Установки порта: 8 бит, 1 стоп, без четности. 125000 бод

Сообщения разделяются по паузе в линии как в Modbus RTU. Сообщение должно начинаться и заканчиваться интервалом тишины, длительностью не менее 3,5 символов при данной скорости передачи. Во время передачи сообщения не должно быть пауз длительностью более 1,5 символа. Для скоростей 500000 и более бод интервал тишины 128 мкс. Проверка целостности осуществляется с помощью [CRC](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) как в Modbus RTU

Длина запроса 255 байт максимум.

Первый байт сообщения: старшие 4 бита адрес устройства в сети младшие 4 бита команда.

Адреса устройств:

1–плотностной

2-глубиномер

3-инклинометр

4-ГК

5-нейтронный

6-БК, БКС и т.д.

7-профилемер

8-индукционные

9-АГК

15-широковещательный запрос

0-зарезервировано для расширенных адресов устройств

Состояния прибора и алгоритм работы:

enum

{

APP\_SET\_TIME,

APP\_CLEAR\_RAM,

APP\_DELAY,

APP\_WORK,

APP\_IDLE,

};

APP\_SET\_TIME-0 обнуление таймера тактирования кадров ( синхронизация с другими модулями), установка времени задержки (число кадров, int32, отрицательное число), переход на пониженное энергопотребление, переход к состоянию APP\_CLEAR\_RAM

APP\_CLEAR\_RAM-1 идет очистка памяти, по окончании переход к APP\_DELAY

APP\_DELAY-2 прибор на задержке, при начале очередного такта со временем задержки = 0, установка APP\_WORK, установка указателя записи во внешнюю память в начало, переход в рабочий режим, полное энергопотребление, начало работы (накопление данных за такт).

APP\_WORK-3 Начало с 1 (первого) такта. Обработка накопленных за такт данных, запись их в память. Накопление данных для 2 (второго) такта.

И т.д. вплоть до прихода команды выключения прибора.

APP\_IDLE-4 прибор выключен

Команды поддерживаемые устройством.

//адрес устройства в сети

#define ADDRESS (ADDRESS\_PROCESSOR << 4)

//\*\*\*\*\*\*\*\*\* команды поддерживаемые устройством \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// установка времени на поверхности если 0 - то выключение прибора

#define CMD\_TIME\_SYNC 0xF5

// синхронизация времени во время работы в скважине

#define CMD\_BEACON 0xFA

// режим повышенной скорости

#define CMD\_TURBO 0xFD

// чтение внешней памяти

#define CMD\_ERAM 0x1 | ADDRESS

// чтение информации об устройстве (дескриптор устройства)

#define CMD\_INFO 0x2 | ADDRESS

// обработка ошибок

#define CMD\_ERR 0xE | ADDRESS // было 6 неверно конфликт с CMD\_WRITE\_EE

// чтение EEPROM памяти

#define CMD\_READ\_EE 0x5 | ADDRESS

// запись EEPROM памяти

#define CMD\_WRITE\_EE 0x6 | ADDRESS

// текущие состояние и данные (Тестовый режим или выдача последних сохраненных данных в рабочем режиме)

#define CMD\_WORK 0x7 | ADDRESS

// bootloader

#define CMD\_BOOT 0x8 | ADDRESS

Команда 0xF5 CMD\_TIME\_SYNC.

Установка задержки, выключение прибора

Широковещательная команда без ответа.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| запрос | 0xF5 | int32L | int32L1 | int32H1 | int32H | CRCL | CRCH |

Данные int32

Если 0, то перевод прибора из любого состояния в режим ожидания команд APP\_IDLE пониженное энергопотребление, иначе: данные отрицательное число, количество тактов до включения прибора, выполняется:

-установка времени задержки

-обнуление таймера тактирования кадров

-если прибор в состоянии больше > APP\_DELAY переход в состояние APP\_SET\_TIME, пониженное энергопотребление.

Команда 0xFА CMD\_BEACON.

Синхронизация времени во время работы в скважине.

Широковещательная команда без ответа.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| запрос | 0xFА | int32L | int32L1 | int32H1 | int32H | CRCL | CRCH |

Команда 0xFD CMD\_TURBO.

Режим повышенной скорости включается при считывании памяти

Широковещательная команда без ответа.

Если нет сообщений 4-5 тактов, то переход на скорость 125000.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| запрос | 0xFD | Speed | CRCL | CRCH |

Speed –скорость передачи 1-0.5M, 2-1M, 3-2.25M, 4-4.5M.

Команда 0x01 CMD\_ERAM

Чтение внешней памяти

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| запрос | 0x(Адр)1 | Адрес начала uint32 (4 байта) | Длина N uint32 (4байта) | CRCL | CRCH |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ответ | 0x(Адр)1 | Данные N байт | CRCL | CRCH |

Команда 0x02 CMD\_INFO

Чтение информации об устройстве.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| запрос | 0x(Адр)2 | Длина N | CRCL | CRCH |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| запрос | 0x(Адр)2 | Длина N | Адрес начала uint16 (2 байта) | CRCL | CRCH |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ответ | 0x(Адр)2 | Данные N байт | CRCL | CRCH |

Команда 0x05 CMD\_READ\_EE

Чтение EEPROM устройства.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| запрос | 0x(Адр)5 | Адрес начала uint16 (2 байта) | Длина N | CRCL | CRCH |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ответ | 0x(Адр)5 | Данные N байт | CRCL | CRCH |

Команда 0x06 CMD\_WRITE\_EE

Запись EEPROM устройства.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| запрос | 0x(Адр)6 | Адрес начала uint16 (2 байта) | Данные N байт | CRCL | CRCH |

|  |  |  |  |
| --- | --- | --- | --- |
| ответ | 0x(Адр)6 | CRCL | CRCH |

Если необходимо провести инициализацию периферии новыми значениями.

Команда 0x07 CMD\_WORK

Текущие состояние и данные

Тестовый режим или выдача последних сохраненных данных в рабочем режиме

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| запрос | 0x(Адр)7 | Длина N | CRCL | CRCH |
| запрос | 0x(Адр)7 | Длина N uint16 (2байта) | CRCL | CRCH |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ответ | 0x(Адр)7 | Данные N байт | CRCL | CRCH |

Команда 0x0Е CMD\_ERR

Обработка ошибок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| запрос | 0x(Адр)Е | NeedClearErr | CRCL | CRCH |

Если NeedClearErr = 0xA5, то сбрасывается флаг ошибки (очищается буфер ошибок?)

см. ниже переменная *uint8\_t* AppState; /// автомат|AU

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ответ | 0x(Адр)Е | номер ошибки | Строка описания | CRCL | CRCH |

После каждого запроса, выдается очередная ошибка из буфера ошибок

Генерация информации об устройстве.

Описание данных, которые выдает прибор в тестовом режиме, записываемые в память в режиме работы, хранящиеся в EEPROM, находятся в отдельном хедер файле пример (Data.h):

#pragma once

namespace adxl354gk

{

typedef struct \_\_attribute\_\_((packed))

{

*uint16\_t* gk; ///гк

} Gk\_t ;

typedef struct \_\_attribute\_\_((packed))

{

Gk\_t GR1; /// ГК|GK1

} EepData\_t;

typedef struct \_\_attribute\_\_((packed))

{

*int16\_t* X;

*int16\_t* Y;

*int16\_t* Z;

} Dat\_t;

typedef struct \_\_attribute\_\_((packed))

{

Dat\_t accel;

Dat\_t magnit;

*int16\_t* T;

float Zenit; /// зенит

float Azimut; /// азимут

float Gtf; /// отклонитель

float Mtf; /// маг\_отклон

*int16\_t* Gtot; /// амплит\_accel

*int16\_t* Mtot; /// амплит\_magnit

} InclW\_t;

typedef struct \_\_attribute\_\_((packed))

{

Dat\_t accel;

Dat\_t magnit;

*int16\_t* T;

} InclR\_t\_old;

typedef struct \_\_attribute\_\_((packed))

{

*uint8\_t* AppState; /// автомат|AU

*int32\_t* time; /// время|WT

InclW\_t dat; /// Inclin|INKLGK

Gk\_t gk; /// ГК|GK1

} WorkData\_t;

typedef struct \_\_attribute\_\_((packed))

{

# define RAM\_SIZE 32 /// varRamSize

*int32\_t* ramtime; /// время|WT

InclW\_t dat; /// Inclin|INKLGK

Gk\_t gk; /// ГК|GK1

} RamData\_t;

typedef struct \_\_attribute\_\_((packed))

{

# define ADDRESS\_PROC 3 /// var\_adr

# define DEV\_INFO "\_\_DATE\_\_ ADXL354 GK" /// var\_info

# define CHIP\_NUMBER 4 /// varChip

# define SERIAL\_NUMBER 1 /// varSerial

# define UART\_SPEED\_MASK 192 /// varSupportUartSpeed

WorkData\_t Wrk; /// WRK

RamData\_t Ram; /// RAM

EepData\_t Eep; /// EEP

} AllDataStruct\_t; /// InclGK1

}

Данный файл перед компиляцией преобразуется с помощью специальной утилиты h2meta

(пример: c:\AVR\h2meta.exe Data.h MetaData.h) в следующий хедер файл (MetaData.h):

#pragma once

const unsigned char \_\_attribute\_\_ ((section(".meta\_data"), used)) cmetaAll[] = {

36,138,1,73,110,99,108,51,0,40,3,39,50,53,46,48,57,46,50,48,49,57,32,65,68,88,76,

51,53,52,32,71,75,0,56,4,57,1,0,62,192,0,36,170,0,87,82,75,0,17,224,226,242,238,236,

224,242,124,65,85,0,3,226,240,229,236,255,124,87,84,0,36,127,0,73,110,99,108,105,

110,124,65,68,88,76,51,53,52,0,36,18,0,97,99,99,101,108,0,2,88,0,2,89,0,2,90,0,36,

19,0,109,97,103,110,105,116,0,2,88,0,2,89,0,2,90,0,2,84,0,4,231,229,237,232,242,0,

4,224,231,232,236,243,242,0,4,238,242,234,235,238,237,232,242,229,235,252,0,4,236,

224,227,95,238,242,234,235,238,237,0,2,224,236,239,235,232,242,95,97,99,99,101,108,

0,2,224,236,239,235,232,242,95,109,97,103,110,105,116,0,36,14,0,195,202,124,71,75,

49,0,18,227,234,0,36,161,0,82,65,77,0,43,10,0,3,226,240,229,236,255,124,87,84,0,36,

127,0,73,110,99,108,105,110,124,65,68,88,76,51,53,52,0,36,18,0,97,99,99,101,108,0,

2,88,0,2,89,0,2,90,0,36,19,0,109,97,103,110,105,116,0,2,88,0,2,89,0,2,90,0,2,84,0,

4,231,229,237,232,242,0,4,224,231,232,236,243,242,0,4,238,242,234,235,238,237,232,

242,229,235,252,0,4,236,224,227,95,238,242,234,235,238,237,0,2,224,236,239,235,232,

242,95,97,99,99,101,108,0,2,224,236,239,235,232,242,95,109,97,103,110,105,116,0,36,

14,0,195,202,124,71,75,49,0,18,227,234,0,36,21,0,69,69,80,0,36,14,0,195,202,124,71,

75,49,0,18,227,234,0};

Создается бинарный массив описывающий структуры данных хедер файла Data.h используемый командой 0x02 CMD\_INFO.

Утилита h2meta.

При использовании различных компиляторов, процессоров, особых настроек проекта для генерации файла MetaData.h используем шаблон, например: c:\AVR\h2meta.exe Data.h MetaData.h Patern.h, где Patern.h:

#pragma once

extern *uint8\_t* ReadMetaData(*uint8\_t*\* p, *uint8\_t* n, *uint16\_t* from);

const unsigned char \_\_attribute\_\_ ((section(".meta\_data"), used)) cmetaAll[] = {

%s};

Вместо %s утилитой подставляется сгенерированный массив метаданных и создается файл MetaData.h.

Утилита понимает кодировку ANSI, пытается распознать UTF-8, и преобразовать в ANSI, может распознавать следующие типы данных, массивы этих типов:

int8\_t

uint8\_t

int16\_t

uint16\_t

int32\_t

uint32\_t

int64\_t

uint64\_t

float

double

пользовательские типы данных:

var\_info - uint8[] – текстовая информация о приборе

var\_adr - uint8 – адрес прибора

varChip - uint8 – тип процессора для бутлоадера

varSerial - uint16 – серийный номер прибора

varRamSize – uint16 память в мегабайтах

varSSDSize – uint32 память в секторах 512 байт для SD карты

varSupportUartSpeed uint16 битовая маска поддерживаемых устройством протоколов (скорости uart) обмена данными:

**type** ESpeed = (S125K = $80, S500K = $40, S1M = $20, S2\_25M = $10, S4\_5M = $08, SD = $4000, USB = $8000);

Структуры должны быть упакованными.

Объявление структур распознается только в таком виде:

typedef struct

{

………

} RamDat……

Распознаются только десятичные числа для пользовательских типов данных.

Для var\_info строка \_\_DATE\_\_ заменяется на текущую дату.

После трех /// распознается:

Для переменных пример:

*int32\_t* time; /// время|WT

time - имя для программы прибора

время – фактическое имя в метаданных, желательно латиницей.

WT - атрибут для форматирования вывода на ПС, только латиницей

Для структур пример:

InclW\_t dat; /// Inclin|ADXL354

dat - имя структуры для программы прибора

Inclin – фактическое имя в метаданных, тип устройства для метрологии, желательно латиницей.

ADXL354 - атрибут для форматирования вывода на ПС, расчёта окончательных данных в тестовом режиме или после считывания данных из памяти, только латиницей.

Для пользовательских данных пример:

# define ADDRESS\_PROC 3 /// var\_adr

ADDRESS\_PROC – имя для программы прибора

3- значение в метаданных и для программы прибора

var\_adr - фактическое имя пользовательского типа данных

утилита ищет корневую структуру с именем AllDataStruct\_t и по ней генерирует метаданные пример:

typedef struct

{

# define ADDRESS\_PROC 3 /// var\_adr

# define DEV\_INFO "\_\_DATE\_\_ ADXL354 GK" /// var\_info

# define CHIP\_NUMBER 4 /// varChip

# define SERIAL\_NUMBER 1 /// varSerial

# define UART\_SPEED\_MASK 192 /// varSupportUartSpeed

WorkData\_t Wrk; /// WRK

RamData\_t Ram; /// RAM

EepData\_t Eep; /// EEP

} AllDataStruct\_t \_\_attribute\_\_((packed)); /// InclGK1

InclGK1 – имя модели прибора

Где WRK – данные прибора выдаваемые в режиме выдачи тестовой информации

RAM- данные записываемые каждый кадр в память в рабочем режиме.

EEP- данные в EEPROM прибора (если есть)

Соглашения для структуры WRK (данные прибора выдаваемые в режиме выдачи тестовой информации).

typedef struct

{

*uint8\_t* AppState; /// автомат|AU – обязательный параметр

*int32\_t* time; /// время|WT – обязательный параметр

InclW\_t dat; /// Inclin|ADXL354 – данные модуля прибора

Gk\_t gk; /// ГК|GK1 – данные модуля прибора

} Work

Переменная AppState /// автомат|AU

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 бит | 7 бит | 6 бит | 5 бит | 4 бит | 3 бит | 2 бит | 1 бит |
| Питание | ошибка |  | | | состояние конечного автомата | | |

состояние конечного автомата

enum

{

APP\_SET\_TIME,

APP\_CLEAR\_RAM,

APP\_DELAY,

APP\_WORK,

APP\_IDLE,

};

APP\_SET\_TIME-0 установка времени задержки

APP\_CLEAR\_RAM-1 идет очистка памяти

APP\_DELAY-2 прибор на задержке

APP\_WORK-3 прибор в работе

APP\_IDLE-4 прибор выключен

8 бит питание – если сброшен, то прибор в режиме пониженного энергопотребления.

Если команда 0x07 запрашивает только первые два параметра: состояние конечного автомата и время, то прибор переходит или остается в режиме пониженного энергопотребления (не относится к режиму работы APP\_WORK!).

Если команда 0x07 запрашивает все данные структуры WRK, то прибор переходит в режим информации, бит питание устанавливается, если запросы не приходят 4-5 тактов, то прибор переходит в спящий режим (пониженного энергопотребления) (не относится к режиму работы APP\_WORK!).

7 бит ошибка – если во время работы были неисправности оборудования, запредельные режимы, то устанавливается данный флаг. Флаг сбрасывается командой 0х0Е, которая читает сообщения об ошибках.

Соглашения для структуры RAM (данные записываемые каждый кадр в память в рабочем режиме).

typedef struct

{

# define RAM\_SIZE 32 /// varRamSize или varSSDSize – обязательный параметр объем памяти

*int32\_t* ramtime; /// время|WT – обязательный параметр номер кадра

InclW\_t dat; /// Inclin|ADXL354 – данные модуля прибора

Gk\_t gk; /// ГК|GK1 – данные модуля прибора

} Ram…