Profile 1	x+38	TLav in Profile 3
x+13, x+14	Overload Time		
x+15	Peak in Profile 2		
x+16	P-P in Profile 2		
x+17	MAX in Profile 2		
x+18	MIN in Profile 2		
x+19	SPL in Profile 2		
x+20	RMS in Profile 2		
x+21	Lden in Profile 2		
x+22	Ltm3 in Profile 2		
x+23	Ltm5 in Profile 2		
x+24	Lav in Profile 2		
x+25	TLav in Profile 2		

x denotes address in MB-MAP INDEX option. All results are placed in registers using Big Endian convention – most significant byte is placed on first index. Results are in dB with last digit being fraction part of the measurement result.

I.7 Modbus protocol exception codes

Code	Name	Brief description
0x01	ILLEGAL FUNCTION	Incorrect function, unrecognized by the device. Error occurs when Request contains a function not listed in this document.
0x02	ILLEGAL DATA ADDRESS	Incorrect data structure address – in an event when any address used in Request message exceeds declared addressing bounds.
0x03	ILLEGAL DATA VALUE	Incorrect input data.
0x04	SLAVE DEVICE FAILURE	Device error – unrecognized critical error.
0x05	ACKNOWLEDGE	Not supported. The Request has been accepted, but for some reason processing it will take more time than usual.
0x06	SLAVE DEVICE BUSY	Not supported. The device is busy.
0x08	MEMORY PARITY ERROR	Not supported. The device attempted to access memory part which has not passed parity check.
0x0A	GATEWAY PATH UNAVAILABLE	Not supported. The gate was unable to determine communication path in order to transmit the message.
0x0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Not supported. The gate did not receive a response from the target device.

I.8 Reference

- <http://www.modbus-IDA.org> – Modbus developer's site
- http://www.modbus-ida.org/docs/Modbus_Application_Protocol_V1_1a.pdf – Modbus protocol specification
- http://www.modbus-ida.org/docs/Modbus_over_serial_line_V1_01.pdf – Modbus over serial line specification
- <http://www.modbus-ida.org/tech.php> – links regarding Modbus implementations