MODBUS Master to SDI 12 Slave Converter

The TBS09S is a converter to connect SDI-12 sensors to a MODBUS master. It can control multiple SDI12 sensors in parallel by individually addressing the connected SDI-12 sensors.

TBS09S MODBUS Master to SDI 12 Slave Converter

Features

- MODBUS Master to SDI 12 Slave Converter
- Multiple SDI-12 sensors can be connected
- SDI-12 Standard V1.4 (reduced commands set)
- MODBUS RTU, 19200 baud
- Up to 32 SDI-12 commands can be programmed
- 5 - 16V supply voltage
- 12mA current consumption when active
- Available in DIN rail and FIBOX housing.

Target Applications

- SDI-12 sensor networks with MODBUS controller

Factory 4, F4, Lot I-3B-1, Saigon HiTech Park, Q.9 | Ho Chi Minh City | Tel +84 287 1099865| office@tekbox.com| www.tekbox.com

MODBUS Master to SDI-12 Slave Converter

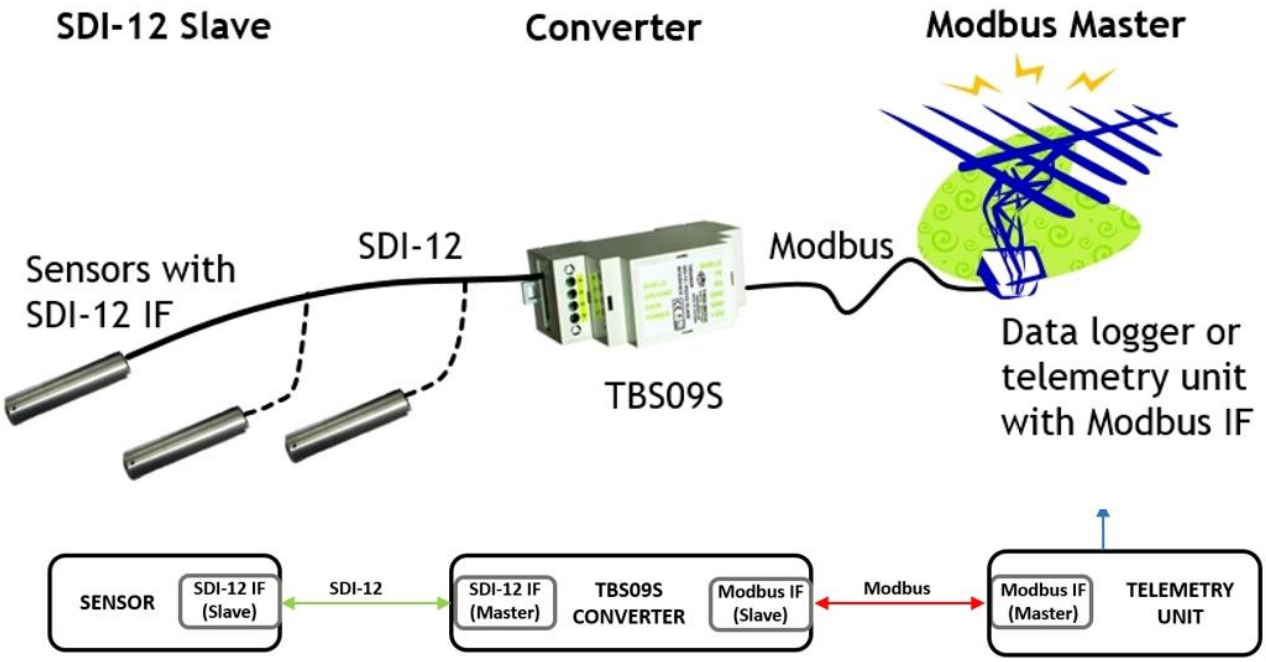
Contents

MODBUS Master to SDI-12 Slave Converter

1 Introduction

The TBS09S is a converter to connect one or multiple SDI-12 sensors to a MODBUS device such as a data logger or telemetry unit. The converter is inserted in between the data logger or RTU with MODBUS interface and the sensor(s) with SDI-12 interface. The designation MODBUS Master to SDI-12 Slave is ambiguous. Looking purely at the converter, the device got a MODBUS slave interface on one side and a SDI-12 master output at the other side. However, looking at its application, the device is a converter between a MODBUS master (data logger, RTU, etc.) and a SDI-12 slave (sensor with SDI-12 interface).

The following diagrams describe a typical use of TBS09S module that bridges a MODBUS telemetry unit with a SDI-12 sensor and highlight how the internal TBS09S MODBUS/SDI-12 layers interact with them.



TBS09S application

2 Product specification

- Application: converter used to interface MODBUS master devices (e.g., RTU) with SDI-12 slave devices (e.g., sensors)
    - The converter embeds MODBUS slave and SDI-12 master modules
- SDI-12 compatibility:
    - Version: v1.4
    - SDI-12 commands not supported: aV!, extended commands, high volume commands, metadata commands.
    - Data command supports up to 32 measurements maximum
- MODBUS compatibility:

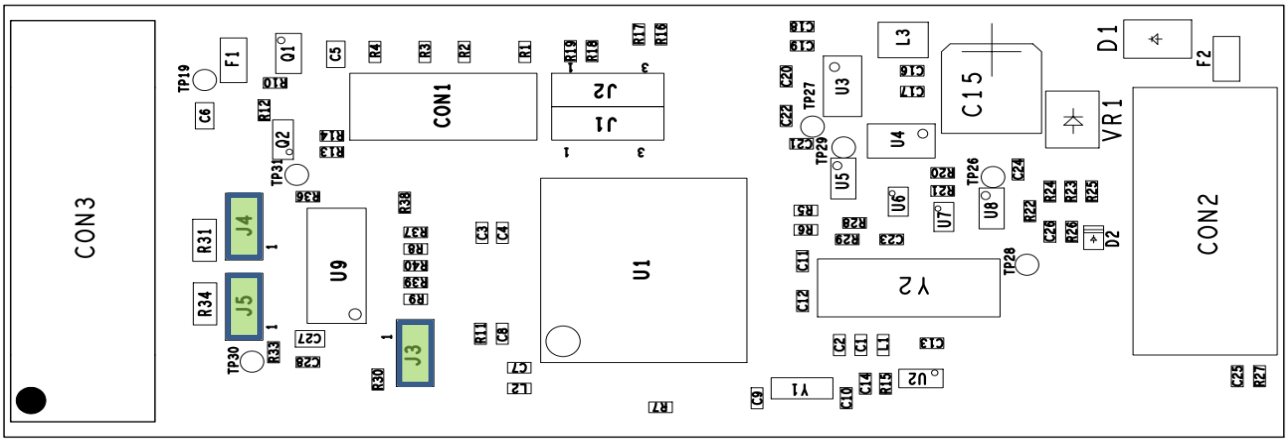
- Protocol: MODBUS RTU
  - Required supported function codes by MODBUS master: 0x01, 0x03, 0x04, 0x05, 0x06
- RS485 configuration:
  - Baud rate: 19200
  - Data length: 8 bits
  - Parity: none
  - Stop bits: 1
- Half or full duplex (configurable)
- 120 Ω termination (configurable)

- Supply voltage: 5 – 16 V
- Power consumption
  - Current drawn by TBS09S without any SDI-12 measurements: 12 mA
  - Warm-up time: 3s
- Form factor:
  - DIN rail
  - FIBOX (IP67 housing suitable for outdoor use)

3 Calibration and settings

TBS09S doesn't require any calibration.
It comes factory-configured to operate by default in half duplex with termination.

This configuration can be changed by the user by setting related jumpers J3/J4/J5 after lifting the housing:



Communication mode:

Configuration	J3	Default configuration
Half Duplex	✓	✓
Full Duplex	×	

MODBUS termination:

Configuration	J4 – J5	Default configuration
120 Ω termination	✓	✓
No termination	×	

4 Connections

TBS09S provides one 4 slots connector (SDI-12) and one 6 slots connector (RS485):

SDI-12 terminal assignment, from top to bottom:

Slot name	Description	Comment
Shield	Cable shield	Connect to sensor's cable shield
Ground	Ground	Connect to ground
Data	SDI-12 data line	Connect to SDI-12 sensor data line
Power	TBS09S supply voltage input	Connect to +12V external power supply

RS485 terminal assignment, from top to bottom:

Slot name	Description	Comment
T+	TXD+ output	Connect to MODBUS master RXD+
T-	TXD- output	Connect to MODBUS master RXD-
R+	RXD+ input	Connect to MODBUS master TXD+ **(*full duplex operation only –***

		must be left unconnected in half duplex)
R-	RXD- input	Connect to MODBUS master TXD- ***(full duplex operation only – must be left unconnected in half duplex)***
G	Ground	Connect to ground
P	SDI-12 sensor supply voltage output (+12V, connected to the converter supply line through a high side FET switch)	Connect to SDI-12 sensor power line.

5 MODBUS configuration

5.1 HW configuration

MODBUS RTU communication over RS485 can be set in either half or full duplex and 120Ω termination can be added as well.

It is advised to use 120Ω termination in case many MODBUS slave devices including TBS09S are sharing the same bus.

5.2 RS485 configuration

RS485 communication parameters are set as follows:

Feature	Default setting
Baud rate	19200 bauds
Data bits	8
Parity	None
Stop bits	1
TBS09S slave address	1

These settings can't be changed except for TBS09S MODBUS address.

For other RS485 configurations, please send a request to Tekbox Customer Support Team.

6 Sending SDI-12 commands through TBS09S

6.1 Limitations

Due to the fact SDI-12 and MODBUS RTU are totally different communication stacks and also due to memory constraints on the module, TBS09S does not fully support SDI-12 protocol as described in v1.4 specification.

Limitations vs SDI-12 v1.4 specification:

- Extended SDI-12 commands are not supported
- Verification command V! is not supported
- Maximum of 32 measurement values vs 99 in SDI-12 standard
- SDI-12 measurement time up to 255s vs 999s in SDI-12 standard
- aCx! commands are supported nevertheless they're not handled as concurrent measurement
- aMx! service request can't be signaled to the MODBUS master
- Data commands are automatically handled by TBS09S so the MODBUS master only needs to retrieve the measurements after sending the measurement command.
- The limitations and communication principles between the MODBUS master and SDI-12 sensors is discussed in detail in a dedicated chapter.

6.2 Supported SDI-12 commands

Supported SDI-12 address range is aligned with SDI-12 v1.4 specification:

- 0 – 9
- A – Z
- a – z

TBS09S embeds a full SDI-12 v1.4 stack, nevertheless only most of SDI-12 v1.3 commands are supported for MODBUS to SDI-12 conversion with some restrictions. Further support might be extended in a future HW&FW upgrade.

The following table lists SDI-12 v1.4 commands and their support by TBS09S and highlights any limitations or restrictions vs SDI-12 standard:

SDI-12 command	Description	TBS09S support	Comments
?!	Address Query	Yes	-
a!	Acknowledge Active	Yes	-
aI!	Send Identification	Yes	-
aAb!	Change Address	Yes	-
aM!	Start Measurement	Yes	Although TBS09S handles the service request, this is transparent for the MODBUS master as it can't be notified about it. Only polling the status register for the *ready* flag could help knowing earlier that the measurements are available.
aMC!	Start Measurement and Request CRC	Yes	
aM1! to aM9!	Additional Measurements	Yes	
aMC1! to aMC9!	Additional Measurements and Request CRC	Yes	

			Measurement time limited to 255s.
aC!	Start Concurrent Measurement	Yes	As TBS09S must receive new SDI-12 measurement commands over MODBUS only once the measurement values have been retrieved, true concurrent measurements are not possible. C! and CC! are merely like executing M!/MC!. Measurement time limited to 255s.
aCC!	Start Concurrent Measurement and Request CRC	Yes	
aC1! to aC9!	Additional Concurrent Measurement	Yes	
aCC1! to aCC9!	Additional Concurrent Measurement and Request CRC	Yes	
aR0! to aR9!	Continuous Measurements	Yes	-
aRC0! to aRC9!	Continuous Measurements and Request CRC	Yes	-
aV!	Start Verification	No	Not supported.
aD0! to aD9!	Send Data	Yes	Command automatically sent by TBS09S, does not have to be programmed. Maximum number of measurement values limited to 32.
	Extended Commands	No	
	High Volume Commands	No	
	Metadata Commands	No	

6.3 MODBUS to SDI-12 communication principles

SDI-12 commands are encapsulated by MODBUS which acts as a communication layer.

Each MODBUS request and response are fully compliant with MODBUS protocol standard.

Each request must be executed sequentially so SDI-12 measurement command is sent and the corresponding data command shall be then sent before executing another measurement command.

The overall measurement procedure can be summarized in few steps:

1. Program SDI-12 command(s) to be executed:
    a. MODBUS function code: **0x06** (Write Single Register)
    b. SDI-12 commands stored in MODBUS holding registers (configuration registers)
    c. Maximum of 32 SDI-12 commands.

d. Write only one commands or a set of commands one by one.
e. Content can be read back with MODBUS function code **0x03** (Read Holding Register)

2. Check for each programmed command the measurement time *ttt* and number of expected values *n/nn*
   a. MODBUS function code: **0x04** (Read Input Register)
   b. Measurement time and number of measurements stored in MODBUS input registers (read-only)
   c. This step is optional (especially when the use already has access to this information from the SDI-12 sensors data sheets).
3. Trigger SDI-12 measurement.
   a. MODBUS function code: **0x05** (Write Single Coil)
   b. Triggers the execution of the corresponding SDI-12 command
4. Retrieve measurement values returned by the executed SDI-12 command.
   a. MODBUS function code:**0x04** (Read Input Register)

6.3.1 TBS09S MODBUS registers mapping

<table>
<tr><th>Addresses</th><th colspan="2">Holding Registers</th><th colspan="2">Input Registers</th><th colspan="2">Coil Registers</th></tr>
<tr><td>0x00</td><td rowspan="10">32 registers to configure up to 32 SDI-12 commands</td><td>Cmd_index_0</td><td rowspan="10">32 registers to hold the measurement time ttt and number of returned parameters n for each programmed SDI-12 command</td><td>tttnn_index0</td><td rowspan="10">32 registers to trigger the execution of corresponding programmed SDI-12 command</td><td>Cmd_index_0</td></tr>
<tr><td>0x01</td><td>Cmd_index_1</td><td>tttnn_index1</td><td>Cmd_index_1</td></tr>
<tr><td>0x02</td><td>Cmd_index_2</td><td>tttnn_index2</td><td>Cmd_index_2</td></tr>
<tr><td>…</td><td>…</td><td></td><td></td></tr>
<tr><td>…</td><td>…</td><td></td><td></td></tr>
<tr><td>…</td><td>…</td><td></td><td></td></tr>
<tr><td>…</td><td>…</td><td></td><td></td></tr>
<tr><td>…</td><td>…</td><td></td><td></td></tr>
<tr><td>0x1E</td><td>Cmd_index_30</td><td>tttnn_index30</td><td>Cmd_index_30</td></tr>
<tr><td>0x1F</td><td>Cmd_index_31</td><td>tttnn_index31</td><td>Cmd_index_31</td></tr>
<tr><td>0x20</td><td colspan="2"></td><td colspan="2">Command status register</td><td colspan="2" rowspan="14"></td></tr>
<tr><td>0x21</td><td colspan="2" rowspan="2"></td><td rowspan="14">64 registers to hold 32 measurement values (32 bits hexadecimal float numbers, big-endian)</td><td>Value_index_0</td></tr>
<tr><td>0x22</td><td rowspan="2">Value_index_1</td></tr>
<tr><td>0x23</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>0x24</td></tr>
<tr><td>…</td><td colspan="2" rowspan="2"></td><td>…</td></tr>
<tr><td>…</td><td>…</td></tr>
<tr><td>0x5B</td><td colspan="2" rowspan="2"></td><td rowspan="2">Value_index_29</td></tr>
<tr><td>0x5C</td></tr>
<tr><td>0x5D</td><td colspan="2" rowspan="2"></td><td rowspan="2">Value_index_30</td></tr>
<tr><td>0x5E</td></tr>
<tr><td>0x5F</td><td colspan="2" rowspan="2"></td><td>Value_index_31</td></tr>
<tr><td>0x60</td></tr>
</table>

5 memory areas are used by TBS09S:

- Holding registers: configuration of SDI-12 commands to be executed
- Input registers contain:
  - Executed SDI-12 command status
  - SDI-12 measurement time and parameters for each programmed SDI-12 command
  - SDI-12 measurement values
- Coil registers: used to trigger a specific SDI-12 command execution

For addresses between 0x00 and 0x1F, it is important to note that corresponding holding register, input register and coil register are related to each other.

For instance, holding register 0x03 contains the SDI-12 command 3M!, input register 0x03 contains *tttn*=0012 for SDI-12 command 3M! and the coil register 0x03 is used to start the execution of SDI-12 command 3M!.

Address	Holding register	Input register	Coil register
0x03	3M!	0012	ON

6.3.2 SDI-12 commands configuration

Holding registers 0x00 to 0x1F are used for that purpose.

Each register can be used to configure a specific SDI-12 address and command:

Holding registers 0x00 to 0x1F structure	
Byte_Hi	SDI-12 address (hexadecimal ASCII value)
Byte_Lo	Encoded SDI-12 command

The SDI-12 address is simply represented by its corresponding ASCII value in hexadecimal:

SDI-12 address	Encoded SDI-12 address
0 - 9	0x30 – 0x39
A – Z	0x41 – 0x5A
a - z	0x61 – 0x7A

The SDI-12 command is encoded based on specific rules depending on its type and following look-up tables can be used to find the encoded command corresponding to a specific SDI-12 command:

Command: aCx!	Encoded Cmd (Byte 2)	Command: aMx!	Encoded Cmd (Byte 2)	Command: aRx!	Encoded Cmd (Byte 2)
aC!	0x73	aM!	0x7D	aR!	0xA2
aC1!	0x74	aM1!	0x7E	aR1!	0xA3
aC2!	0x75	aM2!	0x7F	aR2!	0xA4
aC3!	0x76	aM3!	0x80	aR3!	0xA5
aC4!	0x77	aM4!	0x81	aR4!	0xA6
aC5!	0x78	aM5!	0x82	aR5!	0xA7
aC6!	0x79	aM6!	0x83	aR6!	0xA8
aC7!	0x7A	aM7!	0x84	aR7!	0xA9
aC8!	0x7B	aM8!	0x85	aR8!	0xAA
aC9!	0x7C	aM9!	0x86	aR9!	0xAB

Command: aCCx!	Encoded Cmd (Byte 2)	Command: aMCx!	Encoded Cmd (Byte 2)	Command: aRCx!	Encoded Cmd (Byte 2)
aCC!	0xB6	aMC!	0xC0	aRC!	0xE5
aCC1!	0xB7	aMC1!	0xC1	aRC1!	0xE6
aCC2!	0xB8	aMC2!	0xC2	aRC2!	0xE7
aCC3!	0xB9	aMC3!	0xC3	aRC3!	0xE8
aCC4!	0xBA	aMC4!	0xC4	aRC4!	0xE9
aCC5!	0xBB	aMC5!	0xC5	aRC5!	0xEA
aCC6!	0xBC	aMC6!	0xC6	aRC6!	0xEB
aCC7!	0xBD	aMC7!	0xC7	aRC7!	0xEC
aCC8!	0xBE	aMC8!	0xC8	aRC8!	0xED
aCC9!	0xBF	aMC9!	0xC9	aRC9!	0xEE

Command: aAb!	Encoded Cmd (Byte 2)	Command: aAb!	Encoded Cmd (Byte 2)	Command: aAb!	Encoded Cmd (Byte 2)
aA0!	0x01	aAA!	0x12	aAa!	0x32
aA1!	0x02	aAB!	0x13	aAb!	0x33
aA2!	0x03	aAC!	0x14	aAc!	0x34
aA3!	0x04	aAD!	0x15	aAd!	0x35
aA4!	0x05	aAE	0x16	aAe!	0x36
aA5!	0x06	aAF	0x17	aAf!	0x37
aA6!	0x07	aAG	0x18	aAg!	0x38
aA7!	0x08	aAH	0x19	aAh!	0x39
aA8!	0x09	aAI	0x1A	aAi!	0x3A

aA9!	0x0A	aAJ	0x1B	aAj!	0x3B
		aAK	0x1C	aAk!	0x3C
		aAL	0x1D	aAl!	0x3D
		aAM	0x1E	aAm!	0x3E
		aAN!	0x1F	aAn!	0x3F
		aAO!	0x20	aAo!	0x40
		aAP!	0x21	aAp!	0x41
		aAQ!	0x22	aAq!	0x42
		aAR!	0x23	aAr!	0x43
		aAS!	0x24	aAs!	0x44
		aAT!	0x25	aAt!	0x45
		aAU!	0x26	aAu!	0x46
		aAV!	0x27	aAv!	0x47
		aAW!	0x28	aAw!	0x48
		aAX!	0x29	aAx!	0x49
		aAY!	0x2A	aAy!	0x4A
		aAZ!	0x2B	aAz!	0x4B

Other commands	**Encoded Cmd (Byte 2)**
aI!	0x69
?! or a!	0x00

Below tables are given for information purpose only and provide insights how the encoding rule has been designed (an index 0 to 9 encoded as 0x30 to 0x39 is systematically used for all measurement commands encoding; not applicable for SDI-12 Identification and Change Address commands).

For each command, a full example is provided with each byte values (SDI-12 address and SDI-12 command).

4.3.6.1 aMx! Commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	aM!	0x30 - 0x39 (0 - 9) a = 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0x7D	0x4D + 0x30	0M!	0x30	0x7D
2	aM1!		0x7E	0x4D + 0x31	3M1!	0x33	0x7E
3	aM2!		0x7F	0x4D + 0x32	6M2!	0x36	0x7F
4	aM3!		0x80	0x4D + 0x33	9M3!	0x39	0x80
5	aM4!		0x81	0x4D + 0x34	AM4!	0x41	0x81
6	aM5!		0x82	0x4D + 0x35	HM5!	0x48	0x82
7	aM6!		0x83	0x4D + 0x36	YM6!	0x59	0x83
8	aM7!		0x84	0x4D + 0x37	bM7!	0x62	0x84
9	aM8!		0x85	0x4D + 0x38	mM8!	0x6D	0x85
10	aM9!		0x86	0x4D + 0x39	xM9!	0x78	0x86

4.3.6.2 aMCx! Commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	aMC!	0x30 - 0x39 (0 - 9) a = 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0xC0	0x4D + 0x43 + 0x30	0MC!	0x30	0xC0
2	aMC1!		0xC1	0x4D + 0x43 + 0x31	3MC1!	0x33	0xC1
3	aMC2!		0xC2	0x4D + 0x43 + 0x32	6MC2!	0x36	0xC2
4	aMC3!		0xC3	0x4D + 0x43 + 0x33	9MC3!	0x39	0xC3
5	aMC4!		0xC4	0x4D + 0x43 + 0x34	AMC4!	0x41	0xC4
6	aMC5!		0xC5	0x4D + 0x43 + 0x35	HMC5!	0x48	0xC5
7	aMC6!		0xC6	0x4D + 0x43 + 0x36	YMC6!	0x59	0xC6
8	aMC7!		0xC7	0x4D + 0x43 + 0x37	bMC7!	0x62	0xC7
9	aMC8!		0xC8	0x4D + 0x43 + 0x38	mMC8!	0x6D	0xC8
10	aMC9!		0xC9	0x4D + 0x43 + 0x39	xMC9!	0x78	0xC9

4.3.6.3 aCx! Commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	aC!	a = 0x30 - 0x39 (0 - 9) 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0x73	0x43 + 0x30	0C!	0x30	0x73
2	aC1!		0x74	0x43 + 0x31	3C1!	0x33	0x74
3	aC2!		0x75	0x43 + 0x32	6C2!	0x36	0x75
4	aC3!		0x76	0x43 + 0x33	9C3!	0x39	0x76
5	aC4!		0x77	0x43 + 0x34	AC4!	0x41	0x77
6	aC5!		0x78	0x43 + 0x35	HC5!	0x48	0x78
7	aC6!		0x79	0x43 + 0x36	YC6!	0x59	0x79
8	aC7!		0x7A	0x43 + 0x37	bC7!	0x62	0x7A
9	aC8!		0x7B	0x43 + 0x38	mC8!	0x6D	0x7B
10	aC9!		0x7C	0x43 + 0x39	xC9!	0x78	0x7C

4.3.6.4 aCCx! Commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	aCC!	a = 0x30 - 0x39 (0 - 9) 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0xB6	0x43 + 0x43 + 0x30	0CC!	0x30	0xB6
2	aCC1!		0xB7	0x43 + 0x43 + 0x31	3CC1!	0x33	0xB7
3	aCC2!		0xB8	0x43 + 0x43 + 0x32	6CC2!	0x36	0xB8
4	aCC3!		0xB9	0x43 + 0x43 + 0x33	9CC3!	0x39	0xB9
5	aCC4!		0xBA	0x43 + 0x43 + 0x34	ACC4!	0x41	0xBA
6	aCC5!		0xBB	0x43 + 0x43 + 0x35	HCC5!	0x48	0xBB
7	aCC6!		0xBC	0x43 + 0x43 + 0x36	YCC6!	0x59	0xBC
8	aCC7!		0xBD	0x43 + 0x43 + 0x37	bCC7!	0x62	0xBD
9	aCC8!		0xBE	0x43 + 0x43 + 0x38	mCC8!	0x6D	0xBE
10	aCC9!		0xBF	0x43 + 0x43 + 0x39	xCC9!	0x78	0xBF

4.3.6.5 aRx! Commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	aR!	a = 0x30 - 0x39 (0 - 9) 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0xA2	*0x72 + 0x30	0R!	0x30	0xA2
2	aR1!		0xA3	*0x72 + 0x31	3R1!	0x33	0xA3
3	aR2!		0xA4	*0x72 + 0x32	6R2!	0x36	0xA4
4	aR3!		0xA5	*0x72 + 0x33	9R3!	0x39	0xA5
5	aR4!		0xA6	*0x72 + 0x34	AR4!	0x41	0xA6
6	aR5!		0xA7	*0x72 + 0x35	HR5!	0x48	0xA7
7	aR6!		0xA8	*0x72 + 0x36	YR6!	0x59	0xA8
8	aR7!		0xA9	*0x72 + 0x37	bR7!	0x62	0xA9
9	aR8!		0xAA	*0x72 + 0x38	mR8!	0x6D	0xAA
10	aR9!		0xAB	*0x72 + 0x39	xR9!	0x78	0xAB

(*) Note: Using hex value of 'r' (0x72) instead of 'R' (0x52), to avoid 'Byte 2' having the same value with other commands.

4.3.6.6 aRCx! Commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	aRC!	a = 0x30 - 0x39 (0 - 9) 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0xE5	*0x72 + 0x43 + 0x30	0RC!	0x30	0xE5
2	aRC1!		0xE6	*0x72 + 0x43 + 0x31	3RC1!	0x33	0xE6
3	aRC2!		0xE7	*0x72 + 0x43 + 0x32	6RC2!	0x36	0xE7
4	aRC3!		0xE8	*0x72 + 0x43 + 0x33	9RC3!	0x39	0xE8
5	aRC4!		0xE9	*0x72 + 0x43 + 0x34	ARC4!	0x41	0xE9
6	aRC5!		0xEA	*0x72 + 0x43 + 0x35	HRC5!	0x48	0xEA
7	aRC6!		0xEB	*0x72 + 0x43 + 0x36	YRC6!	0x59	0xEB
8	aRC7!		0xEC	*0x72 + 0x43 + 0x37	bRC7!	0x62	0xEC
9	aRC8!		0xED	*0x72 + 0x43 + 0x38	mRC8!	0x6D	0xED
10	aRC9!		0xEE	*0x72 + 0x43 + 0x39	xRC9!	0x78	0xEE

(*) Note: Using hex value of 'r' (0x72) instead of 'R' (0x52), to avoid 'Byte 2' having the same value with other commands.

4.3.6.7 Other SDI-12 commands

NO	Cmd	Byte 1 (Sensor Address)	Byte 2 (Cmd)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	?!	0X3F	0x00	0x00	?!	0x3F	0x00
2	a!	0x30 - 0x39 (0 - 9) a = 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0x00	0x00	0!	0x30	0x00
3	aI!		0x69	*i = 0x69	3I!	0x33	0x69
4	aAb!		**b - 0x2F	0x30 - 0x39 (0 - 9) b = 0x41 - 0x5A (A - Z) 0x61 - 0x7A (a - z)	0A1!	0x30	0x02
					1Aa!	0x31	0x32
					aAB!	0x61	0x13
					BAZ!	0x42	0x2B

(*) Note: Using hex value of 'i' (0x69) instead of 'I' (0x49), to avoid 'Byte 2' having the same value with other commands.
(**) Note: Subtract the 'b' to '0x2F' instead of plus with hex value 'A' (0x41), to avoid 'Byte 2' having the same value with other commands.

6.3.3 SDI-12 delay and parameters number encoding

This information is stored by TBS09S into Input Registers 0x00 to 0x1F.

Input registers 0x00 to 0x1F structure	
Byte_Hi	SDI-12 command delay (hexadecimal)
Byte_Lo	SDI-12 delay number of parameters (hexadecimal)

Example:

Input register 0x05: value 0x0A08	
Byte_Hi	0x0A ➔ 10 seconds
Byte_Lo	0x08 ➔ 8 parameters

Notes*:*

- Both bytes are set to zero in case ?!, a!, aI! or aAb! SDI-12 command is executed.
- SDI-12 command measurement time is limited to 255s vs 999s in the SDI-12 standard.
- Number of returned measurement values is limited to 32 vs 99 in case of concurrent measurement.
- The SDI-12 delay retrieved from the input register corresponds to the SDI-12 delay returned by the SDI-12 sensor increased by 1s. This is required to take into account internal TBS09S processing time.

6.3.4 Command status register

Whenever a MODBUS request is executed, the status register located at input register address 0x20 is updated.

Input register 0x20 structure (Command status register)	
Byte_Hi	Status code
Byte_Lo	Corresponding holding register address

Status code	
0x00	Unknown
0x11	OK
0xCC	SDI-12 CRC error
0xEE	Invalid command
0xFF	Command process failed

Example:

Holding register 0x15 is programmed with 0x39C3 (9MC3! SDI-12 command).

The command 9MC3! is executed and then the status register value is 0xCC15, which means execution of the SDI-12 command stored in holding register 0x15 (9MC3! in this example) failed due to a CRC error (0xCC).

6.3.5 SDI-12 commands activation

Each SDI-12 command configured in its holding register can be enabled by turning on its corresponding coil register.

Coil registers 0x00 to 0x1F structure	
Byte_Hi	SDI-12 command activation: ON or OFF
Byte_Lo	0x00

SDI-12 command activation	
0x00	OFF
0xFF	ON

Example:

By writing 0xFF00 to coil register 0x12, the SDI-12 command programmed in holding register 0x12 will be executed.

6.3.6 SDI-12 measurement values

Measurement values are stored in Input Registers 0x21 to 0x60.

Each value is encoded over 4 bytes (i.e., 2 registers) in hexadecimal floating point as per IEEE754 (a useful online converter can be found at https://www.h-schmidt.net/FloatConverter/IEEE754.html).

A total of up to 32 values are then stored in TBS09S after the execution of a SDI-12 command. When a value is not available (for instance only 5 measurements are expected), then the first 10 registers are populated whereas other 54 registers left are set to zero.

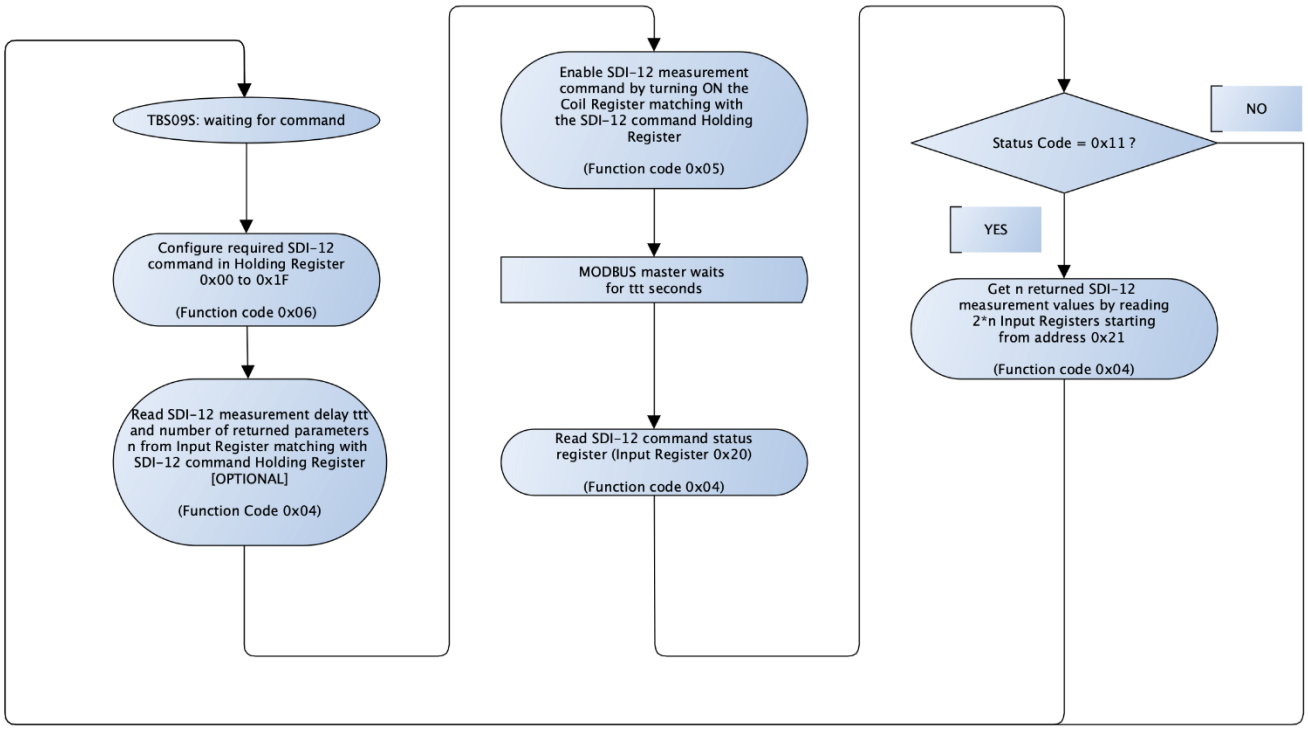
Example:

SDI-12 measurement value index 1	
0x23	0x41BE
0x24	0xF5C3

The value represented as hexadecimal float is 0x41BEF5C3 which is 23.87 in decimal.

6.3.7 TBS09S configuration and measurement flowchart

The following flowchart highlights the steps to follow to configure TBS09S and execute SDI-12 measurements.

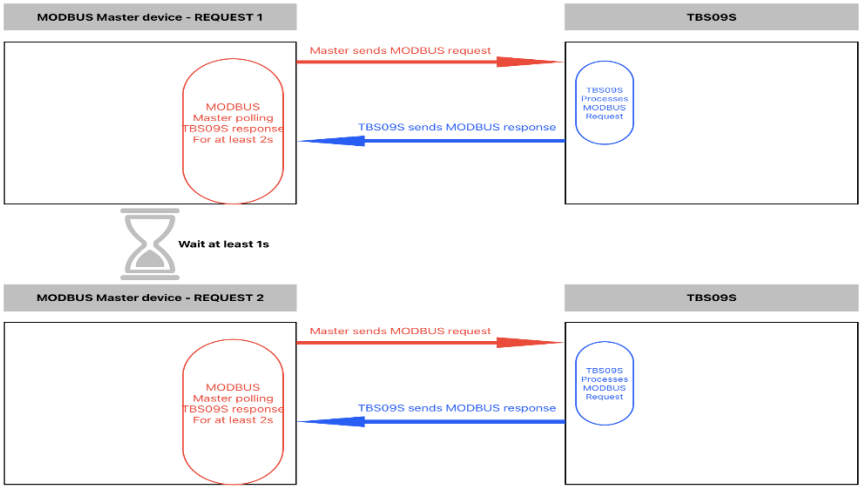


After the MODBUS master has started the SDI-12 measurement procedure by writing to the desired Coil Register, TBS09 is not able to process any other MODBUS command until it has updated the corresponding Input Register with the measurement values.

Would the MODBUS master send any command or poll the Input Registers during that time, a timeout error will occur.

Note:

- **It is mandatory to add a minimum 1s delay between the execution of 2 MODBUS requests (whatever the function code is)**. This is required to cope with internal TBS09S processing time.
- It is also recommended to configure the MODBUS master device with at least 2s for the MODBUS response timeout.
- Below diagram highlights both MODBUS timeout and MODBUS requests delay:



6.4 Miscellaneous commands

6.4.1 Overview

Some extra MODBUS commands are provided by TBS09S to:

- Change TBS09S MODBUS address
- Check TBS09S FW version
- Reload default settings

These commands need to be written to any TBS09S holding register but unlike SDI-12 commands, they are immediately executed and therefore does not require any further commands.

NO	Cmd	Byte 1 (Command)	Byte 2 (Parameter)	Rule of byte 2	Example		
					Cmd	Byte 1	Byte 2
1	Change Slave Address	0xCA	0x00-0xFF	Change slave address from 0-255	addr: 5	0xCA	0x05
					addr: 13	0xCA	0x0D
					addr: 248	0xCA	0xF8
2	Get Device Firmware Version	0xEF	0x00		0xEF00	0xEF	0x00
3	Get Default Setting	0xED	0x00		0xED00	0xED	0x00

However, it is strongly recommended to read back the content of the status register (Input Register 0x20) to ensure the command has been correctly executed.

6.4.2 Change MODBUS address

This command must be very carefully executed as there's no way to retrieve TBS09S MODBUS address if it has been lost, unless by trying all addresses one by one.

It is therefore strongly recommended to read back the content of the status register (Input Register 0x20) to ensure the command has been correctly executed before proceeding further.

Once the command has been successfully executed, the new MODBUS address is applied after turning off and on TBS09S.

6.4.3 Get FW version

After executing this command, the FW version can be retrieved by reading Input Registers 0x21 to 0x33.

The FW version consists in a hexadecimal ASCII string with following format:

021	DS61	Data	3134h
022	DS62		5445h
023	DS63		4B42h
024	DS64		4F58h
025	DS65		564Eh
026	DS66		5442h
027	DS67		5330h
028	DS68		3953h
029	DS69		7276h
030	DS70		4131h
031	DS71		3930h
032	DS72		3031h
033	DS73		3030h

Converted to ASCII: 14TEKBOXVNTBS09SrvA**1900100**

Where:

- 14: SDI-12 standard v1.4
- TEKBOXVN: Tekbox Vietnam
- TBS09S: product name
- rvA: HW revision A
- 1900100: FW version 19.0.01.00

6.4.4 Reset to default settings

This command resets TBS09S to its default parameters:

- RS485: 19200 bauds, no parity, 1 stop bit, 8 data bits
- MODBUS slave address: 1
- All holding registers are cleared

After successful execution of this command (that can be confirmed by checking the status register 0x20), the default configuration is restored after turning off/on TBS09S.

7 TBS09S configuration and communication examples

The following examples are based on Click PLC Koyo C0-12DD1E-2-D communicating with TBS09S.

The initial setup is as follows:

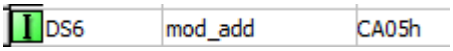
- PLC: MODBUS RTU
- TBS09S MODBUS slave address: 1
- SDI-12 sensor: TBSSPP1 soil moisture and temperature cell, SDI-12 address: T

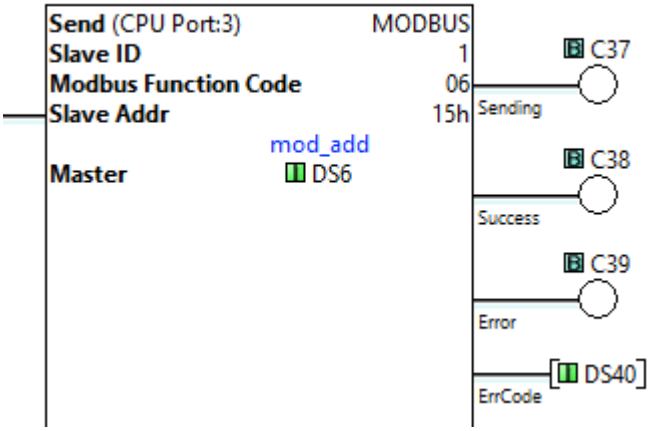
Also in below example, Holding Register 0x15 will be used to store the commands to be executed (could be any other Holding Register as long as it's within the range supported by TBS09S).

After executing any command stored in the Holding Registers, the status register shall be systematically checked to ensure the success of the operation. This will be described only in the example related to TBS09S MODBUS address change however it is strongly recommended to perform this check whenever a command is executed.

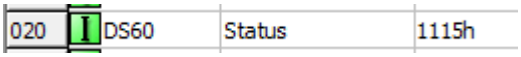
7.1 Change TBS09S MODBUS address

Change TBS09S MODBUS address from 1 to 5 by writing command 0xCA05 to Holding Register 0x15 using MODBUS function code 0x06:





The command has been successfully executed; this can be checked by reading back the status register at Input Register address 0x20:



- 0x11: Success
- 0x15: Holding Register address where the Change MODBUS Address command has been programmed.

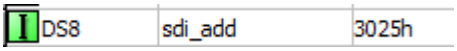
TBS09S is then turned off/on so the new MODBUS slave address is applied.

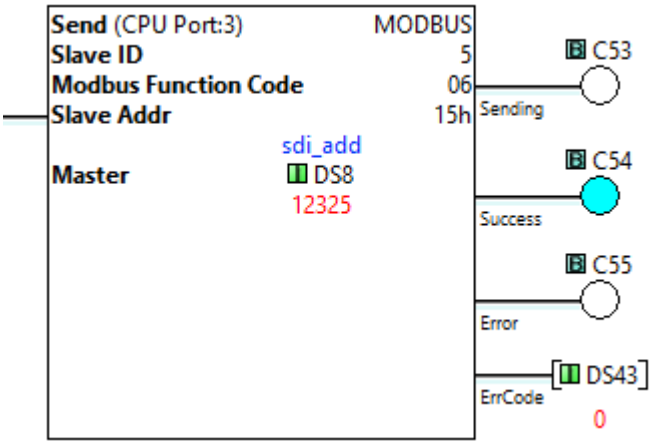
Note:
It shall be noted that there are 2 different kinds of status, irrespective of the command that is executed:

- MODBUS status
  - According to MODBUS standard, a response is sent back by the slave following a MODBUS request sent by the master. In case of error, a specific MODBUS exception response is sent back.
  - In this example, C39 coil is used to signal an error and the exception code is logged into PLC DS40 register.
- TBS09S status
  - This is the value stored in Input Register address 0x20 as described earlier in this document.
  - This status is related to TBS09S processing and more specifically when a command stored in the Holding Registers is executed.

7.2 Change SDI-12 sensor address

SDI-12 sensor address is changed from 0 to T by writing 0x3025 to Holding Register 0x15 using MODBUS function code 0x06:

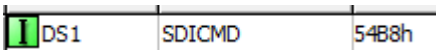


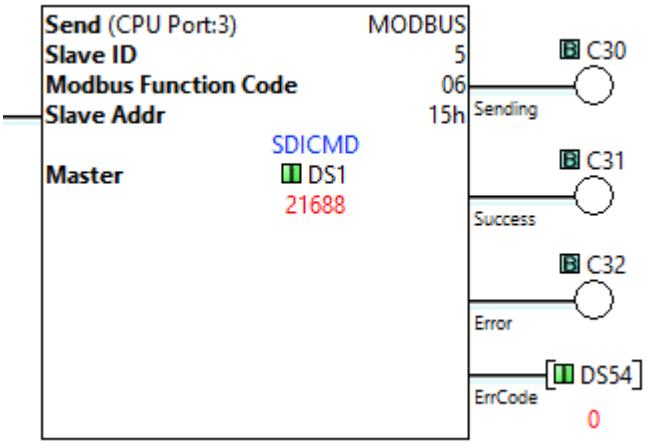


New SDI-12 address T is immediately applied.

7.3 Configure SDI-12 command

The PLC will program SDI-12 command TCC2! by writing 0x54B8 to Holding Register 0x15 using MODBUS function code 0x06.



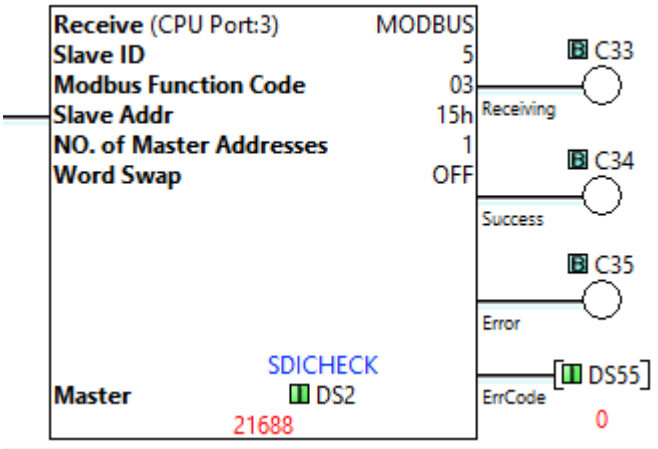


The PLC must then wait 1s before sending the next MODBUS request.

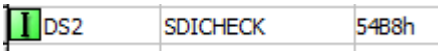
7.4 Read back configured SDI-12 command

This step is optional and is only to illustrate that at any time it's possible to read back SDI-12 commands configuration by reading corresponding Holding Register.

Read back Holding Register 0x15 using MODBUS function code 0x03:



The SDI-12 command has been correctly programmed:

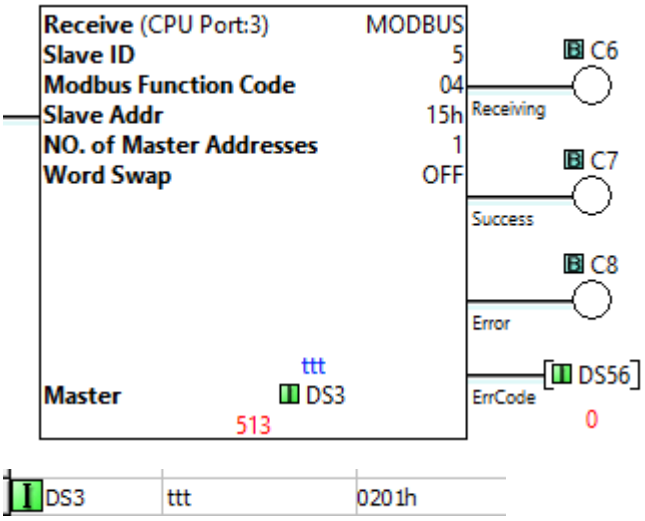


The PLC must then wait 1s before sending the next MODBUS request.

7.5 Read SDI-12 measurement time and number of parameters

After programming the desired SDI-12 command, TBS09S gets SDI-12 parameters *ttt* and *n* and makes them available in Input Register 0x15.

These parameters can be read back by using MODBUS function code 0x04.



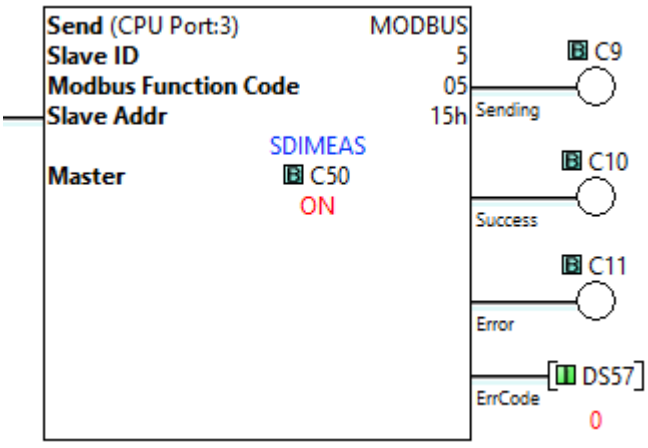
0x0201 is read back:

- 0x02: delay 2s
- 0x01: number of measurement values 1

The PLC must then wait 1s before sending the next MODBUS request.

7.6 Trigger SDI-12 measurement

Command TCC2! will now be executed by turning on the Coil Register 0x15 using MODBUS function code 0x05:



Turning on this coil will trigger the SDI-12 measurement command programmed in Holding Register 0x15 (i.e. TCC2!) and will automatically send SDI-12 data command TD0! Retrieve the measurement values.

The PLC needs then to wait for 2s (SDI-12 delay is 2s in this example) before reading the measurement value in Input Registers from address 0x21.

7.7 Check SDI-12 command status register

After the 2 seconds delay period has expired, it is required to check TBS09S status register (Input Register 0x20) to ensure the command has been successfully processed and executed:

020	DS60	Status	1115h

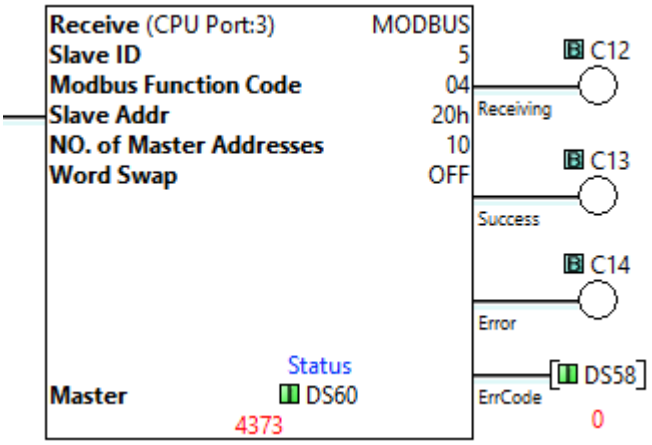
Status register holds 0x1115:

- 0x11: command successfully executed
- 0x15: the SDI-12 command stored in Holding Register 0x15 has been executed

The PLC must then wait 1s before sending the next MODBUS request.

7.8 Retrieve SDI-12 measurement values

The temperature value returned by the sensor can be read by simply reading TBS09S Input Registers from address 0x20 (Status register + measurement values registers) using MODBUS function code 4:



In this example all of the 64-measurement values register are read including the status register.

Since this sensor returns only one measurement value, it is held as a hexadecimal float number in the first 2 measurement registers (addresses 0x21 and 0x22):

020	DS60	Status	1115h
021	DS61	Data	41DFh
022	DS62		020Ch
023	DS63		0000h
024	DS64		0000h
025	DS65		0000h
026	DS66		0000h
027	DS67		0000h

The hexadecimal floating point representation is 0x41DF020C which is +27.876 in decimal.

8 Communication protocols

8.1 SDI-12

SDI-12 is a standard for interfacing data recorders with microprocessor-based sensors. SDI-12 stands for serial/digital interface at 1200 baud. It can connect multiple sensors with a single data recorder on one cable. It supports up to 60 meters cable between a sensor and a data logger.

The SDI-12 standard is prepared by

SDI-12 Support Group
(Technical Committee)
165 East 500 South
River Heights, Utah
435-752-4200
435-752-1691 (FAX)
http://www.sdi-12.org

The standard is available on the website of the SDI-12 Support Group.

8.2 MODBUS

The MODBUS standard is managed by Modbus Organization and the reference MODBUS Application Protocol is available on their website: https://www.modbus.org/

9 Mechanical information

Housing	Length (mm)	Width (mm)	Height (mm)
DIN rail	90.2	36.3	41.9
FIBOX	120	80	55

10 Environmental specification

Symbol	Parameter	Conditions	Min	Max	Unit
T_A	Operating Ambient Temperature Range	DIN rail: standard room humidity levels.	+20	+30	°C
T_{STG}	Storage Temperature Range	DIN rail housing	-20	+80	°C
		FIBOX housing	-40	+140	
-	Waterproofness	DIN rail: no, indoor use only	-		
		FIBOX: yes, IP67			

11 Ordering information

Part Number	Description
TBS09S	MODBUS master to SDI-12 slave converter – FIBOX housing
TBS09SDR	MODBUS master to SDI-12 slave converter – DIN rail housing
TBS09J	7 ports MODBUS junction box (half/full duplex) – FIBOX housing

12 History

Version	Date	Author	Changes
V1.0	9.8.2018	Thinh	Creation of the document
V1.1	9.19.2019	Thinh	Update new command
V1.5	24.6.2020	Hoa Hoang	Updated naming to MODBUS Master to SDI 12 Slave Converter
V1.6	15.7.2020	Mayerhofer	Complete rework of the document
V1.7	17.7.2020	Philippe	Updated link in 5.5
V1.8	2.4.2021	Mayerhofer	Updated drawing in chapter 1
V1.9	22.07.2021	Philippe Hervieu	Fix typo in 5.4.2 (Read returned SDI-12 address)
V1.10	08.08.2022	Philippe Hervieu	120 Ohm termination set by default.
V2.0	07.09.2023	Philippe Hervieu	User manual reworked following FW upgrade.