Протокол сети *“CorNet”*

v1.0

Предназначение

Локальная промышленная сеть ***CorNet*** предназначена для удаленного управления и сбора данных со счетчиков-корректоров объема газа **ОКВГ-01**. В данном документе описан протокол обмена данными и функционирование сети в целом.

Физический уровень

На физическом уровне используется полудуплексный интерфейс RS-485 и экранированная витая пара в качестве среды передачи сигналов. Волновое сопротивление кабеля – 120 Ом. Топология сети – шина с терминаторами (резисторы 120 Ом) на концах сегмента. К одному сегменту RS-485 может быть подключено до 32-х устройств и его длина может достигать 1200 метров. В корректорах **ОКВГ-01** для обеспечения требований по искробезопасности используется модифицированный драйвер RS-485 и на одном сегменте может быть установлено не более шести корректоров.

На Рис.1 показана структура сети ***CorNet***. Корректоры последовательно объединяются по RS-485 и подключаются через модуль преобразователя интерфейсов к последовательному порту компьютера. Модуль преобразователя интерфейсов осуществляет согласование дуплексного интерфейса RS-232 с полудуплексным RS-485. Для управления передачей данных через преобразователь используется сигнал RTS интерфейса RS-232. При RTS > +3V (“space”) происходит передача данных от компьютера к корректорам, а при RTS < -3V (“mark”) в обратном направлении.

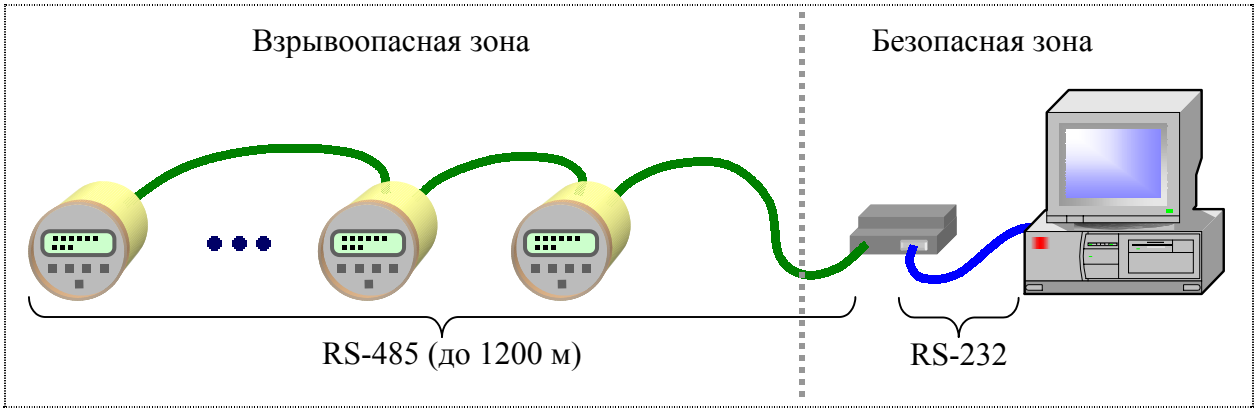

Рис.1 Структура сети ***CorNet***

Данные в сети передаются асинхронно по 10 бит – стартовый бит, 8 бит данных (младший бит вперед) и стоп-бит. Скорость передачи должна быть одинаковой для всех устройств на одном сегменте и выбирается из ряда: 4800, 9600, 19200, 38400.

Сеть в целом может состоять из нескольких сегментов. Каждый из этих сегментов через свой преобразователь интерфейсов подключается к отдельному порту RS-232 компьютера.

Канальный уровень

Работа сети организуется по принципу master-slave. Компьютер под управлением соответствующего программного обеспечения является мастером, а корректоры – это slave-устройства и они могут передавать данные в сеть только по команде мастера.

Каждое slave-устройство в сети должно иметь уникальный адрес из диапазона 1…126. Мастер имеет адрес 0, а адрес 127 используется как широковещательный. Slave обрабатывает только команды, адресованные на его собственный или широковещательный адреса. Команды для других адресов полностью игнорируются.

Для выделения в потоке данных адресных байтов используется процедура байт-стаффинга (экранирующий символ 0xFA). Для передачи адреса мастер формирует последовательность 0xFA,<adr>, при этом адрес не может быть равен 0xFA. При передаче в блоке данных кода 0xFA – он удваивается. На приемной стороне производится обратное преобразование потока – выделяются адресные байты, а сдвоенные 0xFA заменяются одним байтом. Эта процедура действует и при передаче данных от slave к мастеру.

Циклы обмена между мастером и slave могут быть трех видов:

- простая команда;
- команда + запись блока данных в slave;
- команда + чтение блока данных из slave.

В последних двух случаях длина передаваемого или принимаемого блока задается мастером в командном пакете.

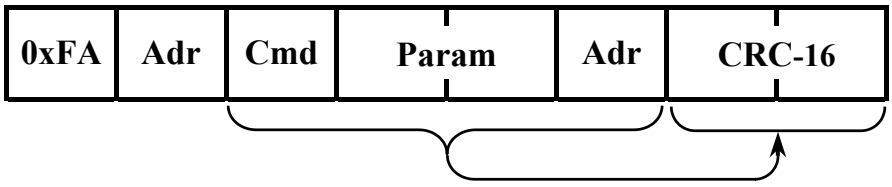

Рис.2 Простая команда

Командный пакет (Рис.2) имеет фиксированную структуру и длину. Сначала мастер формирует адрес slave (**0xFA**,**Adr**), затем передается код команды (**Cmd**), два байта параметров команды (**Param**), и повторно адрес slave (**Adr**). Завершается пакет двумя байтами контрольной суммы (**CRC-16**), подсчитанной для предыдущих четырех байт. Порядок следования байт в полях **CRC-16** и **Param**. – Hi,Lo: сначала старший байт слова, затем младший.

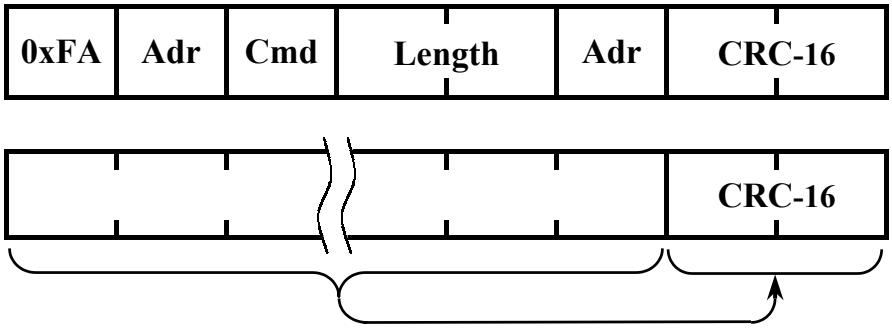

Рис.3 Команда + запись (чтение) блока данных

В командах с записью или чтением блока данных (Рис.3) сначала мастер передает командный пакет, затем передает или принимает пакет данных переменной длины. Пакет данных состоит из блока данных, длина которого задается в поле параметров команды, и двух байт контрольной суммы **CRC-16**.

Для вычисления контрольной суммы CRC-16 используется полином 0x1021 (аналогичный используемому в протоколе XMODEM). Начальное значение контрольной суммы при расчете принимается равным нулю.

При записи блока данных в slave между командой и блоком данных должна быть задержка минимум 0.5 мс (рекомендуемое значение – 1 мс) для того чтобы slave успел обработать команду и подготовиться к приему блока. При чтении блока данных из slave после выдачи последнего байта команды мастер должен максимально быстро (в пределах 0.5 мс) переключить полудуплексный интерфейс RS-485 на прием. При выдаче мастером последовательности команд между командами должна быть минимальная задержка 1 мс (рекомендуемое значение – 3 мс).

Функционирование сети

Сеансы связи

Поскольку корректоры являются устройствами с батарейным питанием и большую часть времени находятся в экономичном режиме, контроль интерфейса осуществляется ими один раз за период измерений (1…10 с). Поэтому вводится понятие сеанса связи – мастер сначала должен привести все slave на сегменте сети в режим готовности к обмену данными. Для инициирования сеанса связи мастер формирует преамбулу – в течении определенного времени выдает в линию последовательность 0xFA,0x00. Длительность преамбулы должна превышать максимальный период измерений установленный в корректорах.

Slave обнаружив ситуацию обмена данными в сети переходит к режиму ожидания адреса. В этом режиме он контролирует все байты, передаваемые по сети, и при получении собственного или широковещательного адреса переходит к режиму приема команды. После завершения обмена данными с мастером или при приеме ошибочной команды, slave возвращается к режиму ожидания адреса.

В течении одного сеанса связи мастер может осуществлять обмен данными с несколькими slave. Все slave на сегменте в течении всего сеанса связи находятся в режиме готовности к обмену данными. Завершение сеанса связи и возврат slave-устройств в экономичный режим работы осуществляется либо по команде мастера, либо по истечении времени таймаута (12 с) после окончания передачи данных в сети (таймаут сеанса).

Диаграмма состояний slave

На Рис.4 приведена диаграмма состояний slave при обмене данными по сети. Как уже было сказано выше, большую часть времени slave находится в экономичном режиме работы (“**SLEEP**”). Проверка интерфейса осуществляется один раз за период измерений. Если будет зафиксирован трафик по сети (во время проверки будет принят хотя бы один байт), slave переходит в режим ожидания адреса (“**WAIT ADDRESS**”). В этом режиме slave принимает все байты, передаваемые по сети, но выделяет в потоке только адресные байты – байты данных игнорируются. При получении своего или широковещательного адреса (127) slave переходит к приему пакета команды (“**GET COMMAND**”).

Если в состоянии “**WAIT ADDRESS**” в течении времени таймаута сеанса (12 с) не будет принято ни одного байта, slave завершает сеанс связи и возвращается в экономичный режим работы.

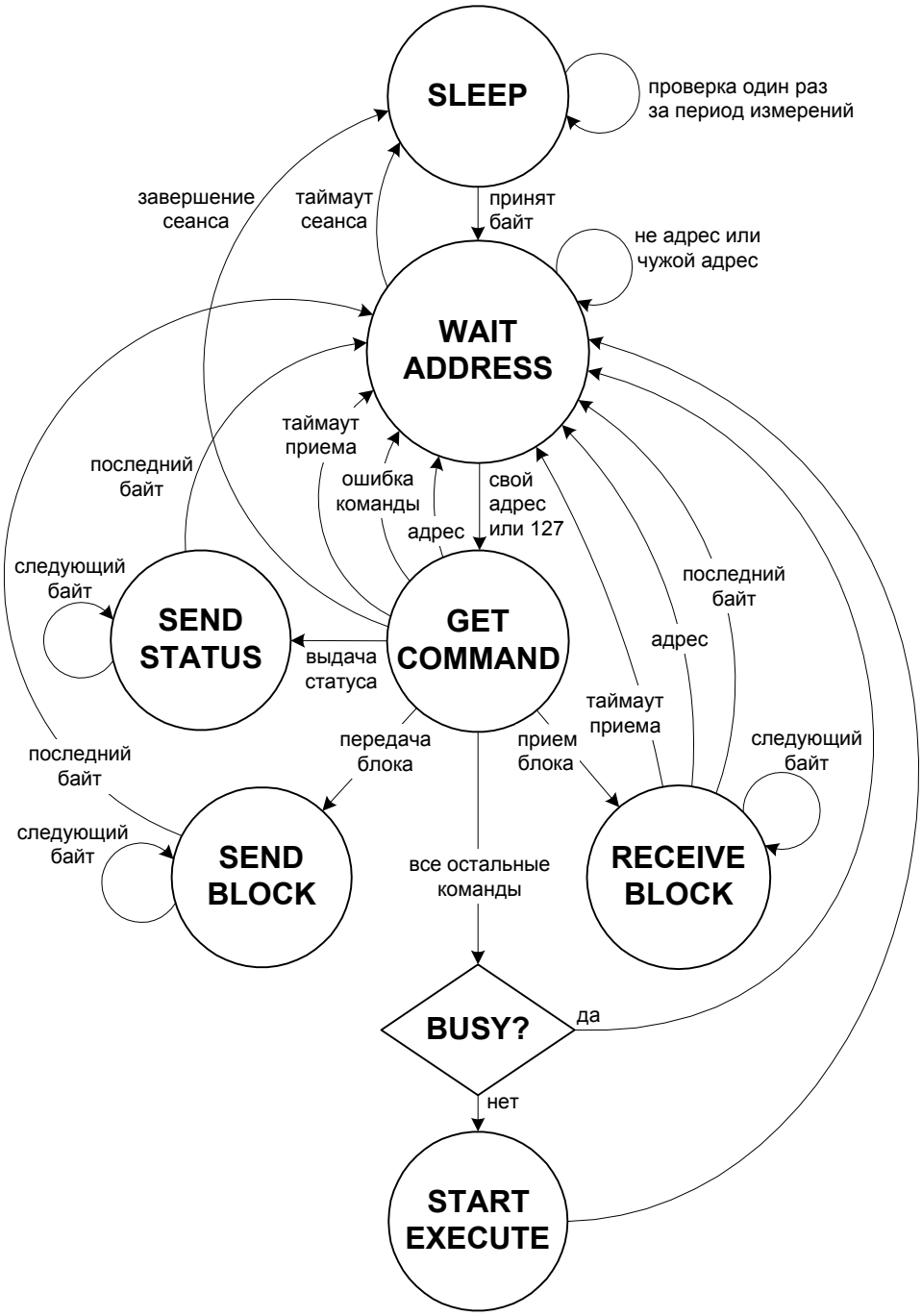

Рис.4 Диаграмма состояний slave

В режиме “**GET COMMAND**” slave принимает пакет из 4-х байт команды и 2-х байт контрольной суммы. Реальное число принятых байт может быть больше из-за процедуры байт-стаффинга (удвоение кода 0xFA). В процессе приема slave подсчитывает контрольную сумму и сравнивает ее с принятой. Также контролируется поле адреса в команде и диапазон допустимых кодов команд. В командах приема и передачи блока данных дополнительно проверяется максимальная допустимая длина блока. При обнаружении несоответствий устанавливаются соответствующие биты в регистре статуса и происходит возврат к состоянию “**WAIT ADDRESS**”. Прекращение приема команды и переход к режиму ожидания адреса происходит также в случае приема адресного байта (любого) или если межсимвольная пауза превысит значение таймаута приема (2 с).

После получения корректной команды slave переходит к ее выполнению. Группа из пяти команд (системные) выполняются с наивысшим приоритетом сразу по их получении.

К системным относятся команды приема и передачи блока, выдачи статуса, завершения сеанса и пустая команда (на диаграмме не показана). Все остальные команды (прикладные) запускаются на выполнение отдельным процессом ("**START EXECUTE**") с меньшим по сравнении с системными командами приоритетом. Прикладные команды не предназначены для приема или передачи данных по сети, они лишь подготавливают данные во внутреннем буфере slave для последующей передачи или используют принятые ранее данные из буфера. Все пересылки данных по сети осуществляются только системными командами приема и передачи блока и выдачи статуса.

Время выполнения прикладных команд не детерминировано и зависит от типа команды и загрузки процессора slave (но не более чем 0.5 с). На время выполнения прикладной команды в регистре статуса интерфейса выставляется флаг занятости "**BUSY**" и последующие прикладные команды будут игнорироваться, пока не завершится выполнение предыдущей команды. Системные команды могут обрабатываться параллельно с выполнением прикладных команд. Таким образом, мастер имеет возможность, периодически проверяя значение статуса, определить момент завершения прикладной команды.

По команде передачи блока slave переходит к режиму "**SEND BLOCK**" и осуществляет передачу указанного в команде числа байт из буфера. После вывода последнего байта блока slave переключается на прием и переходит к состоянию ожидания адреса. Команда выдачи статуса "**SEND STATUS**" выполняется аналогично команде передачи блока, только число байт в блоке фиксировано (2 байта).

По команде приема блока slave переходит к режиму "**RECEIVE BLOCK**" и осуществляет прием в буфер указанного в команде числа байт. После приема последнего байта блока slave переходит к состоянию ожидания адреса. Прием блока может быть прерван досрочно, если будет принят адресный байт (любой) либо межсимвольная пауза превысит значение таймаута приема (2 с).

Процедуры обмена

Исходя из описанных выше принципов работы slave, мастер должен следовать следующим процедурам по обмену данными:

- для передачи данных в slave:
    1) дождаться завершения предыдущей команды (периодически проверяя статус);
    2) переслать блок данных в буфер slave;
    3) выдать прикладную команду, которая будет использовать данные из буфера;
- для считывания данных из slave:
    1) дождаться завершения предыдущей команды;
    2) выдать прикладную команду для подготовки необходимых данных в буфере;
    3) дождаться завершения предыдущей команды;
    4) считать блок данных из буфера slave.

Команды и структуры данных

Ниже дано подробное описание команд и данных корректора **ОКВГ-01**. Данные представлены в виде структур на языке С. При этом действуют следующие соглашения:

- размерности простых типов данных:
  - char, uchar – 1 байт;
  - short, ushort, int, uint – 2 байта;
  - long, ulong, float – 4 байта.
- порядок байт в простых типах данных – big endian (т.е. старший байт по младшему адресу);
- поля в структурах не выравниваются;
- код команды имеет тип uchar, а параметр команды – short.

Сложные типы данных составляются из простых. Полное определение всех используемых типов данных и констант приведено в приложении 1.

Код: 0 **Параметр:** 0 Пустая команда

Описание:

Никаких действий не производится, за исключением установки флагов в регистре статуса.

Данные:

Нет

Код: 1 **Параметр:** 0 Чтение статуса slave

Описание:

Чтение из slave блока данных, состоящего из двух байт статуса (sl_stat_t). Тип ошибки определяется установкой в ‘1’ соответствующего бита (флага) в регистре статуса. Первый байт характеризует предыдущий цикл обмена, а второй – статус выполнения команды. Нулевые значения обоих байт статуса свидетельствуют об отсутствии ошибок.

Данные:

```
typedef
struct {
   uchar io;       // статус предыдущего цикла обмена:
                   //   &0x01 - выход по таймауту
                   //   &0x02 - ошибка в командном пакете (CRC, adr)
                   //   &0x04 - ошибка в блоке данных (CRC)
                   //   &0x08 - прерывание приема пакета
                   //           (получен адресный байт)
   uchar cmd;      // статус выполнения команды:
                   //   &0x10 - неизвестная команда
                   //   &0x20 - ошибка выполнения команды
                   //           (неверные параметры, …)
                   //   &0x80 - исполнение предыдущей команды
                   //           не завершено
   }
sl_stat_t;
```

Код: 2 **Параметр:** <blksize> Запись блока в slave (master -> slave)

Описание:

Запись в slave блока данных размером <blksize>. Размер блока может быть установлен от 0 до максимального размера буфера slave (2040 байт в текущей реализации). Если указанный в команде размер блока превышает размер буфера, в регистре статуса устанавливается флаг ошибки командного пакета и прием блока вообще не осуществляется.

Данные:

Массив байт размером <blksize>

Код: 3 **Параметр:** <blksize> Чтение блока из slave (master <- slave)

Описание:

Чтение из slave блока данных размером <bklsize>. Размер блока может быть установлен от 0 до максимального размера буфера slave (2040 байт). Если указанный в команде размер блока превышает размер буфера, в регистре статуса устанавливается флаг ошибки командного пакета и передача блока не осуществляется.

Данные:

Массив байт размером <blksize>

Код: 4 **Параметр:** 0 Завершение сеанса связи

Описание:

По данной команде slave завершает текущий сеанс связи и переходит в экономичный режим работы ("**SLEEP**" на диаграмме состояний slave).

Данные:

Нет

Код: 5 **Параметр:** 0 Экран и текущее состояние консоли

Описание:

Slave копирует в сетевой буфер текущее содержимое экрана и значения параметров работы консоли (sl_con_t). Данная команда предназначена для реализации режима удаленного терминала.

Данные:

```
typedef
struct {
   uchar mode;     // режим работы консоли: CON_MODE_*
   uchar tmc;      // счетчик таймаута консоли, (cn_tm0..0), sec
   uchar edf;      // флаг режима редактирования строки

      // следующие параметры действительны только при
      // mode == CON_MODE_NORMAL
   uchar st;       // байт статуса консоли: CON_ST_*
   uchar adr;      // текущий адрес в LCD
   uchar crow;     // текущая позиция курсора: строка (0...CON_ROWS-1)
   uchar ccol;     // текущая позиция курсора: столбец (0...CON_COLS-1)
   uchar scr[CON_ROWS*CON_COLS];    // буфер экрана (!!! lcd charset)
   }
sl_con_t;
```

Код: 6 **Параметр:** <scancode> Ввод символа в буфер клавиатуры

Описание:

Значение параметра команды <scancode> вводится в буфер клавиатуры slave. Диапазон значений <scancode> - 0…255. Коды 0…31 предназначены для имитации нажатия отдельных клавиш консоли или их комбинаций. Остальные значения (32…255) используются при вводе данных в режиме редактирования и соответствуют кодировке CP-866. Команда предназначена для реализации режима удаленного терминала.

Данные:

scancode	Клавиши
1	[x]
2	[<]
3	[>]
4	[↵]
5	[o]
8	backspace
17	[o]+[x]
18	[o]+[<]
19	[o]+[>]
20	[o]+[↵]
26	[x]+[<]
27	[x]+[>]

Код: 7 **Параметр:** 0 Выключение индикатора корректора

Описание:

По данной команде slave выключает LCD-индикатор консоли. Для его включения необходимо сымитировать с помощью команды ввода символа в буфер клавиатуры нажатие какой-либо клавиши.

Данные:

Нет

Код: 8 **Параметр:** 0 Конфигурация: информация

Описание:

В сетевой буфер копируется информация о корректоре (sl_info_t) – серийный номер прибора, версия и дата компиляции программного обеспечения, а также информация вводимая пользователем при конфигурации. Символьные строки завершаются нулевым символом ‘\0’.

Данные:

```
typedef
struct {
   uchar s[6][20];   // 6 символьных строк: информация вводимая
                     //   пользователем
   uchar fw[6][20];  // 6 строк: 0,1-название прибора;
                     //          2,3-версия firmware;
                     //          4,5-дата/время компиляции;
   ushort sernum;    // серийный номер прибора
   }
sl_info_t;
```

Код: 8 **Параметр:** 1 Конфигурация: датчик температуры

Описание:

В сетевой буфер копируются параметры конфигурации датчика температуры (sl_temcfg_t).

Данные:

```
typedef
struct {
   uchar type;    // тип датчика: TEM_RTD_*
   short R0;      // сопротивление RTD при 0'C, R(Ohm)*100;
   uchar on;      // флаг подключения датчика: 1-подключен;
                  //    0-не подключен(использ. знач. def);
   float def;     // значение температуры по умолчанию
   float mn;      // граничное значение для alarm: min
   float mx;      // граничное значение для alarm: max
   uchar ps_on;   // флаг измерения температуры датчика давления:
                  // 0 - не подключен, 1 - подключен;
   short CfA[2];  // буфера таблицы передаточной функции канала
   short CfB[2];  // измерения R*100 = F(adc);
   float CfK[2];  // y = (CfK[0] * x) + CfK[1];
   }
sl_temcfg_t;
```

Код: 8 **Параметр:** 2 Конфигурация: датчик давления

Описание:

В сетевой буфер копируются параметры конфигурации датчика давления (sl_presscfg_t).

Данные:

```
typedef
struct {
   uchar type;    // тип датчика: PR_SENS_*
   uchar on;      // флаг подключения датчика: 1-подключен;
                  //    0-не подключен(использ. знач. def);
   float def;     // значение давления по умолчанию, MPa;
   float mn;      // граничное значение для alarm: min, MPa
   float mx;      // граничное значение для alarm: max, MPa
   float vex;     // напряжение запитки моста (паспортное), mV;
   float mvv;     // reserved;
   short CfA[2];  // буфера таблицы передаточной функции
   short CfB[2];  // канала измерения U(mV)*100 = F(adc);
   float CfK[2];  //
      // таблица функции преобразования датчика давления, P = F(u,t);
   short Tt[PR_TSZ_T];           // температура, 'C
   short Tu[PR_TSZ_T][PR_TSZ_U]; // напряжение, mV*100
   short Tp[PR_TSZ_T][PR_TSZ_U]; // давление, MPa*1000
   table_t pFu[PR_TSZ_T];
   table2_t F;
   }
sl_presscfg_t;
```

Код: 8 **Параметр:** 3 Конфигурация: датчик объема

Описание:

В сетевой буфер копируются параметры конфигурации тахометрического датчика (sl_thcfg_t).

Данные:

```
typedef
struct {
   float   vn;    // объем (м3), для которого снималась характеристика
                  // турбинки; объем/(1 имп.): v1[i] = vn/N[i];
   short T[VOL_TSZ]; // период между импульсами турбинки, 1==10msec
   short N[VOL_TSZ]; // число импульсов для объема th.vn
   table_t Ftbl;     // таблица функции преобразования датчика объема
   }
sl_thcfg_t;
```

Код: 8 **Параметр:** 4 Конфигурация: пауза

Описание:

В сетевой буфер копируются параметры конфигурации алгоритма вычислений объема газа за время паузы в работе прибора (sl_pausecfg_t).

Данные:

```
typedef
struct {
   ulong mx;      // макс. длительность отключения питания, 1 == 0.01 ч
   uchar ae;      // флаг разрешения автоматического добавления
                  // к основному объему объема за паузу
   }
sl_pausecfg_t;
```

Код: 8 **Параметр:** 5 Конфигурация: системные параметры

Описание:

В сетевой буфер копируются системные параметры конфигурации (sl_syscfg_t) – период измерений, таймаут консоли, уставки контроля напряжения питания, время начала учетных суток.

Данные:

```
typedef
struct {
   uchar mtu;     // период (sec) измерений и усреднения
   float mtf;     // период (sec) измерений и усреднения
   uchar ut0;     // timeout неактивности пользователя, sec
   uchar uton;    // флаг разрешения timeout-а пользователя
   uchar contrast;// контраст LCD
   ushort vpwmn;  // минимальное напряжение питания, mV
   ushort vpwmx;  // максимальное напряжение питания, mV
   uchar apw;     // флаг подачи аналогового питания:
                  // 0: импульсный режим;  1: включено все время;
      // firmware >= v2.6
   uchar h0;      // время начала учетных суток, час: 0..23
   }
sl_syscfg_t;
```

Код: 8 **Параметр:** 6 Конфигурация: интерфейс

Описание:

В сетевой буфер копируются параметры конфигурации интерфейса (sl_siocfg_t).

Данные:

```
typedef
struct {
   uchar en;      // флаг разрешения работы интерфейса
   uchar baud;    // скорость: SIO_BAUD_*
   uchar adr;     // сетевой адрес прибора: 1...126
   }
sl_siocfg_t;
```

Код: 8 **Параметр:** 7 Конфигурация: выходы

Описание:

В сетевой буфер копируются параметры конфигурации выходных ключей корректора (sl_siocfg_t) – длительность импульсов и функции каждого из выходов.

Данные:

```
typedef
struct {
```

```
   uchar ms;      // длительность импульса, мсек (5..70)
   ushort tms;    // длительность импульса, tmS(x)
   uchar mode[2]; // режим работы ключей: OUT_*
   float v1[2];   // цена одного импульса при OUT_VOL*, м3
   }
sl_outcfg_t;
```

Код: 8 **Параметр:** 8 Константы

Описание:

В сетевой буфер копируются значения констант (sl_cst_t), которые используются для расчета коэффициента сжимаемости газа.

Данные:

```
typedef
struct {
   float    Dn;   // плотность газа при норм. условиях, кг/м3
   float    aCO2; // концентрация углекислого газа в смеси, %
   float    aN2;  // концентрация азота в смеси, %
   float    Pbar; // барометрическое давление, MPa;
   }
sl_cst_t;
```

Код: 8 **Параметр:** 9 Переход на летнее/зимнее время

Описание:

В сетевой буфер копируются параметры перехода на летнее/зимнее время (sl_dtj_t).

Данные:

```
typedef
struct {
   uchar mn;      // месяц, 0,1…12 (0 – переход запрещен)
   uchar wn;      // неделя месяца, 1…5 (5 означает последняя)
   uchar wd;      // день недели, 0..6
   uchar h;       // час, 0..23
   }
dtjump_t;

typedef
struct {
   dtjump_t s;    // дата/время перехода на летнее время
   dtjump_t w;    // дата/время перехода на зимнее время
   }
sl_dtj_t;
```

Код: 9 **Параметр:** 0 Текущие значения измерений

Описание:

В сетевой буфер копируются текущие значения различных измеряемых и вычисляемых параметров (sl_curr_t).

Данные:

```
typedef
struct {
   dtime_t  dt;   // дата/время
   tem_t    tm;   // температура
   press_t  pr;   // давление
   thcnt_t  th;   // счетчик импульсов
   volume_t vl;   // прирост объема
   gaz_t    gz;   // температура,давление,расход,объем
   pause_t  pz;   // пауза
   corr_t   cr;   // коэф. сжимаемости и коррекции
   sys_t    sys;  // напряжение питания, магн.поле
```

```
   event_t  evt[EVT_SZ];   // события
   }
sl_curr_t;
```

Код: 10 **Параметр:** 0 Чтение состояния очередей историй и логов

Описание:

В сетевой буфер копируются текущее состояние очередей историй измерений и логов событий (sl_queue_t). Также по данной команде производится обновление дополнительной копии состояния очередей, которая используется в командах чтения записей из историй и логов.

Данные:

```
typedef
struct {
   xque_t   hm;   // очередь истории минутных измерений
   xque_t   hh;   // очередь истории часовых измерений
   xque_t   hd;   // очередь истории суточных измерений
   xque_t   hmn;  // очередь истории месячных измерений
   xque_t   le;   // очередь лога событий
   xque_t   lc;   // очередь лога изменений
   }
sl_queue_t;
```

Код: 11 **Параметр:** <recnum> Чтение записи из истории минутных измерений

Описание:

В сетевой буфер копируется одна запись из истории минутных измерений (gaz_t). Параметр команды <recnum> определяет номер записи: 0 – последняя по времени запись, 1 – предпоследняя, и т.д. Значение <recnum> должно быть меньше текущего числа записей в истории, иначе в регистре состояния выставляется флаг ошибки и команда не выполняется.

При выполнении данной команды используется дополнительная копия состояния очередей, которая обновляется только по команде чтения текущего состояния очередей. Это сделано для обеспечения целостности данных при считывании блока записей.

Данные:

```
typedef
struct {
   dtime_t     dt;   // дата, время
   float       tem;  // средняя температура, 'C
   float       pr;   // среднее давление (абсолютное), MPa
   float       Q;    // средний расход, м3/ч
   float       Qn;   // средний расход при нормальных условиях, м3/ч
   ulongfloat  V;    // объем, м3
   ulongfloat  Vn;   // объем при нормальных условиях, м3
   }
gaz_t;
```

Код: 12 **Параметр:** <recnum> Чтение записи из истории часовых измерений

Описание:

В сетевой буфер копируется одна запись из истории часовых измерений (gaz_t). Параметр команды <recnum> определяет номер записи: 0 – последняя по времени запись, 1 – предпоследняя, и т.д. Значение <recnum> должно быть меньше текущего числа записей в истории, иначе в регистре состояния выставляется флаг ошибки и команда не выполняется.

При выполнении данной команды используется дополнительная копия состояния очередей, которая обновляется только по команде чтения текущего состояния очередей. Это сделано для обеспечения целостности данных при считывании блока записей.

Данные:

```
typedef
struct {
   dtime_t     dt;   // дата, время
   float       tem;  // средняя температура, 'C
   float       pr;   // среднее давление (абсолютное), MPa
   float       Q;    // средний расход, м3/ч
   float       Qn;   // средний расход при нормальных условиях, м3/ч
   ulongfloat  V;    // объем, м3
   ulongfloat  Vn;   // объем при нормальных условиях, м3
   }
gaz_t;
```

Код: 13 **Параметр:** <recnum> Чтение записи из истории суточных измерений

Описание:

В сетевой буфер копируется одна запись из истории суточных измерений (gaz_t). Параметр команды <recnum> определяет номер записи: 0 – последняя по времени запись, 1 – предпоследняя, и т.д. Значение <recnum> должно быть меньше текущего числа записей в истории, иначе в регистре состояния выставляется флаг ошибки и команда не выполняется.

При выполнении данной команды используется дополнительная копия состояния очередей, которая обновляется только по команде чтения текущего состояния очередей. Это сделано для обеспечения целостности данных при считывании блока записей.

Данные:

```
typedef
struct {
   dtime_t     dt;   // дата, время
   float       tem;  // средняя температура, 'C
   float       pr;   // среднее давление (абсолютное), MPa
   float       Q;    // средний расход, м3/ч
   float       Qn;   // средний расход при нормальных условиях, м3/ч
   ulongfloat  V;    // объем, м3
   ulongfloat  Vn;   // объем при нормальных условиях, м3
   }
gaz_t;
```

Код: 14 **Параметр:** <recnum> Чтение записи из истории месячных измерений

Описание:

В сетевой буфер копируется одна запись из истории месячных измерений (gaz_t). Параметр команды <recnum> определяет номер записи: 0 – последняя по времени запись, 1 – предпоследняя, и т.д. Значение <recnum> должно быть меньше текущего числа записей в истории, иначе в регистре состояния выставляется флаг ошибки и команда не выполняется.

При выполнении данной команды используется дополнительная копия состояния очередей, которая обновляется только по команде чтения текущего состояния очередей. Это сделано для обеспечения целостности данных при считывании блока записей.

Данные:

```
typedef
struct {
   dtime_t     dt;   // дата, время
   float       tem;  // средняя температура, 'C
   float       pr;   // среднее давление (абсолютное), MPa
   float       Q;    // средний расход, м3/ч
   float       Qn;   // средний расход при нормальных условиях, м3/ч
   ulongfloat  V;    // объем, м3
   ulongfloat  Vn;   // объем при нормальных условиях, м3
   }
gaz_t;
```

Код: 15 **Параметр:** <recnum> Чтение записи из лога событий

Описание:

В сетевой буфер копируется одна запись из лога событий (evtlog_t). Параметр команды <recnum> определяет номер записи: 0 – последняя по времени запись, 1 – предпоследняя, и т.д. Значение <recnum> должно быть меньше текущего числа записей в логе, иначе в регистре состояния выставляется флаг ошибки и команда не выполняется.

При выполнении данной команды используется дополнительная копия состояния очередей, которая обновляется только по команде чтения текущего состояния очередей. Это сделано для обеспечения целостности данных при считывании блока записей.

Данные:

```
typedef
struct {
   dtime_t dt;              // дата/время записи
   uchar id;                // идентификатор записи
   uchar bf[LOG_EBUF_SZ];   // буфер параметров записи
   }
evtlog_t;
```

Код: 16 **Параметр:** <recnum> Чтение записи из лога изменений

Описание:

В сетевой буфер копируется одна запись из лога изменений (chglog_t). Параметр команды <recnum> определяет номер записи: 0 – последняя по времени запись, 1 – предпоследняя, и т.д. Значение <recnum> должно быть меньше текущего числа записей в логе, иначе в регистре состояния выставляется флаг ошибки и команда не выполняется.

При выполнении данной команды используется дополнительная копия состояния очередей, которая обновляется только по команде чтения текущего состояния очередей. Это сделано для обеспечения целостности данных при считывании блока записей.

Данные:

```
typedef
struct {
   dtime_t dt;              // дата/время записи
   uchar id;                // идентификатор записи
   uchar bf[LOG_CBUF_SZ];   // буфер параметров записи
   }
chglog_t;
```

Приложение 1. Константы и типы данных ОКВГ-01.

```
/*---- define -------------------------------------------------*/

#define  VERSION     "v1.0"

////////////////////////////////////////////////////////////////

#define  ON    1
#define  OFF   0
#define  YES   1
#define  NO    0


////////////////////////////////////////////////////////////////

#define  NET_ADR_MASTER       0        // адрес мастера
#define  NET_ADR_BROADCAST    127      // широковещательный адрес

#define  NET_DLE              0xFA     // Data Link Escape code (for byte stuffing)

#define  NET_TOUT_RCV      (mS(2000))  // таймаут при приеме

   // net.stio
#define  NET_ST_OK            0x00     // предыдущая команда выполнена нормально
#define  NET_ST_TOUT          0x01     // выход по таймауту
#define  NET_ST_CMDERR        0x02     // ошибка в командном пакете (CRC, adr)
#define  NET_ST_DATERR        0x04     // ошибка в блоке данных (CRC)
#define  NET_ST_BREAK         0x08     // прерывание приема пакета (получен адресный байт)
   // net.stcmd
#define  NET_ST_UNKNOWN       0x10     // неизвестная команда
#define  NET_ST_EXEERR        0x20     // ошибка выполнения команды (неверные параметры, ...)
#define  NET_ST_BSY           0x80     // исполнение предыдущей команды не завершено


   // коды комманд
#define  NET_CMD_NULL         0     // пустая команда
#define  NET_CMD_STAT         1     // выдать байт статуса
#define  NET_CMD_WR           2     // запись блока в slave  (master -> slave)
#define  NET_CMD_RD           3     // чтение блока из slave (master <- slave)
#define  NET_CMD_END          4     // команда завершения сессии
#define  NET_CMD_CONS         5     // экран и текущее состояние консоли (sl_con_t)
#define  NET_CMD_KEYB         6     // ввод символа в буфер клавиатуры
#define  NET_CMD_LCDOFF       7     // выключить индикатор корректора

#define  NET_CMD_GETCFG       8        // получить параметры конфигурации;

   // значения параметра команды NET_CMD_GETCFG
#define  SL_CFG_INFO          0     // конфигурация: информация (sl_info_t)
#define  SL_CFG_TEM           1     // конфигурация: датчик температуры (sl_temcfg_t)
#define  SL_CFG_PRESS         2     // конфигурация: датчик давления (sl_presscfg_t)
#define  SL_CFG_TH            3     // конфигурация: тахометрический датчик (sl_thcfg_t)
#define  SL_CFG_PAUSE         4     // конфигурация: пауза (sl_pausecfg_t)
#define  SL_CFG_SYS           5     // конфигурация: системные параметры (sl_syscfg_t)
#define  SL_CFG_SIO           6     // конфигурация: интерфейс (sl_siocfg_t)
#define  SL_CFG_OUT           7     // конфигурация: выходы (sl_outcfg_t)
#define  SL_CFG_CST           8     // константы (sl_cst_t)
#define  SL_CFG_DTJ           9     // параметры перехода на летнее/зимнее время (sl_dtj_t)

#define  NET_CMD_CURRENT      9     // получить текущие значения параметров (sl_curr_t)

#define  NET_CMD_QUEUE        10    // получить текущее состояние очередей историй измерений
                                    // и логов (sl_queue_t)

   // команды получения записей из историй (NET_CMD_HIST*) или логов (NET_CMD_LOG*);
   // параметром команды задается номер записи: 0 - последняя запись, 1 - предпоследняя, ...
#define  NET_CMD_HISTM        11    // получить одну запись из истории минутных измерений (gaz_t)
#define  NET_CMD_HISTH        12    // получить одну запись из истории часовых измерений (gaz_t)
#define  NET_CMD_HISTD        13    // получить одну запись из истории суточных измерений (gaz_t)
#define  NET_CMD_HISTMN       14    // получить одну запись из истории месячных измерений (gaz_t)
#define  NET_CMD_LOGEV        15    // получить одну запись из лога событий (evtlog_t)
#define  NET_CMD_LOGCH        16    // получить одну запись из лога изменений (chglog_t)


////////////////////////////////////////////////////////////////
```

```
#define  CON_ROWS             2        // rows number of LCD screen
#define  CON_COLS             16       // columns number of LCD screen

   // режимы консоли
#define  CON_MODE_NORMAL      0
#define  CON_MODE_IDLE        1
#define  CON_MODE_SHUTDOWN    2

   // байт статуса консоли
#define  CON_ST_BLINK         0x01     // blink on/off
#define  CON_ST_CURSEN        0x02     // cursor on/off
#define  CON_ST_LCDPOW        0x04     // LCD power on/off
#define  CON_ST_INTEN         0x08     // interrupt enable/disable
#define  CON_ST_IRQ           0x10     // interrupt request флаг
#define  CON_ST_ASCII         0x20     // ascii on/off
#define  CON_ST_SCROLL        0x40     // scroll on/off
#define  CON_ST_TTY           0x80     // TTY mode on/off

////////////////////////////////////////////////////////////////


   // slave keyboard scan codes
#define  KEY_NULL    0
#define  KEY_1       1
#define  KEY_2       2
#define  KEY_3       3
#define  KEY_4       4
#define  KEY_5       5
#define  KEY_6       6

#define  KEY_FBASE   8
#define  KEY_F1      (KEY_1+KEY_FBASE)
#define  KEY_F2      (KEY_2+KEY_FBASE)
#define  KEY_F3      (KEY_3+KEY_FBASE)
#define  KEY_F4      (KEY_4+KEY_FBASE)
#define  KEY_F5      (KEY_5+KEY_FBASE)
#define  KEY_F6      (KEY_6+KEY_FBASE)

#define  KEY_OBASE   16
#define  KEY_O1      (KEY_1+KEY_OBASE)

#define  KEY_O2      (KEY_2+KEY_OBASE)
#define  KEY_O3      (KEY_3+KEY_OBASE)
#define  KEY_O4      (KEY_4+KEY_OBASE)
#define  KEY_O5      (KEY_5+KEY_OBASE)
#define  KEY_O6      (KEY_6+KEY_OBASE)

#define  KEY_EBASE   24
#define  KEY_E1      (KEY_1+KEY_EBASE)
#define  KEY_E2      (KEY_2+KEY_EBASE)
#define  KEY_E3      (KEY_3+KEY_EBASE)
#define  KEY_E4      (KEY_4+KEY_EBASE)
#define  KEY_E5      (KEY_5+KEY_EBASE)
#define  KEY_E6      (KEY_6+KEY_EBASE)

#define  KEY_DUMMY   31

   // keys status bitmap
#define  KEY1_BIT    0x01
#define  KEY2_BIT    0x02
#define  KEY3_BIT    0x04
#define  KEY4_BIT    0x08
#define  KEY5_BIT    0x10
#define  KEY6_BIT    0x20

#define  SL_ESC      KEY_1
#define  SL_LEFT     KEY_2
#define  SL_RIGHT    KEY_3
#define  SL_CR       KEY_4
#define  SL_CYCL     KEY_5
#define  KEY_F       SL_CR
#define  KEY_O       SL_CYCL
#define  KEY_E       SL_ESC
#define  KEY_F_BIT   KEY4_BIT
#define  KEY_O_BIT   KEY5_BIT
#define  KEY_E_BIT   KEY1_BIT
#define  SL_BACKSPACE   KEY_FBASE
#define  SL_BS       SL_BACKSPACE
```

```
#define  F_AR        (KEY_FBASE+SL_RIGHT)
#define  F_AL        (KEY_FBASE+SL_LEFT)
#define  F_CYCL      (KEY_FBASE+SL_CYCL)
#define  F_ESC       (KEY_FBASE+SL_ESC)

#define  O_AR        (KEY_OBASE+SL_RIGHT)
#define  O_AL        (KEY_OBASE+SL_LEFT)
#define  O_CR        (KEY_OBASE+SL_CR)
#define  O_ESC       (KEY_OBASE+SL_ESC)

#define  E_AR        (KEY_EBASE+SL_RIGHT)
#define  E_AL        (KEY_EBASE+SL_LEFT)
#define  E_CYCL      (KEY_EBASE+SL_CYCL)
#define  E_CR        (KEY_EBASE+SL_CR)

//////////////////////////////////////////////////////////////////

   // тип датчика температуры
#define  TEM_RTD_Pt     0
#define  TEM_RTD_Cu     1

#define  TEM_ON         1
#define  TEM_OFF        0

//////////////////////////////////////////////////////////////////

   // тип датчика давления
#define  PR_SENS_ABS    0     // датчик абсолютного давления
#define  PR_SENS_GAGE   1     // датчик избыточного давления

#define  PR_ON          1
#define  PR_OFF         0

#define  PR_TSZ_U       8     // макс. количество точек по напряжению
#define  PR_TSZ_T       8     // макс. количество точек по температуре

//////////////////////////////////////////////////////////////////

#define  VOL_TSZ        32    // макс. размер таблицы функции преобразования турбинки

//////////////////////////////////////////////////////////////////

#define  LOG_ESZ           200   // макс. число записей в логе событий
#define  LOG_CSZ           200   // макс. число записей в логе изменений
#define  LOG_EBUF_SZ       4     // размер буфера параметров лога событий
#define  LOG_CBUF_SZ       8     // размер буфера параметров лога изменений

#define  EVT_TEM           0
#define  EVT_PRESS         1
#define  EVT_POW           2
#define  EVT_MFLD          3
#define  EVT_SZ            4

#define  EVT_NORM          0     // параметр в норме
#define  EVT_MIN           1     // параметр < мин.
#define  EVT_MAX           2     // параметр > макс.

   // идентификаторы лога событий
#define  LOG_NULL          0
#define  LOG_PWOFF         1     // отключение питания процессора
#define  LOG_PWON          2     // включение питания процессора
#define  LOG_RESET         3     // сброс (без отключения питания)
#define  LOG_DTDEF         4     // системное время установлено в default
#define  LOG_CFGDEF        5     // конфигурация установлена в default

#define  LOG_TEM_NORM      6     // температура в норме
#define  LOG_TEM_MIN       7     // температура < мин.
#define  LOG_TEM_MAX       8     // температура > макс.

#define  LOG_PRESS_NORM    9     // давление в норме
#define  LOG_PRESS_MIN     10    // давление < мин.
#define  LOG_PRESS_MAX     11    // давление > макс.

#define  LOG_POW_NORM      12    // напряжение питания в норме
```

```
#define  LOG_POW_MIN       13    // напряжение питания < мин.
#define  LOG_POW_MAX       14    // напряжение питания > макс.

#define  LOG_MFLD_NORM     15    // внешнее магнитное поле отсутствует
#define  LOG_MFLD_MAX      16    // внешнее магнитное поле > макс.

#define  LOG_SUMMER_TIME   17    // переход на летнее время
#define  LOG_WINTER_TIME   18    // переход на зимнее время

#define  LOG_PAUSE_BEGIN   19    // начало паузы
#define  LOG_PAUSE_END     20    // окончание паузы
#define  LOG_PAUSE_AUTOADD 21    // автодобавление к основному объему


   // идентификаторы лога изменений
#define  CHG_NULL          0
#define  CHG_INFO          1     // общая информация

#define  CHG_TEM_TYPE      2     // тип RTD
#define  CHG_TEM_R0        3     // сопротивление RTD при 0'C
#define  CHG_TEM_ONOFF     4     // подключение RTD
#define  CHG_TEM_DEF       5     // температура по умолчанию
#define  CHG_TEM_MIN       6     // минимальная граница температуры
#define  CHG_TEM_MAX       7     // максимальная граница температуры
#define  CHG_TEM_PSEN      8     // измерение температуры датчика давления: on/off

#define  CHG_PR_TYPE       9     // тип датчика давления: abs | gage
#define  CHG_PR_VEX        10    // паспортное напряжение запитки моста
#define  CHG_PR_MVV        11    // reserved
#define  CHG_PR_ONOFF      12    // подключение датчика давления
#define  CHG_PR_DEF        13    // давление по умолчанию
#define  CHG_PR_MIN        14    // минимальная граница давления
#define  CHG_PR_MAX        15    // максимальная граница давления

#define  CHG_SYS_MEASURE   16    // период измерений
#define  CHG_SYS_CONS      17    // consol timeout: on/off
#define  CHG_SYS_TOUT      18    // consol timeout: sec
#define  CHG_SYS_PWMIN     19    // минимальное напряжение питания
#define  CHG_SYS_PWMAX     20    // максимальное напряжение питания

#define  CHG_PZ_MAX        21    // макс. длительность паузы
#define  CHG_PZ_AUTOADD    22    // автоматическое добавление

#define  CHG_VOL_FUNC      23    // функция преобразования турбинки
#define  CHG_PR_FUNC       24    // функция преобразования датчика давления

#define  CHG_ACCESS        25    // уровень доступа
#define  CHG_PWD_OPER      26    // пароль оператора
#define  CHG_PWD_ADMIN     27    // пароль администратора
#define  CHG_PWD_OWNER     28    // пароль владельца

#define  CHG_DENSITY       29    // плотность при нормальных условиях
#define  CHG_CO2           30    // концентрация углекислого газа в смеси
#define  CHG_NITROGEN      31    // концентрация азота в смеси
#define  CHG_PBAR          32    // барометрическое давление

#define  CHG_DTIME         33    // дата/время в RTC
#define  CHG_SUMMER_JUMP   34    // дата/время перехода на летнее время
#define  CHG_WINTER_JUMP   35    // дата/время перехода на зимнее время

#define  CHG_PZ_ADDVOL     36    // добавление объема за паузу к основному

#define  CHG_CLR_VOLUME    37    // обнуление текущих объемов V и Vn

#define  CHG_CALIBR_T1P    38    // одноточечная калибровка канала температуры
#define  CHG_CALIBR_T2P    39    // двухточечная калибровка канала температуры
#define  CHG_CALIBR_P1P    40    // одноточечная калибровка канала давления
#define  CHG_CALIBR_P2P    41    // двухточечная калибровка канала давления
#define  CHG_CALIBR_R      42    // калибровка канала температуры: опорные точки R1,R2
#define  CHG_CALIBR_U      43    // калибровка канала давления: опорные точки U1,U2

#define  CHG_COUNTER       44    // счетчик импульсов

#define  CHG_DEFPWD        45    // пароли по умолчанию

#define  CHG_SIO_ONOFF     46    // интерфейс on/off
#define  CHG_SIO_BAUD      47    // скорость интерфейса
#define  CHG_SIO_ADR       48    // сетевой адрес интерфейса

#define  CHG_PULSE_WIDTH   49    // минимальная длительность импульсов на выходах
```

```
#define  CHG_OUT1_MODE     50    // выход1: режим работы
#define  CHG_OUT2_MODE     51    // выход2: режим работы
#define  CHG_OUT1_VOL      52    // выход1: цена одного импульса для объема
#define  CHG_OUT2_VOL      53    // выход2: цена одного импульса для объема

#define  CHG_SYS_H0        54    // время начала учетных суток



/*---- type ------------------------------------------------*/

typedef  signed   char  schar;
typedef  unsigned char  uchar;
typedef  unsigned short ushort;
typedef  unsigned int   uint;
typedef  unsigned long  ulong;

typedef
struct {
   ulong it;      // целая часть числа
   float fr;      // дробная часть числа
   }
ulongfloat;


////////////////////////////////////////////////////////////


#pragma  ZTC align 1

typedef
struct {
   short n;          // размер таблицы
   uchar xlim;       // =1: ограничение аргумента, если он выходит за границы

   //int *x;           // массив значений аргумента
   uchar xm;         // тип пямяти
   ushort xa;        // адрес

   //int *f;           // массив значений функции
   uchar fm;         // тип памяти
   ushort fa;        // адрес

   float scale;      // масштабный коэффициент
   }
table_t;


typedef
struct {
   short nx;         // размер таблицы по x
   uchar xlim;       // =1: ограничение x

   //int *x;           // массив значений аргумента x
   uchar xm;         // тип памяти
   ushort xa;        // адрес

   //table_t *f;       // массив таблиц одномерных функций z=f(y), (для
                       // каждого элемента массива аргументов x своя функция f)
   uchar fm;         // тип памяти
   ushort fa;        // адрес

   float scale;      // масштабный коэффициент
   }
table2_t;


typedef
struct {
   uchar hs;         // hundredths of a second,  0..99
   uchar s;          // seconds,  0..59
   uchar m;          // minutes,  0..59
   uchar h;          // hours,    0..23
   uchar d;          // day,      1..31
   uchar mn;         // month,    1..12
   ushort y;         // year,     19xx..2xxx
   uchar w;          // weekday,  0..6;  0: sunday
   }
dtime_t;
```

```
typedef
struct {
   ushort i;         /* текущий индекс */
   ushort q;         /* размер очереди (число элементов в очереди) */
   ushort n;         /* размер буфера (число элементов) */
   ushort sz;        /* размер элемента (байт) */
   //void *bf;         /* -> на буфер */
   uchar bfm;        // тип памяти
   ushort bfa;       // адрес
   }
xque_t;


typedef
struct {
   float gC;         // температура газа, 'C
   float gK;         // температура газа, 'K
   float pC;         // температура датчика давления, 'C
   short R;          // измеренное сопротивление RTD, R(Ohm)*100
   }
tem_t;


typedef
struct {
   float gaz;        // давление газа (абсолютное), MPa;
   short U;          // измеренное напряжение моста, mV;
   float vsc;        // масштабирующий коэффициент, приводящий измеренное
                     // вых. напряжение моста в диапазон паспортных значений
   }
press_t;


typedef
struct {
   uchar f;          // флаг изменения состояния счетчика импульсов
   ulong cnt;        // текущее значение счетчика импульсов тахометра
   dtime_t t1;       // время поступления предыдущего импульса
   dtime_t t2;       // время поступления последнего импульса
   ulong hs;         // период поступления импульсов (последн. зн.), hsec
   ulong xhs;        // сумма периодов импульсов, пришедших за время
                     // одного цикла вычислений, hsec
   }
thcnt_t;


typedef
struct {
   float dV;         // текущий прирост объема, м3
   float dVn;        // текущий прирост объема при нормальных условиях, м3
   float Q;          // текущий расход, м3/ч
   float Qn;         // текущий расход при нормальных условиях, м3/ч
   }
volume_t;


typedef
struct {
   dtime_t dt;       // дата, время
   float   tem;      // средняя температура, 'C
   float   pr;       // среднее давление (абсолютное), MPa
   float   Q;        // средний расход, м3/ч
   float   Qn;       // средний расход при нормальных условиях, м3/ч
   ulongfloat  V;    // объем, м3
   ulongfloat  Vn;   // объем при нормальных условиях, м3
   }
gaz_t;


typedef
struct {
   dtime_t dt[2];    // время начала и конца паузы
   ulong   hs;       // продолжительность паузы, hsec
   float   Q;        // средней расход за время паузы, м3/ч
   float   Qn;       // средний расход при нормальных условиях, м3/ч
   float   dV;       // объем за время последней паузы, м3
   float   dVn;      // объем за время последней паузы при норм.условиях, м3
   ulongfloat  V;    // суммарный объем за несколько пауз, м3
```

```
   ulongfloat  Vn;   // суммарный объем за несколько пауз при норм.условиях, м3
   }
pause_t;


typedef
struct {
   float Pkr;        // псевдокритическое давление, кгс/см2
   float Tkr;        // псевдокритическая температура, 'K
   float Pc;         // приведенное давление
   float Tc;         // приведенная температура
   float Z;          // коэффициент сжимаемости газа
   float K;          // коэффициент коррекции
   }
corr_t;


typedef
struct {
   ushort vpow;      // напряжение питания батареи, mV
   uchar mfld;       // состояние датчика внешнего магнитного поля:
                     // 0 - поля нет;  1: - поле есть;
   }
sys_t;


typedef
struct {
   uchar   st;       // текущее состояние "события": 0 - норма
   uchar   id;       // идентификатор записи для лога
   dtime_t dt;       // время начала события
   }
event_t;


typedef
struct {
   dtime_t dt;       // дата/время записи
   uchar   id;       // идентификатор записи
   uchar   bf[LOG_EBUF_SZ];   // буфер параметров записи
   }
evtlog_t;


typedef
struct {
   dtime_t dt;       // дата/время записи
   uchar   id;       // идентификатор записи
   uchar   bf[LOG_CBUF_SZ];   // буфер параметров записи
   }
chglog_t;


////////////////////////////////////////////////////////////////////

   // slave: состояние консоли и содержимое экрана;  команда NET_CMD_CONS
typedef
struct {
   uchar mode;       // режим работы консоли: CON_MODE_*
   uchar tmc;        // счетчик таймаута консоли, (cn_tm0..0), sec
   uchar edf;        // флаг режима редактирования строки

      // следующие параметры действительны только при  mode == CON_MODE_NORMAL
   uchar st;         // байт статуса консоли: CON_ST_*
   uchar adr;        // текущий адрес в LCD
   uchar crow;       // текущая позиция курсора: строка (0...CON_ROWS-1)
   uchar ccol;       // текущая позиция курсора: столбец (0...CON_COLS-1)
   uchar scr[CON_ROWS*CON_COLS];    // буфер экрана (!!! lcd charset)
   }
sl_con_t;


   // slave: информация; команда NET_CMD_INFO
typedef
struct {
   uchar s[6][20];   // 6 символьных строк: информация вводимая пользователем
   uchar fw[6][20];  // 6 строк: 0,1-название прибора; 2,3-версия firmware;
                     // 4,5-дата/время компиляции;
   ushort sernum;    // серийный номер прибора
   }
```

```
sl_info_t;

   // slave: конфигурация датчика температуры; команда NET_CMD_TEMCFG
typedef
struct {
   uchar type;       // тип датчика: TEM_RTD_*
   short R0;         // сопротивление RTD при 0'C, R(Ohm)*100;
   uchar on;         // флаг подключения датчика: 1-подключен;
                     //    0-не подключен(использ. знач. def);
   float def;        // значение температуры по умолчанию
   float mn;         // граничное значение для alarm: min
   float mx;         // граничное значение для alarm: max
   uchar ps_on;      // флаг измерения температуры датчика давления;
                     // 0 - не подключен; 1 - подключен;
   short CfA[2];     // буфера таблицы передаточной функции канала
   short CfB[2];     // измерения R*100 = F(adc);
   float CfK[2];     // y = (CfK[0] * x) + CfK[1];
   }
sl_temcfg_t;


   // slave: конфигурация датчика давления; команда NET_CMD_PRESSCFG
typedef
struct {
   uchar type;       // тип датчика: PR_SENS_*
   uchar on;         // флаг подключения датчика: 1-подключен;
                     //    0-не подключен(использ. знач. pr.def);
   float def;        // значение давления по умолчанию, MPa;
   float mn;         // граничное значение для alarm: min, MPa
   float mx;         // граничное значение для alarm: max, MPa
   float vex;        // напряжение запитки моста (паспортное), mV;
   float mvv;        // reserved

      // буфера таблицы передаточной функции канала измерения U(mV)*100 = F(adc);
   short CfA[2];     //
   short CfB[2];     //
   float CfK[2];     //

      // таблица функции преобразования датчика давления, P = F(u,t);
   short Tt[PR_TSZ_T];           // температура, 'C
   short Tu[PR_TSZ_T][PR_TSZ_U]; // напряжение, mV*100
   short Tp[PR_TSZ_T][PR_TSZ_U]; // давление, MPa*1000
   table_t pFu[PR_TSZ_T];
   table2_t F;
   }
sl_presscfg_t;

   // slave: конфигурация тахометрического датчика; команда NET_CMD_THCFG
typedef
struct {
   float   vn;       // объем (м3), для которого снималась характеристика
                     // турбинки; объем/(1 имп.): dV1 = th.vn / Ni;
   short T[VOL_TSZ]; // период между импульсами турбинки, 1==10msec
   short N[VOL_TSZ]; // число импульсов для объема th.vn
   table_t Ftbl;     // таблица функции преобразования датчика
   }
sl_thcfg_t;


   // slave: конфигурация паузы; команда NET_CMD_PAUSECFG
typedef
struct {
   ulong mx;         // макс. длительность отключения питания, 1 == 0.01 час
   uchar ae;         // флаг разрешения автоматического добавления
                     // к основному объему объема за паузу
   }
sl_pausecfg_t;


   // slave: системные установки; команда NET_CMD_SYSCFG
typedef
struct {
   uchar mtu;        // период (sec) измерений и усреднения
   float mtf;        // период (sec) измерений и усреднения
   uchar ut0;        // timeout неактивности пользователя, sec
   uchar uton;       // флаг разрешения timeout-а пользователя
   uchar contrast;   // контраст LCD
   ushort vpwmn;     // минимальное напряжение питания, mV
   ushort vpwmx;     // максимальное напряжение питания, mV
   uchar apw;        // флаг подачи аналогового питания:
```

```
                    // 0: импульсный режим;  1: включено все время;

      // firmware >= v2.6
   uchar h0;         // время начала учетных суток, час: 0..23
   }
sl_syscfg_t;


   // slave: конфигурация интерфейса; команда NET_CMD_SIOCFG
typedef
struct {
   uchar en;         // флаг разрешения работы интерфейса
   uchar baud;       // скорость: SIO_BAUD_*
   uchar adr;        // сетевой адрес прибора: 1...126
   }
sl_siocfg_t;


   // slave: конфигурация выходных ключей; команда NET_CMD_OUTCFG
typedef
struct {
   uchar ms;         // длительность импульса, мсек (5..70)
   ushort tms;       // длительность импульса, tmS(x)
   uchar mode[2];    // режим работы ключей: OUT_*
   float v1[2];      // цена одного импульса при OUT_VOL*, м3
   }
sl_outcfg_t;


   // slave: значения констант; команда NET_CMD_CST
typedef
struct {
   float    Dn;      // плотность газа при норм. условиях, кг/м3
   float    aCO2;    // концентрация углекислого газа в смеси, %
   float    aN2;     // концентрация азота в смеси, %
   float    Pbar;    // барометрическое давление, MPa;
   }
sl_cst_t;


   // slave: параметры перехода на летнее/зимнее время; команда NET_CMD_DTJ
typedef
struct {
   uchar mn;         // month,   0,1..12  (0 - переход запрещен)
   uchar wn;         // weeknum of month, 1..5  (5 означает последняя )
   uchar wd;         // weekday, 0..6
   uchar h;          // hour,    0..23
   }
dtjump_t;

typedef
struct {
   dtjump_t s;       // дата/время перехода на летнее время
   dtjump_t w;       // дата/время перехода на зимнее время
   }
sl_dtj_t;


   // slave: текущие значения параметров; команда NET_CMD_CURRENT
typedef
struct {
   dtime_t  dt;      // дата/время
   tem_t    tm;      // температура
   press_t  pr;      // давление
   thcnt_t  th;      // счетчик импульсов
   volume_t vl;      // прирост объема
   gaz_t    gz;      // температура,давление,расход,объем
   pause_t  pz;      // пауза
   corr_t   cr;      // коэф. сжимаемости и коррекции
   sys_t    sys;     // напряжение питания, магн.поле
   event_t  evt[EVT_SZ];   // события
   }
sl_curr_t;


   // slave: текущее состояние очередей историй и логов;  команда NET_CMD_QUEUE
typedef
struct {
   xque_t   hm;      // очередь истории минутных измерений
   xque_t   hh;      // очередь истории часовых измерений
```

```
   xque_t   hd;      // очередь истории суточных измерений
   xque_t   hmn;     // очередь истории месячных измерений
   xque_t   le;      // очередь лога событий
   xque_t   lc;      // очередь лога изменений
   }
sl_queue_t;


#pragma  ZTC align


/*---- macros -------------------------------------------------*/

#ifndef  Size
#define  Size(x,y)      (sizeof(x)/sizeof(y))
#endif

#ifndef  min
#define  min(a,b)       (((a) < (b)) ? (a) : (b))
#endif

#define  Hi(x)          ((uchar)(((uint)(x))>>8))
#define  Lo(x)          ((uchar)(x))
#define  HiWord(x)      ((uint)(((ulong)(x))>>16))
#define  LoWord(x)      ((uint)(x))


//////////////////////////////////////////////////////////////////


#define  crc_init()     (crc = 0)
#define  crc_add(c)     (crc = (crc << 8) ^ crctabl[ Hi(crc) ^ (c) ])


//////////////////////////////////////////////////////////////////


#define  net_tx_en()             (sio_rts(net.port, 1))
#define  net_tx_dis()            (sio_rts(net.port, 0))

#define  net_endses(adr)         (net_cmd((adr), NET_CMD_END, 0))
#define  net_lcdoff(adr)         (net_cmd((adr), NET_CMD_LCDOFF, 0))
#define  net_updqueue(adr)       (net_cmd((adr), NET_CMD_QUEUE, 0))


//////////////////////////////////////////////////////////////////


   // delay(uS(x)) | delay(mS(x))
#define  uS(x)    ((ulong)(x))
#define  mS(x)    (((ulong)(x))*1000L)


//////////////////////////////////////////////////////////////////
```

Приложение 2. Вычисление контрольной суммы CRC-16

```
#include <stdio.h>
#include <stdlib.h>
#include "cornet.h"


/*---- variable -------------------------------------------------*/

uint crc;         // текущее значение

uint crctabl[256] = {
   0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,
   0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,
   0x1231,0x0210,0x3273,0x2252,0x52b5,0x4294,0x72f7,0x62d6,
   0x9339,0x8318,0xb37b,0xa35a,0xd3bd,0xc39c,0xf3ff,0xe3de,
   0x2462,0x3443,0x0420,0x1401,0x64e6,0x74c7,0x44a4,0x5485,
   0xa56a,0xb54b,0x8528,0x9509,0xe5ee,0xf5cf,0xc5ac,0xd58d,
   0x3653,0x2672,0x1611,0x0630,0x76d7,0x66f6,0x5695,0x46b4,
   0xb75b,0xa77a,0x9719,0x8738,0xf7df,0xe7fe,0xd79d,0xc7bc,
   0x48c4,0x58e5,0x6886,0x78a7,0x0840,0x1861,0x2802,0x3823,
   0xc9cc,0xd9ed,0xe98e,0xf9af,0x8948,0x9969,0xa90a,0xb92b,
   0x5af5,0x4ad4,0x7ab7,0x6a96,0x1a71,0x0a50,0x3a33,0x2a12,
   0xdbfd,0xcbdc,0xfbbf,0xeb9e,0x9b79,0x8b58,0xbb3b,0xab1a,
   0x6ca6,0x7c87,0x4ce4,0x5cc5,0x2c22,0x3c03,0x0c60,0x1c41,
   0xedae,0xfd8f,0xcdec,0xddcd,0xad2a,0xbd0b,0x8d68,0x9d49,
   0x7e97,0x6eb6,0x5ed5,0x4ef4,0x3e13,0x2e32,0x1e51,0x0e70,
   0xff9f,0xefbe,0xdfdd,0xcffc,0xbf1b,0xaf3a,0x9f59,0x8f78,
   0x9188,0x81a9,0xb1ca,0xa1eb,0xd10c,0xc12d,0xf14e,0xe16f,
   0x1080,0x00a1,0x30c2,0x20e3,0x5004,0x4025,0x7046,0x6067,
   0x83b9,0x9398,0xa3fb,0xb3da,0xc33d,0xd31c,0xe37f,0xf35e,
   0x02b1,0x1290,0x22f3,0x32d2,0x4235,0x5214,0x6277,0x7256,
   0xb5ea,0xa5cb,0x95a8,0x8589,0xf56e,0xe54f,0xd52c,0xc50d,
   0x34e2,0x24c3,0x14a0,0x0481,0x7466,0x6447,0x5424,0x4405,
   0xa7db,0xb7fa,0x8799,0x97b8,0xe75f,0xf77e,0xc71d,0xd73c,
   0x26d3,0x36f2,0x0691,0x16b0,0x6657,0x7676,0x4615,0x5634,
   0xd94c,0xc96d,0xf90e,0xe92f,0x99c8,0x89e9,0xb98a,0xa9ab,
   0x5844,0x4865,0x7806,0x6827,0x18c0,0x08e1,0x3882,0x28a3,
   0xcb7d,0xdb5c,0xeb3f,0xfb1e,0x8bf9,0x9bd8,0xabbb,0xbb9a,
   0x4a75,0x5a54,0x6a37,0x7a16,0x0af1,0x1ad0,0x2ab3,0x3a92,
   0xfd2e,0xed0f,0xdd6c,0xcd4d,0xbdaa,0xad8b,0x9de8,0x8dc9,
   0x7c26,0x6c07,0x5c64,0x4c45,0x3ca2,0x2c83,0x1ce0,0x0cc1,
   0xef1f,0xff3e,0xcf5d,0xdf7c,0xaf9b,0xbfba,0x8fd9,0x9ff8,
   0x6e17,0x7e36,0x4e55,0x5e74,0x2e93,0x3eb2,0x0ed1,0x1ef0
   };

/*---- code -----------------------------------------------------*/

/*****************************************/
/* Вычисление следующего значения CRC      */
/*****************************************/
/* 16 bit CRC remainder based on the CCITT */
/* polynomial (0x1021) as used by XMODEM.  */
/*****************************************/
void crc_add (uchar c)
{
   crc = (crc << 8) ^ crctabl[ Hi(crc) ^ c ];
}


/*************************/
/* Начальное значение CRC */
/*************************/
void crc_init (void)
{
   crc = 0;
}
```

Автор:

Сергей Шендерук
Львовский центр Института космических исследований

phone: (380)-322-654450
www: http://www.isr.lviv.ua/okvg.htm
e-mail: shs@isr.lviv.ua
fido: 2:462/30