Description

The GEKO protocol for communication with heat pump devices and solar panels is not a standard Modbus RTU protocol, but some functionalities are similar. Communication is via RS485, it seems that the communication parameters are 8N1 (at least in the case of a heat pump), but it may also depend on the production date, it is easy to find these parameters when connecting the heat pump with the display into the communication circuit, because in this mode, you don't need to send any messages (you can't even).

The interrogated data should start with bytes 0x69 followed by a byte 0x02 and then 0x01 (or in the reverse order 0x01 0x02). If such a sequence is found, the parameters are most likely correct.

The heat pump for communication with the display uses the baud rate 38400, but this may also depend on the version of the device. In the pump options it is possible to set RS485 parameters, it is a second RS port used for communication with EkoLan. Here the possibilities are greater, because we do not eavesdrop on the transmission, but send a request to the pump and it responds. I also propose to set the speed of 38400, the same as on the display so that it is the same. Additionally, I recommend changing the default parameters of the logical (65535) and physical (255) addresses to something smaller, because I haven't tested how to send such high addresses. I set the logical address to 6 and the physical address to 5.

The protocol frame format consists of 2 parts, they will be described below.

Data types

1. int8 - a normal positive byte, often represented as hex.
2. int16 - 2 bytes of the message, which is a positive value. First, the value of the lower byte is recorded in the protocol, i.e. the value 0x01 is transmitted like this: 0100
3. float16signed - 2 bytes of the message, which may have negative values. Record similar to int16, i.e. the youngest first byte. If the highest bit of the second byte is set (0x80) then the value is negative and it should be calculated: w = w - 0x10000, where w are both bytes converted to int16. Then divide the value by 10 to get a float value with decimal places. It is used to transfer temperature.
4. int32 - 4 bytes of the message, the youngest first.

Format of the 1st part of the protocol frame

Start character	Target physical address	Source physical address				Length of the 2nd part of the message	CRC8
0x69	0x02	0x01	0x84	0	0	0x0c	F6

1. Start character - a fixed value that allows you to find the beginning of the transmission
2. Target and Source physical address - if the heat pump sends a message to the display, these bytes have the form 0x02 0x01, if the display responds, the bytes have the form 0x01 0x02. For transmission on the second port, where the heat pump is a communication client, it has a specific address that can be set on the display (eg 5). In this case, we have to send a query with addresses in the form 0x05 0x01, and the answer is the reverse: 0x01 0x05.
3. Constant Bytes - 0x84, 0, 0 are bytes that never change, so it's hard to tell if there might be something else or not.

4. Length of the 2nd part of the message - the number of bytes to be received after the CRC8 sum, which will constitute the whole message. First we should download 8 bytes, check if everything is correct and then we should download as many bytes as is written in the 7th byte
5. CRC8 - this is not a typical CRC8, it is called CRC-8 / DVB-S2, so you cannot use the standard mechanism for calculating CRC. It is recommended to check this byte to detect transmission errors. This is the CRC of the first 7 bytes, not the entire message.

Format of the 2nd part of the protocol frame

Target logical address (int16)	Source logical address (int16)	Function (int8)	Message type marker (int16)	Remaining bytes	CRC16 (int16)
0x02	0x01	0x70	0x80	687800	3495

1. Target and Source logical address - just like in the first part, only here we have 2 bytes per address, so this one can accept more devices on one line. In the case of the display address, this value is the same as in the physical part, but these values may differ than in the physical part if we use the 2nd port for communication with EkoLan.
2. Function - description of the operation, so far I have met only 4 types:
    a. 0x40 – query for reading registers
    b. 0x50 – response to the query for reading registers, containing bytes of data from these registers or an error
    c. 0x60 – query for writing to registers, contains values that should be written
    d. 0x70 – response to writing to registers (whether it was successful)

    The master device can only send 0x40, 0x60 messages. The slave device can only reply to the 0x40 message with the 0x50 message, and to the 0x60 message with the 0x70 message. In the case of communication with the display, the heat pump is a master, if it wants to download some data from the display, it uses 0x40, if it wants to save something in the display (to be displayed on the screen) it uses 0x60 and sends bytes. The format of these messages is based on registers, each register has a specific address and is used for something. In theory, the addresses in the display could be different than the addresses in the heat pump, but everything indicates that these data are similar, i.e. when asking the heat pump for the same register addresses that the pump sends to the display, you get similar results (possibly they are small differences in one byte, but with registers below).
3. Message type marker - we usually observe the value 0x80 (in the protocol record these are 2 bytes 8000) it means that the next part of the message will concern the register address. I also encountered an unusual answer, where this field was C000 (i.e. 0xC0) - in binary form 1100 0000 0000 0000, it may suggest that this field is a flag. In case of receiving the value 0xC0, it most likely means that the device returned an error that it cannot perform the operation: I sent a query to read 100 registers starting from address 0. Presumably there is no register with the value 0 and therefore the heat pump returns an error. There was only one byte after C000 (except CRC): 04, which may be an error code.
4. Remaining bytes - in the case of the 0xC0 message type, the remaining bytes are described in the section Message type tag. For typical messages, it will be described below.
5. CRC16 – another CRC sum, this time from the 2nd part of the message. This time a different algorithm: CRC-16 / XMODEM, it is a non-standard CRC again, so you have to count manually. The sum consists of 2 bytes and it looks like they are placed sequentially, ie the oldest byte

first and not the youngest as in all other cases (although it may be a different type of CRC). Therefore, the description above that it is int16 is inaccurate.

Format of the remaining bytes

Format of the remaining bytes of functions 0x40 and 0x70 (i.e. query for reading registers and response to writing registers), proper format when the message type marker is 0x80

Number of registers / bytes (int8)	Address of the first byte read / stored (int16)
104	120

In this case, the message is short because it contains only the number of registers and the start address, but not the values of these registers.

Format of the remaining bytes of functions 0x50 and 0x60 (i.e. the response to the query for reading registers and the query for writing registers), proper format when the message type marker is 0x80

Number of registers / bytes (int8)	Address of the first byte read / write (int16)	Register values (int8 array)
4	252	10000000

In the example, the display returned the response 0x50 to the request for 4 bytes read, starting from the address 252. In register 252 there is 0x10, and in registers 253,254,255 there are all zeros.

Communication between the heat pump and the display

Communication on this channel is very large, therefore eavesdropping is difficult, as it is difficult to find the beginning of the frame. A series of messages is sent twice per second. Presumably, the point is that when the pump is working, the display shows flashing markers that something is happening (probably these markers flash every 500ms). The interval between the series of messages is about 360ms, and the whole series lasts about 140ms. Therefore, if you set the read timeout somewhere around 200ms, and ignore the first reading, then the second reading should get a frame from the first character, this knowledge should make it easier to find the start of the transmission.

The message series looks like this:

1. The pump sends a query to the display to read 20 registers starting from address 100.
2. The display responds, it always seems to be the same. It's hard to say if there is a display model or a serial number.
3. The pump sends a record of 104 registers starting at address 120. This is the main message, there are temperatures there. After this message there is a long pause (which is quite a large part of the 140ms for a series of messages), probably the display is not too fast to write it to memory and it takes longer.
4. The display replies with the standard message 0x70 that the bytes have been written successfully.
5. The pump requests to read 4 bytes starting from address 252.
6. The display responds and returns 4 bytes. By default they are: 10000000 and they mean that the display is working normally, there are no changes. The value 08000000 means that the display is turned off. However, the value 11000000 means that the user has changed a parameter in the menu, at this point the communication scheme is different, described below.

If in point 6 there is no flag that something has been changed, the pump waits about 360ms and starts another series of communication.

Additionally, in point 3, i.e. in 104 registers starting from 120, there are no menu parameters (i.e. whether the heat pump is turned on in the settings, or whether the heater is turned on). After entering the menu, the display does not ask about these parameters. This may suggest that these values are in other registers that are only sent during device startup. Changing the menu value in the display seems to have a slightly different process, it is impossible to read from it what actual addresses these configuration registers have.

Change the value of the display menu

Additional communication after point 6 from the previous series of messages when the menu was changed (the heat pump was turned off), the display responded 11000000 to the request for 4 registers starting from 252:

1. The pump sends a request to read 6 bytes starting from the address 256 (this is an address that does not fit into one byte, it may suggest that special functional registers are from 256).
2. The display returns the value 000100000000 when the user turns off the heat pump and the value 000101000000 when the user turns on the heat pump. Register 257 means which device (01 is heat pump, and 02 is heater E). Register 258 means disable or enable.
3. Heat pump sends 2 byte write starting from address 256 to FFFF value. This is quite a strange message, it probably means that the display needs to clear the buffer of what has changed in the menu as the Heat Pump has correctly received this data. I wonder why it writes the value of FFFF when the default values are 0000, maybe it is after the event mask, but I haven't fully figured out this mechanism.
4. The display responds with success of the save and then the heat pump reverts to the standard communication mechanism described in the previous section.

Example from point 2 when the user changes the LOW COP temperature of the heat pump:

04015A000000 - the value 0x5A means 90, i.e. 9.0 (it is a float16signed type). Value 0x01 means that it is related to the heat pump and 0x04 means that it is supposed to operate LOW COP temperature change.

Example from point 2, when the user changes the water temperature for heating with the heat pump:

020186010000 - 0x186 means 390, which is 39.0. The value 0x01 means that it is related to the heat pump and 0x02 means that it is supposed to change the water temperature operation.

Examples for E heater: 000200000000, 0202a4010000

Communication with the pump on the second port

Such communication is possible and even better. It is best to change the parameters as described in the first chapter, as other parameters have not been tested. It is recommended to first eavesdrop on the communication between the pump and the display to make sure that the 8N1 parameters are correct and that the A and B wires are well connected. Because in the case of the second port, the display will not respond if we send the message incorrectly, and in the case of eavesdropping we have bytes of data analyze. For example, if there are many FF characters in the communication, it may be a sign of wrong baud rate parameter setting. If there are so many 1s, this could mean the wrong number of stop bits. In case of error in both the stop bit and parity, the first character 0x69

will always be correct, the switch will start in the following characters. If we cannot find the 0x69 byte, it means bad baud rate or badly connected cables.

To get the same what is sent between the display and the pump, send the message 0x40 to the pump with a request for 104 registers starting from address 120. The pump will respond with the message 0x50 (in the case of a display it is the message 0x60, because there is a record of these values to the display, and here is reading the value from the pump). The register values are more or less the same, only you can see the difference in register 196 (register description below).

In this mode, you can also change the pump configuration, which is a significant advantage. First of all, you must first find what you want to change, I partially analyzed the registers and the deciphered information is in the description below. Not all parameters I was able to find the possibility of changing the time. It seems that only registers above 255 are modifiable, although it is possible to send a record of the time change to 8 registers at address 120, and the answer is correct, but the time does not change. It is possible that there is another time change command, or some larger register (e.g. above 1000).

When storing values, pay attention to the min / max. For example, the temperature in some of them may be changed in the range of 10 - 60 degrees. When, by mistake, instead of 400 I sent 40 (in registers the values are multiplied by 10 to contain the numbers after the decimal point), the heat pump returned the correct answer and changed to 500 (50 degrees) - this is the default value in this field, because the 4.0 that I sent was too small.

Description of registers

The heat pump has registers from 100 to 537. Values smaller or larger cannot be retrieved, suggesting this is the final figure. Basically it seems most registers are double (2 byte), sometimes there are single byte registers, but then they are in groups with each other so that the next registers start with an even number. It follows that the last 2-byte register is 536. Register 538 cannot be retrieved either in two or individually.

Description of registers in the attached excel file

Time program registers (schedule)

For example, the schedule for a heat pump Mon-Fri is on address 314 and consists of 4 bytes. Only the lower 3 contain the schedule, the oldest is always 0. One bit corresponds to each hour, if set, the given device is ready at this time. For example, the value of this register: 0x3FFFC0 in binary notation means:

00111111111111111000000

The youngest bit is 0 - 1 hour and the oldest 23 - 0 hour. In the above binary notation, the youngest bit is on the right, from the notation it can be read that the pump is turned off at 0, 1, 2, 3, 4, 5, then it is turned on until it is turned off again at 22 and 23.

Work status register

This is a very strange register (196), as it looks like it takes 2 bytes, but there are flags inside that have different meanings. The 3 oldest bits are used for communication with the display, in the case of the EkoLan port they have a value of 0. The values in these 3 bits are incremented by 1 with each subsequent message to the display. In the case of heating with the heater, this register has the value 0x1000 (i.e. the bit is set in the quadrant in which these 3 bits are located to control the display). When the circulation pump is running, the register value is 0x2. If the heat pump is running, this value is 0x825. I happened to notice that the value is 0x25 immediately after turning on the pump, which would suggest that bit 0x800 is responsible for the compressor's operation (it does not turn on immediately). In fact, 0x25 is 3 bits, which would suggest that they can stand alone as well. It is possible that one of these bits is responsible for the operation of the fan itself, and another for the operation of the circulation pump (the manual states that the circulation pump can sometimes turn on by itself to prevent jamming, so it would be useful to decode these bits).

It looks more or less bit like this:

1	2	3	4	5	6	7	8
Something related to the heat pump	Circulation pump	Something related to the heat pump		Something related to the heat pump			
9	10	11	12	13	14	15	16
			Compressor	Heater E	Display bit 1	Display bit 2	Display bit 3

The other bits probably represent other devices, possibly one of them is also responsible for anti-legionella.