Motors | Energy | Automation | Coatings

Serial

CFW-11

Communication Manual

Language: English

RS232 / RS485 Serial Communication Manual

Series: CFW-11

Language: English

Document Number: 0899.5741 / 03

Publication Date: 01/2010

Summary

ABOUT THIS MANUAL....5

ABBREVIATIONS AND DEFINITIONS.... 5
NUMERICAL REPRESENTATION 5

1 INTRODUCTION TO SERIAL COMMUNICATION 6

2 ACCESSORY KITS....7

2.1 RS232 7
2.1.1 RS232-01 Kit.... 7
2.1.2 Connector Pin Functions.... 7
2.1.3 Indications and Switches.... 7
2.1.4 Connection with the RS232 Network.... 7
2.1.5 Cables for the RS232 Connection 8
2.2 RS485 8
2.2.1 RS485-01 Kit.... 8
2.2.2 Kit CAN/RS485-01 8
2.2.3 Connector Pin Functions.... 9
2.2.4 Indications and Switches.... 9
2.2.5 Connection with the RS485 Network.... 9
2.3 ANYBUS-CC 9

3. INVERTER PROGRAMMING 10

3.1 SYMBOLS FOR THE PROPRIETIES DESCRIPTION 10
P0105 – 1ST/2ND RAMP SELECTION 10
P0220 – LOCAL/REMOTE SELECTION SOURCE.... 10
P0221 – SPEED REFERENCE SELECTION – LOCAL SITUATION.... 10
P0222 – SPEED REFERENCE SELECTION – REMOTE SITUATION 10
P0223 – FORWARD/REVERSE SELECTION – LOCAL SITUATION.... 10
P0224 – RUN/STOP SELECTION - LOCAL SITUATION 10
P0225 – JOG SELECTION - LOCAL SITUATION 10
P0226 – FORWARD/REVERSE SELECTION – REMOTE SITUATION 10
P0227 – RUN/STOP SELECTION - REMOTE SITUATION 10
P0228 – JOG SELECTION - REMOTE SITUATION 10
P0308 – SERIAL ADDRESS.... 10
P0310 – SERIAL COMMUNICATION RATE.... 11
P0311 – SERIAL INTERFACE BYTE CONFIGURATION 11
P0312 – SERIAL PROTOCOL 11
P0313 – ACTION IN CASE OF COMMUNICATION ERROR 12
P0314 – SERIAL WATCHDOG.... 12
P0316 – SERIAL INTERFACE STATUS 13
P0680 – LOGIC STATUS 13
P0681 – MOTOR SPEED IN 13 BITS.... 14
P0682 – CONTROL WORD VIA SERIAL / USB 15
P0683 – SPEED REFERENCE VIA SERIAL/ USB 16
P0695 – SETTINGS FOR THE DIGITAL OUTPUTS 16
P0696 – VALUE 1 FOR ANALOG OUTPUTS.... 17
P0697 – VALUE 2 FOR ANALOG OUTPUTS.... 17
P0698 – VALUE 3 FOR ANALOG OUTPUTS.... 17
P0699 – VALUE 4 FOR ANALOG OUTPUTS.... 17

4 TP PROTOCOL....19

4.1 PROTOCOL FIELDS.... 19
4.2 TELEGRAM FORMAT.... 20
4.2.1 Reading Telegram.... 20
4.2.2 Writing Telegram 20
4.3 EXAMPLE OF TELEGRAMS USING THE TP PROTOCOL 20

5 MODBUS-RTU PROTOCOL 22

5.1 TRANSMISSION MODES 22
5.2 RTU MODE MESSAGE STRUCTURE 22
5.2.1 Address *22*
5.2.2 Function Code *22*
5.2.3 Data Field *22*
5.2.4 CRC *23*
5.2.5 Time Between Messages *23*
5.3 CFW-11 OPERATION IN THE MODBUS-RTU NETWORK 23
5.3.1 Available Functions and Response Times *24*
5.3.2 Data Addressing and Offset *24*
5.4 DETAILED DESCRIPTION OF THE FUNCTIONS 25
5.4.1 Function 03 – Read Holding Register *25*
5.4.2 Function 06 – Write Single Register *26*
5.4.3 Function 16 – Write Multiple Registers *27*
5.4.4 Function 43 – Read Device Identification *28*
5.4.5 Communication Errors *29*

6 FAULTS AND ALARMS RELATED TO THE SERIAL COMMUNICATION 30

A128/F228 – TIMEOUT FOR SERIAL COMMUNICATION 30

I. APPENDICES 31

APPENDIX A. ASCII TABLE 31
APPENDIX B. CRC CALCULATION USING TABLES 32
APPENDIX C. CRC CALCULATION USING DISPLACEMENT OF REGISTERS 33

About this Manual

This manual supplies the necessary information for the operation of the CFW-11 frequency inverter using the RS232 and RS485 serial interfaces. This manual must be used together with the CFW-11 user manual.

Abbreviations and Definitions

ASCII	American Standard Code for Information Interchange
CRC	Cycling Redundancy Check
EIA	Electronic Industries Alliance
RTU	Remote Terminal Unit

Numerical Representation

Decimal numbers are represented by means of digits without suffix. Hexadecimal numbers are represented with the letter 'h' after the number.

1 Introduction to Serial Communication

In a serial interface the data bits are sent sequentially through a communication channel or bus. Several technologies use the serial communication for data transfer, including the RS232 and RS485 interfaces.

The directions that specify the RS232 and RS485 standards, however, do neither specify the character format, nor its sequence for the data transmission and reception. Therefore, besides the interface, it is also necessary to identify the protocol used for the communication. Among the several existent protocols, one used a lot in the industry is the Modbus-RTU protocol.

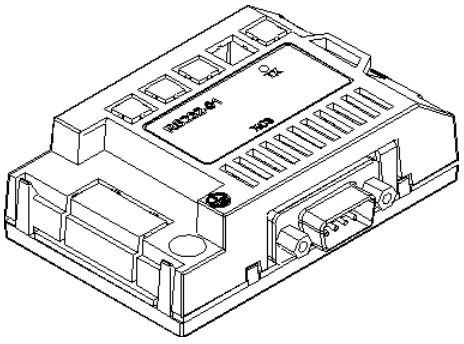
In the sequence the characteristics of the RS232 and RS485 serial interfaces available for the CFW-11 will be presented, as well as the protocols for the use of those interfaces.

2 Accessory Kits

In order to make available a serial interface for the CFW-11 it is necessary to use one of the RS232 or RS485 communication kits described next. Information on the installation of those kits can be obtained in the guide that comes with the kit.

2.1 RS232

2.1.1 RS232-01 Kit



- ☑ WEG part number: 10051958.
- ☑ Composed by the RS232 communication module (drawing at the left), mounting instructions and fixing screw.
- ☑ The interface follows the EIA RS232C standard.
- ☑ It allows the connection from the CFW11 to the network master (point to point).
- ☑ Maximum distance for the devices connection of 10m.

2.1.2 Connector Pin Functions

The RS232 communication module presents a male DB9 connector (XC8) with the following pin assignment:

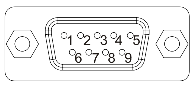


Table 2.1 *- RS232 DB9 connector pin assignment*

Pin	Name	Function
1	Not connected	-
2	RX	Data reception
3	TX	Data transmission
4	Not connected	-
5	GND	Reference for the RS232 circuit
6	Not connected	-
7	Not connected	-
8	Not connected	-
9	Not connected	-

2.1.3 Indications and Switches

- ☑ **TX LED**: LED for the indication of data transmission by the inverter, in green color.

2.1.4 Connection with the RS232 Network

- ☑ The inverter RX and TX signals must be respectively connected to the master TX and RX signals, besides the reference signal (GND) connection.
- ☑ The RS232 interface is very susceptible to interference. For that reason the cable used for the communication must be as short as possible – always less than 10 meters – and must be laid separately from the power input and motor cables.

2.1.5 Cables for the RS232 Connection

In case it is wished, WEG is able to supply the following cables for the connection in RS232 between the CFW-11 inverter and a network master, as a PC for instance:

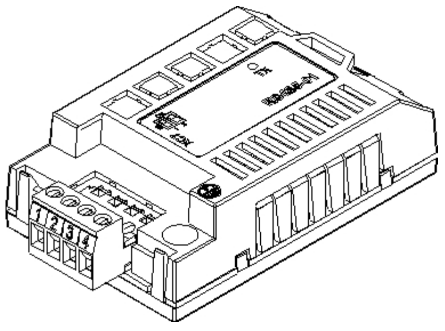
Cable	WEG Part Number
Shielded RS232 cable with a DB9 female connector Length: 3 meters	10050328
Shielded RS232 cable with a DB9 female connector Length: 10 meters	10191117

Other cables, however, can be found in the market – generally called *null-modem* – or assembled according to what is wished for the installation.

2.2 RS485

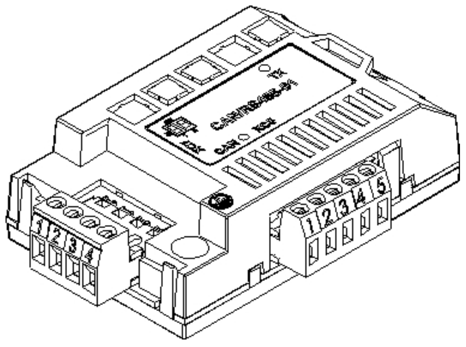
The CFW-11 presents 2 options for using the RS485 interface, as described next.

2.2.1 RS485-01 Kit



- ☑ WEG part number: 10051957.
- ☑ Composed by the RS485 communication module (drawing at the left), mounting instructions and fixing screw.
- ☑ The interface follows the EIA-485 standard.
- ☑ The interface is electrically isolated and with differential signal, which grants more robustness against electromagnetic interference.
- ☑ It allows the connection of up to 32 devices to the same segment. More devices can be connected by using repeaters.[1]
- ☑ A maximum bus length of 1000 meters.

2.2.2 Kit CAN/RS485-01



- ☑ WEG part number: 10051960.
- ☑ Composed by the CAN/RS485-01 communication module (drawing at the left), mounting instruction and fixing screw.
- ☑ It has the same characteristics as the RS485-01 interface, plus a CAN interface, for applications where the operation with both interfaces is necessary.

[1] The limit number of devices that can be connected to the network depends also on the used protocol.

2.2.3 Connector Pin Functions

The RS485 communication module presents a 4 wire *plug*-in connector (XC7) with the following pin assignment:

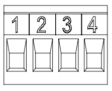
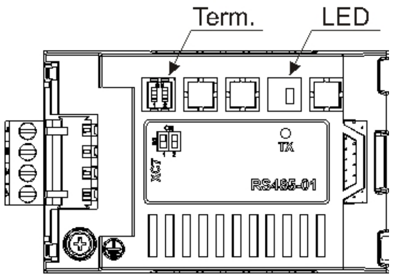


Table 2.2 *- 4 wire RS485 connector pin assignment*

Pin	Name	Function
1	A-Line (-)	RxD/TxD negative
2	B-Line (+)	RxD/TxD positive
3	GND	0V isolated from the RS485 circuit
4	Ground	Ground (shield)

2.2.4 Indications and Switches



☑ **TX LED:** LED for the indication of data transmission by the inverter, in green color.
☑ **Termination resistor (S1):** switch for enabling the termination resistor, necessary for the RS485 interface. This resistor must be enabled (position *ON*) only at the devices located at the extremes of the main bus.

2.2.5 Connection with the RS485 Network

The following point must be observed for the connection of the inverter using the RS485 interface:

☑ It is recommended the use of a shielded cable with a twisted pair of wires.
☑ It is also recommended that the cable has one more wire for the connection of the reference signal (GND). In case the cable does not have the additional wire, then the GND signal must be left disconnected.
☑ The cable must be laid separately (and far away if possible) from the power cables.
☑ All the network devices must be properly grounded, preferably at the same ground connection. The cable shield must also be grounded.
☑ Enable the termination resistors only at two points, at the extremes of the main bus, even if there are derivations from the bus.

2.3 ANYBUS-CC

The RS232 and RS485 interfaces can also be made available by using the Anybus-CC kits available for RS232 or RS485. Refer to the Anybus-CC Communication Manual for information on those kits.

3. Inverter Programming

Next, only the CFW-11 frequency inverter parameters related to the serial communication will be presented.

3.1 Symbols for the Proprieties Description

RO	Reading only parameter
CFG	Parameter that can be changed only with a stopped motor.
Net	Parameter visible on the HMI if the inverter has the network interface installed – RS232, RS485, CAN, Anybus-CC, Profibus – or if the USB interface is connected.
Serial	Parameter visible on the HMI if the inverter has the RS232 or RS485 interface installed.
USB	Parameter visible on the HMI if the inverter USB interface is connected.

P0105 – 1st/2nd Ramp Selection

P0220 – LOCAL/REMOTE Selection Source

P0221 – Speed Reference Selection – LOCAL Situation

P0222 – Speed Reference Selection – REMOTE Situation

P0223 – FORWARD/REVERSE Selection – LOCAL Situation

P0224 – Run/Stop Selection - LOCAL Situation

P0225 – JOG Selection - LOCAL Situation

P0226 – FORWARD/REVERSE Selection – REMOTE Situation

P0227 – Run/Stop Selection - REMOTE Situation

P0228 – JOG Selection - REMOTE Situation

These parameters are used in the configuration of the command source for the CFW-11 inverter local and remote situations. In order that the inverter be controlled through the Serial interface, one of the the options "Serial/USB" available in these parameters, must be selected.

The detailed description of these parameters is found in the CFW-11 Programming Manual.

P0308 – Serial Address

Range:	1 to 247	Default:	1
Proprieties:	CFG, Serial		
Access groups via HMI:			

01 PARAMETER GROUPS
  ∟ 49 Communication
    ∟ 113 Serial RS232 / 485

Description:
It allows the programming of the address used for the inverter serial communication. It is necessary that each device in the network has an address different from all the others. The valid addresses for this parameter depend on the protocol programmed in P0312:

☑ P0312 = 1 (TP) → Valid addresses: 1 to 30.
☑ P0312 = 2 (Modbus-RTU) → Valid addresses: 1 to 247.

P0310 – Serial Communication Rate

Range: 0 = 9600 bits/s
1 = 19200 bits/s
2 = 38400 bits/s
3 = 57600 bits/s

Default: 0

Proprieties: CFG, Serial

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 113 Serial RS232 / 485

Description:
It allows the programming of the wished communication rate for the serial interface, in bits per second. This rate must be the same for all the devices connected to the network.

P0311 – Serial Interface Byte Configuration

Range: 0 = 8 data bits, no parity, 1 stop bit
1 = 8 data bits, parity even, 1 stop bit
2 = 8 data bits, parity odd, 1 stop bit
3 = 8 data bits, no parity, 2 stop bits
4 = 8 data bits, parity even, 2 stop bits
5 = 8 data bits, parity odd, 2 stop bits

Default: 0

Proprieties: CFG, Serial

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 113 Serial RS232 / 485

Description:
It allows the programming of the number of data bits, parity and *stop* bits of the serial interface bytes. This configuration must be identical for all the devices connected to the network.

P0312 – Serial Protocol

Range: 1 = TP
2 = Modbus-RTU

Default: 2

Proprieties: CFG, Serial

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 113 Serial RS232 / 485

Description:
It allows the selection of the desired protocol for the serial interface. The detailed description of those protocols appears in the next topics of this manual.

P0313 – Action in Case of Communication Error

Range: 0 = Inactive
1 = Disable via Start/Stop
2 = Disable via General Enable
3 = Change to Local
4 = Change to LOCAL keeping the commands and the reference
5 = Fault trip

Default: 0

Proprieties: CFG, Net

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 111 Status/Commands

Description:
It allows the selection of the action to be executed by the inverter when a communication error is detected.

Table 3.1 *- Parameter P0313 options*

Options	Description
0 = Inactive	No action is taken and the inverter remains in the existing status.
1 = Disable via Start/Stop	A stop command with deceleration ramp is executed and the motor stops according to the programmed deceleration ramp.
2 = Disable via General Enable	The inverter is disabled by removing the General Enabling and the motor coasts to stop.
3 = Change to Local	The inverter commands change to Local.
4 = Change to LOCAL keeping the commands and the reference	The inverter is changed to the local mode; However, the enabling and reference commands received via the network , in case the inverter had been programmed for start/stop via HMI or 3-wire and reference via HMI or electronic potentiometer, are kept in the local mode.
5 = Fault Trip	Instead of an alarm, a communication error causes a fault at the inverter, so that it becomes necessary to perform the inverter fault reset in order to get it back to normal operation.

It is considered a communication error for the serial interface only the *Timeout* event – A128 alarm/F228 fault. This *Timeout* is programmed via the P0314 parameter.

The actions described in this parameter are executed by means of the automatic writing of the respective bits on the control via serial/USB parameter – P0682. In order to be effective, it is necessary that the inverter be programmed to be controlled via serial. This programming is done by means of parameters P0220 to P0228.

P0314 – Serial Watchdog

Range: 0.0 … 999.0 s **Default:** 0.0 s

Proprieties: CFG, Serial

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 113 Serial RS232/485

Description:
It allows the programming of a time limit for the detection of serial interface communication error. In case the inverter remains without receiving valid telegrams longer than the time programmed in this parameter, it will be considered that a communication error happened, the alarm A128 will be showed on the HMI (or F228 fault, depending on the programming done at P0313) and the option programmed in P0313 will be executed.

After being powered up, the inverter starts counting this time from the first received valid telegram. The value 0.0 disables this function.

P0316 – Serial Interface Status

Range: 0 = Inactive
1 = Active
2 = *Watchdog* error

Default: -

Proprieties: RO

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 113 Serial RS232/485

Description:
It makes it possible to establish if the RS232 or RS485 serial interface board is properly installed and if the serial communication presents errors.

Table 3.2 *- P0316 parameter values*

Options	Description
0 = Inactive	Serial interface inactive. It occurs when the inverter does not have the RS232/RS485 board installed.
1 = Active	RS232/RS485 interface board installed and acknowledged.
2 = *Watchdog* Error	Active serial interface, but a serial communication error has been detected – Alarm A128/Fault F228.

P0680 – Logic Status

Range: 0000h … FFFFh

Default: -

Proprieties: RO

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 111 Status/Commands

Description:
It allows the inverter status monitoring. Each bit represents a specific status:

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3 to 0
Function	Fault condition	Manual/ Automatic	Undervoltage	LOC/REM	JOG	Speed Direction	General Enabling active	Ramp enabled	In Alarm condition	In configuration mode	Second Ramp	Quick Stop Activated	Reserved

Table 3.3 *- Parameter P0680 functions*

Bits	Values
Bits 0 to 4	Reserved.
Bit 4 Quick Stop Activated	**0:** Quick stop command is not activated. **1:** Inverter is executing quick stop command.
Bit 5 Second Ramp	**0:** The inverter is configured to use as acceleration and deceleration ramp for the motor, the first ramp, programmed at the parameters P0100 and P0101. **1:** The inverter is configured to use as acceleration and deceleration ramp for the motor, the second ramp, programmed at the parameters P0102 and P0103.
Bit 6 In configuration mode	**0:** Inverter operating normally. **1:** Inverter in configuration mode. Indicates a special condition when the inverter cannot be enabled: ☑ Executing the self tuning routine. ☑ Executing guided start-up routine. ☑ Executing the HMI copy function. ☑ Executing the flash memory card guided routine. ☑ There is a parameter setting incompatibility. ☑ Without power supply at the inverter power section. **Note:** It is possible to obtain the exact description of the special operation mode at parameter P0692.
Bit 7 Alarm condition	**0:** The inverter is not in alarm condition. **1:** The inverter is in alarm condition. **Note:** The alarm number can be read by means of the parameter P0048 – Current Alarm.
Bit 8 Ramp Enabled (RUN)	**0:** The motor is stopped. **1:** The inverter is driving the motor at the set point speed, or executing either the acceleration or the deceleration ramp.
Bit 9 General Enabling active	**0:** General Enabling is not active. **1:** General enabling is active and the inverter is ready to run the motor.
Bit 10 Speed Direction	**0:** The motor is rotating in reverse mode. **1:** The motor is rotating in direct mode.
Bit 11 JOG	**0:** JOG function inactive. **1:** JOG function active.
Bit 12 LOC/REM	**0:** Inverter in Local mode. **1:** Inverter in Remote mode.
Bit 13 Undervoltage	**0:** No Undervoltage. **1:** With Undervoltage.
Bit 14 Manual/ Automatic	**0:** PID in manual mode. **1:** PID in Automatic mode.
Bit 15 Fault condition	**0:** The inverter is not in a fault condition. **1:** Any fault has been recorded by the inverter. **Note:** The fault number can be read by means of the parameter P0049 – Current Fault.

P0681 – Motor Speed in 13 bits

Range: -32768 … 32767 **Default:** -

Proprieties: RO

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
  ∟ 111 Status/Commands

Description:

It allows monitoring the motor speed. This word uses 13 bit resolution with signal to represent the motor synchronous speed:

☑ P0681 = 0000h (0 decimal) → motor speed = 0 rpm
☑ P0681 = 2000h (8192 decimal) → motor speed = synchronous speed

Intermediate or higher speed values in rpm can be obtained by using this scale. E.g. for a 4 pole 1800 rpm synchronous speed motor, if the value read is 2048 (0800h), then, to obtain the speed in rpm one must calculate:

8192 – 1800 rpm
2048 – value read in P0681

$$\text{Speed in rpm} = \frac{1800 \times 2048}{8192}$$

Speed in rpm = 450 rpm

Negative values in this parameter indicate motor rotating in counterclockwise sense of rotation.

P0682 – Control Word via Serial / USB

Range: 0000h … FFFFh **Default:** 0000h

Proprieties: Serial, USB

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 111 Status/Commands

Description:
It is the inverter serial interface Control word. This parameter can only be changed via serial interface or via USB. For the other sources (HMI, CAN, etc.) it behaves like a reading only parameter.

In order to have those commands executed, it is necessary that the inverter be programmed to be controlled via serial. This programming is done by means of parameters P0105 and P0220 to P0228.

Each bit of this word represents an inverter command that can be executed.

Bits	15 to 8	7	6	5	4	3	2	1	0
Function	Reserved	Fault reset	Quick Stop	Second Ramp Use	LOC/REM	JOG	Direction of Rotation	General Enabling	Start/Stop

Table 3.4 *- Parameter P0682 bit functions*

Bits	Values
Bit 0 Start/Stop	**0:** It stops the motor with deceleration ramp. **1:** The motor runs according to the acceleration ramp until reaching the speed reference value.
Bit 1 General Enabling	**0:** It disables the inverter, interrupting the supply for the motor. **1:** It enables the inverter allowing the motor operation.
Bit 2 Direction of Rotation	**0:** To run the motor in a direction opposed to the speed reference. **1:** To run the motor in the direction indicated by the speed reference.
Bit 3 JOG	**0:** It disables the JOG function. **1:** It enables the JOG function.
Bit 4 LOC/REM	**0:** The inverter goes to the Local mode. **1:** The inverter goes to the Remote mode.
Bit 5 Second Ramp Use	**0:** The inverter uses as acceleration and deceleration ramp for the motor, the first ramp times, programmed at the parameters P0100 and P0101. **1:** The inverter uses as acceleration and deceleration ramp for the motor, the second ramp times, programmed at the parameters P0102 and P0103.
Bit 6 Quick Stop	**0:** Quick Stop command not activated. **1:** Quick Stop command activated. **Obs.:** when the control modes V/f or VVW are selected, the use of this function is not recommended.
Bit 7 Fault reset	**0:** No function. **1:** If in a fault condition, then it executes the inverter reset.
Bits 8 to 15	Reserved.

P0683 – Speed Reference via Serial/ USB

Range:	-32768 … 32767	**Default:**	0
Proprieties:	Serial, USB		

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 111 Status/Commands

Description:
It allows the programming of the speed reference for the inverter via the serial interface. This parameter can only be changed via serial interface or via USB. For the other sources (HMI, CAN, etc.) it behaves like a reading only parameter.

In order to have the reference written in this parameter working, it is necessary that the inverter be programmed to use the speed reference via serial. This programming is done by means of parameters P0221 and P0222.

This word uses a 13 bit resolution with signal to represent the motor speed:

☑ P0683 = 0000h (0 decimal) → speed reference = 0 rpm
☑ P0683 = 2000h (8192 decimal) → speed reference = synchronous speed

Intermediate or higher speed reference values can be programmed by using this scale. E.g. for a 4 pole 1800 rpm synchronous speed motor, to obtain a speed reference of 900 rpm one must calculate:

1800 rpm – 8192
900 rpm – 13 bit reference

$$13 \text{ bit reference} = \frac{900 \times 8192}{1800}$$

13 bit reference = 4096 (value corresponding to 900 rpm in a 13 bit scale)

This parameter also accepts negative values to revert the motor speed direction. The reference speed direction, however, depends also on the control word bit 2 setting – P0682:

☑ Bit 2 = 1 and P0683 > 0: reference for direct speed rotation
☑ Bit 2 = 1 and P0683 < 0: reference for reverse speed rotation
☑ Bit 2 = 0 and P0683 > 0: reference for reverse speed rotation
☑ Bit 2 = 0 and P0683 < 0: reference for direct speed rotation

P0695 – Settings for the Digital Outputs

Range:	0000h … FFFFh	**Default:**	0000h
Proprieties:	Net		

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 111 Status/Commands

Description:
It allows the control of the digital outputs by means of the network interfaces (Serial, USB, CAN, etc.). This parameter cannot be changed via HMI.

Each bit of this parameter corresponds to the desired value for a digital output. In order to have the correspondent digital output controlled according to this content, it is necessary that its function be programmed for "P0695 Content" at parameters P0275 to P0280.

Bits	15 to 5	4	3	2	1	0
Function	Reserved	Setting for DO5	Setting for DO4	Setting for DO3 (RL3)	Setting for DO2 (RL2)	Setting for DO1 (RL1)

***Table 3.5** - P0695 parameter bit functions*

Bits	Values
Bit 0 Setting for DO1 (RL1)	**0:** DO1 output open. **1:** DO1 output closed.
Bit 1 Setting for DO2 (RL2)	**0:** DO2 output open. **1:** DO2 output closed.
Bit 2 Setting for DO3 (RL3)	**0:** DO3 output open. **1:** DO3 output closed.
Bit 3 Setting for DO4	**0:** DO4 output open. **1:** DO4 output closed.
Bit 4 Setting for DO5	**0:** DO5 output open. **1:** DO5 output closed.
Bits 5 to 15	Reserved

P0696 – Value 1 for Analog Outputs

P0697 – Value 2 for Analog Outputs

P0698 – Value 3 for Analog Outputs

P0699 – Value 4 for Analog Outputs

Range: -32768 … 32767 **Default:** 0

Proprieties: Net

Access groups via HMI:

01 PARAMETER GROUPS
∟ 49 Communication
∟ 111 Status/Commands

Description:
It allows the control of the analog outputs by means of network interfaces (Serial, USB, CAN, etc.) This parameter cannot be changed via HMI.

The value written in those parameters is used as the analog output value, providing that the function for the desired analog output be programmed for “Content P0696/ P0697/ P0698/ P0699”, in the parameters P0251, P0254, P0257 or P0260.

The value must be written in a 15 bit scale (7FFFh = 32767)[2] to represent 100% of the output desired value, i.e.:

☑ P0696 = 0000h (0 decimal) → analog output value = 0 %
☑ P0696 = 7FFFh (32767 decimal) → analog output value = 100 %

The showed example was for P0696, but the same scale is also used for the parameters P0697/P0698/P0699. For instance, to control the analog output 1 via serial, the following programming must be done:

☑ Choose a parameter from P0696 to P0699 to be the value used by the analog output 1. For this example we are going to select P0696.
☑ Program the option “Content P0696” as the function for the analog output 1 in P0254.
☑ Using the serial interface, write in P0696 the desired value for the analog output 1, between 0 and 100%, according to the parameter scale.

[2] Refer to the CFW-11 manual for knowing the actual output resolution.

NOTE!
If the analog output is programmed for working from -10 V to +10 V, negative values must be programmed at the specific parameter, so that -32768 to 32767 represent a variation from -10 V to +10 V at the output.

4 TP Protocol

The TP Protocol was developed with the purpose of making it possible the communication with the TP line PLC's. However, due to its flexibility and easiness to use, it has been used in other applications, many times implemented in PLC's and other systems for controlling and monitoring WEG equipment

4.1 Protocol FIELDS

☑ **SXT:** "Start of Transmission" byte. Value: 02h; 2 decimal.
☑ **ETX:** "End of Transmission" byte. Value: 03h; 3 decimal.
☑ **ADR:** Byte of the inverter address in the network, programmable via P0308.
Value Range: from 41h; 65 decimal; 'A' (ASCII) to 5Eh; 94 decimal; '^' (ASCII), representing the addresses 1 … 30 in the network.
Special 1: 40h; 64 decimal; '@' (ASCII) → It allows the writing or reading of all the equipments connected to the network.
Special 2: 5Fh; 95 decimal; '_' (ASCII) → It allows ONLY writing in all the equipments connected to the network, without acceptance or rejection answer.

☑ **COD:** Telegram code byte
Reading: 3Ch (hexadecimal); 60 (decimal); '<' (ASCII)
Writing: 3Dh (hexadecimal); 61 (decimal); '=' (ASCII) without saving the parameter in the EEPROM
Writing: 3Eh (hexadecimal); 62 (decimal); '>' (ASCII) saving the parameter in the EEPROM

☑ **BCC:** Telegram longitudinal checksum byte, EXCLUSIVE OR (XOR) between all the telegram bytes. With the size of 1 byte (00h to FFh hexadecimal)

☑ **DMW:** "Data Master Write". Those are the 4 writing bytes the master sends to the slave, the first 2 represent the parameter number and the last 2 the value to be written in that parameter.
PHi: Byte representing the parameter number high portion.
PLo: Byte representing the parameter number low portion.
VHi: Byte representing the parameter content high portion.
VLo: Byte representing the parameter content low portion.
E.g.: To write 1FFFh in the speed reference (P0683) → PHi = 02h, PLo = ABh, VHi = 1Fh, VLo = FFh.

☑ **DMR:** "Data Master Read". Those are the 2 reading bytes the master sends to the slave representing the parameter to be read.
PHi: Byte representing the parameter number high portion.
PLo: Byte representing the parameter number low portion.
E.g.: to read the value of the output voltage parameter (P0007) → PHi = 00h, PLo = 07h.

☑ **NUM:** This is the byte that represents the number of DMW's or DMR's to be transmitted, according to the telegram COD.
Range: 1 ... 6 (decimal).

☑ **DSV:** "Data Slave Value". Those are the 2 bytes the slave sends to the master, after a request from a master reading parameter, representing the content of the requested parameter.
VHi: Byte representing the high portion of the parameter to be written.
VLo: Byte representing the low portion of the parameter to be written.
Eg.: Answer to the inverter logic status parameter (P0680) reading request → VHi = 13h, VLo = 00h.

☑ **ACK:** Slave acknowledgment byte after a master writing. Value: 06h; 6 decimal.

☑ **NAK:** Slave rejection byte after a master reading or writing. It may occur when the master request a reading or writing from an inexistent parameter, or the value to be written is out of the permitted value range.
Value: 15h; 21 decimal.

4.2 Telegram Format

Next, the format of the telegrams for reading and writing in the parameters will be presented. It is important to realize that each telegram in the TP protocol allows the reading or writing of up to 6 parameters each time. Telegrams with format error or incorrect BCC will be ignored by the inverter, which will not send an answer to the master.

4.2.1 Reading Telegram

Master:

STX	ADR	COD	NUM	**DMR**	...	**DMR**	ETX	BCC

☑ COD: code for reading → 3Ch (hexadecimal); 60 (decimal); '<' (ASCII)
☑ NUM: number of read parameters. Range from 1 ... 6.
☑ DMR: number of the requested parameter. The number of DMR's must be equal to the value configured in the NUM byte.

Slave (CFW-11):

ADR	**DSV**	...	**DSV**	BCC	or	ADR	NAK

☑ DSV: value of the requested parameter. The number of DSV's must be equal to the value configured in the NUM byte.

Remembering that:

DMR	
PHi	PLo

DSV	
VHi	VLo

4.2.2 Writing Telegram

Master:

STX	ADR	COD	NUM	**DMW**	...	**DMW**	ETX	BCC

☑ COD: code for writing
- 3Eh (hexadecimal); 62 (decimal); '>' (ASCII) → saving in the EEPROM
- 3Dh (hexadecimal); 61 (decimal); '=' (ASCII) → without saving in the EEPROM

☑ NUM: number of writing parameters. Range from 1 ... 6.
☑ DMW: number and content for the parameter. The number of DMW's must be equal to the value configured in the NUM byte.

Slave (CFW-11):

ADR	ACK	or	ADR	NAK

Remembering that:

DMW			
PHi	PLo	VHi	VLo

4.3 Example of Telegrams Using the TP Protocol

All the next examples consider that the inverter is programmed with the address 1 (P0308=1), consequently the field ADR is sent with the value 41h (refer to table I.1).

Example 1: the reading of two inverter parameters:
☑ Motor speed – P0002 (assuming P0002 at 1200rpm = 04B0h).
☑ Motor current – P0003 (assuming P0003 at 5,0A = 0032h).

Master:

02h	41h	3Ch	02h	00h	02h	00h	03h	03h	7Fh
STX	ADR	COD	NUM	DMR:P0002		DMR:P0003		ETX	BCC
				Parameter		Parameter			

Slave (CFW-11):

41h	04h	B0h	00h	32h	C7h
ADR	DSV:1200		DSV:50		BCC
	Value		Value		

<u>Example 2</u>: To program 6 parameters for the inverter operation:
- ☑ Writing telegram saving in the EEPROM.
- ☑ P0100 = 50 (100 in decimal = 0064h, 50 in decimal = 0032h)
- ☑ P0101 = 150 (101 in decimal = 0065h, 150 in decimal = 0096h)
- ☑ P0220 = 6 (220 in decimal = 00DChh, 6 in decimal = 0006h)
- ☑ P0222 = 9 (222 in decimal = 00DEh, 9 in decimal = 0009h)
- ☑ P0226 = 5 (226 in decimal = 00E2h, 5 in decimal = 0005h)
- ☑ P0227 = 2 (227 in decimal = 00E3h, 2 in decimal = 0002h)

Master (request):

02h	41h	3Eh	06h	00h	64h	00h	32h	00h	65h	00h	96h
STX	ADR	COD	NUM	DMW:P0100 = 50				DMW:P0101 = 150			
				Parameter		Value		Parameter		Value	

00h	DCh	00h	06h	00h	DEh	00h	09h	00h	E2h	00h	05h
DMW:P0220 = 6				DMW:P0222 = 9				DMW:P0226 = 5			
Parameter		Value		Parameter		Value		Parameter		Value	

00h	E3h	00h	02h	03h	D6h
DMW:P0227 = 2				ETX	BCC
Parameter		Value			

Slave (answer):

41h	06h
ADR	ACK

<u>Example 4</u>: the writing of the enabling command and the speed reference via serial:
- ☑ Writing telegram without saving in the EEPROM.
- ☑ P0682 = 0013h – control via serial with LOC/REM to Remote mode, General Enabling and Start/Stop (682 in decimal = 02AAh).
- ☑ P0683 = 1000h – Speed reference via serial programmed for half the motor synchronous speed (683 in decimal = 02ABh).

Master (request):

02h	41h	3Dh	02h	02h	AAh	00h	13h	02h	ABh	10h	00h	03h	7Dh
STX	ADR	COD	NUM	DMW:0682 = 0013h				DMW:P0683 = 1000h				ETX	BCC
				Parameter		Value		Parameter		Value			

Slave (answer):

41h	06h
ADR	ACK

5 MODBUS-RTU Protocol

The Modbus-RTU protocol was initially developed in 1979. Nowadays, it is a widely spread open protocol, used by several manufactures in many equipments. The CFW-11 inverter Modbus-RTU communication was developed based on the following documents:

- ☑ MODBUS Protocol Reference Guide Rev. J, MODICON, June 1996.
- ☑ MODBUS Application Protocol Specification, MODBUS.ORG, May 8th 2002.
- ☑ MODBUS over Serial Line, MODBUS.ORG, December 2nd 2002.

In those documents is defined the format of the messages used by the elements that are constituent parts of the Modbus network, the services (or functions) that can be made available, and also how those elements exchange data in the network.

5.1 Transmission Modes

Two transmission modes are defined in the protocol specification: ASCII and RTU. The modes define the way the message bytes are transmitted. It is not possible to use the two transmission modes in the same network.

The CFW-11 frequency inverter uses only the RTU mode for the telegram transmission. The bytes are transmitted in hexadecimal format and its configuration depends on the programming done by means of P0311.

5.2 RTU Mode Message Structure

The Modbus-RTU structure uses a master-slave system for message exchange. It allows up to 247 slaves, but only one master. Every communication begins with the master making a request to a slave, which answers to the master what has been asked. In both telegrams (request and answer), the used structure is the same: Address, Function Code, Data and CRC. Only the data field can have a variable size, depending on what is being requested.

Master (request telegram):

Address (1 byte)	Function (1 byte)	Request data field (n bytes)	CRC (2 bytes)

Slave (response telegram):

Address (1 byte)	Function (1 byte)	Answer data field (n bytes)	CRC (2 bytes)

5.2.1 Address

The master initiates the communication sending a byte with the address of the slave to which the message is destined. When sending the answer, the slave also initiates the telegram with its own address. The master can also send a message to the address 0 (zero), which means that the message is destined to all the slaves in the network (*broadcast*). In that case, no slave will answer to the master.

5.2.2 Function Code

This field also contains a single byte, where the master specifies the kind of service or function requested to the slave (reading, writing, etc.). According to the protocol, each function is used to access a specific type of data.

In the CFW-11, parameter related data is available as registers of the *holding* type (referenced starting from address 4000 or '4x').

5.2.3 Data Field

It is a variable size field. The format and contents of this field depend on the used function and the transmitted value. This field is described together with the function description (refer to item 5.4).

5.2.4 CRC

The last part of the telegram is the field for checking the transmission errors. The used method is the CRC-16 (Cycling Redundancy Check). This field is formed by two bytes; where first the least significant byte is transmitted (CRC-), and then the most significant (CRC+). The CRC calculation form is described in the protocol specification; however, information for its implementation is also supplied in the appendices B and C.

5.2.5 Time Between Messages

In the RTU mode there is no specific character that indicates the beginning or the end of a telegram. The indication of when a new message begins or when it ends is done by the absence of data transmission in the network, for a minimum period of 3.5 times the transmission time of a data byte (11 bits[3]). Thus, in case a telegram has initiated after the elapsing of this minimum time, the network elements will assume that the first received character represents the beginning of a new telegram. And in the same manner, the network elements will assume that the telegram has reached its end when after receiving the telegram elements, this time has elapsed again.

If during the transmission of a telegram the time between the bytes is longer than this minimum time, the telegram will be considered invalid because the inverter will discard the bytes already received and will mount a new telegram with the bytes that were being transmitted.

For communication rates higher than 19200 bits/s, the used times are the same as for that rate. The next table shows us the times for different communication transmission rates:

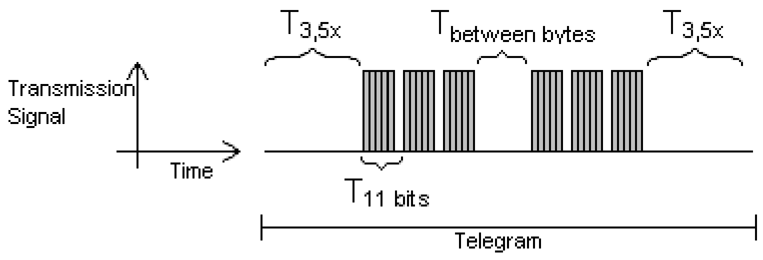


Table 5.1 - *Communication rates and the time periods involved in the telegram transmission*

Communication rate	$T_{11\ bits}$	$T_{3,5x}$
9600 bits/s	1,146 ms	4,010 ms
19200 bits/s	573 μs	2,005 ms
38400 bits/s	573 μs	2,005 ms
57600 bits/s	573 μs	2,005 ms

- ☑ $T_{11\ bits}$ = Time for transmitting one byte of the telegram.
- ☑ $T_{between\ bytes}$ = Time between bytes (must not be longer than $T_{3,5x}$).
- ☑ $T_{3,5x}$ = Minimum interval to indicated beginning and end of a telegram (3,5 x T_{11bits}).

5.3 CFW-11 Operation in the Modbus-RTU Network

The CFW-11 has the following characteristics when operated in Modbus-RTU network:

- ☑ Network connection via RS-232 or RS-485 serial interface (refer to item 2).
- ☑ Address, communication rate and byte format defined by means of parameters (refer to item 3).
- ☑ It allows the inverter programming and control via the access to parameters.

[3] The time of 11 bits is always considered as the time for the transmission of a byte, even if in the parameter P0311 a telegram format where each byte has only 10 bits be programmed.

5.3.1 Available Functions and Response Times

In the Modbus-RTU specification are defined the functions used to access different types of registers. In the CFW-11 the parameters have been defined as being *holding* type registers. In order to access those registers the following services (or functions) have been made available:

☑ *Read Coils*[4]
Description: reading of bit blocks of the *coil* type.
Function code: 01.

☑ *Read Discrete Inputs*[4]
Description: reading of bit blocks of the *discrete input* type.
Function code: 02.

☑ *Read Holding Registers*
Description: reading of register blocks of the *holding register* type.
Function code: 03.

☑ *Read Input Registers*[4]
Description: reading of register blocks of the *input register* type.
Function code: 04.

☑ *Write Single Coil*[4]
Description: writing in a single bit of the *coil* type.
Function code: 05.

☑ *Write Single Register*
Description: writing in a single register of the *holding* type.
Function code: 06.

☑ *Write Multiple Coils*[4]
Description: writing in bit blocks of the *coil* type.
Function code: 15.

☑ *Write Multiple Registers*
Description: writing in register blocks of the *holding register* type.
Function code: 16.

☑ *Read Device Identification*
Description: identification of the drive model.
Function code: 43.

The CFW-11 response time, from the end of transmission of the master until the response of the slave, ranges from 2 to 10 ms for any of the functions above.

5.3.2 Data Addressing and Offset

The CFW-11 data addressing is done with offset equal to zero, i.e., the address number is equivalent to the given number. The parameters are made available starting from the address 0 (zero). The next table illustrates the addressing of the parameters, which can be accessed as *holding* type registers:

[4] Functions used to get access to data used by the SoftPLC function.

Table 5.2 - *Modbus-RTU interface data address*

Parameters		
Parameter number	Modbus data address	
	Decimal	Hexadecimal
P0000	0	0000h
P0001	1	0001h
⋮	⋮	⋮
P0100	100	0064h
⋮	⋮	⋮

NOTE!

☑ All the parameters are treated as holding type registers. Depending on the master that is used, those registers are referenced starting from the base address 40000 or 4x. In this case, the address that must be programmed in the master for a parameter is the address showed in the table above added to the base address. Refer to the master documentation to find out how to access *holding* type registers.

☑ Besides the parameters, other types of data as bit markers, *word* or *float,* can also be accessed using the Modbus-RTU interface. Those markers are used mainly by the SoftPLC function, available for the CFW-11. Refer to the SoftPLC Manual for the description of those markers, as well as for the addresses via Modbus.

5.4 Detailed Description of the Functions

A detailed description of the functions available in the CFW-11 for the Modbus-RTU is provided in this section. In order to elaborate the telegrams it is important to observe the following:

☑ The values are always transmitted in hexadecimal.
☑ The address of a datum, the number of data and the value of registers are always represented in 16 bits. Therefore, it is necessary to transmit those fields using two bytes – superior (*high*) and inferior (*low*).
☑ The telegrams for request, as well as for response, cannot exceed 64 bytes.
☑ The transmitted values are always integer, regardless of having a representation with decimal point. Thus, the value 9.5 would be transmitted via serial as being 95 (5Fh). Refer to the CFW-11 parameter list to obtain the resolution used for each parameter.

5.4.1 Function 03 – Read Holding Register

It reads the content of a group of registers that must be necessarily in a numerical sequence. This function has the following structure for the reading and response telegrams (the values are always in hexadecimal and each field represents a byte):

Request (Master)	Response (Slave)
Slave Address	Slave Address
Function	Function
Address of the initial register (high byte)	Byte Count field
Address of the initial register (low byte)	Datum 1 (high)
Number of registers (high byte)	Datum 1 (low)
Number of registers (low byte)	Datum 2 (high)
CRC-	Datum 2 (low)
CRC+	etc...
	CRC-
	CRC+

Example 1: reading of the motor speed (P0002) and the motor current (P0003) of the CFW-11 located at the address 1 (assuming that P0002 = 1000 rpm and P0003 = 3.5 A).
☑ Address: 1 = 01h (1 byte)
☑ First parameter number: 2 = 0002h (2 bytes)
☑ Value of the fist parameter: 1000 = 03E8h (2 bytes)

☑ Value of the second parameter: 35 = 0023h (2 bytes)

Request (Master)		Response (Slave)	
Field	***Value***	***Field***	***Value***
Slave Address	01h	Slave Address	01h
Function	03h	Function	03h
Initial register (high)	00h	Byte Count	04h
Initial register (low)	02h	P002 (high)	03h
Number of registers (high)	00h	P002 (low)	E8h
Number of registers (low)	02h	P003 (high)	00h
CRC-	65h	P003 (low)	23h
CRC+	CBh	CRC-	3Bh
		CRC+	9Ah

5.4.2 Function 06 – Write Single Register

This function is used to write a value for a single register. It has the following structure (the values are always in hexadecimal and each field represents a byte):

Request (Master)	Response (Slave)
Slave Address	Slave Address
Function	Function
Register address (high byte)	Register address (high byte)
Register address (low byte)	Register address (low byte)
Register value (high byte)	Register value (high byte)
Register value (low byte)	Register value (low byte)
CRC-	CRC-
CRC+	CRC+

Example 2: writing of 900 rpm as the speed reference (P0683) (assuming a synchronous speed of 1800 rpm) for the CFW-11 located at the address 3.

☑ Address: 3 = 03h (1 byte)
☑ Parameter number: 683 = 02AB (2 bytes)
☑ Parameter value: 1000h (2 bytes)

Request (Master)		Response (Slave)	
Field	***Value***	***Field***	***Value***
Slave Address	03h	Slave Address	03h
Function	06h	Function	06h
Register (high)	02h	Register (high)	02h
Register (low)	ABh	Register (low)	ABh
Value (high)	10h	Value (high)	10h
Value (low)	00h	Value (low)	00h
CRC-	F5h	CRC-	F5h
CRC+	B0h	CRC+	B0h

Note that for this function the slave response is an identical copy of the request made by the master.
.

5.4.3 Function 16 – Write Multiple Registers

This function allows writing values for a group of registers, which must be in a numerical sequence. It can also be used to write in a single register (the values are always in hexadecimal and each field represents a byte):

Request (Master)	**Response (Slave)**
Slave Address	Slave Address
Function	Function
Address of the initial register (high byte)	Address of the initial register (high byte)
Address of the initial register (low byte)	Address of the initial register (low byte)
Number of registers (high byte)	Number of registers (high byte)
Number of registers (low byte)	Number of registers (low byte)
Byte Count field (Nr. of data bytes)	CRC-
Datum 1 (high)	CRC+
Datum 1 (low)	
Datum 2 (high)	
Datum 2 (low)	
etc...	
CRC-	
CRC+	

Example 3: writing of the acceleration time (P0100) equal to 1.0s and the deceleration time (P0101) equal to 2.0s, of a CFW-11 located at the address 15.

☑ Values converted to hexadecimal:
- Address: 15 = 0Fh (1 byte)
- First parameter number: 100 = 0064h (2 bytes)
- Value for the fist parameter: 10 = 000Ah (2 bytes)
- Value for the second parameter: 20 = 0014h (2 bytes)

Request (Master)		**Response (Slave)**	
Field	***Value***	***Field***	***Value***
Slave Address	0Fh	Slave Address	0Fh
Function	10h	Function	10h
Initial register (high)	00h	Register (high)	00h
Initial register (low)	64h	Register (low)	64h
Number of registers (high)	00h	Value (high)	00h
Number of registers (low)	02h	Value (low)	02h
Byte Count	04h	CRC-	01h
P100 (high)	00h	CRC+	39h
P100 (low)	0Ah		
P101 (high)	00h		
P101 (low)	14h		
CRC-	E0h		
CRC+	91h		

5.4.4 Function 43 – Read Device Identification

It is an auxiliary function that allows the reading of the product manufacturer, model and firmware version. It has the following structure:

Request (Master)	Response (Slave)
Slave Address	Slave Address
Function	Function
MEI Type	MEI Type
Reading code	Conformity Level
Object number	More Follows
CRC-	Next object
CRC+	Number of objects
	Code of the first object
	Size of the first object
	Value of the first object (n bytes)
	Code of the second object
	Size of the second object
	Value of the second object (n bytes)
	etc...
	CRC-
	CRC+

This function allows the reading of three information categories: Basic, Regular and Extended, and each category is formed by a group of objects. Each object is formed by a sequence of ASCII characters. For the CFW-11 only basic information formed by three objects is available:

- ☑ Object 00h – VendorName: always 'WEG'.
- ☑ Object 01h – ProductCode: Formed by the product code (CFW-11) plus the inverter rated voltage and current (e.g. 'CFW-11 220 - 230 V 10A / 8A').
- ☑ Object 02h – MajorMinorRevision: It indicates the inverter firmware version, in the format 'VX.XX'.

The reading code indicates what information categories are read, and if the objects are accessed in sequence or individually. The CFW-11 supports the codes 01 (basic information in sequence) and 04 (individual access to the objects). The other fields are specified by the protocol, and for the CFW11 they have fixed values.

Example 4: reading of basic information in sequence, starting from the object 01h, from a CFW-11 located at the address 1:

Request (Master)		Response (Slave)	
Field	***Value***	***Field***	***Value***
Slave Address	01h	Slave Address	01h
Function	2Bh	Function	2Bh
MEI Type	0Eh	MEI Type	0Eh
Reading code	01h	Reading code	01h
Object number	01h	Conformity Level	81h
CRC-	70h	More Follows	00h
CRC+	77h	Next object	00h
		Number of objects	02h
		Object code	01h
		Object size	1Bh
		Object value	`'CFW-11 220 - 230 V 10A / 8A'`
		Object code	02h
		Object size	05h
		Object value	`'V4.50'`
		CRC-	B2h
		CRC+	8Fh

In this example the value of the objects was not represented in hexadecimal, but using the corresponding ASCII characters instead. E.g.: for the object 02h, the value 'V4.50' was transmitted as being five ASCII characters, which in hexadecimal have the values 56h ('V'), 34h ('4'), 2Eh ('.'), 35h ('5') and 30h ('0').

5.4.5 Communication Errors

Communication errors may occur in the transmission of telegrams, as well as in the contents of the transmitted telegrams. Depending on the type of error, the CFW-11 may or not send a response to the master. When the master sends a message for an inverter configured in a specific network address, the inverter will not respond to the master if the following occurs:

☑ Parity bit error.
☑ CRC error.
☑ *Timeout* between the transmitted bytes (3.5 times the transmission time of a byte).

In those cases, the master must detect the occurrence of the error by means of the *timeout* while waiting for the slave response. In the event of a successful reception, during the treatment of the telegram, the inverter is able to detect problems and send an error message, indicating the kind of problem found:

☑ Invalid function (Error code = 1): The requested function has not been implemented for the equipment.
☑ Invalid datum address (Error code = 2): the datum address (parameter) does not exist.
☑ Invalid datum value (Error code = 3): It occurs in the following situations:
- The value is out of the permitted range.
- An attempt to write in a datum that cannot be changed (reading only register).

NOTE!
It is important that it be possible to identify at the master what type of error occurred, in order to be able to diagnose problems during the communication.

In the event of any of those errors, the slave must send a message to the master indicating the type of error that occurred. The error messages sent by the slave have the following structure:

Request (Master)	**Response (Slave)**
Slave Address	Slave Address
Function	Function (with the most significant bit in 1)
Data	Error code
CRC-	CRC-
CRC+	CRC+

Example 5: The master requests to the slave at the address 1 the writing in the parameter 99 (nonexistent parameter):

Request (Master)		**Response (Slave)**	
Field	***Value***	***Field***	***Value***
Slave Address	01h	Slave Address	01h
Function	06h	Function	86h
Register (high)	00h	Error code	02h
Register (low)	63h	CRC-	C3h
Value (high)	00h	CRC+	A1h
Value (low)	00h		
CRC-	79h		
CRC+	D4h		

6 Faults and Alarms Related to the Serial Communication

A128/F228 – Serial Communication Timeout

Description:
It is the only alarm/fault related to the serial communication. It indicates that the inverter has stopped receiving valid serial telegrams for a period longer than the programmed in P0314.

Working:
The parameter P0314 allows the programming of a time during which the inverter must receive at least one valid telegram via the RS-232 or RS-485 serial interface – with address and error checking field correct – otherwise, it will be considered that there was any problem in the serial communication. The time counting initiates after the reception of the first valid telegram. This function can be used by any serial protocol supported by the inverter.

After the timeout for serial communication is identified, the alarm A128 or the fault F228, depending on the P0313 programming, will be signalized through the HMI. In case of alarms, if the communication is reestablished and new valid telegrams are received, the alarm indication will be removed from the HMI.

Possible Causes/Correction:
- ☑ Verify factors that could cause failures in the communication (cables, installation, and grounding).
- ☑ Make sure that the master sends telegrams to the inverter in intervals shorter than the programmed in P0314.
- ☑ Disable this function in P0314.

I. APPENDICES

Appendix A. ASCII Table

Table I.1 *- ASCII characters*

Dec	Hex	Chr		Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
0	00	NUL	(Null char.)	32	20	Sp	64	40	@	96	60	`
1	01	SOH	(Start of Header)	33	21	!	65	41	A	97	61	a
2	02	STX	(Start of Text)	34	22	"	66	42	B	98	62	b
3	03	ETX	(End of Text)	35	23	#	67	43	C	99	63	c
4	04	EOT	(End of Transmission)	36	24	$	68	44	D	100	64	d
5	05	ENQ	(Enquiry)	37	25	%	69	45	E	101	65	e
6	06	ACK	(Acknowledgment)	38	26	&	70	46	F	102	66	f
7	07	BEL	(Bell)	39	27	'	71	47	G	103	67	g
8	08	BS	(Backspace)	40	28	(	72	48	H	104	68	h
9	09	HT	(Horizontal Tab)	41	29	)	73	49	I	105	69	i
10	0A	LF	(Line Feed)	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	(Vertical Tab)	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	(Form Feed)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	(Carriage Return)	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	(Shift Out)	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	(Shift In)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	(Data Link Escape)	48	30	0	80	50	P	112	70	p
17	11	DC1	(Device Control 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2	(Device Control 2)	50	32	2	82	52	R	114	72	r
19	13	DC3	(Device Control 3)	51	33	3	83	53	S	115	73	s
20	14	DC4	(Device Control 4)	52	34	4	84	54	T	116	74	t
21	15	NAK	(Negative Acknowledgement)	53	35	5	85	55	U	117	75	u
22	16	SYN	(Synchronous Idle)	54	36	6	86	56	V	118	76	v
23	17	ETB	(End of Trans. Block)	55	37	7	87	57	W	119	77	w
24	18	CAN	(Cancel)	56	38	8	88	58	X	120	78	x
25	19	EM	(End of Medium)	57	39	9	89	59	Y	121	79	y
26	1A	SUB	(Substitute)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	(Escape)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	(File Separator)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	(Group Separator)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	(Record Separator)	62	3E	>	94	5E	^	126	7E	~
31	1F	US	(Unit Separator)	63	3F	?	95	5F	_	127	7F	DEL

Appendix B. CRC Calculation Using Tables

Next, a function using programming language "C" is presented, which implements the CRC calculation for the Modbus-RTU protocol. The calculation uses two tables to supply pre-calculated values of the necessary displacement for the calculation. The algorithm was obtained from and is explained in the documents referred to in the item 5.

```
/* Table of CRC values for high-order byte */
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40 };

/* Table of CRC values for low-order byte */
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04,
0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8,
0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,
0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3, 0x11, 0xD1, 0xD0, 0x10,
0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,
0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C,
0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26, 0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0,
0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,
0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C,
0xB4, 0x74, 0x75, 0xB5, 0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54,
0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98,
0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80, 0x40 };

/* The function returns the CRC as a unsigned short type */
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg;                 /* message to calculate CRC upon    */
unsigned short usDataLen;               /* quantity of bytes in message     */
{
    unsigned char uchCRCHi = 0xFF;      /* high byte of CRC initialized     */
    unsigned char uchCRCLo = 0xFF;      /* low byte of CRC initialized      */
    unsigned uIndex;                    /* will index into CRC lookup table */
    while (usDataLen--)                 /* pass through message buffer      */
    {
        uIndex = uchCRCLo ^ *puchMsgg++; /* calculate the CRC               */
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex};
        uchCRCHi = auchCRCLo[uIndex];
    }

    return (uchCRCHi << 8 | uchCRCLo);
}
```

Appendix C. CRC Calculation Using Displacement of Registers

The algorithm for the calculation of the Modbus-RTU communication CRC using displacement of registers is described in this item. The algorithm was obtained from and is explained in the documents referred to in the item 5.

The CRC calculation is initiated by first loading a 16 bit variable (referenced from now on as CRC variable) with the value FFFFh. Afterwards the following routine is executed step by step:

1. The first byte of the message is submitted (only the data bits – start bit, parity and stop bit are not used) to an XOR (Exclusive OR) logic with the 8 less significant bits of the CRC variable, returning the result in the CRC variable itself.
2. Then, the CRC variable is shifted one position to the right, towards the less significant bit, and the position of the most significant bit is filled with 0 (zero).
3. After this shifting, the *flag* bit (the bit that was shifted out of the CRC variable) is analyzed, occurring the following:
   - ☑ If the bit value is 0 (zero), nothing is done,
   - ☑ If the value of the bit is 1, the content of the CRC variable is submitted to an XOR logic with a constant value of A001h and the result is returned to the CRC variable.
4. The steps 2 and 3 are repeated until eight shifts have been done.
5. The steps 1 to 4 are repeated using the next byte of the message, until all the message has been processed.

The final content of the CRC variable is the CRC field value that is transmitted at the end of the telegram. The less significant part is transmitted first (CRC-) and next the most significant part (CRC+).