IO.HART

APLISENS

PRODUKCJA PRZETWORNIKÓW CIŚNIENIA
I APARATURY POMIAROWEJ

INSTRUKCJA

Implementacja warstw protokołu komunikacyjnego HART

dla przetworników APC..., APR...
i sond SG-25.SMART

WARSZAWA MAJ 2005r.

APLISENS Sp. z o.o., 03-192 Warszawa, ul.Morelowa 7,
tel. (0 22) 814-07-77fax (0 22) 814-07-78,
www.aplisens.pl, e-mail: aplisens@aplisens.pl

W 7 warstwowym modelu OSI protokół komunikacyjny **HART** zajmuje warstwę 1,2 i 7 (warstwa fizyczna, warstwa łącza danych i warstwa aplikacji)

1 warstwa OSI (fizyczna)

W przetwornikach APC-2000, APC-2000/AL, APR-2000, APR-2000/AL, SG-25.SMART warstwą fizyczną jest przewodowa pętla prądowa 4 – 20 mA z równoczesnym wykorzystaniem cyfrowej transmisji opartej o modulację FSK w standardzie BELL 202 (1200/2200 Hz.)

Implementacja warstwy jest zgodna z normą HART Communication Foundation Documents (**HCF_SPEC-54**) rev 8

2 warstwa OSI (łącza danych)

W przetwornikach APC-2000, APC-2000/AL, APR-2000, APR-2000/AL, SG-25.SMART warstwa łącza danych oparta jest o protokół typu Master/Slave, binarny, zorientowany bajtowo z przesyłaniem tokena. Implementacja warstwy jest zgodna z normą HART Communication Foundation Documents (**HCF_SPEC-81**) rev 7.1

7 warstwa OSI (aplikacji.)

W przetwornikach APC-2000, APC-2000/AL, APR-2000, APR-2000/AL, SG-25.SMART warstwa aplikacji zorientowana jest na komendy z predefiniowanym typem danych oraz predefiniowanymi procedurami aplikacyjnymi.

Komendy warstwy 7 OSI można podzielić na 3 grupy:

Typ **UNIVERSAL**	implementacja komend zgodna z normą HCF_SPEC-127 rev 5.2
Typ **COMMON PRACTICE**	implementacja komend zgodna z normą HCF_SPEC-151 rev 7.1
Typ **DEVICE SPECIFIC COMMAND**	opis komend w DTR APLISENS

Implementacja kodów statusów (COMMAND SPECIFIC RESPONSE CODE DEFINITIONS) jest zgodna z HCF_SPEC-307 rev 4.1

Wykaz zaimplementowanych komend (OSI 7 warstwa aplikacji.)

Nr komendy	Typ	Funkcja
UNIVERSAL COMMANDS		
0	UNIVERSAL	Read unique identifier
1	UNIVERSAL	Read primary variable
2	UNIVERSAL	Read current and percent of range
3	UNIVERSAL	Read current and four dynamics variables
6	UNIVERSAL	Write pooling address
11	UNIVERSAL	Read unique identifier associated with TAG
12	UNIVERSAL	Read message
13	UNIVERSAL	Read TAG, DESCRIPTOR, DATE
14	UNIVERSAL	Read PV sensor information
15	UNIVERSAL	Read output information
16	UNIVERSAL	Read final Assembly Number
17	UNIVERSAL	Write message
18	UNIVERSAL	Write tag, descriptor, date
19	UNIVERSAL	Write final assembly number
COMMON PRACTICE COMMANDS		
34	COMMON PRACTICE	Write PV damping value
35	COMMON PRACTICE	Set PV unit code for upper and lower range value
36	COMMON PRACTICE	Set PV upper range value
37	COMMON PRACTICE	Set PV lower range value
38	COMMON PRACTICE	Reset “ configuration changed “ flag
40	COMMON PRACTICE	Enter/exit PV current mode
42	COMMON PRACTICE	Perform master reset
43	COMMON PRACTICE	Set PV zero
44	COMMON PRACTICE	Write PV unit
45	COMMON PRACTICE	Trim PV current DAC zero
46	COMMON PRACTICE	Trim PV current DAC gain
47	COMMON PRACTICE	Write PV transfer function
48	COMMON PRACTICE	Read additional transmitter status
49	COMMON PRACTICE	Write PV sensor serial number
59	COMMON PRACTICE	Write numbers of response preambles
DEVICE SPECIFIC COMMANDS		
128	DEVICE SPECIFIC	Read static data materials
129	DEVICE SPECIFIC	Read device variable trim points
130	DEVICE SPECIFIC	Trim upper sensor calibration
131	DEVICE SPECIFIC	Trim lower sensor calibration
132	DEVICE SPECIFIC	Write control modes
133	DEVICE SPECIFIC	Read control modes
135	DEVICE SPECIFIC	Write coefficient for user characteristic
136	DEVICE SPECIFIC	Read coefficient for user characteristic
138	DEVICE SPECIFIC	Return to factory settings
141	DEVICE SPECIFIC	Write static data configurations
142	DEVICE SPECIFIC	Read static data configurations
231	DEVICE SPECIFIC	Read product codes
233	DEVICE SPECIFIC	Read separator codes
237	DEVICE SPECIFIC	Read operational limits
240	DEVICE SPECIFIC	Write long TAG
241	DEVICE SPECIFIC	Read long TAG
242	DEVICE SPECIFIC	Write start point rad
243	DEVICE SPECIFIC	Read start point rad
244	DEVICE SPECIFIC	Write user’s unit name and rearrange coefficients
245	DEVICE SPECIFIC	Read user’s unit name and rearrange coefficients
246	DEVICE SPECIFIC	Write customer’s security code
247	DEVICE SPECIFIC	Set write protect code

Opis zaimplementowanych komend typu DEVICE SPECIFIC COMMANDS

Komenda 128 Read static data materials

```
COMMAND read_static_data_material
{
NUMBER 128;
OPERATION READ;
TRANSACTION
{
        REQUEST
        {}
        REPLY
        {
                response_code,device_status,
                flange_type,
                flange_material,
                o_ring_material,
                meter_installation,
                drain_vent_material,
                remote_seal_type,
                siepres_remote_seal_fill_fluid,
                remote_seal_isolator_material,
                number_of_remote_seals,
                module_fill_fluid,
                module_isolator_material,
                module_type_code,
                module_range_code,
                pressure_units,
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                                    [no_command_specific_errors];
}
}
```

Komenda 129 Read device variable trim points

```
COMMAND read_last_trim_points
{
NUMBER 129;
OPERATION READ;
TRANSACTION
{
        REQUEST
        {}
        REPLY
        {
                pressure_units,
                sensor_lower_trim_point,        /* IEE754 float, last calibration pressure value */
                sensor_upper_trim_point,        /* IEE754 float, last calibration pressure value */
                current_units,
                current_lower_trim_point,       /* IEE754 float, last calibration current value */
                current_upper_trim_point        /* IEE754 float, last calibration current value */
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                            [no_command_specific_errors];
}
}
```

Komenda 130 Trim upper sensor calibration

```
COMMAND write_upper_sensor_trim
/* Uwaga!, wejście w procedurę kalibracji zeruje podkalibrowanie zera */
{
        NUMBER 130;
        OPERATION WRITE;
        TRANSACTION
{
        REQUEST
        {
                channel_number,                 /* channel_number = 0 or 1 /
                pressure_units,
                sensor_upper_trim_point         /* IEE754 float */
        }
        REPLY
        {
                response_code,device_status,
                channel_number,
                pressure_units,
                sensor_upper_trim_point
        }
}
  RESPONSE_CODES
{
        0,      SUCCESS,                        [no_command_specific_errors];
        2,      DATA_ENTRY_ERROR,               [invalid_selection];
        3,      DATA_ENTRY_ERROR,               [passed_parameter_too_large];
        4,      DATA_ENTRY_ERROR,               [passed_parameter_too_small];
        5,      MISC_ERROR,                     [too_few_data_bytes_recieved];
        6,      MISC_ERROR,                     [span_out_limits];
        7,      MODE_ERROR,                     [in_write_protect_mode];
        11,     MISC_ERROR,                     [excess_correction];
        12,     DATA_ENTRY_ERROR,               [invalid_unit_code];
        14,     DATA_ENTRY_ERROR,               [span_too_small];
}
}
```

Komenda 131 Trim lower sensor calibration

/* Uwaga!, wejście w procedurę kalibracji zeruje podkalibrowanie zera */

```
COMMAND write_lower_sensor_trim
{
        NUMBER 131;
        OPERATION WRITE;
        TRANSACTION
        {
        REQUEST
        {
                channel_number,                 /* channel_number = 0 or 1/
                pressure_units,
                sensor_lower_trim_point         /* IEE754 float */
        }
        REPLY
        {
                response_code,device_status,
                channel_number,
                pressure_units,
                sensor_lower_trim_point
        }
  }
RESPONSE_CODES
  {
        0,      SUCCESS,                        [no_command_specific_errors];
        2,      DATA_ENTRY_ERROR,               [invalid_selection];
        3,      DATA_ENTRY_ERROR,               [passed_parameter_too_large];
        4,      DATA_ENTRY_ERROR,               [passed_parameter_too_small];
        5,      MISC_ERROR,                     [too_few_data_bytes_recieved];
        6,      MISC_ERROR,                     [span_out_limits];
        7,      MODE_ERROR,                     [in_write_protect_mode];
        11,     MISC_ERROR,                     [excess_correction];
        12,     DATA_ENTRY_ERROR,               [invalid_unit_code];
        14,     DATA_ENTRY_ERROR,               [span_too_small];
  }
}
```

Komenda 132 Write control modes

Składnia komendy dla APC-2000 i SG-25.SMART i pochodnych

```
COMMAND write_control_modes
{
NUMBER 132;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
                local_keys_mode_control_codes,
                meter_information_configuration
        }
        REPLY
        {
                response_code,device_status,
                local_keys_mode_control_codes,
                meter_information_configuration
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                    [no_command_specific_errors];
        2,  DATA_ENTRY_ERROR,           [invalid_selection];
        5,  MISC_ERROR,                 [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                 [in_write_protect_mode];
}
}
```

Składnia komendy dla APC-2000/AL. i pochodnych

```
COMMAND write_control_modes
{
NUMBER 132;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
                local_keys_mode_control_codes,      /* 0 = enabled. 1 = disabled */
                meter_information_configuration,    /* 0 = not installed, 1 = integral_LCD */
                meter_display_mode,                 /*
                                                    {BIT0=0,        [normal]},
                                                    {BIT0=1,        [reverse] },
                                                    {BIT1=0,        [right rotation]},
                                                    {BIT1=1,        [left rotation]},
                                                    {BIT2=0,        [entire display normal]},
                                                    {BIT2=1,        [entire display on]},
                                                    {BIT3=0,        [LCD bias ratio 1/5]},
                                                    {BIT3=1,        [LCD bias ratio 1/6]}
                                                    */
                meter_display_electronics_bias_control, /* 0x00-0x1F,  [manual bias control,
                                                                         bias = 0x00 - 0x1F]
                                                    */
                meter_display_variable              /*
                                                    {0,     [current]},
                                                    {1,     [pressure]},
                                                    {2,     [percent_of_range]},
                                                    {3,     [user_rerange]}
                                                    */
```

```
                meter_display_variable_decimal_point        /*
                                                            {1,     [.XXXXX]},
                                                            {2,     [X.XXXX]},
                                                            {3,     [XX.XXX]},
                                                            {4,     [XXX.XX]},
                                                            {5,     [XXXX.X]},
                                                            {6,     [XXXXX.]}
                                                            */

        }
        REPLY
        {
                response_code,device_status,
                local_keys_mode_control_codes,
                meter_information_configuration,
                meter_display_mode,
                meter_display_electronics_bias_control,
                meter_display_variable,
                meter_display_variable_decimal_point
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                        [no_command_specific_errors];
        2,  DATA_ENTRY_ERROR,               [invalid_selection];
        5,  MISC_ERROR,                     [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                     [in_write_protect_mode];
}
}
```

Komenda 133 Read control modes

Składnia komendy dla APC-2000 i SG25.SMART i pochodnych

```
COMMAND read_control_modes
{
NUMBER 133;
OPERATION READ;
TRANSACTION
{
        REQUEST
        {}
        REPLY
        {
                response_code,device_status,
                local_keys_mode_control_codes,
                meter_information_configuration
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                                    [no_command_specific_errors];
}
}
```

Składnia komendy dla APC-2000/AL i pochodnych

```
COMMAND read_control_modes
{
NUMBER 133;
OPERATION READ;
TRANSACTION
{
        REQUEST
        {}
        REPLY
        {
                response_code,device_status,
                local_keys_mode_control_codes,
                meter_information_configuration,
                meter_display_mode,
                meter_display_electronics_bias_control,
                meter_display_variable,
                meter_display_variable_decimal_point
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                                    [no_command_specific_errors];
}
}
```

Komenda 135 Write coefficient for user characteristic (linear approximation)

```
COMMAND write_linear_approximation_coefficient
NUMBER 135;
OPERATION WRITE;
TRANSACTION
{
        REQUEST                         /* n={0,3,6,9,12,15,18) */
        {
        pzp(n),                 /* float IEE754 */
        pzu(n),                 /* float IEE754 */
        pzp(n+1),               /* float IEE754 */
        pzu(n+1),               /* float IEE754 */
        pzp(n+2),               /* float IEE754 */
        pzu(n+2),               /* float IEE754 */
        numer_kontenera         /* 1 bajt określający numer kontenera danych, zakres (0-6).
                                 7 kontenerów  tworzy łącznie zbiór 42 współczynników float */
        }
        REPLY
        {
        response_code,device_status,
        pzp(n),                 /* float IEE754 */
        pzu(n),                 /* float IEE754 */
        pzp(n+1),               /* float IEE754 */
        pzu(n+1),               /* float IEE754 */
        pzp(n+2),               /* float IEE754 */
        pzu(n+2),               /* float IEE754 */
        numer_kontenera         /* 1 bajt określający numer kontenera danych, zakres (0-6).
                                 7 kontenerów  tworzy łącznie zbiór 42 współczynników float */
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                    [no_command_specific_errors];
        2,  DATA_ENTRY_ERROR,           [invalid_selection];
        5,  MISC_ERROR,                 [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                 [in_write_protect_mode];
}
}
```

Komenda 136 Read coefficient for user characteristic

```
COMMAND read_ linear_approximation_coefficient

NUMBER 136;
OPERATION READ;
TRANSACTION
{
        REQUEST                     /* n={0,3,6,9,12,15,18) */
        {
        numer_kontenera             /* 1 bajt określający numer kontenera danych, zakres (0-6).
                                     7 kontenerów  tworzy łącznie zbiór 42 współczynników float */
        }
        REPLY
        {
        response_code,device_status,
        pzp(n),                     /* float IEE754 */
        pzu(n),                     /* float IEE754 */
        pzp(n+1),                   /* float IEE754 */
        pzu(n+1),                   /* float IEE754 */
        pzp(n+2),                   /* float IEE754 */
        pzu(n+2),                   /* float IEE754 */
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                        [no_command_specific_errors];
        2,  DATA_ENTRY_ERROR,               [invalid_selection];
        5,  MISC_ERROR,                     [too_few_data_bytes_recieved];
}
}
```

Komenda 138 Return to factory setting

```
NUMBER 138;
OPERATION COMMAND;
TRANSACTION
{
        REQUEST
        {
                kind_of_trim                    /* 0 = undo zero trim,
                                                   1 = sensor trim recall to factory settings,
                                                   2 = analog output trim recall to factory settings */
        }
        REPLY
        {
                response_code, device_status,
                kind_of_trim
        }
}
  RESPONSE_CODES
{
        0,  SUCCESS,                            [no_command_specific_errors];
        2,  DATA_ENTRY_ERROR,                   [invalid_selection];
        5,  MISC_ERROR,                         [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                         [in_write_protect_mode];
}
}
```

Komenda 141 Write static data configuration

```
COMMAND write_static_data_configuration
{
NUMBER 141;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
                adc_conversion_mode,
                channel_default_counter,
                frequency_filter,
                current_alarm_mode
        }
        REPLY
        {
                response_code,device_status,
                adc_conversion_mode,
                channel_default_counter,
                frequency_filter,
                current_alarm_mode
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                    [no_command_specific_errors];
        5,  MISC_ERROR,                 [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                 [in_write_protect_mode];
}
}
```

Komenda 142 Read static data configuration

```
COMMAND read_static_data_configuration
{
NUMBER 142;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
        }
        REPLY
        {
                response_code,device_status,
                adc_conversion_mode,
                channel_default_counter,
                frequency_filter,
                current_alarm_mode
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                    [no_command_specific_errors];
}
}
```

Komenda 231 Read product code

```
COMMAND read_product_code
{
        NUMBER 231;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                product_code                    /* ASCII 16 */
        }
}
  RESPONSE_CODES
{
        0,      SUCCESS,                                [no_command_specific_errors];
}
}
```

Komenda 233 Read separator code

```
COMMAND read_separator_code
{
        NUMBER 233;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                separator_code                  /* ASCII 16 */
        }
}
  RESPONSE_CODES
{
        0,    SUCCESS,                                  [no_command_specific_errors];
}
}
```

Komenda 235 Read manifold code

```
COMMAND read_manifold_code
{
        NUMBER 235;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                manifold_code                   /* ASCII 16 */
        }
}
  RESPONSE_CODES
{
        0,      SUCCESS,                                [no_command_specific_errors];
}
}
```

Komenda 237 Read operational limits

```
COMMAND read_operational_limits
{
        NUMBER 237;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                eep_max_work_press_uc,          /* max_work_press_unit_code */
                eep_max_work_press,             /* max_work_pressure, IEE754 float */
                eep_max_temp_unit_code,         /* max_temp_unit_code */
                eep_max_temp_value,             /* max_temp_value, IEE754 float */
                eep_min_temp_unit_code,         /* min_temp_unit_code */
                eep_min_temp_value,             /* min_temp_value, IEE754 float */
        }
}
  RESPONSE_CODES
{
        0,    SUCCESS,                          [no_command_specific_errors];
}
}
```

Komenda 240 Write long TAG

```
COMMAND write_long_tag
{
NUMBER 240;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
                long_tag                /* ASCII 24 */
        }
        REPLY
        {
                response_code,device_status,
                long_tag
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                            [no_command_specific_errors];
        5,  MISC_ERROR,                         [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                         [in_write_protect_mode];
}
}
```

Komenda 241 Read long TAG

```
COMMAND read_long_tag
{
        NUMBER 241;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                long_tag                /* ASCII 24 */
        }
}
  RESPONSE_CODES
{
        0,    SUCCESS,                          [no_command_specific_errors];
}
}
```

Komenda 242 Write startpoint rad

```
COMMAND write_startpoint_rad
{
NUMBER 242;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
                startpoint_rad                  /* IEE754, wartość w % zakresu */
        }
        REPLY
        {
                response_code,device_status,
                startpoint_rad
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                            [no_command_specific_errors];
        5,  MISC_ERROR,                         [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                         [in_write_protect_mode];
}
}
```

Komenda 243 Read startpoint rad

```
COMMAND read_startpoint_rad
{
        NUMBER 243;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                startpoint_rad                  /* IEE754, % of range */
        }
}
  RESPONSE_CODES
{
        0,    SUCCESS,                                  [no_command_specific_errors];
}
}
```

Komenda 244 Write user's unit name and rearrange coefficients

```
COMMAND write_user's_unit_name_and_rearrange_coefficients
{
        NUMBER 244;
        OPERATION WRITE;
        TRANSACTION
{
        REQUEST
        {
                user's_unit,                    /* ASCII(16 */
                eep_RrangeShift,                /* MSP430, flip bytes */
                eep_RrangeAmpl                  /* MSP430, flip bytes*/
        }
        REPLY
        {
                response_code,device_status,
                user's_unit,
                eep_RrangeShift,
                eep_RrangeAmpl
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                            [no_command_specific_errors];
        5,  MISC_ERROR,                         [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                         [in_write_protect_mode];
}
}
```

Komenda 245 Read user’s unit name and rearrange coefficients

```
COMMAND read_user's_unit_name_and_rerange_coefficients
{
        NUMBER 245;
        OPERATION READ;
        TRANSACTION
{
        REQUEST
        {

        }
        REPLY
        {
                response_code,device_status,
                user's_unit,
                eep_RrangeShift,
                eep_RrangeAmpl
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                    [no_command_specific_errors];
        5,  MISC_ERROR,                 [too_few_data_bytes_recieved];
        7,  MODE_ERROR,                 [in_write_protect_mode];
}
}
```

Komenda 246 Write customer's security code

```
COMMAND write_customer_security_code
{
NUMBER 246;
OPERATION WRITE;
TRANSACTION
{
        REQUEST
        {
                old_customer_security_code,             /* 4 bytes hex */
                new_customer_security_code              /* 4 bytes hex */
        }
        REPLY
        {
                response_code,device_status,
                new_customer_security_code
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                            [no_command_specific_errors];

}
}
```

Komenda 247 Set write protect code

```
COMMAND set_write_protect_code
{
        NUMBER 247;
        OPERATION WRITE;
        TRANSACTION
{
        REQUEST
        {
                customer_security_code,
                write_protect_code                      /* 0 = no protected, 1 = protected */
        }
        REPLY
        {
                response_code,device_status,
                customer_security_code,
                write_protect_code
        }
}
RESPONSE_CODES
{
        0,  SUCCESS,                                    [no_command_specific_errors];
        2,  DATA_ENTRY_ERROR,                           [invalid_selection];
        5,  MISC_ERROR,                                 [too_few_data_bytes_recieved];
        16, MODE_ERROR                                  [access_restricted];
}
}
```