Современные программные средства связи микроконтроллера с компьютером по интерфейсу RS-232

Алексей КУЗЬМИНОВ
compmicrosys@mail.ru

3. Программирование интерфейса RS-232 в ОС Windows98/XP

3.1. Предварительные замечания

Программирование интерфейса RS-232 в ОС Windows не является чем-то сверхъестественным. Внешний вид программ, написанных на идентичных языках программирования (например, Clarion v3.101 для DOS и Clarion v.6.0 для Windows), также практически одинаков. Исключением (по крайней мере, для языка Кларион) яаляется название экранной формы. Если в Кларионе для DOS экранная форма называется SCREEN («Экран»), то в Кларионе для Windows она называется Window («Окно»). Структуры данных экранных форм также идентичны. Преимущество экранной формы Window перед SCREEN заключается в том, что она обладает большей универсальностью, имеет больше возможностей и, самое главное, эта экранная форма более автоматизирована. Если посмотреть на тексты всех программ, приведенных в статье, то можно заметить, что структура Window в них очень уж «навороченная», поскольку там используется несколько шрифтов и масса другой информации. На первый взгляд кажется, что «грамотно» написать структуру экранной формы Window очень сложно и можно совершить немало ошибок. Но на самом деле, экранную форму Window писать вообще не требуется (!), поскольку она генерируется языком Кларион автоматически. Необходимо только выбрать размер окна (не в тексте программы, а буквально на экране), установить на нем соответствующие кнопки и другие параметры и атрибуты. Все это делается мышью с уже готовым (выбранным по умолчанию) окном. После того как окно «устраивает» программиста, он нажимает определенную кнопку, и экранная форма Window автоматически генерируется (точнее генерируется ее текст на языке Кларион). Более полную информацию по этому вопросу можно найти в документации по языку.

Все остальные «вещи» в программах для DOS и Windows программируются практически одинаково.

Есть, правда, одно свойство языка Clarion v.6.0 (по сравнению, например, с языком Clarion v.5.5 и более ранними версиями), которое необходимо учитывать.

Язык Clarion v.6.0 поддерживает так называемую thread-model, которая связана с многозадачностью операционной системы (в основном это касается Win'XP). Слово thread можно перевести как некий процесс. Если в программе одновременно идут несколько процессов, то Clarion v.6.0 поддерживает такие задачи. Более подробно с подобными вопросами можно ознакомиться в руководстве по этому языку.

Здесь же автору хотелось бы предостеречь программистов от одной (не тривиальной) ошибки, из-за которой программа работать не будет.

Рассмотрим фрагмент программы, который присутствует практически во всех задачах (при программировании в ОС Windows):

```
START
            -
            -
            -
        accept
!-------------------
            case accepted()
               of ?Ok
                  p=1
                  break
               of ?Cancel
                  p=0
                  break
        end ! ot case
!-------------------------
        end ! от accept

        if p=0 then goto E.

        goto START

E       return
```

В приведенном фрагменте используется конструкция accept…end, которая применяется для определения ветвления программы в зависимости от нажатия кнопок OK или Cancel («Выход»). Эти кнопки расположены внизу любого окна. Нажатие кнопки OK приведет к тому, что в программе будет осуществлен переход к метке START. Нажатие кнопки Cancel приведет к переходу к метке E, то есть к выходу из программы. Конструкция на первый взгляд довольно тривиальная и может быть легко упрощена. Например, так, как это сделано в программе на Кларионе для DOS (см. пункт 2.3.4):

```
START
            -
            -
            -
        accept
!-------------------
            case accepted()
               of ?Ok
                  goto START
               of ?Cancel
                  goto E
            end ! ot case
!-------------------------
            end ! от accept
E       return
```

В чем отличие этих двух вариантов?

В первом варианте при нажатии кнопки OK переменной p присваивается значение 1 (p=1), и далее следует оператор break, который организует выход из процесса (thread'a) — accept … end. При нажатии кнопки Cancel переменной p присваивается значение 0 (p=0) и далее точно так же следует оператор break, который аналогично организует выход из процесса (thread'a) — accept … end. После выхода из процесса accept … end анализируется переменная p, и в зависимости от ее значения программа переходит к метке START (при p=1), либо к метке E (при p=0), по которой осуществляется выход из программы.

В отличие от первого варианта второй — более простой. Там нет никакой перменной p, и ветвление происходит сразу: при нажатии кнопки OK программа переходит к метке START (goto START), а при нажатии кнопки Cancel — к метке E (goto E) и далее осуществляется выход из программы.

С точки зрения синтаксиса языка оба варианта безупречны. Мало того, в Clarion v.5.5 (и в Clarion v.3.101 для DOS) оба варианта будут безукоризненно работать и не вызывать накаких проблем.

В Clarion v.6.0 второй вариант может в лучшем случае остановить программу, которая не будет подавать никаких «признаков жизни», и выход из ситуации можно осуществить, нажав классическое Ctrl-Alt-Del, а затем «завершить процесс». В худшем случае второй вариант может «повесить» компьютер, да так, что «оживить» его можно будет, только нажав кнопку RESET на системном блоке. Отчего это?

А оттого, что цикл accept…end — это процесс (thread), который «по умолчанию» организует Clarion v.6.0, выход из которого (процесса) во втором варианте не завершает его. А незавершенный процесс будет продолжаться бесконечно (вот отчего компьютер может «повиснуть»)!

В первом варианте при любом условии (не важно — OK или CANCEL) вначале производится явное завершение процесса accept…end, а затем, когда процесс accept…end завершен, уже анализируется переменная p, и, в зависимости от ее значения, осуществляется соответствующий переход. Но — и в этом вся соль — этот анализ происходит уже после того, как процесс accept…end завершен.

Столь подробное объяснение приводится здесь для того, чтобы незадачливый программист «не попался».

Но это лишь исключительный случай. В основном, программы в Кларион для DOS и Кларион для Windows отличаются очень мало.

3.2. Варианты программирования интерфейса RS-232 в Win'98/XP

Как уже упоминалось, программирование интерфейса RS-232 в операционных системах Windows возможно двумя способами. Первый способ — использование функций API (Application Program Interface) — программного интерфейса для разработки приложений. Второй способ — использование прямых команд ввода/вывода в порт (в данном случае RS-232 или COM-порт).

Использование функций API — стандартный путь, на который указывают все документированные источники по программированию в Windows. Применение API-функций наталкивается на ряд проблем. В частности, API-функции работают очень медленно, и, если речь идет о быстрой реакции на какое-либо кратковременное (например, несколько микросекунд) внешнее воздействие на порт RS-232, то функции API не успевают отследить такое воздействие. Кроме того, некоторые функции работают с явными ошибками, то есть либо выполняют не совсем те действия, которые, например, указаны в описании на конкретную функцию API, либо выполняют дополнительные действия, которые не описаны.

Использование прямых команд ввода/вывода в порт также имеет свои проблемы (на них мы остановимся позже). Однако программирование интерфейса прямыми командами ввода/вывода в порт хорошо известно, поскольку используется в DOS. Кроме того, прямые команды ввода/вывода в порт выполняются очень быстро, так как это «машинные» команды процессора. Они позволяют отследить самые быстротекущие процессы, происходящие с интерфейсом RS-232.

В связи с вышеизложенным, вначале рассмотрим программирование интерфейса RS-232 с помощью функций API, а затем — с помощью прямых команд ввода/вывода в COM-порт.

3.3. Программирование интерфейса RS-232 с помощью функций API

Рассмотрим кратко те функции API, которые имеют непосредственное отношение к программированию интерфейса RS-232 в 32-разрядном режиме. Полное описание всех функций API можно найти в MSDN (об этом уже говорилось ранее).

Прототипирование функций API на Clarion v.6.0 приводится в программе Hello.clw, написанной для работы в Windows98/XP (и приведенной ниже). Там же можно найти числовые значения констант, использующихся в API-функциях. Необходимо отметить, что в Clarion v.5.5 и Clarion v.6.0 есть программа (WINAPI.EXE), в которой также описаны прототипы и значения констант API-функций. Некоторые из прототипов и констант API-функций в этой программе не верны, поэтому пользоваться ими нужно с осторожностью.

Первой функцией API, которой должно предваряться программирование RS-232, является функция открытия порта — CreateFileA (создать файл). Функция возвращает handle порта — числовое значение, которое операционная система приписывает открытому порту. Дословно handle — ручка («потянув» за которую, можно «открыть» порт). Функция CreateFileA работает правильно и претензий не вызывает.

После открытия порта необходимо обратиться к функции построения контрольного блока COM-порта — BuildCommDCBA, в которой требуется перечислить параметры порта (скорость обмена, формат данных, количество стоп-бит и т. п.). Эта функция включает в себя два параметра: ControlString и PortStruct.

ControlString — строка символов, которая, например, может принимать следующее значение: ControlString='Com1:115200,N,8,1', означающее, что будет открыт порт COM1, который будет работать на скорости 115 200 бод, бит паритета не используется, количество бит данных — 8 и количество стоп-бит — 1.

Параметр PortStruct — это массив байт, определяющий более полно структуру работы COM-порта и в который, в частности, входят параметры из ControlString (скорость, бит паритета, количество бит данных и количество стоп-бит). Так вот, если, например, указать в ControlString скорость 115 200, а в PortStruct скорость 9600, то порт будет работать на скорости 9600 бод. Если же в ControlString указать 9600, а в PortStruct 115 200, то скорость работы порта может принять непредсказуемое значение. По этой причине изменять в PortStruct параметры, установленные в ControlString, не рекомендуется.

Необходимо отметить, что функция BuildCommDCBA не производит никаких действий с портом; в ней только перечисляется, какие параметры порта необходимо установить, но установку этих параметров эта функция не производит.

Для установки параметров, указанных в функции BuildCommDCBA, в COM-порт используется функция SetCommState — установить статус порта. Эта функция претензий не вызывает и работает правильно.

Для получения параметров уже работающего порта можно использовать функцию GetCommState — получить статус порта. Эта функция возвращает handle порта, его структуру (PortStruct) и претензий не вызывает.

Для передачи данных по последовательному порту можно использовать две функции API: TransmitCommChar (послать символ через COM-порт), которая выводит в порт один байт, или WriteFile (запись файла), которая может выводить как массив (буфер) байт, так и всего один байт. Эти функции не вызывают претензий и работают правильно.

Чтение информации из порта возможно только функцией ReadFile. Эта функция может прочитать из порта либо один байт, либо массив (буфер) байт только в том случае, если этот байт (или массив байт) поступил в COM-порт. В противном случае функция возвращает ошибку и может, кроме того, «повесить» компьютер на неопределенно долгое время, если байт в порт не поступал. При чтении, например, одного байта, размер входного буфера уменьшается на единицу. Размер входного буфера указывается в функции SetupComm (установка COM-порта), но если, например, в порт поступает байт больше, чем указанный размер буфера (например, 1), то байты не теряются, а накапливаются в буфере, и размер буфера растет. Этот факт является крупной ошибкой, и исправлять ее пока никто не собирается. Возможно, до исправления этой ошибки в конце концов и доберутся разработчики WindowsXP (или последующих версий Windows).

Для очистки входного буфера предусмотрена функция PurgeComm (очистка COM-порта), которая очищает входной буфер от поступивших байт, если, например, они не нужны. Очистить входной буфер можно, применив ту же функцию ReadFile. Если указать количество байт для чтения равным 1, то для очистки буфера размером в 10 байт необходимо 10 раз вызвать функцию ReadFile; если же указать количество байт для чтения равным 10, то эту функцию необходимо вызвать 1 раз.

У читателя может возникнуть вопрос: а зачем вообще очищать входной буфер? Если в порт приходит 10 байт, то и читать нужно 10 байт. Здесь причин несколько.

Во-первых, одним из параметров функции CreateFileA, открывающей COM-порт, является 5-й по счету параметр (всего их 7), который именуется CreationDisposition (создание диспозиции) и который должен принимать значение Open_Existing (то есть открыть существующий). Это означает, что открывается COM-порт со своим уже построенным контрольным блоком (DCB), у которого все параметры приняты по умолчанию, и, в частности,

состояния линий квитирования DTR и RTS, как правило, установлены в высокий уровень. После открытия порта (то есть после того, как отработала функция CreateFileA), но перед тем, как будут выполнены функции BuildCommDCBA и SetCommState, в которых, например, линии DTR и RTS должны быть в состоянии сброса (то есть низкого уровня), эти линии установятся в высокий уровень и будут там находиться до тех пор, пока функции BuildCommDCBA и SetCommState не отработают. После того, как функции BuildCommDCBA и SetCommState отработают, состояния линий DTR и RTS установятся уже в низкий уровень. Таким образом, уровни напряжений на линиях DTR и RTS кратковременно перейдут из низкого состояния в высокий и обратно. Если используется гальванически развязанный интерфейс с питанием от линий RTS и DTR, то на линии RxD появится несколько импульсов. Поскольку это линия данных, это приведет к тому, что в буфер COM-порта введутся несколько байт (1–3), которые (это уже понятно) являются «бросовыми» и которые необходимо сбросить (то есть очистить от них приемный буфер). Сколько этих байт введется и сколько раз надо применить функцию ReadFile, чтобы очистить буфер, — с точностью предсказать невозможно. Если, предположим, ввелось три байта, а функцию ReadFile применили 2 раза, то в буфере будет «сидеть» один неверный байт. Если же ввелось два байта, а функцию применили три раза, то компьютер «повиснет» и будет «ждать» прихода третьего байта (который может не прийти вовсе!).

Во-вторых, если опять же используются развязки с питанием от линий квитирования, то при установке этих линий в соответствующее состояние (уже по программе) на линии данных (RxD) опять может возникнуть несколько импульсов и во входной буфер опять введутся несколько байт. Это приведет к ситуации, описанной выше.

Эти две причины являются некоторым препятствием к применению таких гальванических развязок, если для программирования RS-232 используются функции API.

В-третьих, если сигнал DTR используется для сброса и запуска микроконтроллера (то есть если он управляет сигналом RST), то при кратковременном пребывании сигнала DTR на высоком уровне микроконтроллер может запуститься и изменить (уже по своей программе) состояние линии RxD (например, дать разрешение на передачу байта). Это приведет к поступлению еще одного (неверного) байта во входной буфер. По этой причине, даже если не используются гальванические развязки или используются, но питаются энергией от отдельного источника питания, в микроконтроллере желательно предусмотреть некоторую (небольшую) аппаратную, либо программную задержку для исключения этого эффекта.

И, наконец, в-четвертых. При использовании алгоритма обмена по RS-232, применяющего синхронизацию по линии RxD для разрешения и запрещения компьютеру передавать очередной байт, в буфер опять будут вводиться байты, правда, в этом случае их количество уже строго определено и равно в точности тому количеству байт, синхронизацию поступления которых и обеспечивает такой алгоритм. Если, например, из компьютера в микроконтроллер передается 75 байт, с двумя байтами длины (то есть всего 77), а принимается также 75 байт, то ясно, что первые 77 байт, которые ввелись в буфер, необходимо очистить (применив функцию ReadFile 77 раз, читая по 1 байту, или 1 раз, читая сразу 77 байт). После этого следующий прочитанный байт (78-й) уже будет верен.

Для определения, сколько байт находится во входном буфере, существует структура COMSTAT (статус COM-порта), в которой это количество байт присутствует в переменной CbInQue (размер в байтах входной очереди). Структуру COMSTAT, в частности, заполняет API-функция ClearCommError (сброс ошибки COM-порта). Поскольку, как указывалось, в некоторых случаях предсказать точное значение ненужных байт нельзя, то возможно применение следующего автомата: вызывается API-функция ClearCommError, определяется количество байт во входном буфере CbInQue, затем 1 раз вызывается API-функция ReadFile с CbInQue количеством читаемых байт. Применив API-функцию PurgeComm совместно с описанным автоматом, можно исключить «зависание» компьютера.

Необходимо отметить, что при программировании COM-порта прямыми командами ввода/вывода в порт в DOS и Windows98/XP таких проблем не возникает по нескольким причинам. Во-первых, линии DTR и RTS изначально находятся в низком уровне и при инициализации порта не изменяются без нашего на то ведома (как в функциях API). Во-вторых, размер буфера там аппаратный и умещает только один байт, то есть если в буфере уже есть байт и приходит еще один, то предыдущий байт теряется. В-третьих, прямую команду процессора in, которая очищает аппаратный буфер COM-порта, можно применять хоть 100 раз; при этом компьютер ни разу не зависнет, даже если во входном буфере давно уже нет байта и в помине. API-функции же написаны так, что приходящий байт вводится в программный буфер независимо от того, требуется ли это нам или нет. Это и является причиной вышеописанных «глюков».

Для установки выходных линий (DTR, RTS и TxD) COM-порта в произвольное состояние имеются три функции API, которые претензий не вызывают и работают правильно.

Первая — это EscapeCommFunction (переназначить функцию COM-порта — вспомним клавишу Escape(Esc), находящуюся в самом верхнем левом углу клавиатуры), которая может установить указанные линии в определенное состояние (какую конкретно и в какое состояние — указывается в параметрах для этой функции).

Вторая — это SetCommBreak (установка сигнала Break), которая устанавливает линию TxD в единичное состояние (высокий уровень).

Третья — ClearCommBreak (сброс сигнала Break) сбрасывает линию TxD в нулевое состояние.

Как видно, функция EscapeCommFunction дублирует две последних.

Для чтения состояния входных линий DSR, CTS, DCD и RI существует функция GetCommModemStatus (получить статус модема). Эта функция по-разному работает в Windows98 и WindowsXP. Если, например, линия DSR соединена с линией RxD, и по ней поступает какая-либо информация, то при вызове этой функции в Windows98 она даст правильный результат. В WindowsXP в такой ситуации эта функция ведет себя непредсказуемо, и пользоваться ей не рекомендуется.

Для чтения состояния указанных линий (точнее — для проверки факта изменения этих состояний) и для других целей используются совместно две функции API — SetCommMask (установка маски COM-порта) и WaitCommEvent (ожидание наступления события COM-порта). Эти функции работают правильно и претензий не вызывают, однако во многих справочниках и руководствах приводится неправильное толкование их совместного функционирования.

Функция SetCommMask устанавливает так называемую маску порта, в которой в качестве параметра или параметров указываются те биты, которые отвечают за наступление того или иного события. Если требуется отследить одно событие, то указывается один бит (точнее его название), если несколько — то несколько бит.

Функция WaitCommEvent выполняет две задачи. Первая задача — при наступлении одного из указанных в функции SetCommMask событий она выдает ненулевое значение. Для определения, какое именно событие произошло, функция возвращает параметр, по которому это можно определить, прочитав его. Это ее вторая задача.

Если требуется отследить наступление каждого из двух событий, но в разное время, то в функции SetCommMask необходимо указать два события, два раза вызвать функцию WaitCommEvent, в которой каждый раз необходимо проверить статус и сравнить его с битами того или иного события.

Пример. Отследить два события: поступление байта в COM-порт (EV_RXCHAR) и установку DSR в высокий уровень (EV_DSR).

1. Устанавливаем маску с двумя событиями: EV_RXCHAR+EV_DSR.

```
!----------------------------------------
   loop
    if SetCommMask(CommPort,EV_RXCHAR+EV_DSR) |
    then break. !Маска на прием байта и DSR
    .
!----------------------------------------
```

2. Ожидаем установки DSR в высокий уровень, после чего, например, выполняем команду ClearCommBreak.

```
!----------------------------------------
   loop
    loop until WaitCommEvent(CommPort,EvtStatus,0).
          if band(EvtStatus,EV_DSR) !DSR
          if ClearCommBreak(CommPort) then break.
    .
   .
!----------------------------------------
```

3. Ожидаем приема байта в COM-порт.

```
!----------------------------------------
   loop
    loop until WaitCommEvent(CommPort,EvtStatus,0).
     if band(EvtStatus,EV_RXCHAR) then break.
    .
!----------------------------------------
```

Если требуется отследить только одно событие, то можно установить маску на это событие, а затем просто вызвать функцию WaitCommEvent и не проверять возвращаемый им статус:

1. Устанавливаем маску на поступление байта в COM-порт:

```
!----------------------------------------
   loop
     if SetCommMask(CommPort,EV_RXCHAR)|
     then break. !Маска на прием байта
     .
!----------------------------------------
```

2. Ожидаем наступление этого события (без проверки статуса, как в предыдущем случае):

```
!----------------------------------------
   loop until WaitCommEvent(CommPort,EvtStatus,0).
!----------------------------------------
```

В последнем случае функция WaitCommEvent выполняется намного быстрее, так как содержит не тройную проверку в цикле, а единственную. Во многих руководствах по программированию API-функций, как уже упоминалось по поводу их не совсем правильного толкования, помимо проверки факта наступления события (п. 2) опять, как в предыдущем примере, сравнивается статус порта с битом этого единственного события (EV_RXCHAR), что является лишним и требует дополнительного времени (а иногда счет идет на доли микросекунды!).

После окончания работы с COM-портом его необходимо закрыть. Это делается с помощью вызова функции CloseHandle (закрыть Handle), которая претензий не вызывает и работает правильно.

И последняя функция API, которой хотелось бы коснуться, — установка временной задержки: функция Sleep (спать). Она, вообще говоря, не принадлежит к функциям COM-порта, но в связи с тем, что она отсутствует в Clarion v.6.0, автор рекомендует ее использовать. Вызвать эту функцию просто:

```
Sleep(100)
```

После этого программа будет ждать 100 миллисекунд, а затем будет выполнять следующий оператор.

Приведенные функции API для COM-порта — далеко не полный их перечень. Как уже указывалось, полный перечень находится, например, в MSDN. Для более детального уяснения функций API рекомендуется просмотреть в MSDN хотя бы те, что приведены выше, поскольку именно они используются для программы, приведенной автором далее.

3.3.1. Тестовая программа обмена для компьютера, использующая функции API, в ОС Win'98/XP

После такого несколько сумбурного изложения автором основных функций API, читателю может показаться, что запрограммировать интерфейс RS-232 с их помощью просто невозможно, либо достаточно сложно. Отчасти это так и есть. Тем не менее, автору удалось написать достаточно стабильно работающую программу, которая приведена ниже. Программа выполняет те же функции, что и тестовая программа для DOS (Clarion v.3.101), приведенная в 2.3.4. Она написана на языке Кларион для Windows — Clarion v.6.0. Название программы — HELLO.CLW. Для работы с ней может использоваться любой из микроконтроллеров, работающий с программой, приведенной в 2.3.4.

В программе HELLO.CLW пользователь может выбрать скорость обмена по интерфейсу RS-232 (9600 или 115 200 бод). Разумеется, программа для микроконтроллера должна быть написана так, чтобы указанные скорости обмена по RS-232 микроконтроллера и компьютера совпадали.

Текст программы HELLO.CLW приведен ниже. Вслед за текстом программы приведен текст ее файла-проекта Hello.prj.

Программа HELLO.CLW:

```
----------------------------
!-------------------------------------------------------------
! Программа передачи и приема строки из 75 символов по RS-232
! с помощью API-функций Windows98/XP.
! Используется программа для микроконтроллера: INRS115.c
!-------------------------------------------------------------

PROGRAM

!=======================
            INCLUDE('Equates.CLW')
            INCLUDE('TplEqu.CLW')
            INCLUDE('Keycodes.CLW')
            INCLUDE('Errors.CLW')
!=======================

!-----------------------------------------
! Определение переменных
!-----------------------------------------
C           byte
A           byte
B           byte
S           string(75)
BT          byte
M           byte,dim(75),over(S)
S1          string(80)
M1          byte,dim(80),over(S1)
i           ushort
i0          ushort(1)
j           ushort
k           ushort
STR         string(15)
CH          BYTE
N1          LONG
N           BYTE
CH1         BYTE
NX          ULONG
p           byte
SPEED       string(6)
NP          string(4)
!-----------------------------------------------
! Переменные для частоты и счетчика
!--------------------------------
CSTARTL     ulong
CSTARTH     long
CSTOPL      ulong
CSTOPH      long
DC          long
DELTAC      long
!--------
FREQUENCY GROUP,PRE(FRE)
FREQL       ULONG
FREQH       LONG
.
COUNTER     GROUP,PRE(CNT)
COUNTL      ULONG
COUNTH      LONG
.
!-------------------------------------

!-------------------------------------
! Переменные для API
!-----------------------------------------------------
CommPort            UNSIGNED    !(handle).
PortCString         CSTRING(5)  !'COM1' и т.п.
ControlCString      CSTRING(50) !'COM1:115200,N,8,1' и т.п.
GENERIC_READ        ULONG(80000000H)
GENERIC_WRITE       ULONG(40000000H)
ACCESS              ULONG(0C0000000H)

CREATE_ALWAYS       ULONG(2)
OPEN_EXISTING       ULONG(3)
OPEN_ALWAYS         ULONG(4)
FILE_FLAG_OVERLAPPED ULONG(40000000H)
FILE_FLAG_NOOVERLAPPED ULONG(0)
FILE_FLAG_NO_BUFFERING ULONG(20000000H)

LPERRORS            ULONG

CLRDTR              ULONG(6)
SETDTR              ULONG(5)
CLRRTS              ULONG(4)
SETRTS              ULONG(3)
SETBREAK            ULONG(8)
CLRBREAK            ULONG(9)
RESETDEV            ULONG(7)

ModemStatus         ULONG
MS_CTS_ON           ULONG(10h)
MS_DSR_ON           ULONG(20h)
MS_RING_ON          ULONG(40h)
MS_RLSD_ON          ULONG(80h)

EvtStatus           ULONG
EV_BREAK            ULONG(0040h)
EV_CTS              ULONG(0008h)
EV_CTSS             ULONG(0400h)
EV_DSR              ULONG(0010h)
EV_RING             ULONG(0100h)
EV_RLSD             ULONG(0020h)
EV_RXCHAR           ULONG(0001h)
EV_TXEMPTY          ULONG(0004h)

PURGE_TXABORT       ULONG(1)
PURGE_RXABORT       ULONG(2)
PURGE_TXCLEAR       ULONG(4)
PURGE_RXCLEAR       ULONG(8)

BytesToRead         LONG
!----------------------------


!-----------------------------------------
! Структура          DCB:
!-----------------------------------------
PortStruct          GROUP,PRE(PTS)
DCBlenght           ULONG(28)    ! Длина блока DCB(28 байт).
```

```
BaudRate        ULONG!(800h)  ! Скорость:115200-00020000h,
                              ! 57600-00040000h,
                              ! 9600-00000800h.

fbitFields      ULONG(1)      ! Битовая маска.

wReserved       USHORT        ! Резервный.
XonLim          USHORT(0)     ! Нет Xon.
XoffLim         USHORT(0)     ! Нет Xoff.
ByteSize        BYTE(8)       ! Длина слова 8 бит.
Parity          BYTE(0)       ! Нет паритета.
StopBits        BYTE(1)       ! Стоп-бит: 0-1,1-1.5,2-2.
XonChar         BYTE(0)       ! Символ Xon.
XoffChar        BYTE(0)       ! --//-- Xoff.
ErrorChar       BYTE(0)       ! --//-- Error.
EvtChar         BYTE(0)       ! --//-- Event.
wReserved1      USHORT        ! Резервный.
! Конец структуры.
!-----------------------------------------------

!-----------------------------------------------
! Структура COMSTAT:
!-----------------------------------------------
COMSTAT         GROUP,PRE(CST)
Status          ULONG
CbInQue         ULONG
CbOutQue        ULONG
! Конец структуры.
!-----------------------------------------------
MAP
!===============================================
! Прототипы для API-функций.
!-----------------------------------------------
 MODULE('Comm Prototipes')
!-----------------------------------------
   CreateFileA(*CSTRING,ULONG,ULONG,LONG,ULONG,ULONG,
UNSIGNED),|
   SIGNED,RAW,PASCAL,NAME('CreateFileA')
   BuildCommDCBA(*CSTRING,*STRING),SIGNED,RAW,
PASCAL,Name('BuildCommDCBA')
   SetCommState(SIGNED,*STRING),BOOL,RAW,PASCAL
   GetCommState(SIGNED,*STRING),BOOL,RAW,PASCAL
   TransmitCommChar(UNSIGNED,BYTE),UNSIGNED,PASCAL
   EscapeCommFunction(SHORT,ULONG),LONG,PASCAL
   CloseHandle(SIGNED),BOOL,PROC,PASCAL
   SetCommBreak(SHORT),SHORT,PASCAL
   ClearCommBreak(SHORT),SHORT,PASCAL
   GetCommModemStatus(SHORT,*ULONG),SIGNED,PASCAL
   SetCommMask(SHORT,ULONG),SHORT,PASCAL
   WaitCommEvent(UNSIGNED,*ULONG,LONG),SHORT,RAW,
PASCAL
   ReadFile(UNSIGNED,*BYTE,ULONG,*LONG,LONG),BOOL,
RAW,PASCAL,PROC
   WriteFile(UNSIGNED,*BYTE,ULONG,*LONG,LONG),BOOL,
RAW,PASCAL
   PurgeComm(UNSIGNED,ULONG),UNSIGNED,RAW,PASCAL
!Мой
   ClearCommError(UNSIGNED,*ULONG,*STRING),SHORT,
RAW,PASCAL
!-----------------------------------------------
   END
!---------------
   MODULE('DELAY')
   QueryPerformanceCounter(*STRING),BOOL,RAW,PASCAL
   QueryPerformanceFrequency(*STRING),BOOL,RAW,PASCAL
   Sleep(ULONG),PASCAL
   END
!===============================================
   END !от MAP

WinOut WINDOW('Тест порта RS232 (COM1:115200,N,8,2).
Ввод/вывод — API.'),AT(,,340,164),|
   ONT('MS Sans Serif',8,,FONT:bold),CENTER,CENTERED,
SYSTEM,GRAY,MAX
   BUTTON('Выход'),AT(52,128,50,21),USE(?Cancel),|
   FONT('Tahoma',10,COLOR:Black,FONT:bold,CHARSET:
CYRILLIC),DEFAULT
   IMAGE('CHECK4.ICO'),AT(204,128,18,21),USE(?Image1)
   BUTTON('Продолжить'),AT(224,128,69,21),USE(?Ok),|
   FONT('Tahoma',10,COLOR:Black,FONT:bold,CHARSET:
CYRILLIC),DEFAULT
   IMAGE('CANCEL4.ICO'),AT(32,128,19,21),USE(?Image2)
END
!-----------------------------------------------------------
WinIn WINDOW(' Ввод параметров порта'),AT(,,177,98),|
   FONT('MS Sans Serif',8,,FONT:bold,CHARSET:CYRILLIC),|
   CENTER,CENTERED,SYSTEM,GRAY,MAX,RESIZE
   MENUBAR
   END
   PROMPT('бод'),AT(132,42,18,15),USE(?Prompt1),|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC)
   OPTION('Скорость обмена'),AT(76,10,89,49),USE(SPEED),BOXED,|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC)
   RADIO('115200'),AT(88,22,39,19),|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC)
   RADIO(' 9600'),AT(88,38,34,19),|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC),
VALUE('9600')
   END
   PROMPT('бод'),AT(132,26,18,15),USE(?unnamed),|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC)
   IMAGE('CHECK4.ICO'),AT(24,74,18,22),USE(?Image5)
   BUTTON('Запуск'),AT(48,74,76,20),USE(?Button3),|
   FONT('Tahoma',11,,FONT:bold,CHARSET:CYRILLIC),DEFAULT
   OPTION('<185> Порта'),AT(12,10,53,49),USE(NP),BOXED,|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC)
   RADIO('COM1'),AT(20,22,32,18),USE(?unnamed:1),|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC),|
   TIP('COM1'),VALUE('COM1')
   RADIO('COM2'),AT(20,38,33,17),USE(?unnamed:2),|
   FONT('MS Sans Serif',10,,FONT:bold,CHARSET:CYRILLIC),|
   TIP('COM2'),VALUE('COM2')
   END
END

!-----------------------------------------------------------

   CODE

   NP='COM1'
   SPEED='115200'
   open(WinIn)
   p=0
   accept
   if accepted()=(?button3) then p=1; break.
   end
   if p=0 then return.
   !-----------------------------
! Вычисление временной задержки в 25 мкс.
!-----------------------------------
   loop until QueryPerformance       !Определение частоты
   Frequency(FREQUENCY).             !работы счетчика.
   DC=int(25*FRE:FREQL/1000000)      !Вычисление времени счета
                                     !в 25 мкс.
!---------------------------------------

!--------------------
   do INIT              !Инициализация RS-232 и микроконтроллера.

   loop until(SetCommMask(CommPort,EV_TXEMPTY)).
                        ! Установка маски на вывод байта

   close(WinIn)

   k=0
   OPEN(WinOut)
!---------------------------
   disable(?Ok)
   disable(?Cancel)
START
   !-----------------------------------------------------------------
S='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw
xyzАБВГДЕЖЗИКЛМНОПРСТУФХЦЧ'

   k=k+1
   show(5,30,'Начало передачи из PC')
   show(5,10,'')
   show(5,10,'k=')
   type(k)
   !------------------
   i=1
   C=M[1]
   M[1]=75
   do OUTBYTE                   !Передача мл. байта длины (75)
   M[1]=0
   do OUTBYTE                   !Передача ст. байта длины (0)
   !-------------------------
   M[1]=C                       !Передача 75 байт строки.
   loop i=1 to 75               !в микроконтроллер.
   do OUTBYTE
   .
   !-------------------------
   show(5,40,S)
   show(5,50,'Конец передачи из PC')
   show(5,60,'-----------------------------------------')
   !-----------------------------------------
   sleep(5) !Для 9600 дополнительно.
   sleep(5)
   !-----------------------------------------------------------------
   loop until(PurgeComm(CommPort,PURGE_RXCLEAR)).
                                !Очистка вх. буфера.
   !-----------------------------------------------------------------
   loop until(SetCommMask(CommPort,EV_RXCHAR)).
                                ! Установка маски
                                ! только(!) на прием байта.
   !-----------------------------------------------------------------
   show(5,70,'Начало передачи из микроконтроллера')
   S1=''
   !-----------------------------------------------------------------
   loop i=1 to 75                           !Прием 75 байт строки
   do INBYTE                                !из
   .                                        !микроконтроллера.
   !-----------------------------------------------------------------
   show(5,80,S1)
   show(5,90,'Конец передачи из микроконтроллера')
   show(5,100,'-----------------------------------------')
!-----------------------------------
   enable(?Ok)
   enable(?Cancel)
   p=0
   accept
!-------------------
   case accepted()
        of ?Ok
           !blank
           !display
        disable(?Ok)
        disable(?Cancel)
           p=1
           break
        of ?Cancel
           p=0
           break
   end!ot case
!-------------------
   end! От accept

   if p=0 then goto E.

   blank
   loop until(SetCommMask(CommPort,EV_TXEMPTY)).
                                ! Установка маски
   i=1
   M[1]=40h
   do OUTBYTE
   goto START

E
!-------------------------------
   loop until(EscapeCommFunction(CommPort,CLRDTR)).
                                !Сброс микроконтроллера.
   loop until(PurgeComm(CommPort,PURGE_RXCLEAR)).
                                !Очистка вх. буфера.
   loop until(CloseHandle(CommPort)). !Закрытие CommPort'a.
   close (WinOut)
RE
   Return

!-------------------------------------
!Конец программы
!-------------------------------------

!===============================================
!                ПОДПРОГРАММЫ
!-----------------------------------------------------------------

!-----------------------------------------
! П/п инициализации RS-232 и микроконтроллера.
!-----------------------------------------
INIT            routine
!-----------------------------------------
! Инициализация COM-порта
!-----------------------------------------
   PortCString=NP

   !PortCString='COM1'
   !PortCString='COM2'
   CommPort=CreateFileA(PortCString,ACCESS,0,NULL,OPEN_EX-
ISTING,|
   FILE_FLAG_NOOVERLAPPED,NULL)
                                !Открытие порта
   loop until(GetCommState(CommPort,PortStruct)).
                                !Получение статуса порта.
   ControlCString=NP & ':' & SPEED & ',N,8,2'
   loop until(BuildCommDCBA(ControlCString,PortStruct)).
                                ! Построение DCB.
   PTS:fbitFields=1             ! Битовая маска.
   loop until(SetCommState(CommPort,PortStruct)).
                                !Установка статуса порта.
!-----------------------------------------
! Инициализация микроконтроллера
!-----------------------------------------------------------------
   loop until(EscapeCommFunction(CommPort,CLRDTR)).
                                !Сброс микроконтроллера.
   Sleep(100)                   !Задержка 100 мс.
   loop until(ClearCommBreak(CommPort)).
                                !Сброс TxD — запрет передачи.
   loop until(EscapeCommFunction(CommPort,SETDTR)).
                                !Запуск микроконтроллера
   Sleep(100)
   loop until(PurgeComm(CommPort,PURGE_RXCLEAR)).
                                !Очистка входного буфера.
!-------------------------------------------------
! Конец инициализации
!-------------------------------------------------

!-----------------------------------------
! П/п ввода байта (байт в M1[i] )
```

```
!-----------------------------------------
INBYTE      routine
!-----------------------------------------------------------------------------
   loop until(SetCommBreak(CommPort)).
                                    !Установка TxD — разрешение
                                    !передачи.
   loop until WaitCommEvent(CommPort,EvtStatus,0).
                                    !Ожидание прихода
                                    !байта(EvtStatus=EV_RXCHAR).
   loop until(ClearCommBreak(CommPort)).
                                    !Сброс TxD — запрет передачи.
   loop until(ReadFile(CommPort,M1[i],1,N1,NULL)).
                                    !Чтение байта данных.
!-----------------------------------------
! П/п вывода байта (байт в M[i] )
!-----------------------------------------
OUTBYTE                             routine
!-----------------------------------------------------------------------------
Loop                                !Ожидание разрешения
   loop until (GetCommModemStatus(CommPort,ModemStatus)).
                                    !передачи,
   if BAND(ModemStatus,MS_DSR_ON) then break.
                                    !т.е. установки DSR
   .                                !(MS_DSR_ON).
   loop until(TransmitCommChar(CommPort,M[i])).
                                    !Вывод байта.
   loop until WaitCommEvent(CommPort,EvtStatus,NULL).
                                    !Ож-е конца вывода байта
   !(EvtStatus=EV_TXEMPTY).
!-------------------------------------------------!
DEL25 routine
   loop until QueryPerformanceCounter(COUNTER).
                                    !Чтение счетчика.
   CSTARTL=CNT:COUNTL               !Занесение начального
                                    !значения в
   CSTARTH=CNT:COUNTH               !две стартовых long-
                                    !переменных.
   Loop                             !
   loop until QueryPerformanceCounter(COUNTER).
                                    ! Ожидание
   if CNT:COUNTH=CSTARTH            !
   DELTAC=CNT:COUNTL-CSTARTL        ! прошествия
   Else                             !
   DELTAC=0ffffffffffffffffh-CSTARTL+CNT:COUNTL+1
                                    ! времени
   .                                !
   if DELTAC>DC then break.         ! DC(=25 мкс.)
   .                                !
!=============================================
DELAY35 routine
   loop 21000 times.
!-----------------------------------------------------------------------------

   Файл-проект Hello.prj
----------------------------------
-- Project File Title
#noedit
#system win32
#model clarion lib
#set RELEASE = on
#pragma debug(vid=>off)
#pragma optimize(cpu=>386)
#pragma optimize(speed=>on)
#pragma define(NULL=>off)
#compile «hello.clw»
#pragma link(«hello.lib»)
#pragma link(«WindowsShell.Manifest»)
#link «hello.exe»
```

Программа работает в ОС Win'98 SE2 и Win'XP SP2.

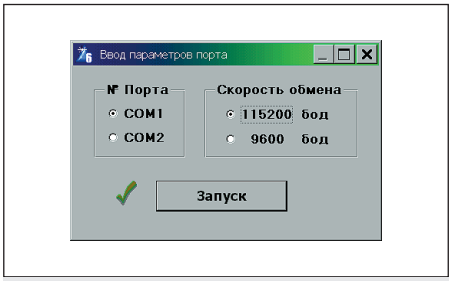

Рис. 6. Окно выбора параметров порта (Win'98)

При запуске в Win'98 на экране монитора появляется окно, показанное на рис. 6. Необходимо выбрать номер COM-порта, к которому подключено соответствующее устройство (или одна из макетных плат, описанных ранее). Соответствующая скорость обмена должна быть запрограммирована в микроконтроллер. Питание устройства (или макетной платы) должно быть включено. При выборе 1-го COM-порта и скорости обмена в 115 200 бод (как на рис. 6) и нажатии кнопки «Запуск» на экран выводится окно, показанное на рис. 7. Нажатие кнопки «Продолжить» приведет к повторному запуску программы передачи и приема строки. При этом количество перезапусков отражается переменной k, показанной в верхнем левом углу окна (k = 179, так как при большем количестве нажатий на кнопку «Продолжить» указательный палец начинает неметь :)). Если сравнить окно, показанное на рис. 7, с окном, показанным на рис. 5 (см. «КиТ» № 8`2006), то можно заметить их логическую (но не внешнюю) идентичность. Это не удивительно, поскольку программы выполняют одну и ту же функцию, но в разных ОС (DOS и Win'98).

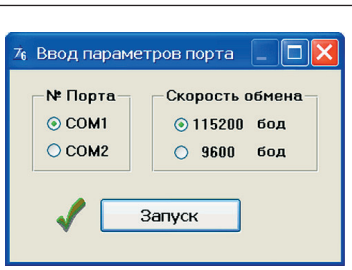

Рис. 8. Окно выбора параметров порта (Win'XP)

При запуске программы в Win'XP на экране монитора появляется окно, показанное на рис. 8. После выбора необходимой скорости обмена, номера COM-порта и нажатии кнопки «Запуск» на экран выводится окно, показанное на рис. 9.

Продолжение следует

Литература

1. Баррингтон Брюс Б. Как создавался Кларион // Мир ПК. 1993. № 2.
2. Кузьминов А. Ю. Интерфейс RS-232. Связь между компьютером и микроконтроллером. От DOS к Windows98/XP М.: ДМК-ПРЕСС, 2006 (в печати).
3. Кузьминов А. Ю. Интерфейс RS-232. Связь между компьютером и микроконтроллером. М.: Радио и связь, 2004.
4. Кузьминов А. Ю. Однокристальные микроЭВМ — основа удаленных систем сбора и обработки сигналов, поступающих с датчиков // Электроника и компоненты. 1998. № 2
5. Кузьминов А. Ю. Новые MCS51 — совместимые микроконтроллеры и их применение в системах сбора информации с датчиков // Контрольно-измерительные приборы и системы. 1997. № 6. 1998. № 7.
6. Кузьминов А. Ю. Удаленные системы сбора информации с датчиков на базе однокристальных микроЭВМ // Автоматизация и производство. 1996. № 3.
7. Кузьминов А. Ю. Универсальная система сбора и обработки данных АСИР-3 // Мир ПК. 1996. № 6.
8. Орлов А. Два звучных слова — Clarion и Delphi // Мир ПК. 1996. № 6.
9. Фролов А. В., Фролов Г. В. Программирование модемов. М.: ДИАЛОГ-МИФИ, 1993.
10. www.analog.com
11. www.atmel.com
12. www.maxim-ic.com
13. www.semiconductor-philips.com
14. www.silabs.com
15. www.ti.com
16. www.msdn.microsoft.com/library
17. www.gapdev.com
18. www.sysinternal.com

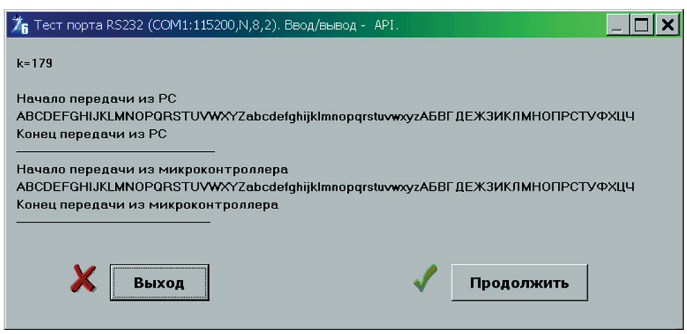

Рис. 7. Основное окно работы программы (Win'98)

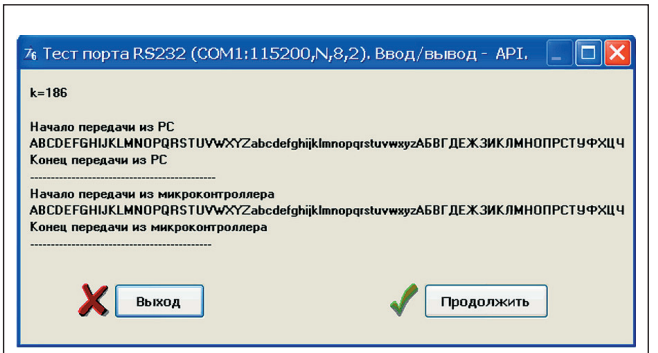

Рис. 9. Основное окно работы программы (Win'XP)