Драйвер обмена с контроллером вывода телесигнализации на диспетчерский щит КЩ-32

Руководство пользователя

Содержание

1.Назначение драйвера

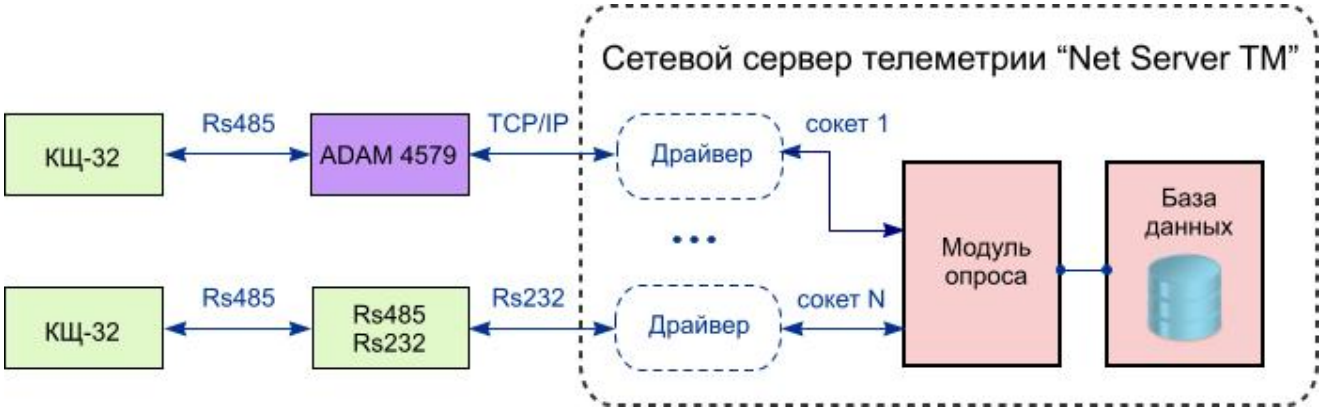


Рис. 1. Структурная схема взаимодействия драйвера с «модулем опроса».

Драйвер служит интерфейсным модулем между контроллером «КЩ-32» (далее по тексту устройством) и «модулем опроса», который входит в состав сервера телеметрии «Net server TM». (См. Рис.1.) «Модуль опроса» общается с драйвером с помощью пакетов-посылок (заказов). Более подробно о структуре пакетов изложено в п.5. Между драйвером и устройством обмен информацией осуществляется с использованием протокола, описанного в п.6. При запуске «модуля опроса» автоматически запускается драйвер с помощью командной строки, сформированной на основании заданных в конфигурации объектов. В командной строке указываются параметры инициализации, среди них: системный IP-адрес и порт для соединения с устройством по протоколу TCP/IP, либо порт и параметры последовательного соединения с устройством, параметры для инициализации диагностики и др. При некорректной инициализации (отсутствие необходимого параметра инициализации, либо присвоение ему некорректного значения) драйвер завершает работу.

После запуска драйвер пытается установить соединение с устройством. В случае неудачного соединения с устройством драйвер повторяет попытки соединения с периодом 20 сек. до нормального завершения. В случае удачного соединения с устройством, драйвер регулярно его опрашивает.

Драйвер, получив от «модуля опроса» заказ, отвечает квитанцией или ответом с состоянием связи с устройством. Если заказ влечёт команду устройству, драйвер формирует и отправляет устройству посылку.

В каждый момент времени драйвер обрабатывает не более одного информационного запроса. При некорректном заказе драйвер выдает «модулю опроса» признак ошибки обработки заказа. Информационные запросы и команды управления задвижкой выдаются «модулем опроса» по разным сокетам. Драйвер завершает свою работу:

- при закрытии «модуля опроса»;
- при отсутствии заказов от «модуля опроса» в течение времени, заданного в конфигурации.

Для запуска драйвера требуется установленный интерпретатор языка Tcl (Tcl/Tk версии 8.4 и выше), например ActiveTcl 8.4.4.0.

Требования к среде для установки ActiveTcl 8.4.4.0:

ОС:	Аппаратное оборудование:	Версия ОС:
Irix	SGI (Mips)	6.3+
HP-UX	HP (PARISC)	10.20+
Linux	Intel	Red Hat 7.0+ or SuSE 6.0+ *
Solaris	Sun (Sparc)	2.5+
Windows	Intel	NT 3.51/NT 4.0/2000/XP

Свободное место на диске: 2000 Мб

* рекомендуемые требования

2. Краткое описание контроллера «КЩ-32»

Характеристики интерфейса RS485
Скорость обмена 2400...57600 бит/с
Длина линии связи до 1000 м

3. Конфигурация драйвера

3.1. Командная строка вызова.

./**k32ort SERIAL**=dev,speed,parity,data_b,stop_b **PORT**=Nport
DEVICES=N1,...,Nn [**TKILL**=suic_tout] [**LOG**=log_file]
[**DEBUG**=dbg_val] [**BASE**=base_file] [**CONF**=config_file]
[**STMCONF**=stm_file]
или
./**k32port IP**=ip_addr_or_name:ip_port **PORT**=Nport
DEVICES=N1,...,Nn [**TKILL**=suic_tout] [**LOG**=log_file]
[**DEBUG**=dbg_val] [**BASE**=base_file] [**CONF**=config_file]
[**STMCONF**=stm_file]

где k32port – имя запускаемого модуля (имя драйвера);
dev – устройство, обслуживающее COM-порт;
speed – скорость в бодах;
parity – чётность (всегда n, сохранена для «общности» формы);
data_b – количество бит в байте (всегда 8, сохранено для «общности» формы);
stop_b – количество стоповых бит (1 или 2);
ip_addr_or_name:ip_port – ip-адрес и порт (сокет), через который драйвер ведёт диалог с устройствами (драйвер является клиентом);
Nport – сокетный порт верхнего уровня, по которому поступают информационные запросы и отправляются ответы;
base_file – файл текущего состояния щита;
stm_file – файл конфигурации сервера телеметрии;
N1...Nn= имена устройств, с первой встреченной в имени десятичной цифры начинается адрес устройства;
suic_tout – таймаут в секундах «самоубийства» программы при отсутствии сокетных запросов (умолчание: 0 – никогда);
log_file – файл журнала (умолчание: стандартный вывод – экран);
config_file – файл конфигурации (умолчание: tekport.conf);
dbg_val – битовое поле (hex) разрешения вывода в журнал отладочной

информации:
FD_OK 1 (OK message)
FD_CPACK 2 (device dialog)
FD_MESS 4 (result message)
FD_INPACK 8 (socket dialog - in packets)
FD_OUTPACK 10 (socket dialog - out packets)
FD_TIME 20 (time output)

Примеры:
k32port SERIAL=/dev/ttyS1,19200,n,8,2 PORT=7720 BASE=k32_b/ad_tc DEVICES=7 TKILL=3600
k32port IP=10.0.1.27:5201 PORT=7720 BASE=k32_b/ad_tc DEVICES=1,tc16,22 LOG=dep.log DEBUG=18

Строка запуска также поясняется при вызове драйвера без аргументов:
./k32port

3.2. Список параметров

Запрос	**Ответ**	**Назначение**
«ts=*ts_name* par= *ts_val*»	квитанция	Значение ТС для вывода на щит
«dev=*Ndev* act=state»	«out=»	Состояние связи с устройством
«dev=*Ndev* act=teston»	квитанция	Включение теста устройства
«dev=*Ndev* act=testoff»	квитанция	Выключение теста устройства
«dev=*Ndev* act=kvit»	квитанция	Квитирование устройства

3.3. Файл конфигурации

Файл k32port.conf (имя может быть специфицировано в строке запуска драйвера) регулярно перепрочитывается (с интервалом 10 секунд).
Строка относящаяся к устройству dev имеет вид:

dev [**polltout**=12] [**livetout**=30] [**debug**=ffff] [**log**=]

Указанные значения – программные умолчания; реально таймауты берутся из файла конфигурации сервера. Примеры:
1 debug=18
s50 livetout=600 polltout=3

dev - dev, указываемый при обращении к драйверу через сокет;
polltout - таймаут в секундах опроса устройства;
livetout - таймаут определения отсутствия связи с устройством;
debug - режим вывода отладочных сообщений (см. строку запуска);
log - файл журнала.
Область действия **debug** & **log** - весь драйвер, а не конкретное устройство (но в строке устройства обслуживаемого драйвером).

4. Конфигурация сервера.

Создание конфигурации осуществляется с помощью программы **STMConf** (C:\Igel\StmConf\StmConf.exe). В данном разделе будет описаны настройки, необходимые для взаимодействия сервера телеметрии и драйвера. Подробное описание работы с конфигуратором **StmConf** и создание конфигураций содержит документ «Конфигуратор сервера телеметрии StmConf. Руководство пользователя». Также дополнительную информацию можно найти web-справочнике, нажав F1 в IgelView3, раздел Средства конфигурации > Конфигуратор сервера телеметрии StmConf.

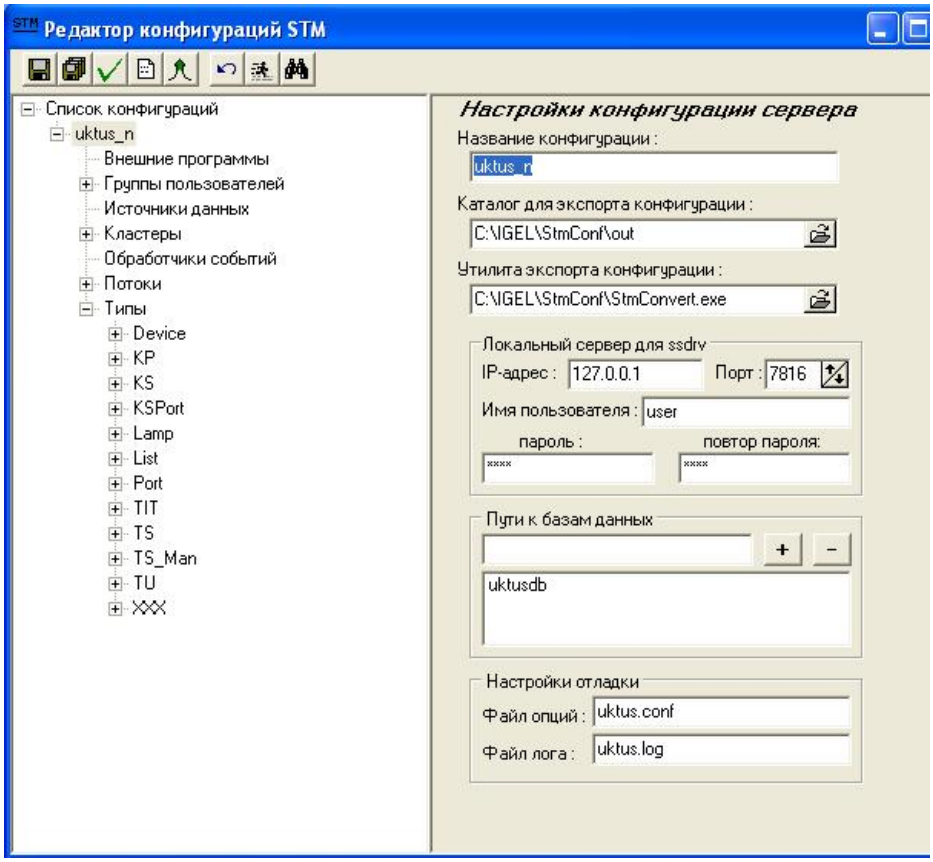
4.1. Типы объектов и устройств. В конфигурации должны присутствовать следующие типы

KSPort – Приборный порт, описывает режимы работы с конвертерами ADAM 4579 или MOXA;

KS – Описание прибора;

Lamp – Управление состоянием ламп щита.

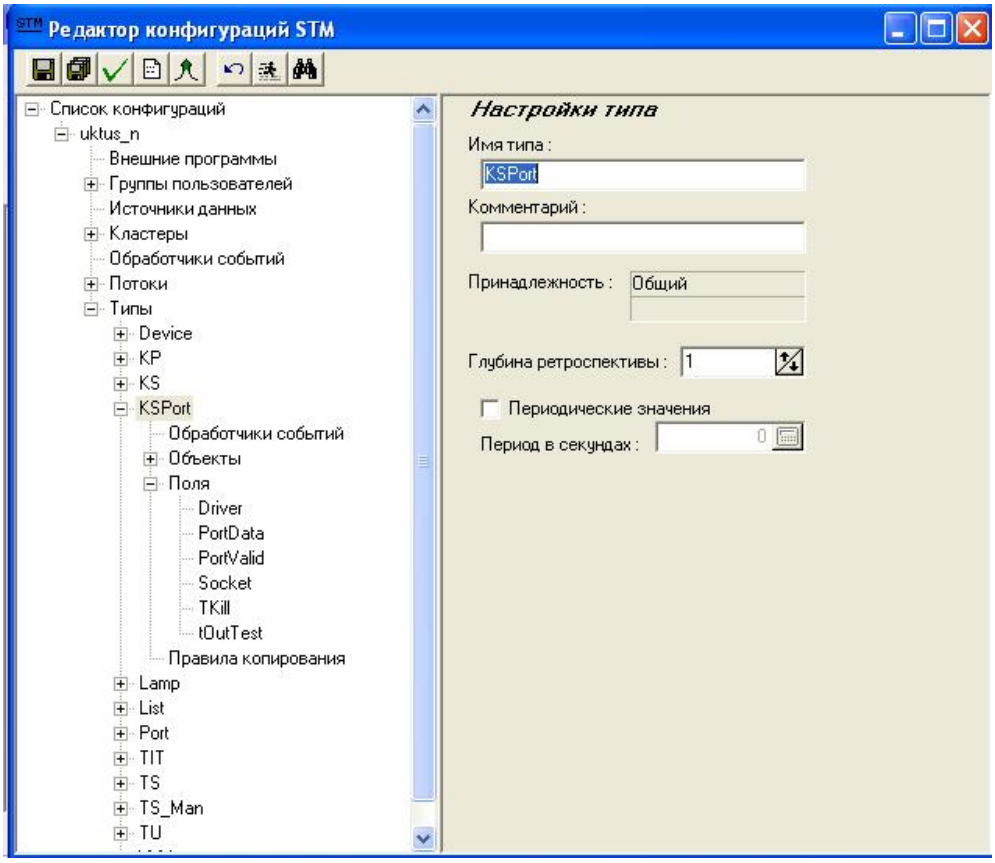
На основании введенной информации в объектах типа **KSPort** и **KS** «модуль опроса» сформирует командные строки и запустит копии драйвера.



4.2. Структура типов. Типы должны иметь следующую структуру полей:

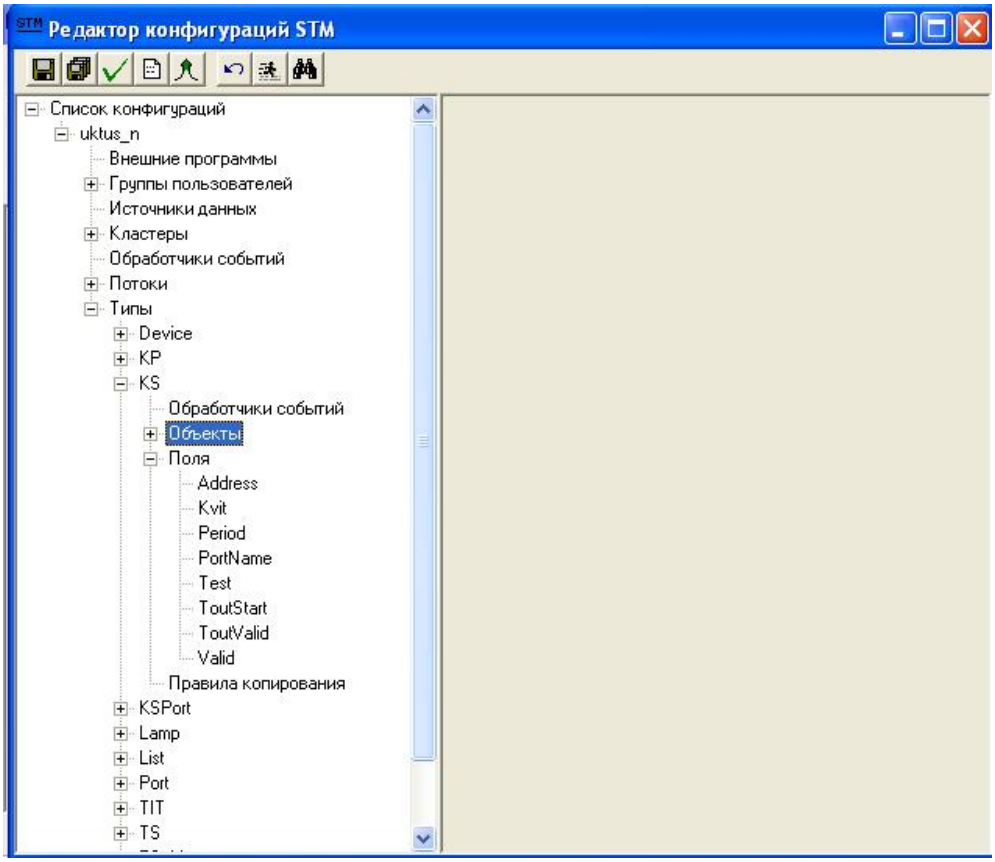
KSPort

Тип поля	Наим. Поля	Комментарий	Пример значения
String	Driver	Командная строка запуска драйверного модуля	/stm/k32port LOG=130Ps25st1.log DEBUG=FF
String	PortData	Характеристики порта	10.12.20.16:4001
uInt	socket	Номер серверного сокета	7777
uLong	Tkill	таймаут “самоубийства” драйвера при отсутствии запросов (в сек.)	3600
Long	tOutTest	таймаут тестовых запросов в (мсек.)	80000



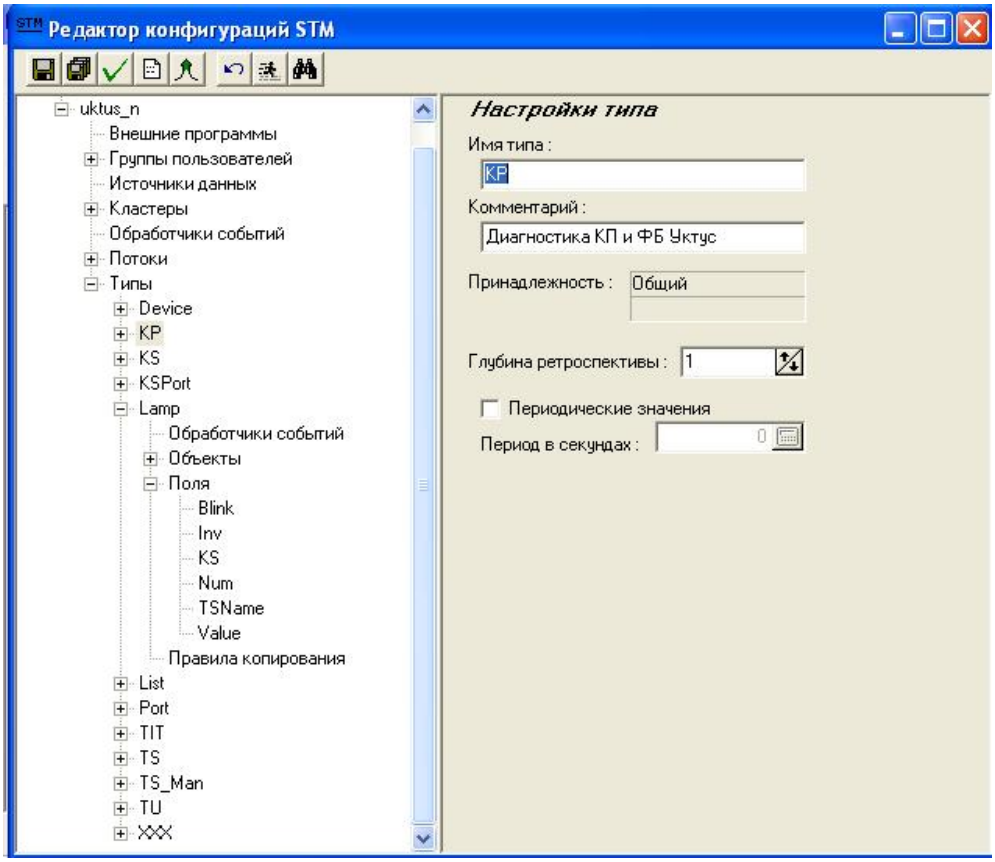
KS

Тип поля	Наим. Поля	Комментарий	Пример значения
Int	Address	Адрес устройства	15
Byte	Kvit	Признак квитирования контроллера	0
uLong	Period	Период опроса контроллера	5
String	PortName	Имя объекта типа KSport, которому соответствует прибор	port1
Byte	Test	Включение теста ламп	0



Lamp

Тип поля	Наим. поля	Коммент.	Пример значения
Byte	Blink	Признак мигания	0
Byte	Inv	Инверсия	1
String	KS	Имя контроллера	KS1
Byte	Num	Номер лампы	9
String	TSName	Имя телесигнала	Земля-4
Byte	Value	Значение телесигнала	1


Редактор конфигураций STM
uktus_n
Внешние программы
Группы пользователей
Источники данных
Кластеры
Обработчики событий
Потоки
Типы
Device
KP
KS
KSPort
Lamp
Обработчики событий
Объекты
Поля
Blink
Inv
KS
Num
TSName
Value
Правила копирования
List
Port
TIT
TS
TS_Man
TU
XXX
Настройки типа
Имя типа :
KP
Комментарий :
Диагностика КП и ФБ Уктус
Принадлежность :
Общий
Глубина ретроспективы :
1
Периодические значения
Период в секундах :
0

5. Структура пакетов обмена между «Модулем опроса» и драйвером.

Структура пакетов обмена между «модулем опроса» и драйвером подробно описана в документе “«Модуль опроса». Руководство пользователя“. Типы запросов, ответов и значения параметров запросов приводятся в таблице 1, 2 и 3.

Таблица 1. Типы запросов.

Пакет	Описание
{num=N}\n	Пакет контроля работоспособности соединения
{num=N ts=x par=y}\n	Сообщение со значением y телесигнала с именем x
{num=N dev=x act=state}\n	Запрос состояния связи с устройством x
{num=N dev=x act=teston}\n	Включение теста устройства x
{num=N dev=x act=testoff}\n	Выключение теста устройства x
{num=N dev=x act=kvit}\n	Квитирование устройства x

Таблица 2. Типы ответов.

Пакет	Описание
{num=N}\n	Пакет контроля работоспособности соединения
{num=N dev=x out=y}\n	Ответ на запрос о состояния связи с устройством x (y=0/1)
{num=N dev=x}\n	Квитанция на тест и квитирование для устройства x
{num=N ts=x}\n	Квитанция на сообщение со значением для телесигнала с именем x

Таблица 3. Возможные значения параметров запроса.

Пара-метр	Описание	Значения
Num	Номер запроса по порядку. Целое число.	Меняется циклически до 1000000
Par	Значение телесигнала	0-1
Dev	Имя (адрес) устройства на порту	Целое число от 1 до 254
ts	Имя телесигнала	1-32

6. Протокол обмена драйвера с контроллером «КЩ-32».

Обмен данными между КЩ32 и драйвером организован на базе протокола MOD_BUS режим RTU. Предусматривается два режима передачи данных:

1 – режим передачи данных с квитанцией.

Переданные данные драйвера воспринимаются конкретным контроллером в ответ контроллер передает квитанцию, подтверждающую прием данных.

2 – режим передачи дейтаграммы, т.е. данные предназначены для всех контроллеров подключенных к общему каналу связи.

Формат данных:

Адрес	Функция	Данные	CRC
1 - байт	1 - байт	N – байт	2 - байта

Функция 0x01h:

Передается состояние выходных ключей контроллера (N=4)

Формат 1-го байта: 1 бит соответствует 1-му ключу,
. . . . . ,
8 бит соответствует 8-му ключу.
Формат 2-го байта: 1 бит соответствует 9-му ключу,
. . . . . ,
8 бит соответствует 16-му ключу,
Формат 3-го байта: 1 бит соответствует 17-му ключу,
. . . . . ,
8 бит соответствует 24-му ключу,
Формат 4-го байта: 1 бит соответствует 25-му ключу,
. . . . . ,

8 бит соответствует 32-му ключу.

Функция 0x02h:

Загружаются признаки мигания (N=4)

формат аналогичен состоянию выходных ключей

Функция 0x03h:

Квитирование индивидуальное для контроллера

Адрес	Код	CRC
1 - байт	0x03h	2 - байта

Функция 0x04h:

Команда на включение тестового режима

Адрес	Код	CRC
1 - байт	0x04h	2 - байта

Формат квитанции:

Адрес	Код	CRC
1 - байт	1 байт	2 - байта

Код равен коду 0x77 плюс 8-й бит как признак загрузки массива признаков мигания в ОЗУ и 4-й бит как признак нажатия кнопки “КВИТИРОВАНИЕ”.

4 бит = “1” кнопка “КВИТИРОВАНИЕ” нажата.
4 бит = “0” кнопка “КВИТИРОВАНИЕ” не нажата.
8 бит = “1” массив мигания загружен.
8 бит = “0” массив мигания требует загрузки.

Дейтограммы

Необходимы для проведения общего квитирования и тестирования.

Код функций тот же самый, только код адреса равен 0xFFh,
И в ответ контроллер не посылает квитанцию.

Дейтограмма 1:

Квитирование общее

Адрес	Код	CRC
0xFFh	0x03h	2 - байта

Дейтограмма 2:

Тестирование общее включить

Адрес	Код	CRC
0xFFh	0x04h	2 - байта

Дейтограмма 3:

Снять тестирование (общее для всех контроллеров на порту)

Адрес	Код	CRC
0xFFh	0x05h	2 - байта

Расчёт CRC

```
//--------------------------------------------------------------------------
/* Table of CRC values for high-order byte */
static const unsigned char CRCH[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
};

/* Table of CRC values for low-order byte */
static const unsigned char CRCL[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
```

```
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40
};

//**********************************************************  get_crc
word get_crc(byte *p, word len) {
unsigned char uchCRCH = 0xFF; /* high CRC byte initialized   */
unsigned char uchCRCL = 0xFF; /* low CRC byte  initialized   */
unsigned char uIndex;          /* will index into CRC lookup table */
while (len--)  {          /* pass through message buffer */
  uIndex = uchCRCH ^ *p++; /* calculate the CRC */
  uchCRCH = uchCRCL ^ CRCH[uIndex];
  uchCRCL = CRCL[uIndex];
  }
return uchCRCH << 8 | uchCRCL;
}
```