

Novar-line PF-controller remote link communication description

Programmer handbook

11/2005

Power factor controllers of types Novar-106/114/206/214/314RS can be equipped with RS-232 or RS-485 remote link. Controller operation can be monitored and controlled over the remote link from supervising system (master, usually a PC).

This handbook describes the communication from point of view of application programmer. Basic knowledge of controller parameters and C-language syntax is supposed.

1.1 Data structures

The data interchanged between controller and master are organized to following structures :

- „Status“ ... contains actual controller state information (type, serial number, state of outputs, alarms, errors etc.); is read-only
- „EEStatus“ ... contains another state information such like maximum registered values, number of switching operations and switch-on time of all outputs etc.; is read-only
- „NovarStatus“ ... contains basic information of controller state and actual values of quantities measured; especially for on-line visualisation purposes; is read-only
- „Config“ ... contains actual state of controller parameters; can be both read and written
- „NovarSetMap“ ... virtual structure, that serves for selected controller functions' starting; is write-only

The master can get information of controller state by reading a structure. By writing to appropriate structure it can change any of parameters, start some of controller function etc.

Description of the structures follows in separate chapter below .

1.2 Communication protocols

The communication between a master and the controller (slave) runs over asynchronous serial link (COM) with RS-232 or RS-485 interface. If the RS-485 interface is used several controllers can be connected to a single link. At the master side a RS-232/RS-485 converter with automatic transmission direction switch can be used, or the converter direction need to be controlled by the master program.

The controllers are shipped with proprietary “KMB” protocol preset as default. Optionally the Modbus-RTU protocol can be set.

The communication rate can be set in range 300-9600 Bd (9600 Bd as default).

1.2.1 KMB communication protocol

The communication channel is set to 8 bits, no parity, one stop-bit.

Communication type is “Master-Slave”, i.e. after receiving message-command from master (PC) the controller processes it and transmits message-answer back to the master.

The message format is as follows :

1. Device address (1 byte)
2. Message length (in bytes) without final message checksum. (1 byte). It has the value of (3 + message body length).
3. Message type (1 byte).
4. Message body - its length differs each from other according the message type. Some messages can have no body (i.e. these messages' body length is zero).
5. Message checksum (1 byte) - sum of all preceding bytes modulo 256 (1 byte).

When receiving correct command, the slave (controller) processes it and transmits answer. If it can process the command successfully, the message-type value in the answer is cleared to zero. Otherwise the message-type value contains error-specification.

The slave must answer during 600 ms after receiving a command from the master, available gap of up to 4 bytes transmit time between bytes during transmitting can occur.

1.2.1.1 Message types

Following message types can be used for reading/writing data structures :

message no. (hex)	message type
0x14	Device Status Read - Status, EEStatus
0x16	Device Setting Read – Config
0x17	Device Setting Write - Config
0x30	Device Status Read – NovarStatus
0x31	Device Function Start – NovarSetMap Write

1.2.1.1.1 Device Status Read (Status, EEStatus) – 0x14

Message no. 0x14. The slave returns in answer the Status (34 bytes long) and the EEStatus (70 bytes) structures, i.e. 104 bytes.

Example :

Master must send following sequence of bytes (address value 1 is supposed) :

| 0x01| 0x03 | 0x14 | checksum = 0x18 |

The command has no body.

Answer :

| 0x01| 0x6B| 0x00 | message body = Status+EEStatus (104 bytes) | checksum |

(2nd byte, i.e. message length without checksum, is 104 + 3 = 107 = 0x6B).

1.2.1.1.2 Device Setting Read (Config) – 0x16

Message no. 0x16. The slave returns the Config structure (66 bytes).

Command :

| 0x01| 0x03 | 0x16 | checksum = 0x1A |

Answer :

| 0x01| 0x45| 0x00 | message body = Config (66 bytes) | checksum |

1.2.1.1.3 Device Setting Write (Config) – 0x17

Message no. 0x17. The master can write the Config structure (66 bytes) to slave with this command.

Command :

| 0x01| 0x45 | 0x17 | message body = Config (66 bytes) | checksum |

Answer :

| 0x01| 0x03| 0x00 | checksum = 0x04 |

Comment : DeviceAddr and RemoteBDRate variables cannot be set over communication link, the values written to the variables are arbitrary.

1.2.1.1.4 Device Status Read (NovarStatus) – 0x30

Message no. 0x30. The slave returns state information in the NovarStatus structure (35 bytes), necessary especially for on-line monitoring and visualisation purposes.

Command :

| 0x01| 0x03 | 0x30 | checksum = 0x34 |

Answer :

| 0x01| 0x26| 0x00 | message body = NovarStatus (35 bytes) | checksum |

1.2.1.1.5 Device Function Start (with NovarSetMap) – 0x31

Message no. 0x31. Some of controller functions can be started by writing to the NovarSetMap virtual structure (8 bytes). The functions are selected by setting appropriate values to 1.

Command :

| 0x01| 0x0B | 0x31 | message body = NovarSetMap (8 bytes) | checksum |

Answer :

| 0x01| 0x03| 0x00 | checksum = 0x04 |

1.2.2 Modbus-RTU protocol

Standard Modbus-RTU communication protocol can be used optionally for connection with master. Except slave address a and communication rate, parity bit function can be set (even / odd / none parity).

The slave must answer during 600 ms after receiving a command from the master, available gap of up to 1.5 bytes transmit time between bytes during transmitting can occur.

Broadcast-mode is not supported.

Implemented Modbus functions are listed in following table :

function no.	function	application
03	Read Holding Registers	Config read – registers 40101-40133 (addressed 100 – 132)
04	Read Input Registers	Status+EEStatus read – registers 30101-30152 (addressed 100 – 151) NovarStatus read – registers 30201-30218 (addressed 200 – 217)
06	Preset Single Register	Config write – registers 40101-40133 (addressed 100 – 132) NovarSetMap write – registers 40201-40203 (addressed 200 – 202)
08	Diagnostics – 00 – Return Query Data 01 – Restart Comm Option 02 – Return Diagnostic Register 04 – Force Listen Only Mode 10 – Clear Ctrs & Diag. Register 11 – Return Bus Message Count	basic diagnostics
16	Preset Multiple Registers	same as 06 - Preset Single Register
17	Report Slave ID	slave identification

The data structure access is available by reading/writing from/to appropriate registers (see table above). Every structure corresponds with appropriate continuous group of registers.

Example :

Controller state reading (NovarStatus) through „Read Input Registers“ function, address 1 supposed :

Command :

| 0x01| 0x04 | 0x00 | 0xC8| 0x00 | 0x12 | CRCLo | CRCHi |

NovarStatus is placed from input–register no. 30201(address 200 = 0xC8), structure length is 35 bytes = 18 registers (= 0x12 = 36 bytes, last byte has no meaning).

Response :

| 0x01| 0x04 | 0x24 | register 30201 Hi| reg. 30201 Lo| reg. 30202 Hi | reg. 30202 Lo |
..... | reg. 30218 Hi | reg. 30218 Lo | CRCLo | CRCHi |

Behind address (0x01) and function number (0x04) byte count (0x24 = 36) and the register contents follow. The contents corresponds with the NovarStatus as follows :

reg. 30201 Hi	-	SoftVersion –Hi
reg. 30201 Lo	-	SoftVersion –Lo
reg. 30202 Hi	-	DeviceNo - Hi
reg. 30202 Lo	-	DeviceNo – Lo
.....	-
reg. 30218 Hi	-	RegTime
reg. 30218 Lo	-	no meaning

(end of example)

1.3 Structures description

The syntax is conform to C-language. Long/ulong and int/uint variables are stored in high-low order (high byte is followed with low byte).

//=====

Status :

```
typedef struct {
    uchar   HWEError; /* hardware error info */
                /* bit0...EPROM error */
                /* bit1...RAM error */
                /* bit2...EEPROM error */
                /* bit3...calibration error */
    uchar   OutputSwitchNo[14]; /* number of switch-ons, lower values */
    uint    Event; /* actual nonstandard event info */
                /* bit0...1=undercurrent */
                /* bit1...1=overcurrent */
                /* bit2...1=out of compensation */
                /* bit3...1=no voltage */
                /* bit4...1=THD exceeded */
                /* bit5...1=switching nuber limit exceeded */
                /* bit6...1=no sense */
                /* bit7...1=reverse voltage */
                /* bit8...1=step error */

    uint    ActRelayState; /* actual relay state, 1=ON */
    uint    ReqRelayState; /* scheduled relay state */
    uchar   State; /* state */
                /* bit0-3...control process state */
                /* bit4...1= connection unknown */
                /* bit5...1= step values unknown */
    uint    AlarmSigActive; // active alarm signalling
    uint    AlarmActionActive; // active alarm action
                // both coded in the same way as „Event“
    uint    BadSteps; /* map of error steps (which are in standby mode) */
    uint    SoftVersion; /* lowbyte....software version */
                /* highbyte...special version*/
    uint    DeviceNo; /* serial number
    uint    DeviceType; /* device type :
                /* 2....NOVAR-314RS */
                /* 3....NOVAR-206 */
                /* 4....NOVAR-214 */
                /* 5....NOVAR-106 */
                /* 6....NOVAR-114 */

} SType; /* Status -34 bytes */
```

//=====

EEStatus :

```
typedef struct {
    uint    PrecisedSteps; /* precised steps map*/
                /* 1=precised step, 0=not precised yet */
    char    MinCos; /* minimum (1-minute)cos-value since last clearing */
    uchar   Res0; /* no sense
    uchar   MaxTHD; /* maximum (1-minute)THD value since last clearing */
                // MaxTHD coding:
                // 0 -100 ... 0.00 to 50.0 with 0,5% steps
                // 101-200 ... 52.5 to 300.0 with 2,5% steps
                // 201-250 ... 310 to 800% with 10% steps
                // 255 ... value undefined
    uchar   Res1; /* no sense
    uchar   MaxHar[6]; /* max,(1-min)3-5-7-11-13-17 harmonic value */
```

```

// MaxHar coding:
//      0 -100 ... 0.00 to 10.0 with 0,1% steps
//     101-200 ... 10.5 to 60.0 with 0,5% steps
//     201-254 ... 62.5 to 195.0 with 2,5% steps
//     255 ... value undefined
uint   OutputSwitchNo64[14]; // number of step switching in 64-units („higher“ values )
// OutputSwitchNo need to be added to get
// real number of switching operations
uint   ManualStepValue; /* step map in manual mode; 0=ON, 1=OFF */
uint   OutputSwitchOnTime2H[14]; /* switch-on time of each step */
// in 2-hour units */
// max. range 65000, i.e. 130000 hours */
} EEType; /* EEStatus - 70 bytes */
//=====

```

NovarStatus :

```

typedef struct {
uint   SoftVersion; /* low byte...software version */
// high byte...special version number(usually zero) */
uint   DeviceNo; /* serial number */
uint   DeviceType; /* device type, the same as in the Status */
uint   MTP; /* CT ratio, 1-6000 */
// bits 14-0...CT primary nominal value in 5A-units */
// bit 15...0= CT secondary nom. value 1A (CT-ratio xxx/1A) */
// 1= CT secondary nom. value 5A (CT-ratio xxx/5A) */
uchar  Fr; // no sense
uint   I; /* eff. current (on secondary side of the CT) in 0,25mA units*/
// as all current values, the Ck and step values, it must be */
// multiplied with the CT-ratio to get primary values */
uint   I50; /* fundamental harmonic current value in 0,25mA-units*/
int    Ir; /* fund. harmonic active component current value in 0,25mA-units*/
int    Ii; /* fund. harmonic reactive component current value in 0,25mA-units*/
// both signed, negative means export of active energy or */
// capacitive character of reactive energy */
int    Fi; // no sense
char   Kos; /* cos fi in 0,01-units, negative=capacitive character */
// if(Kos == 0) -> cos fi = 0.00L */
// . */
// if(Kos == 99) -> cos fi = 0.99L */
// if(Kos == 100) -> cos fi = 1.00 */
// if(Kos == -99) -> cos fi = 0.99C*/
// . */
// if(Kos == -1) -> cos fi = 0.01C */
// if(Kos == -100) -> cos fi = 0.00C*/
// . */
// if(Kos == 127)-> cos fi undefined (for ex. when I==0) */
uchar  THD; /* THD, for coding see EEStatus */
uchar  Har[6]; /* 3-5-7-11-13-17 harmonic component current values */
// for coding see EEStatus */
uchar  Res0; // reserve
uint   ActRelayState; /* output relay state map (bits 0 to 13), 1=on */
uchar  Res1; // reserve
uchar  Res2; // reserve
uchar  RegState; /* controller state */
// STATEMASK 0x0F /* lower 4 bits-controller state : */
// /* specifies actual controller state */
// STATEINIT 0x00 /* after-reset state */
// STATETEST 0x01 /* start-up test in progress */
// STATEUIMODERE 0x02 /* UIMode-recognition ("AP") in progress */
// STATEUIMODEUK 0x03 /* UIMode-unknown ("P=0") */
// STATECLVALUESRE 0x04 /* CLValues-recog. ("AC") in progress */
// STATECLVALUESUK 0x05 /* CLValues-unknown ("C=0") */
// STATERUN 0x06 /* normal controll in progress */

```

```

// STATESTANDBYCLOFF 0x07 /* standby-outputs OFF (excl. fixed-steps) */
// STATESTANDBYALLOFF 0x08 /* standby - all outputs OFF */
// STATEIDLE 0x09 /* idle - no measured data available */
// STATEMANUAL 0x0F /* manual mode */
/* upper 4 bits – non-standard states mask */
// STATEUIMODEUNKNOWN 0x10 /* UIMode unknown (“P=0”) */
// STATECLVALUESUNKNOWN 0x20 /* CLValues unknown (“C=0”) */
// STATEVOLTAGEBAD 0x40 /* voltage too low („U=0“) */
// STATECURRENTLOW 0x80 /* current too low („I=0“) */
uchar StateLEDs; /* panel indicating LED-diodes state */
/* bit0...TrendL */
/* bit1...TrendLFlash */
/* bit2...TrendC */
/* bit3...TrendCFlash */
/* bit4...PwrReverse */
/* bit5...Alarm */
/* bit6...reserve */
/* bit7...Error */
uchar RegTime; /* time to next control action in %; it decreases from 100 to 0 */
} NSType; /* NovarStatus-35 bytes */
//=====

```

Config :

```

typedef struct {
char ReqCos; /* target cos, range from -90 to +80 */
/* 0x7F... value unknown */
uchar SwitchDelayL; /* control period for undercompensation */
uchar SwitchDelayC; /* control period for overcompensation */
/* control periods coding : */
/* 0...5 sec */
/* 1...10 sec */
/* 2...15 sec */
/* 3...20 sec */
/* 4...30 sec */
/* 5...60 sec */
/* 6...120 sec */
/* 7...180 sec */
/* 8...300 sec */
/* 9...600 sec */
/* 10..1200 sec */
/* otherwise...value not valid */
/* since version 2.1 change: */
/* another bit 7...0= square-proportional control time */
/* ...1= linear-proportional control time */
char RegBandShift; /* no sense, set to 1 ! */
char RegBandLinLimit; /* no sense, set to 1 ! */
} RegParType;

typedef struct {
uchar RegMode; /* control mode setting */
/* bit0...0>manual, 1=automat */
/* bit1...0=evaluation of tarif2 input */
/* bit2...1=automatic step value recognition active */
/* bit3...1= password not retained- required*/
/* 0= password retained - not required*/
/* bit4...res. */
/* bit5...res. */
/* bit6...res. */
/* bit7...res. */
uchar Res0; // no sense
RegParType RegPar[2]; /* second set of values valid for 2nd tariff */
uint MTP; /* CT ratio 1-9950 */
/* bits 14-0...primary value in 5A-units */
/* bit 15...0= CT- nominal secondary current 1A */

```

```

/*      1= 5A      */
uchar  SwitchBlockDelay; /* reconnection block time */
/*      0...5  sec      */
/*      1...10 sec      */
/*      2...20 sec      */
/*      3...30 sec      */
/*      4...60 sec      */
/*      5...120 sec     */
/*      6...300 sec     */
/*      7...600 sec     */
/*      8...1200 sec    */
/*      otherwise...not valid */
uchar  UIMode; /* connection type */
/* bits 2-0: */
/* for bit 3=1, i.e. phase voltage : */
/*      1...U10 */
/*      2...U20 */
/*      3...U30 */
/*      4...U01 */
/*      5...U02 */
/*      6...U03 */
/* for bit 3=0, i.e. line voltage : */
/*      1...U12 */
/*      2...U23 */
/*      3...U31 */
/*      4..U21 */
/*      5..U32 */
/*      6..U13 */
/* bits 7-4 : */
/* if bits 0-3 out of range: */
/*      upper nibble=0.....aut. recognition unsuccessful */
/*      upper nibble=1-F...UIMode not set yet, automatic */
/*      recognition process will be started */
uchar  CSRatio; /* compensation step ratio */
/*      0x00...-individual settings */
/*      1...1:1:1:1:1 */
/*      2...1:1:2:2:2 */
/*      3...1:1:2:2:4 */
/*      4...1:1:2:3:3 */
/*      5...1:1:2:4:4 */
/*      6...1:1:2:4:8 */
/*      7...1:2:2:2:2 */
/*      8...1:2:3:3:3 */
/*      9...1:2:3:4:4 */
/*      10...1:2:3:6:6 */
/*      11...1:2:4:4:4 */
/*      12...1:2:4:8:8 */
uchar  Ck; /* from 0,02 to 2A in 0,01A-steps, coded in the same */
/* way as current */
uchar  Steps; /* number of inductive and capacitive steps used */
/* bits 3-0...CSteps; bits 7-4...LSteps */
uchar  QuickSteps; /* for Novar-314RS only: number of transistor group */
/* steps used; for other types without sense */
/* tranzistorove sekce; pro ostatni typy bez vyznamu */
int  CLVal[14]; /* step values*/
/* coded in the same way as current (see NovarStatus) */
/* capacitors-positive value, inductor-negative value */
/* value 0x7FFF...the step value is unknown */
uint  FixedSteps; /* bit map of fixed steps, 0=fixed step */
uint  FixedStepValue; /* step value of fixed steps; 0=ON, 1=OFF */
char  LCosMargin; /* limit cos-value for decompensation choke operation */
/* coded in the same way as the cos fi (see NovarStatus) */
uchar  QuickControlSpeed; /* for Novar-314RS only: transistor group control */
/* speed and reconnection block time */
// value  contr./sec  block. time [s]

```

```

// 0      1      10      (default)
// 1      1      5.0
// 2      1      2.0
// 3      1      1.0
// 4      2      10
// 5      2      5.0
// 6      2      2.0
// 7      2      1.0
// 8      2      0.5
// 9      3      10
// 10     3      5.0
// 11     3      2.0
// 12     3      1.0
// 13     3      0.6
// 14     3      0.3
// 15     4      10
// 16     4      5.0
// 17     4      2.0
// 18     4      1.0
// 19     4      0.7
// 20     4      0.5
// 21     4      0.2
// 22     5      10
// 23     5      5.0
// 24     5      2.0
// 25     5      1.0
// 26     5      0.8
// 27     5      0.6
// 28     5      0.4
// 29     5      0.2

uint AlarmSig; /* alarm signalling map */
/* bit0...0=undercurrent */
/* bit1...0=overcurrent */
/* bit2...0=compensation error */
/* bit3...0=no voltage */
/* bit4...0=THD exceeded */
/* bit5...0=switching number limit reached */
/* bit6...reserve*/
/* bit7...0=reverse energy*/
/* bit8...0=step error */

uint AlarmAction; /* alarm action map, similar to AlarmSig */
uchar THDLimit; /* THD limit value for alarm signalling/action */
/* coded in the same way as the THD (see EEStatus) */

uchar SwitchNoLimit; /* switch number limit value for alarm signalling */
/* from 10000 do 2000000, in 10000-units*/

uchar PWeight; /* no sense */
uchar QWeight; /* no sense*/
uchar DeviceAddr; /* address */
uchar RemoteBdRate; /* low nibble = Bd-rate */
/* 2...300 Bd */
/* 3...600 Bd */
/* 4..1200 Bd */
/* 5..2400 Bd */
/* 6..4800 Bd */
/* 7..9600 Bd */
/* high-nibble = protocol: */
/* bit7...reserve (0) */
/* bit6...0 = protocol KMB */
/* 1 = ModBus RTU */
/* bits 5,4...parity (for ModBus only) */
/* bit5.....0=no parity(i.e. 2stopbits)*/
/* 1=parity: */
/* bit4.....0=even */
/* 1=odd */

uint ConfigCRC; /* no sense, may be set to any value */

```

```

        } CType;      /* Config - 66 bytes */
//=====

NovarSetMap :

typedef struct {
    uchar ClearLimit;      /* maximum values clearing and mode setting (value 1 active) */
                           /* maximum values clearing :
                           // bit0= MinCos
                           // bit1= MaxTHD
                           // bit2= MaxHar
    uint ClearSwitchNo;    /* bit0-13... clear switching number of appropriate step
    uchar Switch;          /* controller action command
                           // bit0=lock editation(password will be required)
                           // bit1=go to control mode(if controller in manual mode)
                           // bit2=controller reinitialisation
                           // bit3=clear HWEError-info
    uint ClearSwitchOnTime; /* bit0-13... clear switch-on time of appropriate step
    } NovarSetMapType;    /* NovarSetMap - 8 bytes */
//=====

```