Протокол мережі *“CorNet”*

v1.0

Призначення

Локальна промислова мережа ***CorNet*** призначена для віддаленого управління і збору даних з лічильників-коректорів об’єму газу **ОКВГ-01**. В даному документі описано протокол обміну даними і функціонування мережі в цілому.

Фізичний рівень

На фізичному рівні використовується напівдуплексний інтерфейс RS-485 і екранована вита пара як середовище передачі сигналів. Хвильовий опір кабеля – 120 Ом. Топологія мережі – шина с термінаторами (резистори 120 Ом) на кінцях сегменту. До одного сегменту RS-485 може бути підключено до 32-ох пристроїв і його довжина може досягати 1200 метрів. В коректорах **ОКВГ-01** для виконання вимог щодо іскробезпеки, використовується модифікований драйвер RS-485, тому на одному сегменті може бути встановлено не більше шести коректорів.

На Рис.1 показано структуру мережі ***CorNet***. Коректори послідовно об’єднуються між собою за допомогою RS-485 і під’єднуються через модуль перетворювача інтерфейсів до послідовного порта комп’ютера. Модуль перетворювача інтерфейсів здійснює узгодження дуплексного інтерфейсу RS-232 з напівдуплексним RS-485. Для управління передачею даних через перетворювач використовується сигнал RTS интерфейсу RS-232. При RTS > +3V (“space”) відбувається передача даних від комп’ютера до коректора, а при RTS < -3V (“mark”) у зворотньому напрямку.

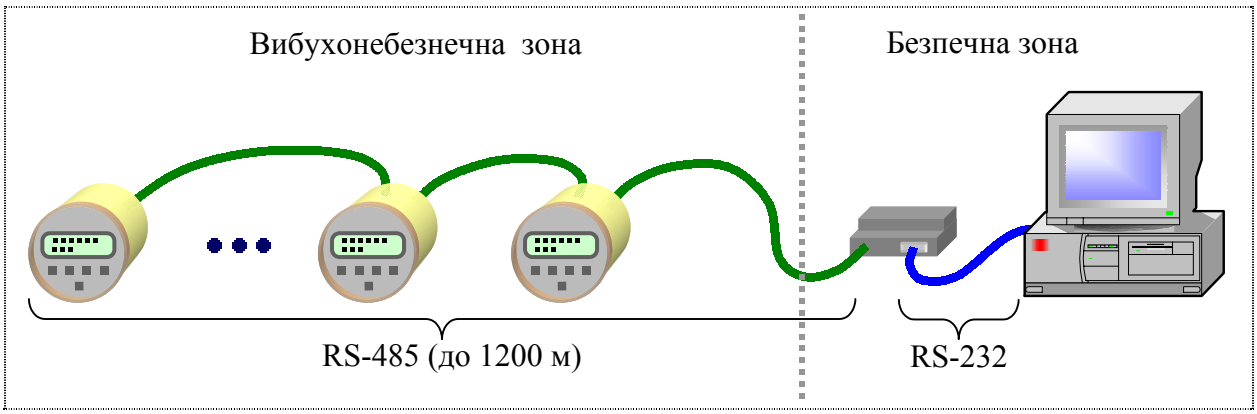

Рис.1 Структура мережі ***CorNet***

Дані в мережі передаються асинхронно по 10 біт – стартовий біт, 8 біт даних (молодший біт перший) і стоп-біт. Швидкість передачі повинна бути однаковою для всіх приладів на одному сегменті і вибирається з можливих: 4800, 9600, 19200, 38400.

Мережа в цілому може складатися з декількох сегментів. Кожен з яких, через свій перетворювач інтерфейсів, підключається до окремого порта RS-232 комп’ютера.

Канальний рівень

Работа мережі організована за принципом master-slave. Комп’ютер під управлінням відповідного програмного забезпечення є master, а коректори – це slave-пристрої і вони можуть передавати дані в мережу тільки по команді майстра.

Кожний slave-пристрій в мережі повинен мати унікальну адресу з діапазону 1…126. Майстер має адресу 0, а адреса 127 є спільною (широкомовною) для всіх пристроїв. Slave обробляє тільки команди, адресовані на його власну або широкомовну адресу. Команди для інших адрес ігноруються.

Для виділення в потоці даних адресних байт використовується процедура байт-стафінгу (екрануючий символ 0xFA). Для передачі адреси майстер формує послідовність 0xFA,<adr>, при цьому адреса не може бути рівна 0xFA. При передачі в блоці даних коду 0xFA – він подвоюється. На боці приймача проводиться обернене перетворення потоку, а саме, виділяються адресні байти, а подвоєні 0xFA замінюються одним байтом. Така ж процедура відбувається і при передачі даних від slave до майстра.

Цикли обміну між майстром і slave можуть бути трьох видів:

- проста команда;
- команда + запис блоку даних в slave;
- команда + читання блоку даних з slave.

В останніх двох випадках довжина блоку, який передається чи приймається, задається майстром в командному пакеті.

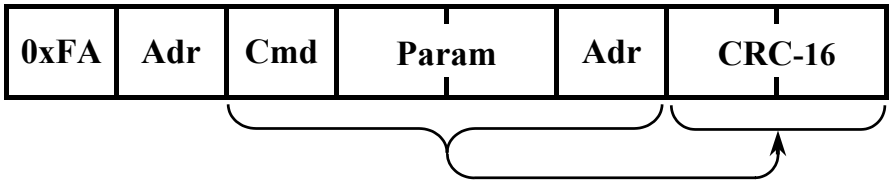

Рис.2 Проста команда

Командний пакет (Рис.2) має фіксовану структуру і довжину. Спочатку майстер формує адресу slave (**0xFA**,**Adr**), потім передає код команди (**Cmd**), два байта параметрів команди (**Param**), і повторно адресу slave (**Adr**). Завершується пакет двома байтами контрольної суми (**CRC-16**), яка обчислена для вищенаведених чотирьох байт. Послідовність байт в полях **CRC-16** і **Param**. – Hi,Lo: спочатку старший байт слова, потім молодший.

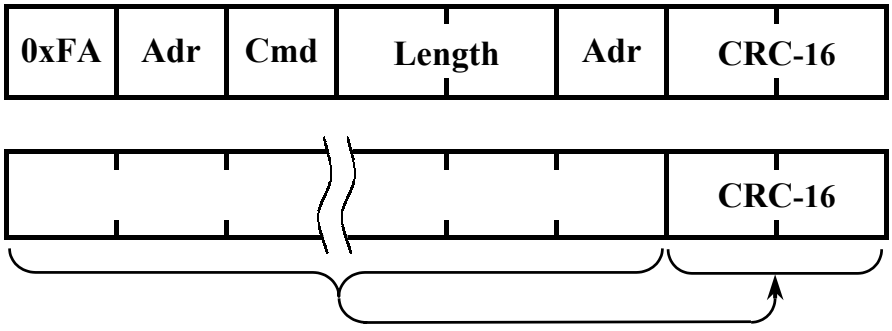

Рис.3 Команда + запис (читання) блоку даних

В командах з записом чи читанням блоку даних (Рис.3) майстер спочатку передає командний пакет, а потім передає чи приймає пакет даних змінної довжини. Пакет даних складається з блоку даних, довжина якого задається в полі параметрів команди, і двох байт контрольної суми **CRC-16**.

Для обчислення контрольної суми CRC-16 використовується поліном 0x1021 (аналогічний тому, який використовується в протоколі XMODEM). Початкове значення контрольної суми при розрахунках приймається рівним нулю.

При запису блоку даних в slave, між командою і блоком даних повинна бути затримка мінімум 0.5 мс (рекомендоване значення – 1 мс) для того щоб slave встиг обробити команду та підготуватися до приймання блоку. При читанні блоку даних з slave після видачі останнього байта команди майстер повинен максимально швидко (в межах 0.5 мс) переключити напівдуплексний інтерфейс RS-485 на приймання. При видачі майстром послідовності команд – між командами повинна бути мінімальна затримка 1 мс (рекомендоване значення – 3 мс).

Функціонування мережі

Сеанси зв'язку

Оскільки коректори є пристроями з батарейним живленням і більшу частину часу знаходяться в економічному режимі, контроль інтерфейсу здійснюється ними один раз за період вимірювань (1…10 с). Тому вводиться поняття сеансу зв'язку – майстер спочатку повинен перевести всі slave на сегменті мережі в режим готовності до обміну даними. Для ініціювання сеансу зв'язку майстер формує преамбулу – на протязі певного часу видає в лінію послідовність 0xFA,0x00. Тривалість преамбули повинна перевищувати максимальний період вимірювань встановлений в коректорах.

Slave виявивши ситуацію обміну даними в мережі переходить в режим очікування адреси. В якому він контролює всі байти, які передаються по мережі, і при отриманні власної або широкомовної адреси переходить в режим приймання команди. Після завершення обміну даними з майстром, або при прийманні помилкової команди, slave повертається в режим очікування адреси.

Протягом одного сеансу зв'язку майстер може здійснювати обмін даними з декількома slave. Всі slave на сегменті протягом одного сеансу зв'язку знаходяться в режимі готовності до обміну даними. Закінчення сеансу зв'язку і повернення slave-пристроїв в економний режим роботи здійснюється або за командою майстра, або по закінченні часу таймаута (12 с) після завершення передачі даних в мережі (таймаут сеансу).

Діаграма станів slave

На Рис.4 наведена діаграма станів slave обміну даними по мережі. Як було сказано вище, більшу частину часу slave знаходиться в економному режимі работи ("**SLEEP**"). Перевірка інтерфейсу здійснюється один раз за період вимірювань. Якщо буде зафіксовано трафік по мережі (під час перевірки буде прийнято хоча би один байт), slave переходить в режим очікування адреси ("**WAIT ADDRESS**"). В цьому режимі slave приймає всі байти, які передаються по мережі, але виділяє в потоці тільки адресні байти – байти даних ігноруються. При отриманні своєї або широкомовної адреси (127) slave переходить до приймання пакету команди ("**GET COMMAND**").

Якщо в стані "**WAIT ADDRESS**", протягом часу таймауту сеансу (12 с) не буде прийнято жодного байта, slave завершує сеанс зв'язку і повертається в економний режим роботи.

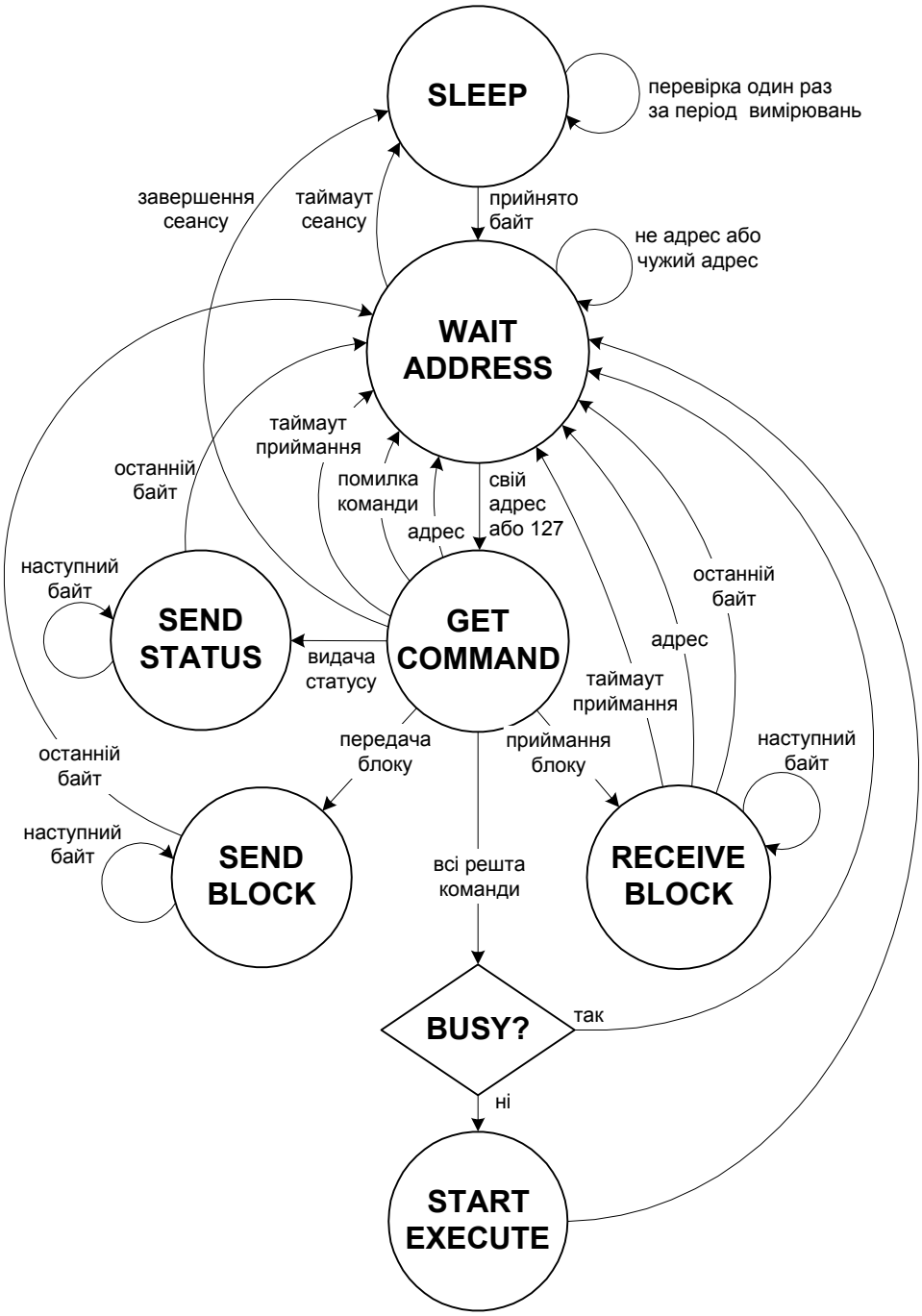

Рис.4 Діаграма станів slave

В режимі "**GET COMMAND**" slave приймає пакет з 4-х байт команди і 2-х байт контрольної суми. Реальна кількість прийнятих байт може бути більша через процедуру байт-стафінга (подвоєння коду 0xFA). В процесі приймання slave обчислює контрольну суму і порівнює її з прийнятою. Також контролюється поле адреси в команді і діапазон допустимих кодів команд. В командах приймання і передачі блоку даних додатково перевіряється максимально допустима довжина блоку. При виявлені невідповідностей встановлюються відповідні біти в регістрі статусу і відбувається повернення до стану "**WAIT ADDRESS**". Зупинка приймання команди і перехід до режиму очікування адреси відбувається також у випадку приймання адресного байта (будь-якого) або якщо міжсимвольна пауза перевищить значення таймаута приймання (2 с).

Після отримання коректної команди slave переходить до її виконання. Група з п'яти команд (системні команди) виконуються, з найвищим пріоритетом, відразу при їх надхо-

дженні. До системних належать команди приймання та передачі блоку, видачі статусу, завершення сеансу і пуста команда (на діаграмі не показана). Всі решта команди (прикладні) запускаються на виконання окремим процесом ("**START EXECUTE**") з меншим, у порівнянні з системними командами, пріоритетом. Прикладні команди не призначені для приймання чи передачі даних по мережі, вони лише готують дані у внутрішньому буфері slave для подальшої їх передачі або використовують прийняті раніше дані з буфера. Всі пересилки даних по мережі здійснюються тільки системними командами приймання і передачі блоку та видачі статусу.

Час виконання прикладних команд не детермінований і залежить від типу команди та завантаження процесора slave (але не більше ніж 0.5 с). На час виконання прикладної команди в регістрі статусу інтерфейсу виставляється прапорець зайнятості "**BUSY**" і наступні прикладні команди будуть ігноруватися, поки не завершиться виконання попередньої команди. Системні команди можуть оброблятись паралельно з виконанням прикладних команд. Таким чином, майстер має можливість, періодично перевіряючи значення статусу, визначити момент завершення прикладної команди.

По команді передачі блоку slave переходить в режим "**SEND BLOCK**" і здійснює передачу вказаної в команді кількості байт з буфера. Після виводу останнього байта блоку slave переключається на приймання і переходить в стан очікування адреси. Команда видачі статусу "**SEND STATUS**" виконується аналогічно команді передачі блоку, тільки кількість байт в блоці є фіксована (2 байта).

По команді приймання блоку slave переходить в режим "**RECEIVE BLOCK**" і здійснює приймання в буфер вказаної в команді кількості байт. Після приймання останнього байта блоку slave переходить в стан очікування адресу. Приймання блоку може бути перерване достроково, якщо буде прийнято адресний байт (будь-який) або міжсимвольна пауза перевищить значення таймаута приймання (2 с).

Процедури обміну

Виходячи з вищеописаних принципів роботи slave, майстер повинен дотримуватись наступних правил обміну даними:

- для передачі даних в slave:
    1) дочекатися завершення попередньої команди (періодично перевіряючи статус);
    2) переслати блок даних в буфер slave;
    3) видати прикладну команду, яка буде використовувати дані з буфера;
- для читання даних з slave:
    1) дочекатися завершення попередньої команди;
    2) видати прикладну команду для підготовки необхідних даних в буфері;
    3) дочекатися завершення попередньої команди;
    4) прочитати блок даних з буфера slave.

Команди і структури даних

Нижче наведено детальний опис команд і даних коректора **ОКВГ-01**. Дані представлені у вигляді структур на мові С. При цьому діють такі узгодження:

- розміри простих типів даних:
  - char, uchar – 1 байт;
  - short, ushort, int, uint – 2 байта;
  - long, ulong, float – 4 байта.
- порядок байт в простих типах даних – big endian (тобто, старший байт за молодшою адресою);
- поля в структурах не вирівнюються;
- код команди має тип uchar, а параметр команди – short.

Складні типи даних утворюються з простих. Повне визначення всіх типів даних, які використовуються, і констант наведено в додатку 1.

Код: 0 **Параметр:** 0 Порожня команда

Опис:

Ніяких дій не виконується, за виключенням встановлення прапорця в регістрі статусу.

Дані:

Відсутні.

Код: 1 **Параметр:** 0 Читання статусу slave

Опис:

Читання з slave блоку даних, який складається з двох байт статусу (sl_stat_t). Тип помилки визначається встановленням в ‘1’ відповідного біту (прапорця) в регістрі статусу. Перший байт характеризує попередній цикл обміну, а другий – статус виконання команди. Нульові значення обох байт статусу свідчать про відсутність помилок.

Дані:

```
typedef
struct {
   uchar io;       // статус попереднього циклу обміну:
                   //   &0x01 - вихід за таймаутом
                   //   &0x02 - помилка в командному пакеті (CRC, adr)
                   //   &0x04 - помилка в блоці даних (CRC)
                   //   &0x08 – зупинка приймання пакету
                   //           (прийнято адресний байт)
   uchar cmd;      // статус виконання команди:
                   //   &0x10 - невідома команда
                   //   &0x20 - помилка виконання команди
                   //           (неправильні параметри, …)
                   //   &0x80 – виконання попередньої команди
                   //           не закінчено
   }
sl_stat_t;
```

Код: 2 **Параметр:** <blksize> Запис блоку в slave (master -> slave)

Опис:

Запис в slave блоку даних розміром <blksize>. Розмір блоку може бути встановлений від 0 до максимального розміру буфера slave (2040 байт в даній реалізації). Якщо вказаний в команді розмір блоку перевищує розмір буфера, в регістрі статусу встановлюється прапорець помилки командного пакету і приймання блоку не відбувається.

Дані:

Масив байт розміром <blksize>

Код: 3 **Параметр:** <blksize> Читання блоку із slave (master <- slave)

Опис:

Читання з slave блоку даних розміром <blksize>. Розмір блоку може бути встановлений від 0 до максимального розміру буфера slave (2040 байт). Якщо вказаний в команді розмір блоку перевищує розмір буфера, в регістрі статусу встановлюється прапорець помилки командного пакету і передача блоку не відбувається.

Дані:

Масив байт розміром <blksize>

Код: 4 **Параметр:** 0 Закінчення сеансу зв'язку

Опис:

За даною командою slave закінчує поточний сеанс зв'язку і переходить в економний режим роботи ("**SLEEP**" на діаграмі станів slave).

Дані:

Відсутні.

Код: 5 **Параметр:** 0 Екран і поточний стан консолі

Опис:

Slave копіює в мережевий буфер поточний вміст екрану і значення параметрів роботи консолі (sl_con_t). Дана команда призначена для реалізації режиму віддаленого терміналу.

Дані:

```
typedef
struct {
   uchar mode;     // режим роботи консолі: CON_MODE_*
   uchar tmc;      // лічильник таймауту консолі, (cn_tm0..0), sec
   uchar edf;      // прапорець режиму редагування стрічки

      // наступні параметри дійсні тільки при
      // mode == CON_MODE_NORMAL
   uchar st;       // байт статусу консолі: CON_ST_*
   uchar adr;      // поточна адреса в LCD
   uchar crow;     // поточна позиція курсора: стрічка (0...CON_ROWS-1)
   uchar ccol;     // поточна позиція курсора: стовбець (0...CON_COLS-1)
   uchar scr[CON_ROWS*CON_COLS];    // буфер екрану (!!! lcd charset)
   }
sl_con_t;
```

Код: 6 **Параметр:** <scancode> Ввід символа в буфер клавіатури

Опис:

Значення параметра команди <scancode> вводиться в буфер клавіатури slave. Діапазон значень <scancode> - 0…255. Коди 0…31 призначені для імітації натискання окремих клавіш консолі або їх комбінацій. Решта значень (32…255) використовуються при вводі даних в режимі редагування і відповідають кодуванню CP-866. Команда призначена для реалізації режиму віддаленого терміналу.

Дані:

scancode	Клавіша
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 Виключення індикатора коректора

Опис:

За даною командою slave виключає LCD-індикатор консолі. Для його включення необхідно зимітувати за допомогою команди вводу символа в буфер клавіатури натиснення будь-якої клавіші.

Дані:

Відсутні.

Код: 8 **Параметр:** 0 Конфігурація: інформація

Опис:

В мережевий буфер копіюється інформація про коректор (sl_info_t) – серійний номер пристрою, версія і дата компіляції програмного забезпечення, а також інформація, яка вводиться користувачем при конфігуруванні. Символьні стрічки завершуються нульовим символом ‘\0’.

Дані:

```
typedef
struct {
   uchar s[6][20];   // 6 символьних стрічок: інформація, яка вводиться
                     //    користувачем
   uchar fw[6][20];  // 6 стрічок: 0,1-назва пристрою;
                     //            2,3-версія firmware;
                     //            4,5-дата/час компіляції;
   ushort sernum;    // серійний номер пристрою
   }
sl_info_t;
```

Код: 8 **Параметр:** 1 Конфігурація: давач температури

Опис:

В мережевий буфер копіюються параметри конфігурації давача температури (sl_temcfg_t).

Дані:

```
typedef
struct {
   uchar type;    // тип давача: TEM_RTD_*
   short R0;      // опір RTD при 0'C, R(Ohm)*100;
   uchar on;      // прапорець підключення давача: 1-підключений;
                  //    0-не підключений(викор. знач. def);
   float def;     // значення температури по замовчуваню
   float mn;      // граничне значення для alarm: min
   float mx;      // граничне значення для alarm: max
   uchar ps_on;   // прапорець вимірювання температури давача тиску:
                  // 0 - не підключений, 1 - підключений;
   short CfA[2];  // буфера таблиці передаточної функції каналу
   short CfB[2];  // вимірювання R*100 = F(adc);
   float CfK[2];  // y = (CfK[0] * x) + CfK[1];
   }
sl_temcfg_t;
```

Код: 8 **Параметр:** 2 Конфігурація: давач тиску

Опис:

В мережевий буфер копіюються параметри конфігурації давача тиску (sl_presscfg_t).

Дані:

```
typedef
struct {
   uchar type;    // тип давача: PR_SENS_*
   uchar on;      // прапорець підключення давача: 1-підключений;
                  //    0-не підключений(викор. знач. def);
   float def;     // значення тиску по замовчуваню, MPa;
   float mn;      // граничне значення для alarm: min, MPa
   float mx;      // граничне значеняе для alarm: max, MPa
   float vex;     // напруга живлення моста (паспортна), mV;
   float mvv;     // reserved;
   short CfA[2];  // буфера таблиці передаточної функції
   short CfB[2];  // каналу вимірювань U(mV)*100 = F(adc);
   float CfK[2];  //
      // таблиця функції перетворення давача тиску, P = F(u,t);
   short Tt[PR_TSZ_T];           // температура, 'C
   short Tu[PR_TSZ_T][PR_TSZ_U]; // напруга, mV*100
   short Tp[PR_TSZ_T][PR_TSZ_U]; // тиск, MPa*1000
   table_t pFu[PR_TSZ_T];
   table2_t F;
   }
sl_presscfg_t;
```

Код: 8 **Параметр:** 3 Конфігурація: давача об’єму

Опис:

В мережевий буфер копіюються параметри конфігурації тахометричного давача (sl_thcfg_t).

Дані:

```
typedef
struct {
   float   vn;    // об’єм (м3), для якого знімались характеристика
                  // турбінки; об’єм/(1 имп.): v1[i] = vn/N[i];
   short T[VOL_TSZ]; // період між імпульсами турбінки, 1==10msec
   short N[VOL_TSZ]; // кількість імпульсів для об’єму th.vn
   table_t Ftbl;     // таблиця функції перетворення давача об’єму
   }
sl_thcfg_t;
```

Код: 8 **Параметр:** 4 Конфігурація: пауза

Опис:

В мережевий буфер копіюються параметри конфігурації алгоритму обчислень об'єму газу за час паузи в роботі приладу (sl_pausecfg_t).

Дані:

```
typedef
struct {
   ulong mx;     // макс. тривалість відключення живлення, 1 == 0.01 год
   uchar ae;     // прапорець дозволу автоматичного додавання
                 // до основного об'єму об'єму за паузу
   }
sl_pausecfg_t;
```

Код: 8 **Параметр:** 5 Конфігурація: системні параметри

Опис:

В мережевий буфер копіюються системні параметри конфігурації (sl_syscfg_t) – період вимірювань, таймаут консолі, установки контролю напруги живлення, час початку облікової доби.

Дані:

```
typedef
struct {
   uchar mtu;      // період (sec) вимірювань і усереднення
   float mtf;      // період (sec) вимірювань і усереднення
   uchar ut0;      // timeout неактивності користувача, sec
   uchar uton;     // прапорці розширення timeout-а користувача
   uchar contrast;// контраст LCD
   ushort vpwmn;  // мінімальна напруга живлення, mV
   ushort vpwmx;  // максимальна напруга живлення, mV
   uchar apw;      // прапорець подачі аналогового живлення:
                   // 0: імпульсний режим;  1: включено весь час;
      // firmware >= v2.6
   uchar h0;       // година початку облікової доби, година: 0..23
   }
sl_syscfg_t;
```

Код: 8 **Параметр:** 6 Конфігурація: інтерфейс

Опис:

В мережевий буфер копіюються параметри конфігурації інтерфейсу (sl_siocfg_t).

Дані:

```
typedef
struct {
   uchar en;       // прапорець дозволу роботи інтерфейсу
   uchar baud;     // швидкість: SIO_BAUD_*
   uchar adr;      // мережевий адрес приладу: 1...126
   }
sl_siocfg_t;
```

Код: 8 **Параметр:** 7 Конфігурація: виходи

Опис:

В мережевий буфер копіюються параметри конфігурації вихідних ключів коректора (sl_siocfg_t) – тривалість імпульсів і функції кожного з виходів.

Дані:

```
typedef
struct {
```

```
   uchar ms;      // тривалість імпульсу, мсек (5..70)
   ushort tms;    // тривалість імпульсу, tmS(x)
   uchar mode[2]; // режим роботи ключів: OUT_*
   float v1[2];   // ціна одного імпульсу при OUT_VOL*, м3
   }
sl_outcfg_t;
```

Код: 8 **Параметр:** 8 Константи

Опис:

В мережевий буфер копіюються значення констант (sl_cst_t), які використовуються для розрахунку коефіцієнта стискуваності газу.

Дані:

```
typedef
struct {
   float    Dn;   // густина газу при норм. умовах, кг/м3
   float    aCO2; // концентрація вуглекислого газу в суміші, %
   float    aN2;  // концентрація азоту в суміші, %
   float    Pbar; // барометричний тиск, МРа;
   }
sl_cst_t;
```

Код: 8 **Параметр:** 9 Перехід на літній/зимовий час

Опис:

В мережевий буфер копіюються параметри переходу на літній/зимовий час (sl_dtj_t).

Дані:

```
typedef
struct {
   uchar mn;      // місяць, 0,1…12 (0 – перехід заборонено)
   uchar wn;      // тиждень місяця, 1…5 (5 означає остання)
   uchar wd;      // день тиждня, 0..6
   uchar h;       // година, 0..23
   }
dtjump_t;

typedef
struct {
   dtjump_t s;    // дата/час переходу на літній час
   dtjump_t w;    // дата/час переходу на зимовий час
   }
sl_dtj_t;
```

Код: 9 **Параметр:** 0 Поточне значення вимірювань

Опис:

В мережевий буфер копіюються поточні значення різних виміряних і обчислених параметрів (sl_curr_t).

Дані:

```
typedef
struct {
   dtime_t  dt;   // дата/час
   tem_t    tm;   // температура
   press_t  pr;   // тиск
   thcnt_t  th;   // лічильник імпульсів
   volume_t vl;   // приріст об'єму
   gaz_t    gz;   // температура, тиск, витрата ,об'єм
   pause_t  pz;   // пауза
   corr_t   cr;   // коеф. стискуваності і корекції
```

```
   sys_t    sys;  // напруга живлення, магн.поле
   event_t  evt[EVT_SZ];   // події
   }
sl_curr_t;
```

Код: 10 **Параметр:** 0 Читання стану черг історій і логів

Опис:

В мережевий буфер копіюється поточний стан черг історій вимірювань і логів подій (sl_queue_t). Також за даною командою проводиться обновлення додаткової копії стану черг, яка використовується в командах читання записів з історій та логів.

Дані:

```
typedef
struct {
   xque_t   hm;   // черга історії хвилинних вимірювань
   xque_t   hh;   // черга історії годинних вимірювань
   xque_t   hd;   // черга історії добових вимірювань
   xque_t   hmn;  // черга історії місячних вимірювань
   xque_t   le;   // черга логу подій
   xque_t   lc;   // черга логу змін
   }
sl_queue_t;
```

Код: 11 **Параметр:** <recnum> Читання запису з історії хвилинних вимірювань

Опис:

В мережевий буфер копіюється один запис з історії хвилинних вимірювань (gaz_t). Параметр команди <recnum> визначає номер запису: 0 – останній за часом запис, 1 – передостанній, і т.д. Значення <recnum> повинно бути меньшим за поточну кількість записів в історії, інакше в регістрі стану встановлюється прапорець помилки і команда не виконується.

При виконанні даної команди використовується додаткова копія стану черг, яка обновлюється тільки по команді читання поточного стану черг. Це зроблено з метою забезпечення цілісності даних при зчитуванні блоку записів.

Дані:

```
typedef
struct {
   dtime_t     dt;   // дата, час
   float       tem;  // середня температура, 'C
   float       pr;   // середній тиск (абсолютний), МРа
   float       Q;    // середня витрата, м3/ч
   float       Qn;   // середня витрата при нормальних умовах, м3/ч
   ulongfloat  V;    // об'єм, м3
   ulongfloat  Vn;   // об'єм при нормальних умовах, м3
   }
gaz_t;
```

Код: 12 **Параметр:** <recnum> Читання запису з історії годинних вимірювань

Опис:

В мережевий буфер копіюється один запис з історії годинних вимірювань (gaz_t). Параметр команди <recnum> визначає номер запису: 0 – останній за часом запис, 1 – передостанній, і т.д. Значення <recnum> повинно бути меньшим за поточну кількість записів в історії, інакше в регістрі стану встановлюється прапорець помилки і команда не виконується.

При виконанні даної команди використовується додаткова копія стану черг, яка обновлюється тільки по команді читання поточного стану черг. Це зроблено з метою забезпечення цілісності даних при зчитуванні блоку записів.

Дані:

```
typedef
struct {
   dtime_t     dt;   // дата, час
   float       tem;  // середня температура, 'C
   float       pr;   // середній тиск (абсолютний), МРа
   float       Q;    // середня витрата, м3/ч
   float       Qn;   // середня витрата при нормальних умовах, м3/ч
   ulongfloat  V;    // об'єм, м3
   ulongfloat  Vn;   // об'єм при нормальних умовах, м3
   }
gaz_t;
```

Код: 13 **Параметр:** <recnum> Читання запису з історії добових вимірювань

Опис:

В мережевий буфер копіюється один запис з історії добових вимірювань (gaz_t). Параметр команди <recnum> визначає номер запису: 0 – останній за часом запис, 1 – передостанній, і т.д. Значення <recnum> повинно бути меньшим за поточну кількість записів в історії, інакше в регістрі стану встановлюється прапорець помилки і команда не виконується.

При виконанні даної команди використовується додаткова копія стану черг, яка обновлюється тільки по команді читання поточного стану черг. Це зроблено з метою забезпечення цілісності даних при зчитуванні блоку записів.

Дані:

```
typedef
struct {
   dtime_t     dt;   // дата, час
   float       tem;  // середня температура, 'C
   float       pr;   // средній тиск (абсолютний), МРа
   float       Q;    // середня витрата, м3/ч
   float       Qn;   // середня витрата при нормальних умовах, м3/ч
   ulongfloat  V;    // об'єм, м3
   ulongfloat  Vn;   // об'єм при нормальних умовах, м3
   }
gaz_t;
```

Код: 14 **Параметр:** <recnum> Читання запису з історії місячних вимірювань

Опис:

В мережевий буфер копіюється один запис з історії місячних вимірювань (gaz_t). Параметр команди <recnum> визначає номер запису: 0 – останній за часом запис, 1 – передостанній, і т.д. Значення <recnum> повинно бути меньшим за поточну кількість записів в історії, інакше в регістрі стану встановлюється прапорець помилки і команда не виконується.

При виконанні даної команди використовується додаткова копія стану черг, яка обновлюється тільки по команді читання поточного стану черг. Це зроблено з метою забезпечення цілісності даних при зчитуванні блоку записів.

Дані:

```
typedef
struct {
   dtime_t     dt;   // дата, час
   float       tem;  // середня температура, 'C
   float       pr;   // середній тиск (абсолютний), МРа
```

```
   float       Q;    // середня витрата, м3/ч
   float       Qn;   // середня витрата при нормальних умовах, м3/ч
   ulongfloat  V;    // об'єм, м3
   ulongfloat  Vn;   // об'єм при нормальних умовах, м3
   }
gaz_t;
```

Код: 15 **Параметр:** <recnum> Читання запису з логу подій

Опис:

В мережевий буфер копіюється один запис з логу подій (evtlog_t). Параметр команди <recnum> визначає номер запису: 0 – останній за часом запис, 1 – передостанній, і т.д. Значення <recnum> повинно бути меньшим за поточну кількість записів в логу, інакше в регістрі стану встановлюється прапорець помилки і команда не виконується.

При виконанні даної команди використовується додаткова копія стану черг, яка обновлюється тільки по команді читання поточного стану черг. Це зроблено з метою забезпечення цілісності даних при зчитуванні блоку записів.

Дані:

```
typedef
struct {
   dtime_t dt;              // дата/час запису
   uchar id;                // ідентифікатор запису
   uchar bf[LOG_EBUF_SZ];   // буфер параметрів запису
   }
evtlog_t;
```

Код: 16 **Параметр:** <recnum> Читання запису з логу змін

Опис:

В мережевий буфер копіюється один запис з логу змін (chglog_t). Параметр команди <recnum> визначає номер запису: 0 – останній за часом запис, 1 – передостанній, і т.д. Значення <recnum> повинно бути меньшим за поточну кількість записів в логу, інакше в регістрі стану встановлюється прапорець помилки і команда не виконується.

При виконанні даної команди використовується додаткова копія стану черг, яка обновлюється тільки по команді читання поточного стану черг. Це зроблено з метою забезпечення цілісності даних при зчитуванні блоку записів.

Дані:

```
typedef
struct {
   dtime_t dt;              // дата/час запису
   uchar id;                // ідентифікатор запису
   uchar bf[LOG_CBUF_SZ];   // буфер параметрів запису
   }
chglog_t;
```

Додаток 1. Константи і типи даних 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 майстра
#define  NET_ADR_BROADCAST    127      // широкомовна адреса

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

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

   // net.stio
#define  NET_ST_OK            0x00     // попередня команда виконана нормально
#define  NET_ST_TOUT          0x01     // вихід по таймауту
#define  NET_ST_CMDERR        0x02     // помилка в командному пакеті (CRC, adr)
#define  NET_ST_DATERR        0x04     // помилка в блоці данних (CRC)
#define  NET_ST_BREAK         0x08     // переривання приймання пакету (прийнято адресний байт)
   // net.stcmd
#define  NET_ST_UNKNOWN       0x10     // невідома команда
#define  NET_ST_EXEERR        0x20     // помилка виконання команди (неправильні параметри, ...)
#define  NET_ST_BSY           0x80     // виконання попередньої команди не закінчено


   // коди команд
#define  NET_CMD_NULL         0     // порожня команда
#define  NET_CMD_STAT         1     // видати байт статусу
#define  NET_CMD_WR           2     // запис блоку в slave  (master -> slave)
#define  NET_CMD_RD           3     // читання блоку із slave (master <- slave)
#define  NET_CMD_END          4     // команда завершення сесії
#define  NET_CMD_CONS         5     // екран і поточний стан консолі (sl_con_t)
#define  NET_CMD_KEYB         6     // ввід символа в буфер клавіатури
#define  NET_CMD_LCDOFF       7     // виключити індикатор коректора

#define  NET_CMD_GETCFG       8     // отримати параметри конфігурації;

   // значення параметра команди NET_CMD_GETCFG
#define  SL_CFG_INFO          0     // конфігурація: інформація (sl_info_t)
#define  SL_CFG_TEM           1     // конфігурація: давач температури (sl_temcfg_t)
#define  SL_CFG_PRESS         2     // конфігурація: давач тиску (sl_presscfg_t)
#define  SL_CFG_TH            3     // конфігурація: тахометричний давач (sl_thcfg_t)
#define  SL_CFG_PAUSE         4     // конфігурація: пауза (sl_pausecfg_t)
#define  SL_CFG_SYS           5     // конфігурація: системні параметри (sl_syscfg_t)
#define  SL_CFG_SIO           6     // конфігурація: інтерфейс (sl_siocfg_t)
#define  SL_CFG_OUT           7     // конфігурація: виходи (sl_outcfg_t)
#define  SL_CFG_CST           8     // константи (sl_cst_t)
#define  SL_CFG_DTJ           9     // параметри переходу на літній/зимовий час (sl_dtj_t)

#define  NET_CMD_CURRENT      9     // отримати поточне значення параметрів (sl_curr_t)

#define  NET_CMD_QUEUE        10    // отримати поточне значення черг історій вимірювань
                                    // і логів (sl_queue_t)

   // команди отримання записів з історій (NET_CMD_HIST*) або логів (NET_CMD_LOG*);
   // параметром команди задається номер запису: 0 – останній запис, 1 - передостанній, ...
#define  NET_CMD_HISTM        11    // отримати один запис з історії хвилинних вимірювань (gaz_t)
#define  NET_CMD_HISTH        12    // отримати один запис з історії годинних вимірювань (gaz_t)
#define  NET_CMD_HISTD        13    // отримати один запис з історії добових вимірювань (gaz_t)
#define  NET_CMD_HISTMN       14    // отримати один запис з історії місячних вимірювань (gaz_t)
#define  NET_CMD_LOGEV        15    // отримати один запис з логу подій (evtlog_t)
#define  NET_CMD_LOGCH        16    // отримати один запис з логу змін (chglog_t)


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

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

   // режими консолі
#define  CON_MODE_NORMAL      0
#define  CON_MODE_IDLE        1
#define  CON_MODE_SHUTDOWN    2

   // байт статуса консолі
#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 давача температури
#define  TEM_RTD_Pt     0
#define  TEM_RTD_Cu     1

#define  TEM_ON         1
#define  TEM_OFF        0

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

   // тип давача тиску
#define  PR_SENS_ABS    0     // давач абсолютного тиску
#define  PR_SENS_GAGE   1     // давач надлишкового тиску

#define  PR_ON          1
#define  PR_OFF         0

#define  PR_TSZ_U       8     // макс. кількість точок по напрузі
#define  PR_TSZ_T       8     // макс. кількість точок по температурі

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

#define  VOL_TSZ        32    // макс. розмір таблиці функції перетворення турбінки

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

#define  LOG_ESZ           200   // макс. кількість записів у логу подій
#define  LOG_CSZ           200   // макс. кількість записів у логу вимірювань
#define  LOG_EBUF_SZ       4     // розмір буфера параметрів логу подій
#define  LOG_CBUF_SZ       8     // розмір буфера параметрів логу змін

#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 нормі
#define  EVT_MIN           1     // параметр < мін.
#define  EVT_MAX           2     // параметр > макс.

   // ідентифікатори лога подій
#define  LOG_NULL          0
#define  LOG_PWOFF         1     // відключення живлення процесора
#define  LOG_PWON          2     // включення живлення процесора
#define  LOG_RESET         3     // скид (без відключення живлення)
#define  LOG_DTDEF         4     // системний час встановлено в default
#define  LOG_CFGDEF        5     // конфігурація встановлена в default

#define  LOG_TEM_NORM      6     // температура в нормі
#define  LOG_TEM_MIN       7     // температура < мін.
#define  LOG_TEM_MAX       8     // температура > макс.

#define  LOG_PRESS_NORM    9     // тиск в нормі
#define  LOG_PRESS_MIN     10    // тиск < мін.
#define  LOG_PRESS_MAX     11    // тиск > макс.

#define  LOG_POW_NORM      12    // напруга живлення в нормі
```

```
#define  LOG_POW_MIN       13    // напруга живлення < мін.
#define  LOG_POW_MAX       14    // напруга живлення > макс.

#define  LOG_MFLD_NORM     15    // зовнішнє магнітне поле відсутнє
#define  LOG_MFLD_MAX      16    // зовнішнє магнітне поле > макс.

#define  LOG_SUMMER_TIME   17    // перехід на літній час
#define  LOG_WINTER_TIME   18    // перехід на зимовий час

#define  LOG_PAUSE_BEGIN   19    // початок паузи
#define  LOG_PAUSE_END     20    // закінчення паузи
#define  LOG_PAUSE_AUTOADD 21    // автододавання до загального об'єму


   // ідентифікатори логу змін
#define  CHG_NULL          0
#define  CHG_INFO          1     // загальна інформація

#define  CHG_TEM_TYPE      2     // тип RTD
#define  CHG_TEM_R0        3     // опір RTD при 0'C
#define  CHG_TEM_ONOFF     4     // підключення RTD
#define  CHG_TEM_DEF       5     // температура по замовчуванню
#define  CHG_TEM_MIN       6     // мінімальна межа температури
#define  CHG_TEM_MAX       7     // максимальна межа температури
#define  CHG_TEM_PSEN      8     // вимірювання температури давача тиску: on/off

#define  CHG_PR_TYPE       9     // тип давача тиску: abs | gage
#define  CHG_PR_VEX        10    // паспортна напруга живлення моста
#define  CHG_PR_MVV        11    // reserved
#define  CHG_PR_ONOFF      12    // підключення давача тиску
#define  CHG_PR_DEF        13    // тиск по замовчуванню
#define  CHG_PR_MIN        14    // мінімальна межа тиску
#define  CHG_PR_MAX        15    // максимальна межа тиску

#define  CHG_SYS_MEASURE   16    // період вимірювань
#define  CHG_SYS_CONS      17    // consol timeout: on/off
#define  CHG_SYS_TOUT      18    // consol timeout: sec
#define  CHG_SYS_PWMIN     19    // мінімальна напруга живлення
#define  CHG_SYS_PWMAX     20    // максимальна напруга живлення

#define  CHG_PZ_MAX        21    // макс. тривалість паузи
#define  CHG_PZ_AUTOADD    22    // автоматичне додавання

#define  CHG_VOL_FUNC      23    // функція перетворення турбінки
#define  CHG_PR_FUNC       24    // функція перетворення давача тиску

#define  CHG_ACCESS        25    // рівень доступу
#define  CHG_PWD_OPER      26    // пароль оператора
#define  CHG_PWD_ADMIN     27    // пароль адміністратора
#define  CHG_PWD_OWNER     28    // пароль власника

#define  CHG_DENSITY       29    // густина при нормальних умовах
#define  CHG_CO2           30    // концентрація вуглекислого газу в суміші
#define  CHG_NITROGEN      31    // концентрація азоту в суміші
#define  CHG_PBAR          32    // барометричний тиск

#define  CHG_DTIME         33    // дата/година в RTC
#define  CHG_SUMMER_JUMP   34    // дата/година переходу на літній час
#define  CHG_WINTER_JUMP   35    // дата/година переходу на зимовий час

#define  CHG_PZ_ADDVOL     36    // додавання об'єму за паузу до загального об'єму

#define  CHG_CLR_VOLUME    37    // обнулення поточних об'ємів V і Vn

#define  CHG_CALIBR_T1P    38    // одноточкова калібровка каналу температури
#define  CHG_CALIBR_T2P    39    // двоточкова калібровка каналу температури
#define  CHG_CALIBR_P1P    40    // одноточкова калібровка каналу тиску
#define  CHG_CALIBR_P2P    41    // двоточкова калібровка каналу тиску
#define  CHG_CALIBR_R      42    // калібровка каналу температури: опорні точки R1,R2
#define  CHG_CALIBR_U      43    // калібровка каналу тиску: опорні точки U1,U2

#define  CHG_COUNTER       44    // лічильник імпульсів

#define  CHG_DEFPWD        45    // паролі по замовчуванню

#define  CHG_SIO_ONOFF     46    // інтерфейс on/off
#define  CHG_SIO_BAUD      47    // швидкість інтерфейсу
#define  CHG_SIO_ADR       48    // мережевий адрес інтерфейсу

#define  CHG_PULSE_WIDTH   49    // мінімальна тривалість імпульсів на виходах
```

```
#define  CHG_OUT1_MODE     50    // вихід1: режим роботи
#define  CHG_OUT2_MODE     51    // вихід2: режим роботи
#define  CHG_OUT1_VOL      52    // вихід1: ціна одного імпульсу для об'єму
#define  CHG_OUT2_VOL      53    // вихід2: ціна одного імпульсу для об'єму

#define  CHG_SYS_H0        54    // година початку облікової доби



/*---- 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;      // ціла частина числа
   float fr;      // дробова частина числа
   }
ulongfloat;


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


#pragma  ZTC align 1

typedef
struct {
   short n;          // розмір таблиці
   uchar xlim;       // =1: обмеження аргументу, якщо він виходить за межі

   //int *x;           // масив значень аргументу
   uchar xm;         // тип пам'яті
   ushort xa;        // адреса

   //int *f;           // масив значень функцій
   uchar fm;         // тип пам'яті
   ushort fa;        // адреса

   float scale;      // маштабуючий коефіцієнт
   }
table_t;


typedef
struct {
   short nx;         // розмір таблиці по x
   uchar xlim;       // =1: обмеження x

   //int *x;           // масив значень аргументу x
   uchar xm;         // тип пам'яті
   ushort xa;        // адреса

   //table_t *f;       // масив таблиць одновимірних функцій z=f(y), (для
                       // кожного елемента масиву аргументів x своя функція f)
   uchar fm;         // тип пам'яті
   ushort fa;        // адреса

   float scale;      // маштабуючий коефіцієнт
   }
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;         /* поточний індекс */
   ushort q;         /* розмір черги (кількість елементів в черзі) */
   ushort n;         /* розмір буфера (кількість елементів) */
   ushort sz;        /* розмір елемента (байт) */
   //void *bf;         /* -> на буфер */
   uchar bfm;        // тип пам'яті
   ushort bfa;       // адреса
   }
xque_t;


typedef
struct {
   float gC;         // температура газу, 'C
   float gK;         // температура газу, 'K
   float pC;         // температура давача тиску, 'C
   short R;          // виміряний опір RTD, R(Ohm)*100
   }
tem_t;


typedef
struct {
   float gaz;        // тиск газу (абсолютний), MPa;
   short U;          // виміряна напруга містка, mV;
   float vsc;        // маштабуючий коефіцієнт, який приводить виміряну
                     // вих. напругу містка до діапазону паспортних значень
   }
press_t;


typedef
struct {
   uchar f;          // прапорець зміни стану лічильника імпульсів
   ulong cnt;        // поточне значення лічильника імпульсів тахометра
   dtime_t t1;       // дата/час надходження попереднього імпульсу
   dtime_t t2;       // дата/час надходження останнього іимпульсу
   ulong hs;         // період надходження импульсів (остан. зн.), hsec
   ulong xhs;        // сума періодів імпульсів, які надійшли за час
                     // одного циклу обчислень, hsec
   }
thcnt_t;


typedef
struct {
   float dV;         // поточний приріст об'єму, м3
   float dVn;        // поточний приріст об'єму при нормальних умовах, м3
   float Q;          // поточна витрата, м3/ч
   float Qn;         // поточна витрата при нормальних умовах, м3/ч
   }
volume_t;


typedef
struct {
   dtime_t dt;       // дата, час
   float   tem;      // середня температура, 'C
   float   pr;       // середній тиск (абсолютний), MPa
   float   Q;        // середня витрата, м3/ч
   float   Qn;       // середня витрата при нормальних умовах, м3/ч
   ulongfloat  V;    // об'єм, м3
   ulongfloat  Vn;   // об'єм при нормальних умовах, м3
   }
gaz_t;


typedef
struct {
   dtime_t dt[2];    // година початку і кінця паузи
   ulong   hs;       // тривалість паузи, hsec
   float   Q;        // середня витрата під час паузи, м3/ч
   float   Qn;       // середня витрата при нормальних умовах, м3/ч
   float   dV;       // об'єм за час останньої паузи, м3
   float   dVn;      // об'єм за час останньої паузи при норм.умовах, м3
   ulongfloat  V;    // сумарний об'єм за декілька пауз, м3
```

```
   ulongfloat  Vn;   // сумарний об'єм за декілька пауз при норм.умовах, м3
   }
pause_t;


typedef
struct {
   float Pkr;        // псевдокритичний тиск, кгс/см2
   float Tkr;        // псевдокритична температура, 'K
   float Pc;         // приведений тиск
   float Tc;         // приведена температура
   float Z;          // коефіцієнт стискуваності газу
   float K;          // коефіцієнт корекції
   }
corr_t;


typedef
struct {
   ushort vpow;      // напруга живлення батареї, mV
   uchar mfld;       // стан давача зовнішнього магнітного поля:
                     // 0 - поле є;  1: - поле відсутнє;
   }
sys_t;


typedef
struct {
   uchar   st;       // поточний стан "події": 0 - норма
   uchar   id;       // ідентифікатор запису для лога
   dtime_t dt;       // година початку події
   }
event_t;


typedef
struct {
   dtime_t dt;       // дата/час запису
   uchar   id;       // ідентифікатор запису
   uchar   bf[LOG_EBUF_SZ];   // буфер параметрів запису
   }
evtlog_t;


typedef
struct {
   dtime_t dt;       // дата/час запису
   uchar   id;       // ідентифікатор запису
   uchar   bf[LOG_CBUF_SZ];   // буфер параметрів запису
   }
chglog_t;


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

   // slave: стан консолі і вміст екрану;  команда NET_CMD_CONS
typedef
struct {
   uchar mode;       // режим роботи консолі: CON_MODE_*
   uchar tmc;        // лічильник таймаута консолі, (cn_tm0..0), sec
   uchar edf;        // прапорець режиму редагування стрічки

      // наступні параметри дійсні тільки при  mode == CON_MODE_NORMAL
   uchar st;         // байт статусу консолі: CON_ST_*
   uchar adr;        // поточний адрес в LCD
   uchar crow;       // поточна позиція курсору: стрічка (0...CON_ROWS-1)
   uchar ccol;       // поточна позиція курсору: стовбець (0...CON_COLS-1)
   uchar scr[CON_ROWS*CON_COLS];    // буфер екрану (!!! lcd charset)
   }
sl_con_t;


   // slave: інформація; команда NET_CMD_INFO
typedef
struct {
   uchar s[6][20];   // 6 символьних стрічок: інформація яка вводиться користувачем
   uchar fw[6][20];  // 6 стрічок: 0,1-назва приладу; 2,3-версія firmware;
                     // 4,5-дата/час компіляції;
   ushort sernum;    // серійний номер приладу
   }
```

```
sl_info_t;

   // slave: конфігурація давача температури; команда NET_CMD_TEMCFG
typedef
struct {
   uchar type;       // тип давача: TEM_RTD_*
   short R0;         // опір RTD при 0'C, R(Ohm)*100;
   uchar on;         // прапорець підключення давача: 1-підключений;
                     //    0-не підключений(викор. знач. def);
   float def;        // значення температури по замовчуванню
   float mn;         // граничне значення для alarm: min
   float mx;         // граничне значення для alarm: max
   uchar ps_on;      // прапорець вимірювання температури давача тиску;
                     // 0 - не підключений; 1 - підключений;
   short CfA[2];     // буфера таблиці передаточної функції каналу
   short CfB[2];     // вимірювання R*100 = F(adc);
   float CfK[2];     // y = (CfK[0] * x) + CfK[1];
   }
sl_temcfg_t;


   // slave: конфігурація давача тиску; команда NET_CMD_PRESSCFG
typedef
struct {
   uchar type;       // тип давача: PR_SENS_*
   uchar on;         // прапорець підключення давача: 1-підключений;
                     //    0-не підключений(викор. знач. pr.def);
   float def;        // значення тиску по замовчуванню, MPa;
   float mn;         // граничне значення для alarm: min, MPa
   float mx;         // граничне значення для alarm: max, MPa
   float vex;        // напруга живлення моста (паспортна), mV;
   float mvv;        // reserved

      // буфера таблиці передаточної функції каналу вимірювання U(mV)*100 = F(adc);
   short CfA[2];     //
   short CfB[2];     //
   float CfK[2];     //

      // таблиця функції перетворення давача тиску, P = F(u,t);
   short Tt[PR_TSZ_T];           // температура, 'C
   short Tu[PR_TSZ_T][PR_TSZ_U]; // напруга, mV*100
   short Tp[PR_TSZ_T][PR_TSZ_U]; // тиск, MPa*1000
   table_t pFu[PR_TSZ_T];
   table2_t F;
   }
sl_presscfg_t;

   // slave: конфігурація тахометричного давача; команда NET_CMD_THCFG
typedef
struct {
   float   vn;       // об'єм (м3), для якого знімалась характеристика
                     // турбінки; об'єм/(1 имп.): dV1 = th.vn / Ni;
   short T[VOL_TSZ]; // період між імпульсами турбінки, 1==10msec
   short N[VOL_TSZ]; // кількість імпульсів для об'єму th.vn
   table_t Ftbl;     // таблиця функції перетворення давача
   }
sl_thcfg_t;


   // slave: конфігурація паузи; команда NET_CMD_PAUSECFG
typedef
struct {
   ulong mx;         // макс. тривалість відключення живлення, 1 == 0.01 год
   uchar ae;         // прапорець дозволу автоматичного додавання
                     // до загального об'єму за паузу
   }
sl_pausecfg_t;


   // slave: системні установки; команда NET_CMD_SYSCFG
typedef
struct {
   uchar mtu;        // період (sec) вимірювань і усереднення
   float mtf;        // період (sec) вимірювань і усереднення
   uchar ut0;        // timeout неактивності користувача, sec
   uchar uton;       // прапорець дозволу timeout-а користувача
   uchar contrast;   // контраст LCD
   ushort vpwmn;     // мінімальна напруга живлення, mV
   ushort vpwmx;     // максимальна напруга живлення, mV
   uchar apw;        // прапорець подачі аналогового живлення:
```

```
                    // 0: імпульсний режим;  1: включене завжди;

      // firmware >= v2.6
   uchar h0;         // година початку облікової доби, год: 0..23
   }
sl_syscfg_t;


   // slave: конфігурація інтерфейсу; команда NET_CMD_SIOCFG
typedef
struct {
   uchar en;         // прапорець дозволу роботи інтерфейсу
   uchar baud;       // швидкість: SIO_BAUD_*
   uchar adr;        // мережевий адрес приладу: 1...126
   }
sl_siocfg_t;


   // slave: конфігурація вихідних ключів; команда NET_CMD_OUTCFG
typedef
struct {
   uchar ms;         // тривалість імпульсу, мсек (5..70)
   ushort tms;       // тривалість імпульсу, tmS(x)
   uchar mode[2];    // режим роботи ключів: OUT_*
   float v1[2];      // ціна одного імпульсу при OUT_VOL*, м3
   }
sl_outcfg_t;


   // slave: значення констант; команда NET_CMD_CST
typedef
struct {
   float    Dn;      // густина газу при норм. умовах, кг/м3
   float    aCO2;    // концентрація вуглекислого газу в суміші, %
   float    aN2;     // концентрація азоту в суміші, %
   float    Pbar;    // барометричний тиск, MPa;
   }
sl_cst_t;


   // slave: параметри переходу на літній/зимовий час; команда NET_CMD_DTJ
typedef
struct {
   uchar mn;         // month,   0,1..12  (0 - перехід заборонено)
   uchar wn;         // weeknum of month, 1..5  (5 означає остання )
   uchar wd;         // weekday, 0..6
   uchar h;          // hour,    0..23
   }
dtjump_t;

typedef
struct {
   dtjump_t s;       // дата/час переходу на літній час
   dtjump_t w;       // дата/час переходу на зимовий час
   }
sl_dtj_t;


   // slave: поточне значення параметрів; команда NET_CMD_CURRENT
typedef
struct {
   dtime_t  dt;      // дата/час
   tem_t    tm;      // температура
   press_t  pr;      // тиск
   thcnt_t  th;      // лічильник імпульсів
   volume_t vl;      // приріст об'єму
   gaz_t    gz;      // температура, тиск, витрата, об'єм
   pause_t  pz;      // пауза
   corr_t   cr;      // коеф. стискуваності і корекції
   sys_t    sys;     // напруга живлення, магн.поле
   event_t  evt[EVT_SZ];   // події
   }
sl_curr_t;


   // slave: поточний стан черг історій та логів;  команда NET_CMD_QUEUE
typedef
struct {
   xque_t   hm;      // черга історії хвилинних вимірювань
   xque_t   hh;      // черга історії годинних вимірювань
```

```
   xque_t   hd;      // черга історії добових вимірювань
   xque_t   hmn;     // черга історії місячних вимірювань
   xque_t   le;      // черга логу подій
   xque_t   lc;      // черга логу змін
   }
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)


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

Додаток 2. Обчислення контрольної суми CRC-16

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


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

uint crc;         // поточне значення

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 ----------------------------------------------------*/

/*****************************************/
/* Обчислення наступного значення 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 ];
}


/*************************/
/* Початкове значення CRC */
/*************************/
void crc_init (void)
{
   crc = 0;
}
```

Автор:

Сергій Шендерук
Львівський центр Інституту космічних досліджень

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