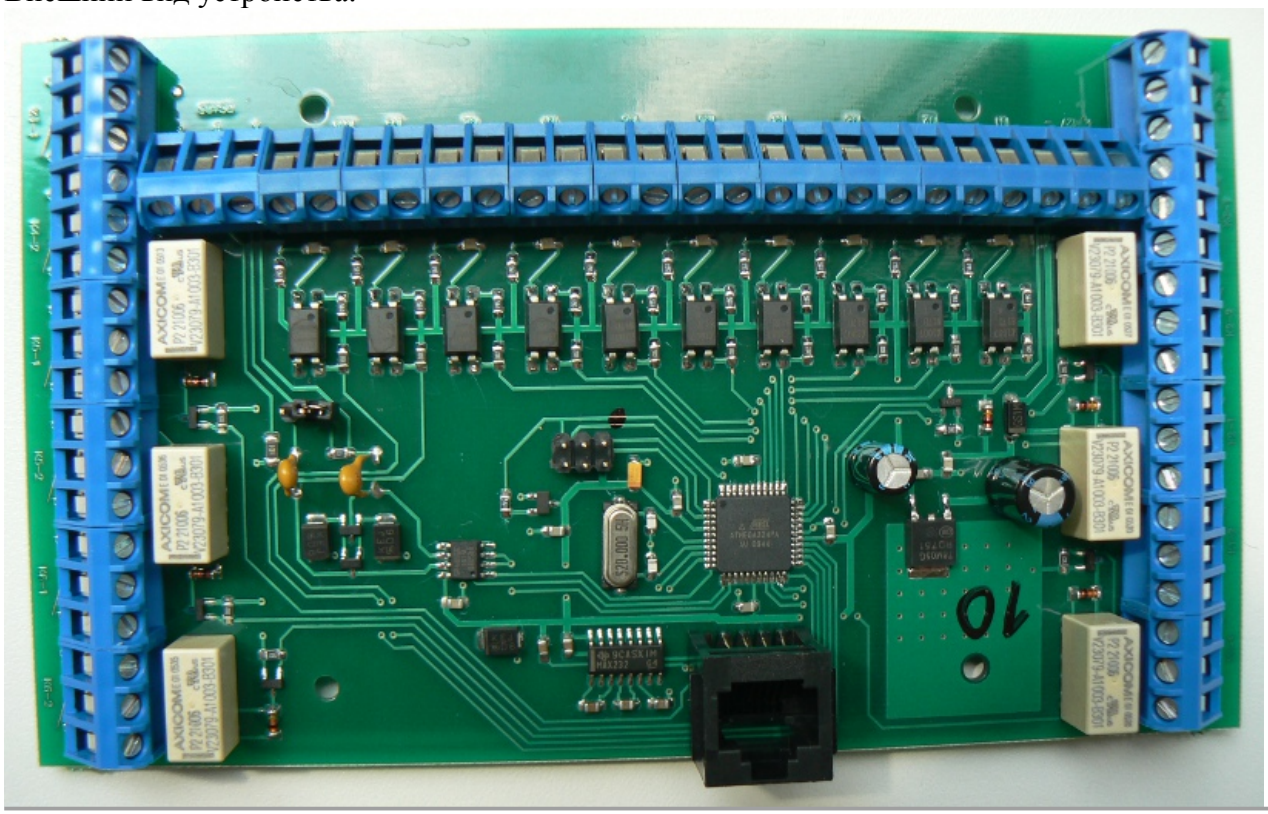
"Eyes&Hands V4: Описание программируемой платы контроля и управления (датчики и реле, RS232+RS485)"

(Mixcom. mxc@mixcom.com.ua 2012)

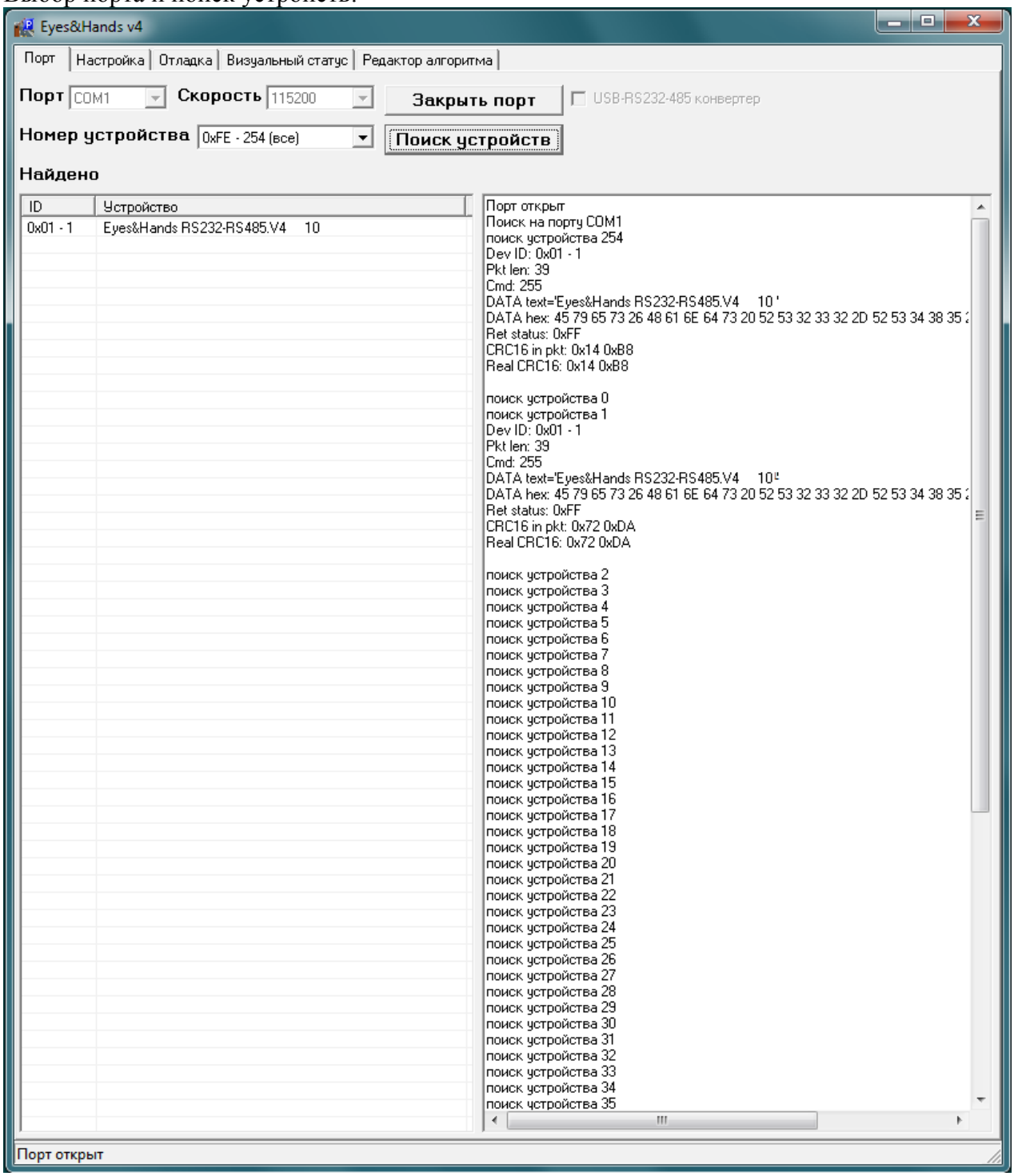
Для мониторинга и управления нужно устройство следящее за датчиками и реле. На рынке есть такие устройства, но они или очень примитивны или дорогие, но даже дорогие не способны полностью удовлетворить типичные потребности в автоматизации. Поэтому было принято решение разработать такое устройство, которое полностью покроет все наши потребности. Было разработано устройство с возможностью подключения 10 датчиков через оптопары с индикацией (такая мелочь, а как удобно), 6 реле, 2 независимых порта - RS232 и RS485, и 16 виртуальных реле-датчиков, 32шт 32 битных счетчиков или таймеров с точностью до 0.0001 сек. Удобно конфигурируемый автономный режим, и самое главное, в устройство программируется алгоритм работы: автономной, или совместной с компьютером. Для этого был разработан свой, удобный язык программирования. Написана удобная программа мониторинга и управления, программирования алгоритмов и отладки.

Внешний вид устройства:

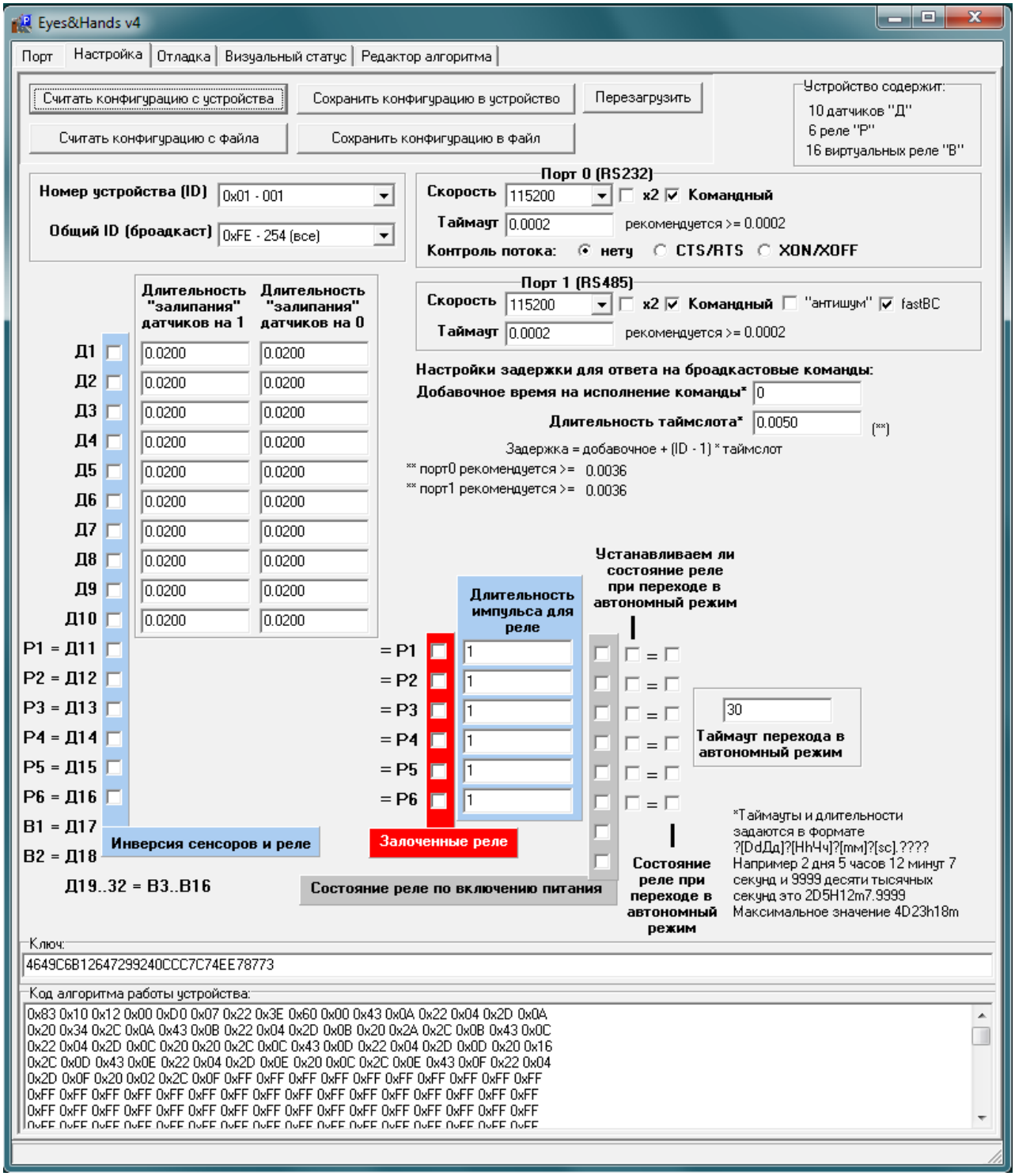


Программа конфигурации, мониторинга и отладки:

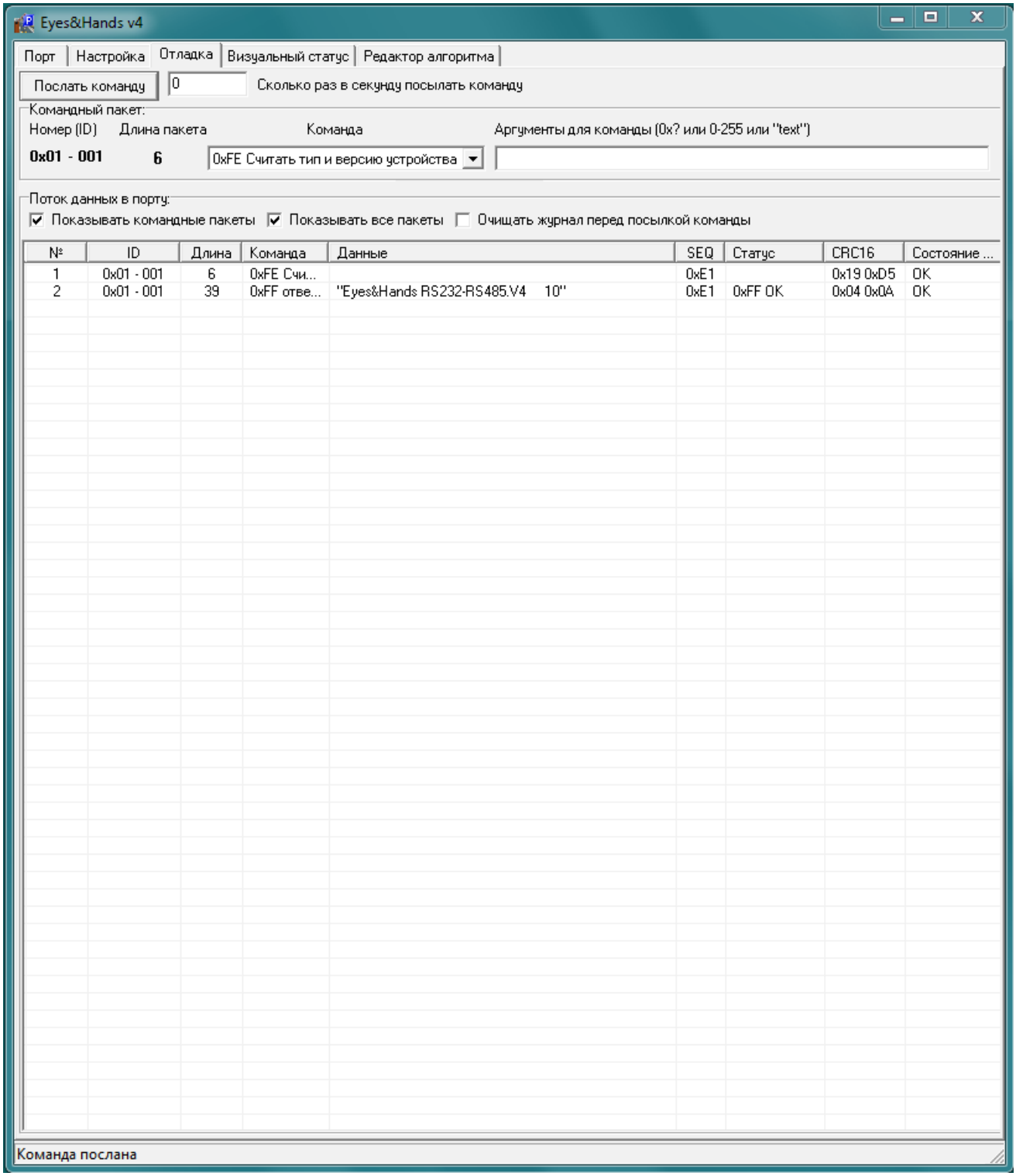
Выбор порта и поиск устройств:



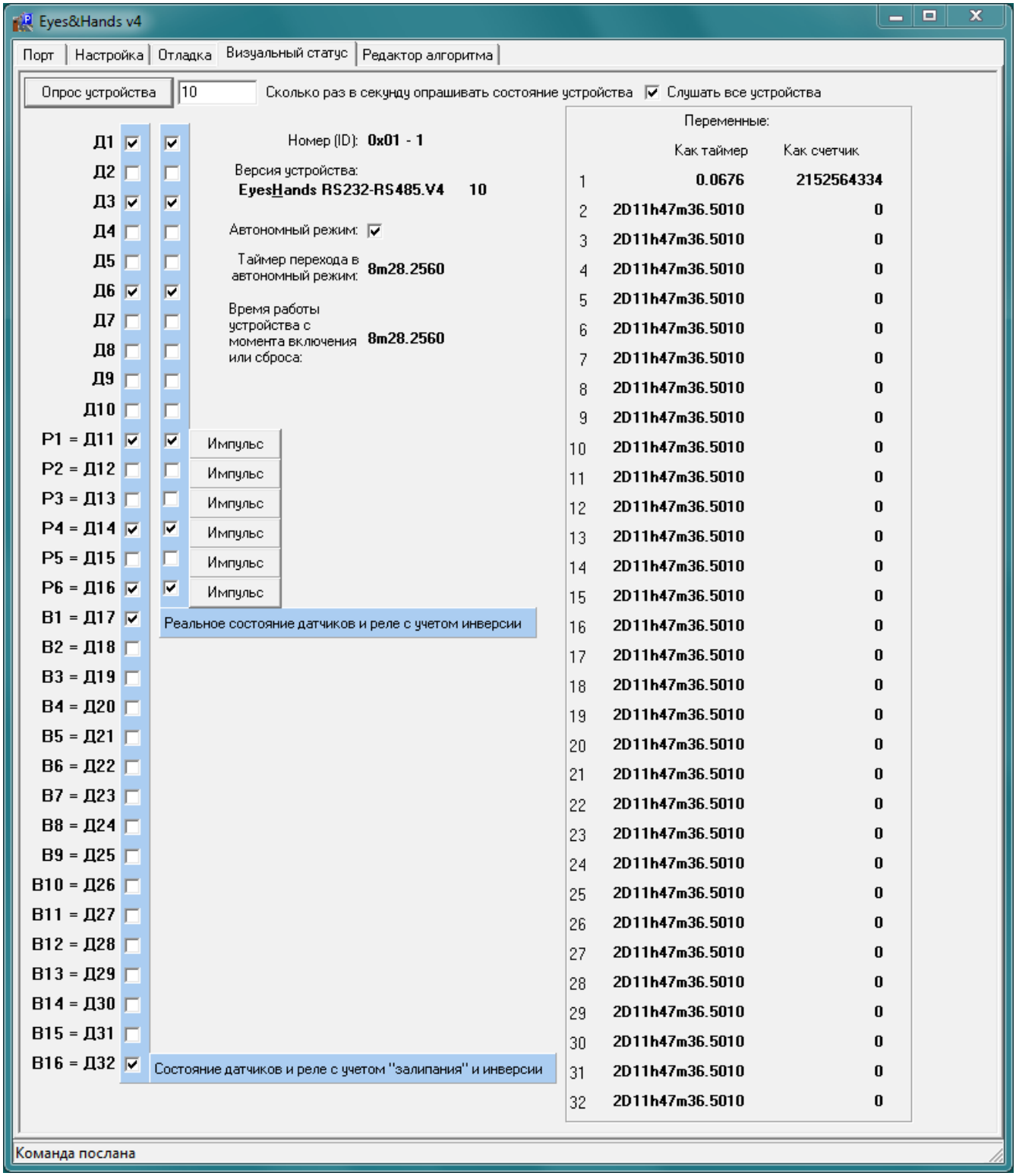
Конфигурация:



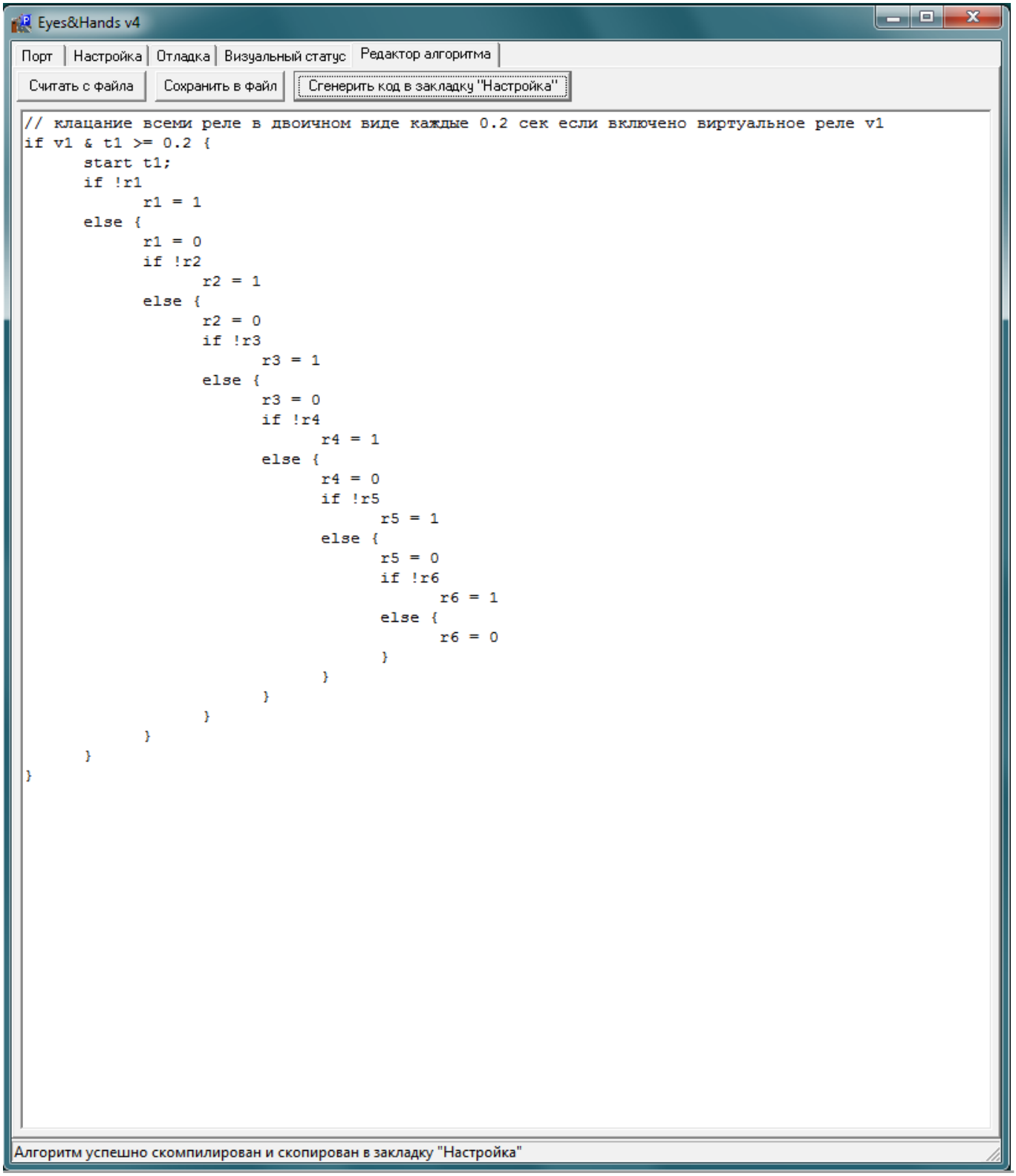
Командный интерфейс:



Визуальный мониторинг:



Программирование алгоритмов:



Команды:

Получить название устройства, тип, версию и серийный номер
CMD=0xFE
ответ например:
RET_DATA="Eyes&Hands RS232-RS485.V4 1"

Сброс устройства (перезагрузка)
CMD=0x28

Опрос состояния

CMD=0x02 DATA=
ответ
RET_DATA=
4 байта - битовое логическое состояние датчиков, реле и логических реле
2 байта - битовое реальное состояние датчиков и реле
4 байта - last1 - бит=1 если с последнего опроса состояния датчик был = 1
4 байта - last0 - бит=0 если с последнего опроса состояния датчик был = 0
4 байта - last01 - бит=1 если с последнего опроса состояния датчик переходил из состояния 0 в 1
4 байта - last10 - бит=0 если с последнего опроса состояния датчик переходил из состояния 1 в 0
4 байта - timer last - когда в последний раз был опрос состояния датчиков
4 байта - timer - текущее значение таймера
2 байта - порт 0 - состояние приемного буфера: длина данных в буфере + 0x8000 - если остановлена удаленная передача (flow control к нам)
2 байта - порт 0 - состояние передающего буфера: длина данных в буфере + 0x8000 - если остановлена наша передача (flow control от нас)
4 байта - порт 0 - количество байт пропущенных при приеме из-за переполнения буфера
2 байта - порт 1 - состояние приемного буфера: длина данных в буфере + 0x8000 - если остановлена удаленная передача (flow control к нам)
2 байта - порт 1 - состояние передающего буфера: длина данных в буфере + 0x8000 - если остановлена наша передача (flow control от нас)
4 байта - порт 1 - количество байт пропущенных при приеме из-за переполнения буфера

Выборочный опрос сотояния
CMD=0x02
DATA=
2 байта - битовое поле для массового опроса состояния устройств реле и датчики:
  0x0001 - слать логическое состояние датчиков, реле и логических реле
  0x0002 - слать реальное состояние датчиков и реле
  0x0004 - слать last1
  0x0008 - слать last0
  0x0010 - слать last01
  0x0020 - слать last10
  0x0040 - timer last
  0x0080 - timer
  0x0100 - слать состояние порта 0
  0x0200 - слать состояние порта 1
  0x0400 - не сбрасывать состояние полей last* и timer last
2 байта - резерв (битовое поле для массового опроса состояния устройств индукционных петель)
1 байт - резерв (битовое поле для массового опроса состояния устройств RFID считывателей)
ответ
RET_DATA=
все в зависимости от битового поля - смотрите "Опрос состояния"

Чтение EEPROM
CMD=0x36 DATA=2 байта смещение, 1 байт длина (макс 32)
RET_DATA=считанные данные

Запись EEPROM
CMD=0x34 DATA=2 байта смещение, 1-32 байта данные для записи

Временно сменить свой ID на случайный (1-253)
(до сброса или выключения, в EEPROM память не сохраняется). Применяется когда на одном порте есть несколько устройств с одинаковым ID, чтобы их временно разнести. Потом проводится поиск устройств и прописывается им другие ID. Потом сброс или выключение, и устройства будут иметь ID прописанный в EEPROM.

CMD=0xFC опционально DATA=1 байт маска (randomID & маска)

Установить реле, датчики, виртуальные реле
CMD=0x70 DATA=4 байта - битовая маска что меняем, 4 байта - битовые состояние что включаем-выключаем, 1 байт - побитно куда подать импульсы на реальные реле
Описание:
- если нам надо включить реле №1 тогда в маске бит этого реле выставляем = 1, в бит состояния = 1
- если нам надо выключить реле №2 тогда в маске бит этого реле выставляем = 1, в бит состояния = 0
- если нам надо дать импульс на реле №3 тогда в маске бит этого реле выставляем = 1, бит состояния не важно, в бит импульса = 1
- если меняем значение в реальном датчике - то для изменения состояние временно меняется бит инверсии (отладка)

Чтение счетчиков/таймеров
CMD=0x20 DATA=2 байта - адрес (0x880 - первый счетчик/таймер, 0x884 - второй, и т.д.), 1 байт - длина считываемых байт
Например: считать все счетчики/таймеры CMD=0x20 DATA=0x80 0x08 0x80
RET_DATA=считанные данные

Запись счетчиков/таймеров
CMD=0x22 DATA=2 байта - адрес (0x880 - первый счетчик/таймер, 0x884 - второй, и т.д.), 1...128 байт - данные
Например: изменить все счетчики/таймеры CMD=0x22 DATA=0x80 0x08 128 байт данные

Изменить настройки порта
CMD=0xF0 DATA=1 байт - номер порта, 4 байта - скорость и алгоритмы работы (смотрите описание EEPROM)

Чтение с порта
(когда порт находиться в RAW режиме, не в командном !) для приема в устройстве есть буфер в 512 байт
CMD=0xF2 DATA=1 байт - номер порта
или
CMD=0xF2 DATA=1 байт - номер порта, 1 байт - длина сколько хотим считать
ответ
RET_DATA=
2 байта - состояние приемного буфера: длина данных в буфере + 0x8000 - если остановлена удаленная передача (flow control к нам)
2 байта - состояние передающего буфера: длина данных в буфере + 0x8000 - если остановлена наша передача (flow control от нас)
4 байта - количество байт пропущенных при приеме из-за переполнения буфера
0...? - считанные данные

Запись в порт
(когда порт находиться в RAW режиме, не в командном !) для передачи в устройстве есть буфер в 256 байт
CMD=0xF4 DATA=1 байт - номер порта + 0x80=flush in + 0x40 flush out, 0...? байт - данные на передачу
ответ
RET_DATA=
1 байт - количество байт, принятых в буфер для передачи
2 байта - состояние приемного буфера: длина данных в буфере + 0x8000 - если остановлена удаленная передача (flow control к нам)

2 байта - состояние передающего буфера: длина данных в буфере + 0x8000 - если остановлена наша передача (flow control от нас)
4 байта - количество байт пропущенных при приеме из-за переполнения буфера

Карта EEPROM (@ смещение, внутренний таймер = 0.1мс = 10000 раз в сек, при изменении некоторых параметров необходима перегрузка):

```
// DEVID платы
uchar EE_DEVID @ 0
// броадкастовый адрес для плат (0xFF у Netronix)
uchar EE_DEVID_BROADCAST @ 1
uchar EE_DEVID_BROADCAST @ 1

#define RS_BAUD_X2    0x80000000
// алгоритм приема "чужих" пакетов
#define RS_RX_ALGO2    0x40000000
// fast broadcast - алгоритм работает только вместе с RS_RX_ALGO2 означает что если мы в ожидании своего таймслота
// и получили пакет от предыдущего устройства - то можно сразу отвечать, не ждать дальше
#define RS_RX_FASTBC    0x20000000
#define RS_HW_FLOW    0x10000000
#define RS_SF_FLOW    0x08000000
#define RS_RAW        0x04000000
#define RS_BAUD_MASK    0x00ffffff
// скорость ком порта RS232
uint32 EE_BAUD0 @ 2
// скорость ком порта RS485
uint32 EE_BAUD0 @ 6
// таймаут чтения с RS232
uint32 EE_USART_read_timeout0 @ 10
// таймаут чтения с RS485
uint32 EE_USART_read_timeout1 @ 14
// таймаут на таймслот на ответ при броадкасте по RS портам
uint16 EE_BROADCAST_timeslot @ 18
uint16 EE_BROADCAST_ADD_timeslot @ 20

// инверсия состояния датчиков и релюх (замкнутые контакты - срабатывание по размыканию)
uint16 EE_INVERSE_SENSORS @ 22

// состояние реле после включения питания платы
uchar EE_RELAYS_DEFAULT_STATUS @ 24
// побитно если бит=1 - состояние залочено и не работает импульсный режим
uchar EE_RELAYS_LOCKED @ 25

// таймаут для перехода в автономный режим
uint32 EE_AUTO_TOUT @ 26

// какие реле изменяем при переходе в автономный режим
uchar EE_AUTO_DEF_MASK @ 30
// на что изменяем
uchar EE_AUTO_DEF_VAL @ 31

// длительности импульсного режима
uint32 EE_RELAYS_IMPULSE_DELAY[MAX_RELAYS] @ 32
// "залипание" 1
uint32 EE_SENSORS_TOUT1[MAX_REAL_SENSORS] @ 56
// "залипание" 0
```

uint32 EE_SENSORS_TOUT0[MAX_REAL_SENSORS] @ 96

uchar EE_COMMANDS[872] @ 136

Примечание: EEPROM память допускает минимум 100000 циклов перезаписей

Примечание: если вы случайно прописали в устройство скорость порта которую не поддерживает ваш компьютер то можно сбросить параметры порта в стандартные 115200 если при включении держать закороченными контакты 1 и 6 на 6-ти контактном разъеме) от 5 до 10 секунд и раскоротить - то в EEPROM запишется конфигурация для портов по умолчанию. Если держать дольше чем 10 секунд то сброса не произойдет.

Описание файлов:

*.eeprom	Сохраненные конфигурации устройств (EEPROM)
*.algo	Исходные тексты алгоритмов
$$$CurrentAlgoritm$$$.algo	Временный файл, можно удалить (текст алгоритма с окна редактирования)
$$$CurrentAlgoritm$$$.pre	Временный файл, можно удалить (текст алгоритма после препроцессора)
$$$CurrentAlgoritm$$$.bin	Временный файл, можно удалить (бинарный код алгоритма)

Описание языка программирования алгоритмов:

Принцип работы устройства:
Устройстве в цикле все время повторяет последовательность действий:
1) по прерыванию (по событию) принять команду с ком порта — обработать — послать ответ
2) проверить датчики и реле, отработать механизм «залипания» контактов для датчиков, отработать импульсный режим для реле
3) выполнить алгоритм
4) повторить с пункта 1

В устройстве в EEPROM памяти доступно 872 байта для описания неких действий (алгоритма). Устройство понимает низкоуровневые команды в специальном формате (машинный код). Для удобства описания алгоритмов был создан свой язык и написан компилятор. Для удобства программиста который будет описывать алгоритм этот компилятор также понимает выходной формат текстовых распространенных препроцессоров, в частности к какому файлу принадлежит каждая строка и ее номер, чтобы в процессе компиляции алгоритма видеть в каком файле и в какой строке возникла ошибка.

Примечание: скорость работы зависит от прописанного алгоритма, в среднем <=7 тысяч полных циклов в секунду. если в алгоритме ошибка, и он зациклился, то после 65536 низкоуровневых команд алгоритма исполнение алгоритма прервется и выполнение алгоритма будет запрещено до сброса или выключения питания для того чтобы устройство не повисло а работало и можно было перепрошить новый алгоритм.

Текстовый препроцессор — это программа которая понимает макросы и подстановки, умеет вставлять в файл текст из другого указанного файла и т.д.. Текстовый препроцессор сам по себе довольно мощный, тут не будет описание всех его возможностей. Можете прочесть документацию тут http://mcpp.sourceforge.net/. Для описания алгоритмов возможности препроцессора не обязательно использовать, он служит только для удобства.

Тут будут описаны только самые важные и нужные вещи в препроцессоре:

Подгрузка, включение, вложение, «инклуды» других файлов:

#include “test.txt”
#include “test2.txt”

считать содержимое файла text.txt и вставить его вместо этой строчки и продолжить ее обработку. При чем в этом файле тоже могут быть вложения.

Примечание: в случае возникновения ошибки во вложенном файле в редакторе алгоритмов в программе EyesHands.exe он не будет загружен вместо исходного текста. Вам необходимо самому открыть указанный файл в текстовый редактор и найти нужную строку и столбец и исправить ошибку.

Подстановка текста, «дефайны»:

#define OsnovnoyDatchik d7
#define OsnovnoeRele r2
#define Timeout 10m
#define Uslovie OsnovnoyDatchik == 1 && t1 > Timeout

потом можно писать так

if(OsnovnoyDatchik == 1 && t1 > Timeout) OsnovnoeRele = Impulse

этот текст препроцессор после подстановки значений превратит в

if(d7 == 1 && t1 > 10m) r2 = Impulse

или даже так, с двойной подстановкой

if(Uslovie) OsnovnoeRele = Impulse

этот текст препроцессор после подстановки также превратит в

if(d7 == 1 && t1 > 10m) r2 = Impulse

Макросы:

#define SetRele(rele, znachenie) rele = znachenie
#define DatchikRele(datchik, rele) \
if(datchik = 01) rele = 1 \
if(datchik = 10) rele = 0

потом можно писать так

SetRele(r1,i)
DatchikRele(d1,r1)

после обработки этого текста препроцессором получим текст

r1 = i
if(d1 = 01) r1 = 1
if(d1 = 10) r1 = 0

Примечание: \в конце строки означает что строка не закончилась и к ней надо добавить следующую.

Комментарии

// комментарии в одной строке, все что после символов // до конца строки игнорируется
***Relay1 = Impulse** // на реле1 подаем импульс*
/ а это многострочный комментарий, может быть и из одной строки и из многих*
это примечание для примера
**/*
***Relay1 = Impulse** /* на реле1 подаем импульс */*

Описание языка

Примечание:

- в ***[]*** указано что можно не писать, например ***go[to]*** означает что можно писать и ***go*** и ***goto***, это одно и тоже
- можно писать и большими буквами и маленькими, разницы нет
- ***(a|b)*** означает что тут возможно написание чего то одного — или ***a*** или ***b***
- для удобства, если кто привык к синтаксису языка C и других - в конце любой команды можно ставить ;
- значение таймера описывается в формате число[Dd]число[Hh]число[m]число[s].число где D или d это дней, H или h это часов, m это минут, s это секунд, s можно не писать, число после . это десятые, сотые, тысячные и десяти тысячные секунды. Точность таймера равна 0.1 миллисекунде = 100 наносекунд (1/10000 секунды). В данном устройстве максимальное значение таймера может быть 0xffffffff / 10000 секунд что примерно = 5 суткам
- Примеры значений таймера:
- 0.0001 или .0001 = 1/10000 секунды = 100 наносекунд
- 0.19 или .19 = 19/100 секунды = 190 миллисекунд
- 7.99 = 7 секунд и 99 сотых секунд (почти 8 секунд)
- 15m20 = 15 минут и 20 секунд
- 3h = 3 часа
- 1d5m3.15 = 24 часа 5 минут 3 секунды и 15 сотых секунд
- 3600 = 3600 секунд что равно одному часу
- 72m = 72 минуты
- и т.д.

В данном устройстве доступны:

- 10 датчиков (сенсоров) ***(D|S[ensor])1-10***
- 6 реле ***R[elay]1-6***, также доступно как датчик 11-16
- 16 виртуальных реле ***V[irtual]1-16*** или также доступно как реле 7-22 и как датчик 17-32
- 32шт 32-битных беззнаковых счетчиков ***C[ounter]1-32*** или как таймеров ***T[imer]1-32***
- 1 автостатус (это 32 виртуальное реле) ***Auto***
- 1 автотаймер (обнуляется автоматически когда на устройство приходит по ком порту команда получение статуса, то есть это время когда устройство последний раз получало команду от управляющего компьютера) ***AT*** или ***AutoT[imer]*** или как ***T[imer]33***

Описание синтаксиса и команд:

end
закончить исполнение алгоритма, после этой команды следующие за end команды не исполняются в этом цикле (смотри "Принцип работы устройства")

label:
метка, пометить место меткой, метка может быть любым словом начинаючимся с A-Z и _ и может содержать также 0-9

go[to] label
прыжок, переход на метку label, - продолжить исполнение команд с метки label

go[to](y|yes|1|true) label
прыжок, переход на метку label, - продолжить исполнение команд с метки label только если перед этим была команда if и резальтат условия был «да», если последнее условие было «нет» пропустить эту команду (вам скорей всего не понадобится эта команда)

go[to](n|no|0|false) label
прыжок, переход на метку label, - продолжить исполнение команд с метки label только если перед этим была команда if и резальтат условия был «нет», если последнее условие было «да» пропустить эту команду (вам скорей всего не понадобится эта команда)

call label
прыжок или переход в подпрограмму на метку label, - продолжить исполнение команд с метки label до команды ret, потом вернуться назад и продолжить исполнение команд после команды call

call(y|yes|1|true) label
прыжок или переход в подпрограмму на метку label, - продолжить исполнение команд с метки label до команды ret, потом вернуться назад и продолжить исполнение команд после команды call (только если перед этим была команда if и резальтат условия был «да», если последнее условие было «нет» пропустить эту команду (вам скорей всего не понадобится эта команда))

call(n|no|0|false) label
прыжок или переход в подпрограмму на метку label, - продолжить исполнение команд с метки label до команды ret, потом вернуться назад и продолжить исполнение команд после команды call (только если перед этим была команда if и резальтат условия был «нет», если последнее условие было «да» пропустить эту команду (вам скорей всего не понадобится эта команда))

ret
return - возврат из подпрограммы, если мы прыгнули сюда командой call* то вернуться назад и продолжить исполнение, иначе прервать работу алгоритма в этом цикле (смотри "Принцип работы устройства")

Управление реле: (только если не залочено)
R1 = 1|y|yes|on // включить
R1 = 0|n|no|off // выключить
R1 = i[mpulse] // подать импульс (включить на указанный в конфигурации период и потом автоматически выключить)

Изменение счетчиков:
C1 = число
C1 + число
C1 - число

Обнуление-перезапуск таймеров (таймер по сути как секундомер — но никогда не останавливается):
Reset|Start|Restart|Clear T1

if условие если_да [else если_нет]
Условный блок, выполнить что то ***если_да*** или ***если_нет*** если верно некое ***условие.*** Условие может состоять из одного или нескольких условий. Операции сравнение в условиях:

равно	== или =

не равно	***!***= или <>
меньше	<
больше	>
меньше или равно	<= или =<
больше или равно	>= или =>

Значение датчиков для сравнения могут быть:

- ***1|y|yes|on*** включен
- ***0|n|no|off*** выключен
- ***01*** только что включен
- ***10*** только что выключен.

Примеры одиночных условий:
Примечание: перед каждым условием можно писать ***!*** или ***not*** что означает наоборот (инверсия условия). (например ***D1 = 1*** то же самое что и ***! D1 = 0***)
Синтаксис:
[!] датчик или
[!] датчик =|==|!=|<> значение датчика
Сравнение значения датчиков:
D1
D32
R4
V20
Auto
и т.д.
что означает что данный датчик включен, то есть это одно и то же что и
D1 == 1
D32 == 1
R4 == 1
V20 == 1
Auto == 1
или например одиночное условие «если датчик 1 выключен»:
D1 = 0 что одно и то же что и ***!D1*** или ***! D1*** или ***NOT D1*** или ***! D1 == ON*** и т.д.
пример: если только что перешли в автономный режим:
Auto == 01 // верно только 1 раз при переходе в автономный режим
Auto == 1 или ***Auto*** верно все время пока находимся в автономном режиме
R4 = 10 // означает реле 4 только что перешло из состояние 1 в состояние 0, то есть другими словами только что выключилось
Проверка залоченности реле:
Locked R1 // верно если реле 1 «залочено» (отключено) в конфигурации
! Locked R1 // верно если реле 1 не «залочено»
еще интересный пример:
! D1 = 01 или ***D1 != 01*** или ***D1 <> 01*** что одно и тоже означает верно если датчик 1 только что не перешел из состояния 0 в состояние 1
Сравнение счетчиков:
Примечание: значение счетчика может быть от 0 и до 4294967295
Синтаксис:
[!] счетчик =|==|!=|<>|<|>|<=|=<|>=|=> число
Примеры:

C1 = 0
! Counter12 < 627354
c2 >= 7
и т. д.
Сравнение таймеров:
Синтаксис:
[!] таймер >=|=>|>|< значение таймера (смотри 3.3)
Примеры:
AT > 10m
верно если таймер AT больше или равно 10 минутам
t12 < .5
верно если не прошло пол секунды
и т. д.

Из одиночных условий можно составлять множественные проверки а также группировать их в ***()*** и описывать некие условия для удобства. Примеры:
верно если верно хотя бы одно из условий (или)
условие1 | или || или ***OR условие2***
верно если верны все условия (и)
условие1 & или ***&&*** или ***AND условие2***
пример использования ***()***
(d1 | d2) & (d3 | d4)
верно если верно d1 или d2 и d3 или d4
результат проверки группированных условий можно также инвертировать ***!*** или ***NOT***, например:
!(d1|d2|d3|d4)
верно если датчики 1-4 все выключены. Это условие можно также описать множеством других вариантов, например:
d1 = 0 & d2 = 0 & d3 = 0 & d4 = 0
или
!d1 & !d2 & !d3 & !d4
и т. д.
Все это в итоге одно и то же и сгенеренный код алгоритма в бинарном виде будет один и тот же. Выбирайте способ какой удобнее, разницы нет.

Еще примеры:
!(d1 & !(r4 | !Locked r4 & (c1 >= 75 | AT < 7)) & t5 > 10m)

Примеры использования условного блока и рекомендованный синтаксис:
Примечание: если в ***если_да*** и ***если_нет*** нужно исполнить несколько команд то сгруппируйте их в ***{}*** также можете всегда использовать ***{}*** для удобства.
If !Auto END
if(!Auto) END
if!(Auto) END
if(Auto == Off) end
это одно и тоже, если датчик авторежима выключен тогда закончить исполнение алгоритма.
```
If(d1 == 1) { // исполняем если датчик 1 включен
   r1 = 1
   r2 = 0
} else { // исполняем если датчик 1 выключен
   r1 = 0
   r2 = 1
}
```
В условном блоке можно включать все команды без ограничения, то есть if тоже. Например:
```
if !Auto {
   if d1
```

```
    r1 = 0
  else
    r1 = 1
} else c2 + 1
```
Пример использования подпрограмм:
```
if что то call Error
if что то другое call Error
END
Error:
r1 = 0
r2 = 0
c1 = 0
RET
```
и т. д.
Мы по метке Error описали некие действия которые необходимо делать несколько раз. Чтобы не писать одно и тоже много раз — мы тут этот повторяющийся кусок кода вынесли в подпрограмму.

Примечание: хотя тут можно использовать препроцессор вот так:
```
#define Error \
  { r1 = 0 \
  r2 = 0 \
  c1 = 0 }
if что то Error
if что то другое Error
```
но в итоговом сгенеренном (скомпилированном) бинарном коде будет 2 раза описаны команды r1 = 0, r2 = 0, c1 = 0 а если использовать подпрограмму — то 1 раз и хотя немного медленнее но зато бинарный код алгоритма будет меньше по размеру, а учитывая что места в EEPROM памяти под алгоритм не много — то меньше размер на 6 байт гораздо лучше чем например скорость исполнения главного цикла не 5000 раз в секунду а например 4980 раз. (числа даны примерные но они близки к реальным)

Итак рассмотрим несколько полезных примеров используя все что описано выше:
(учитывая что памяти в EEPROM не много, особого смысла использовать возможность препроцессора подгружать (включать) содержимое других файлов нету)

Задача 1: необходимо при переходе в автономный режим как бы скоммутировать провода от кнопки на датчике 1 к реле номер 1
Реализация:

```
// определяем для удобства к какому датчику у нас подключена кнопка
#define KNOPKA d1
// и реле
#define RELE r1
// если сейчас не «авто» режим
if(!Auto) {
  // мы только что вышли ! из «авто» режима, не забываем выключить реле :)
  if Auto = 10 RELE = 0
  // если не «авто» режим — нам больше делать нечего
  END
}
// если кнопка только что нажата — включить реле
if(KNOPKA = 01) RELE = 1
// если кнопка только что отпущена — выключить реле
if(KNOPKA = 10) RELE = 0
```

Итак в авто режиме мы на реле подаем состояние кнопки, по сути то же самое как будто кнопка подсоединена в реле напрямик. И не забываем отключить реле если вдруг вы вышли из авто режима а реле было включено.

Задача 2: имеем:

- петля1 — датчик1 (перед шлагбаумом на улице)
- петля2 — датчик2 (после шлагбаума на территории)

Необходимо посчитать сколько машин въехало (счетчик c1) и выехало (счетчик c2).
Реализация:

```
#define Petla1 d1
#define Petly2 d2
#define Zaehalo c1
#define Viehalo c2

if(Petlya1 == 10) {
   if(v2) {
      v2 = 0
      Viehalo + 1
   }
   v1 = 1
}
if(Petlya2 == 10) {
   if(v1) {
      v1 = 0
      Zaehalo + 1
   }
   v2 = 1
}
```

Но тут есть сложности в подсчете, а именно, невозможно точно подсчитать количество заехавших и выехавших в данной конфигурации. Пример машина заехала на петлю1 постояла у закрытого шлагбаума и отъехала назад как только петля1 выключилась у нас v1 = 1. Через некоторое время — например даже через день !!! на петлю2 заехала машина и ! неважно отъехала она назад или выехала с территории но ! как только она съедет с петли2 у нас сработает правило и мы посчитаем что вчерашняя машина заехала ! И хотя абсолютно точно машины невозможно посчитать с данными датчиками но все же такие вот «подвисшие» во времени ситуации можно легко обходить с помощью таймеров. Для примера применен немного другой синтаксис.

```
#define Petla1 d1
#define Petly2 d2
#define Zaehalo c1
#define Viehalo c2
// ждем проезда машины 3 секунды!
#define Zhdem 3

/*
```

```
v1 = 1 - значит машина съехала с петли1 и мы ждем когда она заедет и съедет с петли2
или машина съехала с петли2 и мы ждем когда она заедет и съедет с петли1
но !!! таймаут истек ! То есть машина съехала с петли больше чем 3 секунды назад но так
и проехала по следующей петле, обнуляем виртуальное реле 1 - v1 — по сути сбрасываем
ожидание
*/
if(v1 & t1 >= Zhdem) v1 = 0
if Petlya1 = 10 {
   if v1 {
      v1 = 0
      Viehalo + 1
   } else {
      v1 = 1
      clear t1
   }
}
if Petlya2 = 10 {
   if v1 {
      v1 = 0
      Zaehalo + 1
   } else {
      v1 = 1
      clear t1
   }
}
```

Отличный пример, уже намного лучше чем первый, без учета времени. Подумайте, почему мы тут не используем v2 ? Почему тут достаточно v1 и v2 использовать нету никакого смысла ?
Также подумайте — когда при данном алгоритме может возникнуть ситуация когда машина проехала а мы ее не посчитали.
(Подсказка: машина проезжает петлю, но останавливается на следующей больше чем на указанные 3 секунды, и хотя кажется что надо просто увеличить таймаут — но лучше все же обрабатывать состояние по срабатывании петель то есть по = 01)