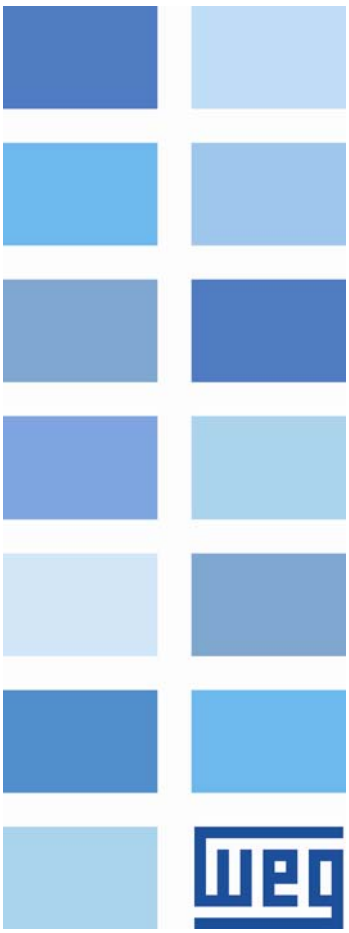
Modbus RTU

CFW700

User's Manual



Modbus RTU User's Manual

Series: CFW700

Language: English

Document Number: 10001123706 / 01

Publication Date: 05/2011

CONTENTS

ABOUT THIS MANUAL

This manual supplies the necessary information for the operation of the CFW700 frequency inverter using the RS232 and RS485 serial interfaces. This manual must be used together with the CFW700 user manual.

ABBREVIATIONS AND DEFINITIONS

ASCII	American Standard Code for Information Interchange
CRC	Cycling Redundancy Check
EIA	Electronic Industries Alliance
TIA	Telecommunications Industry Association
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 RS485 serial interface available for the product will be presented, as well as the Modbus RTU protocol for the use of this interface.

2 NETWORK CONNECTIONS

The CFW700 frequency inverter has a standard RS485 interface. Information about the connection and installation of the inverter to the network is presented bellow.

2.1 RS485

2.1.1 RS485 Interface Characteristics

- The interface follows the EIA-485 standard.
- It operates as a slave in the Modbus RTU network.
- It allows communication baud rates from 9600 up to 57600 Kbit/s.
- 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.

2.1.2 Connector pinout

The RS485 interface is available at the XC1 connector with the following connections:

Table 2.1: *RS485 connector pinout*

Pin	Name	Function
10	A-Line (-)	RxD/TxD negative
9	B-Line (+)	RxD/TxD positive
8	GND	0V isolated from the RS485 circuit

2.1.3 Terminating resistor

It is necessary to enable a terminating resistor at both ends of the main bus for each segment of the RS485 network. There are switches in the CFW700 frequency inverter that can be activated (by placing both switches S2 to ON) to enable the terminating resistor.

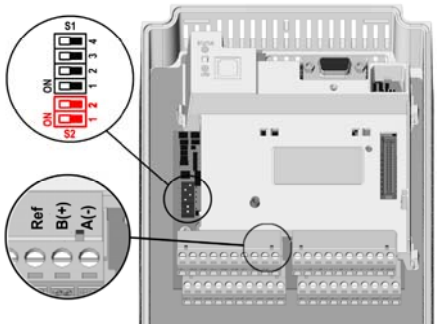


Figure 2.1: *Terminating Resistor and RS485 connector*

2.1.4 Connection with the RS485 Network

The following points must be observed for the connection of the device 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.

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

- Enable the termination resistors only at two points, at the extremes of the main bus, even if there are derivations from the bus.

3 PROGRAMMING

Next, the CFW700 frequency inverter parameters related to the Modbus RTU communication will be presented.

3.1 SYMBOLS FOR THE PROPERTIES DESCRIPTION

RO Reading only parameter
CFG Parameter that can be changed only with a stopped motor.

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 CFW700 frequency inverter local and remote situations. In order that the device be controlled through the Modbus RTU interface, the options 'serial' available in these parameters, must be selected.

The detailed description of these parameters is found in the CFW700 Programming Manual.

P0308 – SERIAL ADDRESS

Range:	1 to 247	**Default:** 1
Properties:	CFG	
Access groups via HMI:	NET	

Description:
It allows programming 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.

P0310 – SERIAL BAUD RATE

Range:	0 = 9600 bits/s 1 = 19200 bits/s 2 = 38400 bits/s 3 = 57600 bits/s	Default: 1
Properties:	CFG	
Access groups via HMI:	NET	

Description:
It allows programming the baud rate for the serial communication interface, in bits per second. This baud 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, even parity, 1 stop bit 2 = 8 data bits, odd parity, 1 stop bit 3 = 8 data bits, no parity, 2 stop bits 4 = 8 data bits, even parity, 2 stop bits 5 = 8 data bits, odd parity, 2 stop bits	**Default:** 0
Properties:	CFG	
Access groups via HMI:	NET	

Description:
It allows programming 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.

P0313 – COMMUNICATION ERROR ACTION

Range:	0 = Inactive 1 = Disable via Run/Stop 2 = Disable via General Enable 3 = Change to Local 4 = Change to Local keeping commands and reference 5 = Causes a Fault	**Default:** 0
Properties:	CFG	
Access groups via HMI:	NET	

Description:
It allows the selection of the action to be executed by the device, if it is controlled via network and a communication error is detected.

Table 3.1: *P0313 options*

Options	Description
0 = Inactive	No action is taken and the inverter remains in the existing status.
1 = Disable via Run/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 commands and reference	The inverter commands change to Local, but the status of the enabling and speed reference commands received via network are kept, providing that the inverter has been programmed to use in Local mode the commands via HMI and speed reference via either HMI.
5 = Causes a Fault	Instead of an alarm, the communication error causes an inverter fault, so that an inverter fault reset becomes necessary in order to restore normal operation.

The following events are considered communication errors:

Serial communication (RS485):
- A128 alarm/F228 fault: Serial communication timeout

The actions described in this parameter are executed by means of the automatic writing of the selected actions in the respective bits of the interface control words. Therefore, in order that the commands written in this parameter be effective, it is necessary that the device be programmed to be controlled via the used network interface. This programming is achieved by means of parameters P0220 to P0228.

P0314 – SERIAL WATCHDOG

Range: 0.0 to 999.0s **Default:** 0.0
Properties: CFG
Access groups via HMI: NET

Description:
It allows programming a time limit for the detection of serial interface communication error. If the inverter remains without receiving valid telegrams longer than the time programmed in this parameter, it will be considered that a communication error has occurred, the alarm A128 will be showed on the HMI 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 **Default:** -
1 = Active
2 = Watchdog error
Properties: RO
Access groups via HMI: NET

Description:
It allows identifying whether the RS485 serial interface board is properly installed, and whether the serial communication presents errors.

Table 3.2: *P0316 options*

Options	Description
0 = Inactive	Inactive serial interface. It occurs when the device does not have the RS485 board installed. Not used for CFW700.
1 = Active	Installed and acknowledged RS485 interface board.
2 = Watchdog error	The serial interface is active, but a serial communication error has been detected - A128 alarm/F228 fault.

P0680 – STATUS WORD

Range: 0000h to FFFFh **Default:** -
Properties: RO
Access groups via HMI: NET

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

Bits	15	14	13	12	11	10	9	8	7	6	5	4 to 0
Function	Fault condition	Reserved	Undervoltage	LOC/REM	JOG	Speed direction	Active General Enable	Motor Running	Alarm condition	In configuration mode	Second ramp	Reserved

Table 3.3: *P0680 parameter bit functions*

Bits	Values
Bits 0 to 3	Reserved
Bit 4 Active fast stop	0: The fast stop command is not active. 1: The drive is executing the fast stop command.
Bit 5 Second ramp	0: The drive is configured to use the first ramp values, programmed in P0100 and P0101, as the motor acceleration and deceleration ramp times. 1: The drive is configured to use the second ramp values, programmed in P0102 and P0103, as the motor acceleration and deceleration ramp times.
Bit 6 In configuration mode	0: The drive is operating normally. 1: The drive is in the configuration mode. It indicates a special condition during which the inverter cannot be enabled: ▪ Executing the self-tuning routine ▪ Executing the oriented start-up routine ▪ Executing the HMI copy function ▪ Executing the flash memory card self-guided routine ▪ There is a parameter setting incompatibility ▪ There is no power at the drive power section Note: It is possible to obtain the exact description of the special operation mode at the parameter P0692.
Bit 7 Alarm condition	0: The drive is not in alarm condition. 1: The drive is in alarm condition. Note: The alarm number can be read by means of the parameter P0048 – Present Alarm.
Bit 8 Motor Running	0: The motor is stopped. 1: The drive is running the motor at the set point speed, or executing either the acceleration or the deceleration ramp.
Bit 9 Active General Enable	0: General Enable is not active. 1: General Enable is active and the inverter is ready to run the motor.
Bit 10 Speed direction	0: The motor is running in the reverse direction. 1: The motor is running in the forward direction.
Bit 11 JOG	0: Inactive JOG function. 1: Active JOG function.
Bit 12 LOC/REM	0: Drive in Local mode. 1: Drive in Remote mode.
Bit 13 Undervoltage	0: No Undervoltage. 1: With Undervoltage.
Bit 14 Reserved	Reserved
Bit 15 Fault condition	0: The drive is not in a fault condition. 1: The drive has detected a fault. Note: The fault number can be read by means of the parameter P0049 – Present Fault.

P0681 – MOTOR SPEED IN 13 BITS

Range: - 32768 to 32767 **Default:** -
Properties: RO
Access groups via HMI: NET

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
- 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 => Speed in rpm

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

Speed in rpm = 450 rpm

Negative values in this parameter indicate that the motor is running in the reverse direction.

P0682 – SERIAL CONTROL WORD

Range: 0000h a FFFFh **Default:** 0000h
Properties: -
Access groups via HMI: NET

Description:
It is the device Modbus RTU interface control word. This parameter can only be changed via serial interface. For the other sources (HMI, etc.) it behaves like a read-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 achieved 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	Fast stop	Second ramp	LOC/REM	JOG	Speed direction	General enable	Run/Stop

Table 3.4: *P0682 parameter bit functions*

Bits	Values
Bit 0 Run/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 enable	0: It disables the drive, interrupting the supply for the motor. 1: It enables the drive allowing the motor operation.
Bit 2 Speed direction	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 drive goes to the Local mode. 1: The drive goes to the Remote mode.
Bit 5 Second ramp	0: The drive uses the first ramp values, programmed in P0100 and P0101, as the motor acceleration and deceleration ramp times. 1: The drive is configured to use the second ramp values, programmed in P0102 and P0103, as the motor acceleration and deceleration ramp times.
Bit 6 Fast stop	0: It does not execute the fast stop command. 1: It executes the fast stop command. Note: This function is not allowed with control types (P0202) V/f or VVW.
Bit 7 Fault reset	0: No function. 1: If in a fault condition, then it executes the drive reset.
Bits 8 to 15	Reserved.

P0683 – SERIAL SPEED REFERENCE

Range: -32768 a 32767 **Default:** 0
Properties: -
Access groups via HMI: NET

Description:
It allows programming the motor speed reference via the Modbus RTU interface. This parameter can only be changed via serial interface. For the other sources (HMI, etc.) it behaves like a read-only parameter.

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

This word uses a 13-bit resolution with signal to represent the motor synchronous speed.

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

Intermediate or higher 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

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

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 - P0682 - bit 2 setting:

- Bit 2 = 1 and P0683 > 0: reference for forward direction
- Bit 2 = 1 and P0683 < 0: reference for reverse direction
- Bit 2 = 0 and P0683 > 0: reference for reverse direction
- Bit 2 = 0 and P0683 < 0: reference for forward direction

P0695 – DIGITAL OUTPUT SETTING

Range: 0000h to 001Fh **Default:** 0000h
Properties: Net
Access groups via HMI: NET

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

Each bit of this parameter corresponds to the desired value for one 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 P0279.

Bits	15 to 5	4	3	2	1	0
Function	Reserved	DO5 setting	DO4 setting	DO3 setting	DO2 setting	DO1 setting

Table 3.5: *P0695 parameter bit functions*

Bits	Values
Bit 0 DO1 setting	0: DO1 output open. 1: DO1 output closed.
Bit 1 DO2 setting	0: DO2 output open. 1: DO2 output closed.
Bit 2 DO3 setting	0: DO3 output open. 1: DO3 output closed.
Bit 3 DO4 setting	0: DO4 output open. 1: DO4 output closed.
Bit 4 DO5 setting	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

Range:	-32768 to 32767	**Default:** 0
Properties:	RW	
Access groups via HMI:	NET	

Description:
They allow the control of the analog outputs by means of network interfaces (Serial, CAN, etc.) These parameters cannot be changed via HMI.

The value written in these parameters is used as the analog output value, providing that the function for the desired analog output be programmed for "P0696 / P0697 value", at the parameters P0251, P0254.

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. For instance, to control the analog output 1 via serial, the following programming must be done:

- Choose a parameter from P0696, P0697 to be the value used by the analog output 1. For this example, we are going to select P0696.
- Program the option "P0696 value" as the function for the analog output 1 in P0254.
- Using the network interface, write in P0696 the desired value for the analog output 1, between 0 and 100%, according to the parameter scale.

NOTE!
If the analog output is programmed for working from -10V to 10V, negative values for this parameter must be used to command the output with negative voltage values, i.e., -32768 to 32767 represent a variation from -10V to 10V at the analog output.

[2] Refer to the CFW700 manual for the product actual output resolution.

4 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 CFW700 frequency 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, December 28th 2006.
- MODBUS over Serial Line, MODBUS.ORG, December 20th 2006.

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.

4.1 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 CFW700 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.

4.2 MESSAGE STRUCTURE FOR RTU MODE

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 (n bytes)	CRC (2 bytes)

Slave (response telegram):

Address (1 byte)	Function (1 byte)	Response Data (n bytes)	CRC (2 bytes)

4.2.1 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.

4.2.2 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.

For the available list of supported functions, refer to item 5.

4.2.3 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).

4.2.4 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 Appendix B.

4.2.5 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). 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 frequency 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:

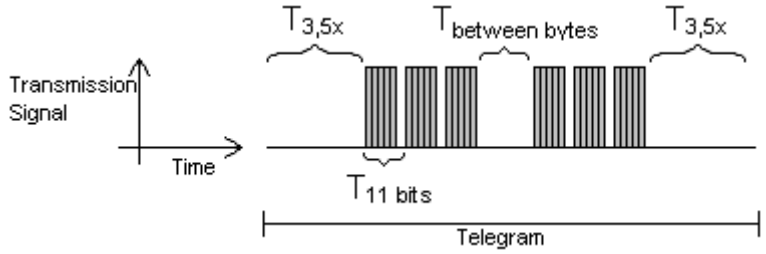


Table 4.1: *Communication rates and the time periods involved in the telegram transmission*

Baud rate	T11 bits	T3,5x
1200 bits/s	9.167 ms	32.083 ms
2400 bits/s	4.583 ms	16.042 ms
4800 bits/s	2.292 ms	8.021 ms
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.
- $T_{3,5x}$ = Minimum interval to indicated beginning and end of a telegram (3.5 x T11bits).

5 OPERATION IN THE MODBUS RTU NETWORK – SLAVE MODE

The CFW700 frequency inverter has the following characteristics when operated in Modbus RTU network:

- Network connection via RS485 serial interface.
- Address, communication rate and byte format defined by means of parameters.
- It allows the device programming and control via the access to parameters.

5.1 AVAILABLE FUNCTIONS AND RESPONSE TIMES

In the Modbus RTU specification are defined the functions used to access different types of data. In the CFW700 the parameters have been defined as being holding type registers. In order to access those data the following services (or functions) have been made available:

- Read Coils[3]
  Description: reading of bit blocks of the coil type.
  Function code: 01.

- Read Discrete Inputs[1]
  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[1]
  Description: reading of register blocks of the input register type.
  Function code: 04.

- Write Single Coil[1]
  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[1]
  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 device model.
  Function code: 43.

The 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.

5.2 MEMORY MAP

The CFW700 Modbus communication is based on the reading/writing of the equipment parameters. All the drive parameters list is made available as holding type 16-bit registers. The data addressing is done with the offset equal to zero, which means that the parameter number corresponds to the register number. The following table illustrates the parameters addressing, which can be accessed as holding type register.

[3] Functions used to access SoftPLC data.

Table 5.1: *Modbus RTU Memory Map*

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

It is necessary to know the inverter list of parameters to be able to operate the equipment. Thus, it is possible to identify what data are needed for the status monitoring and the control of the functions. The main parameters are:

Monitoring (reading):
- P0680 (holding register 680): Status word
- P0681 (holding register 681): Motor speed

Command (writing):
- P0682 (holding register 682): Command Word
- P0683 (holding register 683): Speed Reference

Refer to the Programming Manual for a complete parameter list of the equipment.

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.
- It should be noted that read-only parameters can only be read from the equipment, while other parameters can be read and written through the network.
- 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 CFW700. Refer to the SoftPLC Manual for the description of those markers, as well as for the addresses via Modbus.

6 DETAILED DESCRIPTION OF THE FUNCTIONS

A detailed description of the functions available in the CFW700 frequency inverter 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 CFW700 parameter list to obtain the resolution used for each parameter.

6.1 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 request and response telegrams (each field represents a byte):

Request (Master)	**Response (Slave)**
Slave Address	Slave Address
Function	Function
Address of the initial register (high byte)	Byte count
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: reading of the motor speed (P0002) and the motor current (P0003) of slave at address 1 (assuming that P0002 = 1000 rpm and P0003 = 3.5 A).

- Address: 1 = 01h (1 byte)
- Number of the first parameter: 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	P0002 (high)	03h
Number of registers (high)	00h	P0002 (low)	E8h
Number of registers (low)	02h	P0003 (high)	00h
CRC-	65h	P0003 (low)	23h
CRC+	CBh	CRC-	3Bh
		CRC+	9Ah

6.2 FUNCTION 06 – WRITE SINGLE REGISTER

This function is used to write a value for a single register. It has the following structure (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)
Value for the register (high byte)	Value for the register (high byte)
Value for the register (low byte)	Value for the register (low byte)
CRC-	CRC-
CRC+	CRC+

Example: writing of 900 rpm as the speed reference (P0683) (assuming a synchronous speed of 1800 rpm) for the slave at address 3.

- Address: 3 = 03h (1 byte)
- Parameter number: 683 = 02AB (2 bytes)
- Value for parameter: 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.

6.3 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 (each field represents a byte):

Request (Master)	**Response (Slave)**
Slave Address	Slave Address
Function	Function
Initial register address (high byte)	Initial register address (high byte)
Initial register address (low byte)	Initial register address (low byte)
Number of registers (high byte)	Number of registers (high byte)
Number of registers (low byte)	Number of registers (low byte)
Byte count (number of data bytes)	CRC-
Datum 1 (high)	CRC+
Datum 1 (low)	
Datum 2 (high)	
Datum 2 (low)	
etc...	
CRC-	
CRC+	

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

- 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	Initial register (high)	00h
Initial register (low)	64h	Initial register (low)	64h
Number of registers (high)	00h	Number of registers (high)	00h
Number of registers (low)	02h	Number of registers (low)	02h
Byte count	04h	CRC-	01h
P0100 (high)	00h	CRC+	39h
P0100 (low)	0Ah		
P0101 (high)	00h		
P0101 (low)	14h		
CRC-	E0h		
CRC+	91h		

6.4 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 CFW700 frequency inverter, only basic information formed by three objects is available:

- Objeto 00h – VendorName: represents the product manufacturer.
- Objeto 01h – ProductCode: formed by the product code (CFW700), plus the inverter rated voltage and current (ex. 'CFW700 220 - 230 V 10A').
- Objeto 02h – MajorMinorRevision: it indicates the product 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 CFW700 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 CFW700 they have fixed values.

Example: reading of basic information in sequence, starting from the object 02h, from a CFW700 at 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	02h	Conformity Level	81h
CRC-	70h	More Follows	00h
CRC+	77h	Next object	00h
		Number of objects	01h
		Object code	02h
		Object size	05h
		Object value	'V1.00'
		CRC-	3Ch
		CRC+	53h

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 'V1.00' was transmitted as being five ASCII characters, which in hexadecimal have the values 56h ('V'), 31h ('1'), 2Eh ('.'), 30h ('0') and 30h ('0').

6.5 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 slave 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 product 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 slave may 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 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/bit).

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: the master requests to the slave at the address 1 the writing in the register 2900 (nonexistent register):

Request (Master)		**Response (Slave)**	
Field	*Value*	*Field*	*Value*
Slave Address	01h	Slave Address	01h
Function	06h	Function	86h
Register (high)	0Bh	Error code	02h
Register (low)	54h	CRC-	C3h
Value (high)	00h	CRC+	A1h
Value (low)	00h		
CRC-	CAh		
CRC+	3Eh		

7 FAULTS AND ALARMS RELATED TO THE MODBUS RTU COMMUNICATION

A128/F228 – SERIAL COMMUNICATION TIMEOUT

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

Working:
The parameter P0314 allows the programming of a period during which the slave 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.

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 slave 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 4.

```
/* 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,
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);
}
```

WEG Equipamentos Elétricos S.A.
Jaraguá do Sul – SC – Brasil
Fone 55 (47) 3276-4000 – Fax 55 (47) 3276-4020
São Paulo – SP – Brasil
Fone 55 (11) 5053-2300 – Fax 55 (11) 5052-4212
automacao@weg.net
www.weg.net