

Теплосчетчик «МТР-06» позволяет вести учет параметров по 3-м независимым контурам, в каждый из которых может входить до 2-х точек измерения параметров ("подающая" и "обратная"), на которых измеряются расходы, температуры и давления теплоносителя. Ежечасно производится сохранение измеренных и вычисленных величин в по всем контурам в архиве.

Архив представляет собой кольцевой буфер, в который каждый час записывается срез состояния прибора, описываемый такими структурами данных:

```
struct time_date
{
    uchar    hour ,
            day  ,
            month,
            year ;
} ;

struct u_data
{
    struct time_date timedate;
    float  Q [3], // тепловая энергия нарастающим итогом по контурам
            // Гкал
    M [6], // масса нарастающим итогом по точкам измерения
            // (по две на контур - подающая и обратная), т
    t [6], // среднечасовые температуры по точкам измерения
            // (по две на контур - подающая и обратная),
            // град С
    p [6]; // среднечасовые давления по точкам измерения
            // (по две на контур - подающая и обратная), атм
    unsigned long  T [3]; // время исправной работы каждого контура
            // мин
    unsigned char  Nstart; // число включений. перезапусков
    unsigned long  Err[3]; // Код состояния каждого контура
    unsigned char  event, // тип события
            min, // минуты - для времени события
            Tfmn[3], // время блокировки по расх < отс
            Tfmx[3], // время блокировки по расх > max
            Tdtm[3], // время блокировки по dt < min
            arc_fmt, // Формат архива (расширенный или обычный)
            res3 ; // Зарезервировано
    unsigned int    crc ; // Пока не используется
};
```

Длина структуры u_data равна 128 байт.

Поле *event* может содержать следующие значения:

```
#define  EVENT_HOUR_END      0xA0 // Обычная часовая запись
#define  EVENT_DAY_END      0xA1 // Окончание учетных суток
#define  EVENT_MONTH_END    0xA2 // Окончание учетного месяца
#define  EVENT_POWER_ON     0xAA // Запись о включении питания
#define  EVENT_POWER_OFF    0xAB // Запись о выключении питания
```

Время исправной работы учитывается в минутах. Для записей, у которых поле *event* содержит значение `EVENT_HOUR_END`, `EVENT_POWER_ON` или `EVENT_POWER_OFF`, значение времени исправной работы (поле *T*) сохраняется в виде нарастающего итога. Поля *Tfmn*, *Tfmx* и *Tdtm* содержат значения времени за соответствующий последний час работы. Поле *Err* содержит суперпозицию кодов ошибок за последний час работы. Для записей, у которых поле *event* содержит значение `EVENT_DAY_END`, поле *T* содержит время исправной работы за указанные сутки. Поля *Tfmn*, *Tfmx* и *Tdtm* **не использовать** (см. описание расширенной части архивной записи)! Поле *Err* содержит суперпозицию кодов ошибок за указанные сутки.

Если поле *arc_fmt* содержит значение 2 или выше (кроме 255), то формат архива расширенный. Это означает, что архивная запись состоит из двух частей. Первая часть записи («стандартная») как и прежде описывается структурой данных, приведенной выше. Вторая часть («расширенная») описывается структурой *u_data_tail*:

```
struct u_data_tail
{
    float pre_Q[3], // Младшая часть накопителей Q по трем системам
          pre_M[6], // Младшая часть накопителей M по шести каналам
          V[6],     // Объем
          pre_V[6], // Расширяет разрядность V
          t_cold,   // Температура холодной воды
          t_amb;    // Температура окружающей среды
    unsigned char ch[128-92-2];
    unsigned int crc; // 2
};
```

Начиная с версии архива 5 в структуру добавлены поля *Terr*[3] и *Top*, содержащие счётчики времени (нарастающим итогом) в состоянии ошибки по каждому контуру учёта и времени (нарастающим итогом) во включённом состоянии соответственно. В версии архива 6 в структуру добавлены счётчики в нахождение контура учёта в состоянии специфической ошибки, повторяющие одноименные поля из стандартной части архивной записи, но имеющие удвоенную разрядность, позволяющие их использовать для суточных (поле *event* содержит значение `EVENT_DAY_END`) и месячных (поле *event* содержит значение `EVENT_MONTH_END`):

- *Tfmn* — время блокировки учёта, когда расход меньше заданного в конфигурации минимального значения;
- *Tfmx* — время блокировки учёта, когда расход больше заданного в конфигурации максимального значения;

- Tdtm — время блокировки учёта, когда значение разности температур меньше заданного в конфигурации минимального значения.

Все счётчики времени ведут учёт в минутах.

```
struct u_data_tail
{
    float pre_Q[3], // Младшая часть накопителей Q по трем системам
          pre_M[6], // Младшая часть накопителей M по шести каналам
          V[6],      // Объем
          pre_V[6],  // Расширяет разрядность V
          t_cold,    // Температура холодной воды
          t_amb;     // Температура окружающей среды
    unsigned long Terr[3]; // 12 104
    unsigned long Ton;     // 4 108
    unsigned short
        Tfmn[3], // 6 114 время блокировки по расх < отс
        Tfmx[3], // 6 120 время блокировки по расх > max
        Tdtm[3]; // 6 126 время блокировки по dt < min
    unsigned int crc; // 2
};
```

Поля pre_Q и pre_M расширяют разрядность полей Q и M соответственно структуры u_data. Для того, чтобы использовать младшие части накопителей, можно, например, преобразовать значения в полях Q и M к значениям с удвоенной разрядностью (к примеру тип *double* языка C/C++), после чего прибавить к ним значения полей pre_Q и pre_M соответственно. Аналогично можно использовать поля V и pre_V структуры u_data_tail.

Считывание второй части архивной записи также производится при помощи функции 64H протокола ModBus, но тип операции при этом должен быть равен 6 (для старых версий ПО — 02.xx.xx — при версии архива 2 и выше используется тип операции 3).

При включении программа сдвигает указатель записи на следующую ячейку архива и записывает туда запись с кодом EVENT_POWER_ON. В течении часа прибор постоянно пишет запись с кодом EVENT_POWER_OFF в одну и ту же ячейку, а в конце часа пишет поверх этой записи запись с кодом EVENT_HOUR_END. Таким образом точно до одной минуты можно определить время, в течении которого прибор был выключен. Записи с остальными кодами можно игнорировать.

Поиск записи с требуемыми датой и временем выполняется перебором. Однако, при нормальной работе (когда не происходило выключений) можно предполагать наличие в архиве только 24 часовых записей + 1 записи суточной на одни сутки и таким образом вычислять примерное расположение требуемой записи. Если такое

предположение окажется неверным, выполнять "уточняющий поиск" в нужном направлении.

Начиная с версии протокола 12 введена возможность считывать отдельные суточные и месячные архивы. Количество записей в них можно считать при помощи функции 66h и типов операций 9 и 10 соответственно. Формат архивных записей такой же, как и для обычного архива. Считывание самих архивных записей выполняется при помощи функции 64h, но тип операций должен быть 7 — 10 для суточного и 11 — 14 для месячного архивов (см. описание протокола обмена).

Используемые форматы представления чисел:

unsigned char, uchar	беззнаковое целое, 8 бит
unsigned int	беззнаковое целое, 16 бит, старший байт расположен по младшему адресу (big endian)
unsigned long	беззнаковое целое, 32 бита, старший байт расположен по младшему адресу (big endian)
float	соответствуют числу с плавающей точкой по стандарту IEEE-754, однако расположение байт обратное

Настройки интерфейса RS-232:

Скорость	9600
Число бит данных	8
Контроль четности	нет
Число стоповых бит	1

Целостность каждой половины архивной записи можно контролировать вычисляя 16-битный Cyclical Redundancy Check (CRC16) и сравнивая это значение с соответствующим полем архивной записи (поле crc структур u_data и u_data_tail). Для вычисления CRC16 можно воспользоваться следующим кодом на языке программирования C/C++:

```
/* Table of CRC values for high-order byte */
unsigned char auchCRCHi[] = {
0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x81, 0x91, 0xa1,
0xb1, 0xc1, 0xd1, 0xe1, 0xf1,
0x12, 0x02, 0x32, 0x22, 0x52, 0x42, 0x72, 0x62, 0x93, 0x83, 0xb3,
0xa3, 0xd3, 0xc3, 0xf3, 0xe3,
0x24, 0x34, 0x04, 0x14, 0x64, 0x74, 0x44, 0x54, 0xa5, 0xb5, 0x85,
0x95, 0xe5, 0xf5, 0xc5, 0xd5,
0x36, 0x26, 0x16, 0x06, 0x76, 0x66, 0x56, 0x46, 0xb7, 0xa7, 0x97,
0x87, 0xf7, 0xe7, 0xd7, 0xc7,
0x48, 0x58, 0x68, 0x78, 0x08, 0x18, 0x28, 0x38, 0xc9, 0xd9, 0xe9,
0xf9, 0x89, 0x99, 0xa9, 0xb9,
```

```

0x5a, 0x4a, 0x7a, 0x6a, 0x1a, 0x0a, 0x3a, 0x2a, 0xdb, 0xcb, 0xfb,
0xeb, 0x9b, 0x8b, 0xbb, 0xab,
0x6c, 0x7c, 0x4c, 0x5c, 0x2c, 0x3c, 0x0c, 0x1c, 0xed, 0xfd, 0xcd,
0xdd, 0xad, 0xbd, 0x8d, 0x9d,
0x7e, 0x6e, 0x5e, 0x4e, 0x3e, 0x2e, 0x1e, 0x0e, 0xff, 0xef, 0xdf,
0xcf, 0xbf, 0xaf, 0x9f, 0x8f,
0x91, 0x81, 0xb1, 0xa1, 0xd1, 0xc1, 0xf1, 0xe1, 0x10, 0x00, 0x30,
0x20, 0x50, 0x40, 0x70, 0x60,
0x83, 0x93, 0xa3, 0xb3, 0xc3, 0xd3, 0xe3, 0xf3, 0x02, 0x12, 0x22,
0x32, 0x42, 0x52, 0x62, 0x72,
0xb5, 0xa5, 0x95, 0x85, 0xf5, 0xe5, 0xd5, 0xc5, 0x34, 0x24, 0x14,
0x04, 0x74, 0x64, 0x54, 0x44,
0xa7, 0xb7, 0x87, 0x97, 0xe7, 0xf7, 0xc7, 0xd7, 0x26, 0x36, 0x06,
0x16, 0x66, 0x76, 0x46, 0x56,
0xd9, 0xc9, 0xf9, 0xe9, 0x99, 0x89, 0xb9, 0xa9, 0x58, 0x48, 0x78,
0x68, 0x18, 0x08, 0x38, 0x28,
0xcb, 0xdb, 0xeb, 0xfb, 0x8b, 0x9b, 0xab, 0xbb, 0x4a, 0x5a, 0x6a,
0x7a, 0x0a, 0x1a, 0x2a, 0x3a,
0xfd, 0xed, 0xdd, 0xcd, 0xbd, 0xad, 0x9d, 0x8d, 0x7c, 0x6c, 0x5c,
0x4c, 0x3c, 0x2c, 0x1c, 0x0c,
0xef, 0xff, 0xcf, 0xdf, 0xaf, 0xbf, 0x8f, 0x9f, 0x6e, 0x7e, 0x4e,
0x5e, 0x2e, 0x3e, 0x0e, 0x1e
} ;
/* Table of CRC values for low-order byte */
unsigned char auchCRCLo[] = {
0x00, 0x21, 0x42, 0x63, 0x84, 0xa5, 0xc6, 0xe7, 0x08, 0x29, 0x4a,
0x6b, 0x8c, 0xad, 0xce, 0xef,
0x31, 0x10, 0x73, 0x52, 0xb5, 0x94, 0xf7, 0xd6, 0x39, 0x18, 0x7b,
0x5a, 0xbd, 0x9c, 0xff, 0xde,
0x62, 0x43, 0x20, 0x01, 0xe6, 0xc7, 0xa4, 0x85, 0x6a, 0x4b, 0x28,
0x09, 0xee, 0xcf, 0xac, 0x8d,
0x53, 0x72, 0x11, 0x30, 0xd7, 0xf6, 0x95, 0xb4, 0x5b, 0x7a, 0x19,
0x38, 0xdf, 0xfe, 0x9d, 0xbc,
0xc4, 0xe5, 0x86, 0xa7, 0x40, 0x61, 0x02, 0x23, 0xcc, 0xed, 0x8e,
0xaf, 0x48, 0x69, 0x0a, 0x2b,
0xf5, 0xd4, 0xb7, 0x96, 0x71, 0x50, 0x33, 0x12, 0xfd, 0xdc, 0xbf,
0x9e, 0x79, 0x58, 0x3b, 0x1a,
0xa6, 0x87, 0xe4, 0xc5, 0x22, 0x03, 0x60, 0x41, 0xae, 0x8f, 0xec,
0xcd, 0x2a, 0x0b, 0x68, 0x49,
0x97, 0xb6, 0xd5, 0xf4, 0x13, 0x32, 0x51, 0x70, 0x9f, 0xbe, 0xdd,
0xfc, 0x1b, 0x3a, 0x59, 0x78,
0x88, 0xa9, 0xca, 0xeb, 0x0c, 0x2d, 0x4e, 0x6f, 0x80, 0xa1, 0xc2,
0xe3, 0x04, 0x25, 0x46, 0x67,
0xb9, 0x98, 0xfb, 0xda, 0x3d, 0x1c, 0x7f, 0x5e, 0xb1, 0x90, 0xf3,
0xd2, 0x35, 0x14, 0x77, 0x56,
0xea, 0xcb, 0xa8, 0x89, 0x6e, 0x4f, 0x2c, 0x0d, 0xe2, 0xc3, 0xa0,
0x81, 0x66, 0x47, 0x24, 0x05,
0xdb, 0xfa, 0x99, 0xb8, 0x5f, 0x7e, 0x1d, 0x3c, 0xd3, 0xf2, 0x91,
0xb0, 0x57, 0x76, 0x15, 0x34,
0x4c, 0x6d, 0x0e, 0x2f, 0xc8, 0xe9, 0x8a, 0xab, 0x44, 0x65, 0x06,
0x27, 0xc0, 0xe1, 0x82, 0xa3,
0x7d, 0x5c, 0x3f, 0x1e, 0xf9, 0xd8, 0xbb, 0x9a, 0x75, 0x54, 0x37,
0x16, 0xf1, 0xd0, 0xb3, 0x92,
0x2e, 0x0f, 0x6c, 0x4d, 0xaa, 0x8b, 0xe8, 0xc9, 0x26, 0x07, 0x64,
0x45, 0xa2, 0x83, 0xe0, 0xc1,
0x1f, 0x3e, 0x5d, 0x7c, 0x9b, 0xba, 0xd9, 0xf8, 0x17, 0x36, 0x55,
0x74, 0x93, 0xb2, 0xd1, 0xf0
} ;

unsigned int CRC16(unsigned char * puchMsg,
                  unsigned int usDataLen)

```

```

{
    uchCRCHi = 0; uchCRCLo = 0;
    while (usDataLen) /* pass through message buffer */
    {
        uIndex = uchCRCHi ^ *puchMsg++ ; /* calculate the CRC */
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
        uchCRCLo = auchCRCLo[uIndex] ;
        usDataLen--;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}

```