Усовершенствованный протокол обмена по интерфейсу RS-232

Алексей Кузьминов (Москва)

В статье описаны программные средства усовершенствованного протокола обмена информацией компьютера с микроконтроллером по интерфейсу RS-232 с побайтной синхронизацией. Обращение к низкоуровневым командам процессора в подключаемых подпрограммах позволяет снизить уровень ошибок.

Введение

Протокол обмена информацией компьютера с микроконтроллером по интерфейсу RS-232 с аппаратной побайтной синхронизацией линиями данных [1] успешно используется в компьютерных системах сбора и обработки информации [2, 3]. Однако возросшие скоростные характеристики как компьютеров, так и микроконтроллеров потребовали усовершенствования этого протокола. При этом программные средства, предназначенные для микроконтроллеров, остались прежними.

Аппаратные средства, используемые для обмена, описаны в [2]. В предлагаемой статье предложены программные средства, предназначенные для компьютера. Во-первых, это программирование аппаратно независимой временной задержки длительностью около 25 мкс, которая позволяет увеличить стабильность передачи и практически убрать «дрожание» фронтов сигнала. Во-вторых, это использование низкоуровневых команд процессора компьютера для всего процесса ввода/вывода. Если раньше подпрограммы ввода/вывода байта были написаны на языке высокого уровня (Clarion v.6.0) и каждое обращение к командам прямого ввода из порта компьютера (in) и вывода в порт (out) было организовано с помощью тройной цепочки Clarion → внешняя подпрограмма на C++(external) → внутренняя подпрограмма на C++ (static) → команда процессора ввода/вывода из порта (in/out), то сейчас эти подпрограммы полностью перенесены в низкоуровневые команды процессора и весь процесс ввода/вывода байта (т.е. анализ состояния портов, временные задержки, прямой ввод/вывод и т.п.) организован непосредственно в кодах процессора компьютера. В связи с тем что низкоуровневые команды процессора выполняются на порядок быстрее операторов языка высокого уровня Clarion, надёжность обмена возрастает.

Протокол обмена компьютера с микроконтроллером

Стандартный протокол обмена по интерфейсу RS-232 с аппаратной синхронизацией каждого байта линиями квитирования DTR-DSR и RTS-CTS хорошо известен. Однако при сопряжении компьютера с микроконтроллером выходные линии квитирования (DTR и RTS) интерфейса RS-232 не могут быть использованы по прямому назначению, т.к. в этом случае они применяются для других целей. Напомним, что линия DTR предназначена для сброса микроконтроллера, а линия RTS – для его перевода в режим программирования в системе (In-System-Programming, ISP). В зависимости от типа микроконтроллера, линия RTS также управляет определённым выводом микроконтроллера, переводя его в режим ISP.

Когда осуществляется передача информации из компьютера в микроконтроллер, используются выходная линия TxD компьютера и входная линия RxD микроконтроллера, а входная линия RxD компьютера и выходная линия TxD микроконтроллера не используются. Суть модернизации протокола состоит в том, чтобы использовать эти линии для аппаратной побайтной синхронизации, аналогично линиям DTR-DSR или RTS-CTS.

В стандартном протоколе обмена, если, например, аппаратная синхронизация осуществляется линиями квитирования DTR-DSR (они соединены между собой), то линия DTR приёмника разрешает или запрещает передачу байта передатчиком, а линия DSR передатчика используется для анализа состояния линии DTR приёмника. Если состояние линии DTR приёмника разрешающее, то передатчик выводит информационный байт, в противном случае – не выводит байт до тех пор, пока на линии DTR приёмника не установится разрешающее состояние.

В модернизированном протоколе аппаратная синхронизация осуществляется аналогичным образом, только вместо линии DTR приёмника, разрешающей или запрещающей передачу байта передатчиком, используется его линия TxD, которая при приёме байта не задействована. Линия TxD приёмника, так же как и линия DTR (в стандартном протоколе), может быть переведена в любое состояние (разрешающее или запрещающее). В то же время линия RxD передатчика, которая также не задействована при передаче байта, может быть использована для анализа состояния линии TxD приёмника (аналогично линии DSR в стандартном протоколе обмена), т.к. линии TxD приёмника и RxD передатчика также соединены вместе.

На рисунке 1 показана временная диаграмма передачи конкретного байта – символа 'Q', имеющего ASCII-код, равный 51h или 01010001b. Как видно из рисунка 1, в интерфейсе RS-232 принят порядок передачи байта младшим битом вперёд (LSB-First). При передаче кода 01010001b вначале передаётся старт-бит (= 0), младший бит 1(1), затем идут три следующих бита (2(0), 3(0) и (0)), далее бит 5(1), бит 6(0), бит 7(1) и, наконец, бит 8(0). После этого следует стоп-бит (1), а если их два – то ещё один стоп-бит (1). Причём единичному состоянию (1) соответствует низкий уровень сигнала (–10 В), а нулевому (0) – высокий (+10 В). Поэтому практически все преобразователи интерфейса RS-232 ↔ TTL, помимо преобразования уровней, инвертируют сигналы.

В предложенной схеме передатчик (в данном случае компьютер) ждёт разрешения передачи байта от микроконтроллера, анализируя свою линию RxD (DSR). Как только такое разрешение получено, через время реакции

$T_{рр}$ он начинает передавать байт. Приёмник (микроконтроллер), анализируя свою линию RxD (TxD компьютера) и получив старт-бит, через время $T_{рн}$ запускает подпрограмму задержки на 25 мкс, после чего приёмник сбрасывает разрешение (т.е. устанавливает запрет на передачу следующего байта). Когда байт полностью выведен, через время $T_{рк}$ передатчик (компьютер) вновь приступает к анализу своей линии DSR на предмет разрешения или запрещения передачи следующего байта, и процесс повторяется.

Реальные осциллограммы передачи байта, снятые двухканальным цифровым осциллографом и соответствующие временной диаграмме на рисунке 1, приведены на рисунке 2. Сравнив реальную осциллограмму с временной диаграммой, можно убедиться, что они логически одинаковы.

На рисунке 3 приведены реальные осциллограммы приёма байта компьютером. Из сравнения рисунков 2 и 3 следует, что функциональное назначение линий TxD и RxD при передаче и приёме изменяется на противоположное, а логика работы аппаратной синхронизации остаётся прежней.

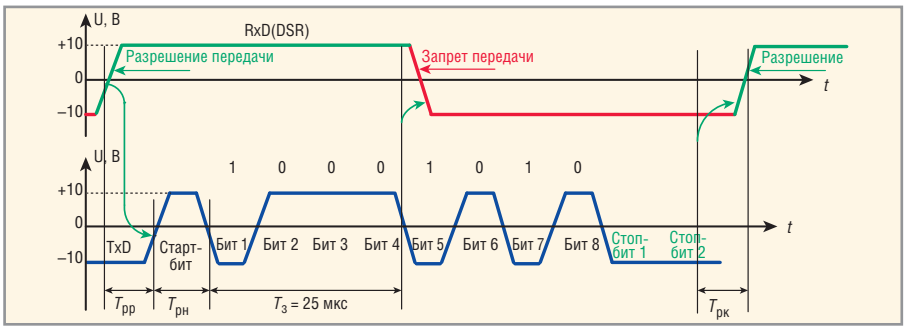

Рис. 1. Временная диаграмма передачи байта – символа 'Q' (уровни RS-232)

$T_{рр}$ – время реакции на разрешение передачи, $T_{рн}$ – время реакции на начало передачи, $T_з$ – время задержки, $T_{рк}$ – время реакции на конец передачи

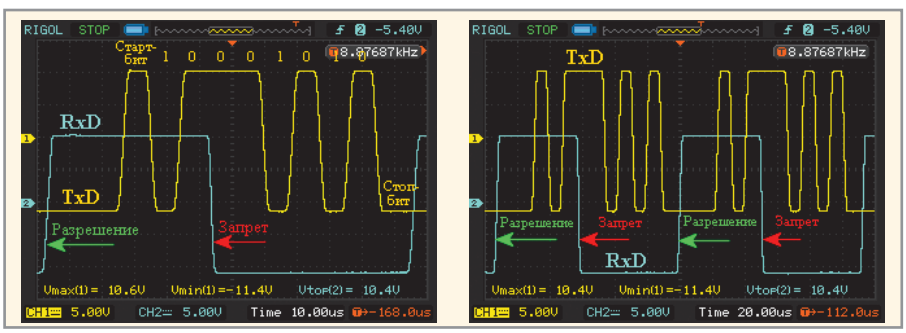

Рис. 2. Осциллограмма передачи байта – символа 'Q' (уровни RS-232)

а) развёртка 10 мкс/дел., б) развёртка 20 мкс/дел.

Необходимо отметить, что достоверность обмена может быть значительно повышена, если передатчик сделает дополнительный анализ линии RxD на сброс разрешения. Как видно из временной диаграммы на рисунке 1, сброс

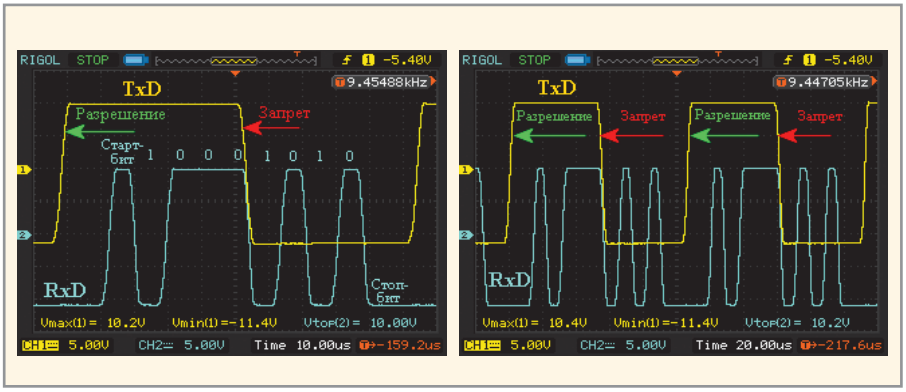

Рис. 3. Осциллограмма приёма байта – символа 'Q' (уровни RS-232)
а) развёртка 10 мкс/дел., б) развёртка 20 мкс/дел.

разрешения (т.е. установка приёмником линии RxD передатчика в низкий уровень) происходит где-то в середине интервала передачи байта. Если передатчик проверит свою линию RxD на предмет сброса разрешения (т.е. её переключения в низкий уровень), то этот факт для передатчика будет означать, что приёмник начал принимать передаваемый байт и поэтому сбросил разрешение. Другими словами, дополнительная проверка сброса разрешения позволяет передатчику определить, что приёмник работает правильно и переключает линию RxD передатчика из высокого уровня в низкий (и обратно).

Следует отметить, что во время передачи байта, т.е. от начала старт-бита и до конца последнего стоп-бита, ни процессор передатчика, ни процессор приёмника не принимают никакого участия в непосредственном обмене данными, поскольку и в компьютере, и в микроконтроллере этим занимается UART, т.е. аппаратные средства. Поэтому все проверки состояния линий, переключения их из одного состояния в другое, организация задержек и т.п. не оказывают влияния на скорость передачи. Например, задержка в 25 мкс и проверка передатчиком сброса разрешения проводятся во время передачи и приёма байта и не замедляют обмен.

Если посмотреть ещё раз на временную диаграмму и осциллограммы (рис. 1 и 2 соответственно), то можно заметить, что график зависимости напряжения от времени на линии RxD представляет собой почти симметричный меандр. Длительность импульса составляет примерно половину времени передачи одного байта (при скорости обмена в 115 200 бод частота меандра составляет около 9 кГц – реальное значение частоты отражено в правом верхнем углу рисунка 2 «8.87687kHz» и рисунка 3 «9.44705kHz»). Такой импульс легко воспринимается преобразователем интерфейса.

Теперь несколько слов о том, каким образом компьютер может анализировать линию разрешения. Дело в том, что в интерфейсе RS-232 компьютера непосредственное чтение и анализ состояния самой линии RxD (т.е. лог. 1 или лог. 0) не предусмотрены. Но если соединить эту линию RxD с какой-либо входной линией квитирования (например, DSR), состояние которой поддаётся чтению, можно считывать состояния линий DSR и RxD. Когда же линия RxD используется для передачи данных, состояние линии DSR не требуется считывать и анализировать. На рисунке 1 это подчёркнуто обозначением RxD(DSR).

Формирование машиннонезависимых временных задержек

Из рисунка 1 видно, что микроконтроллер формирует временную задержку, равную 25 мкс. Программирование такой временной задержки в микроконтроллере не представляет труда, поскольку всегда известна его тактовая частота.

В компьютере для формирования короткой, аппаратно независимой задержки в Windows 98/XP ранее [1] автор использовал так называемый «счётчик производительности» – QueryPerformanceCounter, который постоянно работает с определённой частотой *F* – QueryPerformanceFrequency. Эта частота различается для ОС Windows 98 и Windows XP и составляет около 2 и 7 МГц соответственно (для компьютера на базе Intel P4/1,7 ГГц). Частота счёта жёстко привязана к временной базе системного таймера. Для получения необходимой временной задержки точное значение частоты *F* выяснять не обязательно – достаточно разделить эту частоту на 1 000 000, чтобы получить временную базу для счётчика в 1 мкс. Умножив это значение, например, на 25 и заставив счётчик считать до этого значения, можно получить аппаратнонезависимую временную задержку в 25 мкс.

Для получения временной задержки автор применил более простой способ. Он заключается в использовании достаточно стабильного по времени выполнения команды ввода из порта RS-232, соответствующего регистру состояния модема (адрес порта 3feh). С этой целью на языке Clarion была написана нижеприведённая подпрограмма задержки DEL, использующая ввод с порта 3feh (который часто требуется анализировать):

```
!---------------------
! Подпрограмма задержки
!---------------------
DEL routine
loop 15 times
!times=15,Тзад.=20мкс
|times=20,Тзад.=26.6мкс
A=INP(3feh)
!times=75,Тзад.=100мкс
|26.6/8.68≈3бита(115200 бод)
.
!------------------------
```

Вставив в программу меандра подпрограмму DEL, автор проверил импульсы осциллографом и был обрадован стабильностью их длительности и чистотой фронтов. Подобный эксперимент был проведён, помимо Windows XP, и в Windows 98, и даже в DOS.

Для точной оценки длительности автором была написана программа на турбо-бейсике DOS (TB.EXE фирмы Borland), mtimer.bas, предназначенная для определения числа *N*, записав которое в регистр cx процессора компьютера (счётчик цикла), можно получить задержку длительностью в 1 мс. Хотя операторы бейсика и не отличаются высокой скоростью выполнения, в TB есть полезный оператор/функция – микротаймер (mtimer), использующий аппаратный таймер компьютера и позволяющий оценить время выполнения какой-либо программы или подпрограммы в микросекундах. Как оператор, mtimer запускает аппаратный таймер компьютера, как

функция – останавливает его работу и возвращает время работы в микросекундах.

В качестве подпрограммы, время работы которой требовалось оценить, использовалась подключаемая подпрограмма 1msec.asm в *.com-формате (1msec.com), написанная на ассемблере. В этой подпрограмме производится ввод содержимого порта состояния COM-порта (3feh) в аккумулятор командой in 800 раз, для чего в регистр cx (счётчик цикла) заносится число 800 и осуществляется 800-кратный цикл (оператором loop).

Для получения подпрограммы 1msec.com использовался транслятор ассемблера TASM.EXE и линкер TLINK.EXE фирмы Borland. Для трансляции и получения *.com-файла необходимо выполнить две командные строки:

```
TASM.EXE 1msec.asm
TLINK 1msec.obj /t
```

Программа mtimer.bas запускает таймер, вызывает подпрограмму MSEC (являющуюся подключаемой подпрограммой 1msec.com), после её завершения останавливает таймер, вычисляет время выполнения в микросекундах и выводит результат на экран. Ниже приведены тексты обеих программ.

Программа mtimer.bas

```
sub MSEC inline
$inline "1msec.com"
end sub
'------
mtimer
call MSEC
T%=mtimer
cls
N%=fix(800/T%*1000)
print""
print " N=";N%;" T=";T%;"мкс"
print using " 1 цикл= #.##
мкс";T%/800
print " Для t=25 мкс, cx=";
print using "##";int(25/(T%/800))
```

Программа 1msec.asm

```
CSEG segment ;Сегмент кода
assume
cs:CSEG,ds:CSEG,es:CSEG,ss:CSEG
org 100h
START: push bp ; сохранение ре-
гистров
mov bp,sp
push cx
;-------------
mov dx,3feh
mov cx,800
in al,dx
loop $-1
;-------------
pop cx
pop bp
CSEG ends
end START
```

После запуска программы mtimer.exe (или при использовании опции RUN в самом турбо-бейсике) на экран выводится следующее окно (см. рис. 4). Необходимо отметить, что для выполнения программы mtimer.exe в Windows XP в режиме эмуляции DOS, необходимо перед её запуском запустить программу portmon.exe и провести с ней необходимые манипуляции [1], чтобы разрешить прямой ввод/вывод в COM-порт.

Из рисунка 4, в частности, следует, что команда процессора ввода байта из COM-порта компьютера (in) выполняется за время 1,25 мкс и что для получения временной задержки в 25 мкс в регистр cx необходимо загрузить число 20.

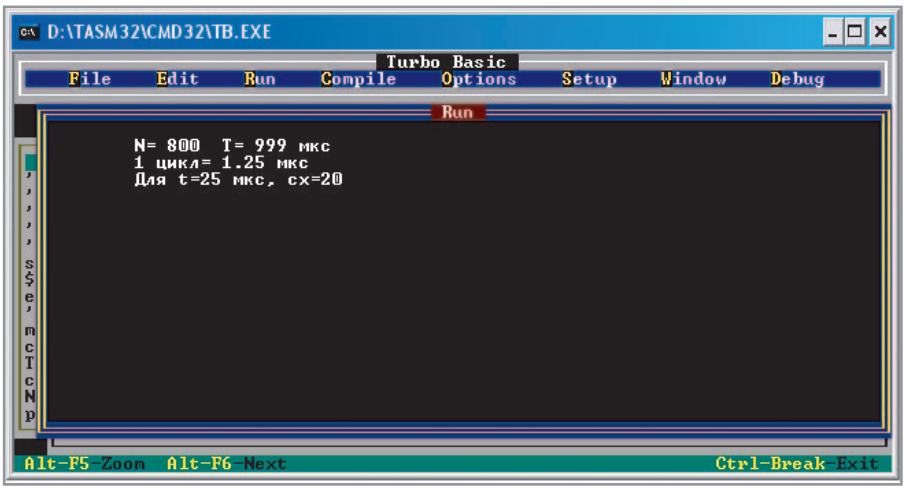

Рис. 4. Окно RUN турбо-бейсика при запуске программы mtimer.exe

Программа mtimer.exe запускалась и в DOS, и в Windows 98/XP (в режиме эмуляции DOS), и везде давала практически одинаковый результат. Это позволяет использовать команду процессора для ввода байта из COM-порта как временную базу для организации коротких, аппаратно независимых временных задержек длительностью от нескольких микросекунд до десятков микросекунд. Получение бóльших задержек не вызывает затруднения, поскольку для них в MSDN Library существует функция sleep(), аргументом которой является число миллисекунд (эта функция использована в тестовой программе testRS232.clw – см. ниже).

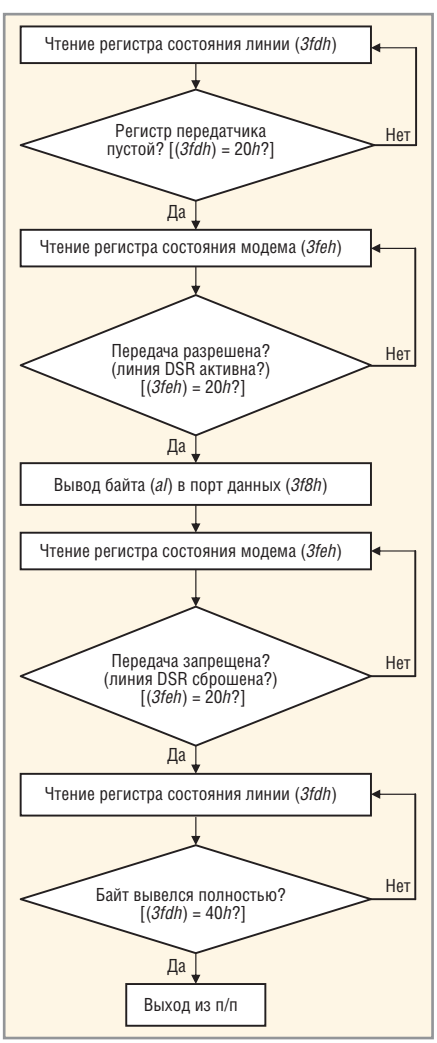

Рис. 5. Блок-схема подпрограммы синхронного вывода байта из компьютера

Подпрограммы ввода/вывода байта для компьютера

В соответствии с временной диаграммой, показанной на рисунке 1, на рисунке 5 приведена блок-схема протокола передачи байта из компьютера с побайтной аппаратной синхронизацией линиями данных. В соответствии с этой блок-схемой в [1] использована подпрограмма вывода байта на языке Clarion v.6.0, которая приведена ниже:

```
!--------------------------------
! Подпрограмма вывода байта (байт
в переменной OUTB)
!--------------------------------
OUTBYTE routine
loop until band(INP(3fdh),20h).
!Ожидание готовности передатчика
(transmitter empty).
loop until band(INP(3feh),20h).
!Ожидание разрешения передачи -
установки DSR.
B=INP(3feh) !Очистка 3feh для
сброса бита DeltaDSR(02h).
OUTP(3f8h,OUTB) !Вывод байта.
loop until band(INP(3feh),02h).
!Ожидание запрета передачи - из-
менения состояния DSR.
loop until band(INP(3fdh),40h).
!Ожидание выхода байта из PC -
(OK to send).
!--------------------------------
```

В этой подпрограмме, в свою очередь, используются две подпрограммы прямого ввода и вывода в порты INP() и OUTP(), написанные на встроенном в Clarion языке C++ (In.cpp и Out.cpp). Эти две подпрограммы также приведены ниже.

```
Подпрограмма In.cpp.
#pragma save
#pragma call(inline => on,
reg_param => (dx), reg_return =>
(ax))
//-------------------------------
static unsigned char inportb(un-
signed int Port_number)=
{
0xEC, // in al,dx
};
#pragma restore
extern "C" unsigned char INP(un-
signed int Port_number)
{
unsigned char byte;
byte=inportb(Port_number);
return byte;
}
```

Вызов этой подпрограммы, в частности, осуществляется из вышеприведённой подпрограммы (routine) вывода байта OUTBYTE (Clarion v.6.0) командой INP().

```
Подпрограмма Out.cpp.
#pragma save
#pragma call(inline => on,
reg_param => (dx,ax))
//-------------------------------
static void outportb(unsigned int
port, unsigned char byt)=
{
0xEE, // out dx,al
};
#pragma restore
extern "C" void OUTP(unsigned int
Port_Number, unsigned char byte)
{
outportb(Port_Number,byte);
}
```

Вызов этой подпрограммы также осуществляется из вышеприведённой подпрограммы (routine) вывода байта OUTBYTE командой OUTP().

Если ещё раз взглянуть на подпрограмму вывода байта OUTBYTE, написанную на Clarion v.6.0, то можно заметить, что вызов подпрограммы INP() производится 1-м, 2-м, 3-м, 5-м и 6-м операторами, причём 1-й, 2-й, 5-й и 6-й – это операторы цикла, где обращение к подпрограмме INP() происходит многократно. Вызов подпро-

граммы OUTP() производится 4-м оператором.

Рассмотрим подробнее саму процедуру вызова, например, в 1-м операторе loop until band(INP(3fdh),20h). В самом начале идёт обращение к подпрограмме, которая в программе In.cpp имеет название INP() и атрибут «внешний» (extern "C"):

```
extern "C" unsigned char INP(un-
signed int Port_number)
```

В эту подпрограмму из Clarion передаётся значение адреса порта (3fdh) в переменную Port_number. Далее подпрограмма INP() вызывает подпрограмму inportb(Port_number), имеющую атрибут static:

```
static unsigned char inportb(un-
signed int Port_number)=
{
0xEC, // in al,dx
};
```

Значение адреса порта (3fdh) из подпрограммы INP() ещё раз передаётся в подпрограмму inportb() в процессорном регистре dx с помощью директивы reg_param => (dx). Далее выполняется подключаемая подпрограмма (уже третья), имеющая машинный (объектный) код 0xEC, который является кодом операции ввода in в регистр al содержимого адреса порта dx. Результат ввода возвращается в регистр ax (точнее, в al) директивой reg_return => (ax). Этот результат вначале передаётся из подпрограммы inportb() в подпрограмму INP(), а оттуда уже в подпрограмму OUTBYTE. Все эти действия производятся ради выполнения единственной машинной команды ввода из порта in al,dx с объектным кодом 0xEC, отсутствующей в языке Clarion.

Если бы подпрограмма INP() выполнялась только один раз, то такой «тройной» вызов был бы оправдан, но обращение к этой подпрограмме из подпрограммы OUTBYTE происходит многократно. Это же касается и подпрограммы OUTP(). И многократно всё описанное повторяется сотни и тысячи раз в программах, использующих подпрограмму OUTBYTE.

Если процессор может выполнить машинные команды ввода из порта (0xEC) и вывода в порт (0xEE), то почему бы его не заставить выполнить сразу всю подпрограмму (в машинных кодах) вывода байта? В этом случае вызов такой подпрограммы будет производиться только один раз из Clarion, а все операции ввода/вывода, анализ состояния портов и т.п. перейдут в подпрограмму в машинных кодах. Кроме того, выполнение готовой кодовой последовательности команд в процессоре происходит на порядок быстрее.

Подпрограмма вывода байта на встроенном в Clarion v.6.0 языке TopSpeed C++ (Outbasm.cpp), использующая подключаемую вставку в машинных кодах, приведена на сайте журнала.

Сравнив комментарии к этой подпрограмме вывода байта в ассемблерной мнемонике с подпрограммой OUTBYTE() на языке Clarion, а также с вышеприведённой блок-схемой (см. рис. 5), можно заметить, что они логически одинаковы. Вызов подпрограммы OUTBASM (unsigned char byte) из Clarion производится следующим образом:

```
OUTASM(OUTB)
```

В качестве параметра в вызываемую подпрограмму необходимо подставить выводимый байт в переменную OUTB. Внешняя подпрограмма OUTBASM вызывает статическую подпрограмму outpasm (unsigned char byt), в которую в качестве параметра передаёт значение выводимого байта (byt) через регистр bx (точнее, через его младший байт bl) с помощью директивы reg_param => (bx).

Чтобы вся конструкция работала, для неё необходимо создать прототип:

```
MODULE('IN/OUT.CPP')
OUTASM(BYTE),NAME('_OUTBASM')
END
```

Кроме того, в файл-проект необходимо включить подпрограмму Outbasm.cpp.

Теперь рассмотрим подпрограмму ввода байта в компьютере. Соответствующая блок-схема, отражающая осциллограммы (см. рис. 3), показана на рисунке 6. Подпрограмма ввода байта на встроенном в Clarion v.6.0 языке TopSpeed C++ (Inbasm.cpp), использующая подключаемую вставку в машинных кодах, приведена на сайте журнала.

Если сравнить комментарии к кодовой части подключаемой подпрограммы inbtasm() с вышеприведённой блок-схемой (см. рис. 6), то можно заметить их полную логическую идентичность.

Вызов внешней подпрограммы extern "C" unsigned char INBASM() из Clarion производится следующим образом:

```
B=INASM()
```

В переменной B – результат ввода байта. Результат ввода байта возвращается в подпрограмму INBASM регистре ax (точнее, al) с помощью прагмы reg_return => (ax), а из подпрограммы INBASM – уже в Clarion в переменную B.

Рис. 6. Блок-схема подпрограммы синхронного ввода байта в компьютер

Чтобы вся конструкция работала, для неё необходимо создать прототип:

```
MODULE('IN/OUT.CPP')
INASM(),BYTE,NAME('_INBASM')
END
```

Кроме того, в файл-проект необходимо включить подпрограмму Inbasm.cpp.

Тестовая программа ввода/вывода строки байтов для компьютера

Суть этой тестовой программы проста: из компьютера в микроконтроллер передаётся строка, состоящая из 75 байтов (символов), которая принимается микроконтроллером, затем им же модифицируется (изменяется прядок следования символов на обратный), затем эта модифицированная строка передаётся микроконтроллером в компьютер, принимается компьютером и выводится на экран для сравнения с исходной строкой (которая предварительно выводится на экран).

Ядром программы являются следующие операторы:

```
loop i=1 to 75
OUTASM(M[i]) !Вывод 75 байт в
микроконтроллер
.
!------- В этом месте идет пере-
ключение с вывода на ввод.-------
B=INP(3f8h) !Холостой ввод для
сброса бита DR-data ready.
loop while band(INP(3fdh),1).
!Ожидание сброса бита "DR"-data
ready.
Sleep(2) ! Задержка 2 мсек.
!-------------------------------
loop i=1 to 75
M1[i]=INASM() !Ввод 75 байт в
компьютер.
.
```

В остальном тексте программы определяются различные переменные, прототипы функций, открытие порта RS-232 и его инициализация, инициализация микроконтроллера, открытие окна, в котором отражается результат работы программы (см. рис. 7), и т.п. Полный текст программы testRS232.clw вместе с файл-проектом приведен на интернет-странице журнала (www.soel.ru).

Что касается аппаратных средств, то они подробно описаны в [2]. Конкретно эта программа проверялась с микроконтроллером P89LPC982.

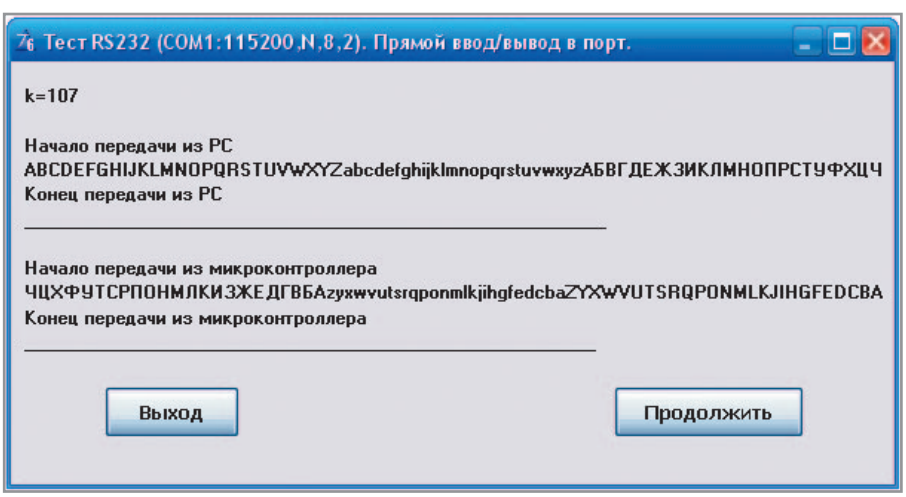

Рис. 7. Снимок с экрана, отображающий работу программы testRS232.exe

В левом верхнем углу рисунка 7 отображается количество передач (или количество нажатий на кнопку «Продолжить» – в данном случае автор нажал на неё 107 раз).

Прежде чем запускать эту программу в Windows XP, требуется произвести настройку COM-порта, воспользовавшись панелью управления. Свойства COM-порта, которые следует настроить, – отсутствие управления потоком и отсутствие буферов [1]. Кроме того, для прямого ввода/вывода в этот порт необходимо установить на компьютере один из драйверов, которые разрешают производить такой ввод/вывод, например, программу USERPORT (UserPort.sys), и вписать адреса портов для прямого ввода/вывода.

Получение машинных кодов для подключаемых подпрограмм

Довольно давно существует упоминавшийся выше компилятор языка Бейсик – турбо-бейсик (TB.EXE v.1.0 от Borland International), работающий в DOS и позволяющий использовать подключаемую подпрограмму, но не в машинных кодах, а непосредственно как объектный файл в *.com-формате. Сопряжение такой подпрограммы с самим языком Бейсик достаточно простое и описано в руководстве по его использованию. Чтобы получить файл подпрограммы на ассемблере в *.com-формате, можно воспользоваться транслятором с ассемблера, например, турбо-ассемблером TASM той же фирмы Borland (TASM – Turbo Assembler v.4.1 (или v.1.0) Borland International, TLINK – Turbo Link v.7.1.30.1 (или v.2.0) Borland International).

Какая программа должна быть написана на турбо-бейсике и какие подключаемые подпрограммы она должна содержать, чтобы решить поставленную задачу? Проще всего, если программа на TB будет выполнять те же действия, что и программа testRS232.clw (ввод/вывод строки байтов по RS-232 в микроконтроллер), приведённая выше на языке Clarion, и в качестве подключаемой подпрограммы в *.com-формате будут использоваться подпрограммы приёма. Тогда с помощью основной программы на TB можно проверять их работоспособность, выводя сообщения на монитор.

Далее будут приведены программа на турбо-бейсике (TESTRS.BAS) и две подключаемые подпрограммы на ассемблере – p86wr.asm (вывод строки байтов по RS-232 из компьютера в микроконтроллер) и p86rd.asm (приём строки байтов по RS-232 из микроконтроллера). Программа TESTRS.BAS содержит оттранслированные программы p86wr.com и p86rd.com как подключаемые подпрограммы в *.com-формате. С текстами этих программ и *.bat-файлами, предназначенными для их трансляции с целью получения *.com- и *.lst-форматов этих программ, можно ознакомиться на интернет-странице журнала.

После запуска оттранслированной программы TESTRS.EXE на экран монитора выводится окно, показанное на рисунке 8. Перед запуском этой программы следует запустить программу portmon.exe и провести с ней необходимые манипуляции [1], чтобы разрешить прямой ввод/вывод в COM-порт. Сравнив окна, показанные на рисунках 7 и 8, можно сделать вывод, что программа TESTRS.BAS работает аналогично программе testRS232.clw.

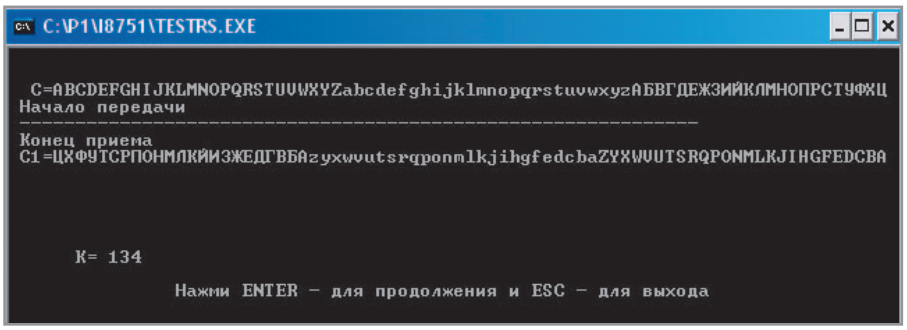

Рис. 8. Результат работы программы TESTRS.EXE

Проанализируем файл-листинг подпрограммы p86rd.asm – файл P86RD.LST в той его части, которая касается макроса INBYTE. Ниже приведён фрагмент листинга программы с этим макросом:

```
62 00000124 INBYTE ;в al находится считанный байт
1 63 00000124 B6 03 mov dh,03h
1 64 00000126 B2 FE mov dl,0feh ; Итого: dx=3feh
1 65 00000128 EC in al,dx ; Очистка регистра 3feh
1 66 00000129 B2 FB mov dl,0fbh ; Итого: dx=3fbh.
1 67 0000012B B0 47 mov al,47h ; Установка линии TxD
1 68 0000012D EE out dx,al ; (разрешение передачи байта).
1 69 0000012E B2 FE mov dl,0feh ; Итого: dx=3feh
1 70 00000130 EC in al,dx ; Ожидание
1 71 00000131 A8 02 test al,02h ; старт-бита
1 72 00000133 74 FB jz $-3 ;
1 73 00000135 33 C9 xor ecx,ecx ;32-разрядный режим.
1 74 00000137 B1 14 mov cl,20 ;В cx/ecx число, определяющее задержку.
1 75 00000139 EC in al,dx
1 76 0000013A E2 FD loop $-1
1 77 0000013C B2 FB mov dl,0fbh ; Итого: dx=3fbh.
1 78 0000013E B0 07 mov al,07h ;
```

```
Сброс линии TxD-
1 79 00000140 EE out dx,al ; запрет передачи.
1 80 00000141 B2 FD mov dl,0fdh ; Итого: dx=3fdh
1 81 00000143 EC in al,dx ; Проверка факта
1 82 00000144 A8 01 test al,01h ; прихода
1 83 00000146 74 FB jz $-3 ; байта.
1 84 00000148 B2 F8 mov dl,0f8h ; Итого: dx=3f8h
1 85 0000014A EC in al,dx ; Ввод байта из порта данных (3f8h) в al.
```

Теперь просмотрим файл p86rd каким-либо файловым менеджером (Frigate, NortonCommander, Total Commander и т.п.) в HEX-формате:

```
00000000: 66 55 66 8B EC 06 1E
66|67 C4 7E 06 66 67 8B 16 |
00000010: 00 00 8E DA 66 26 67
8B|75 02 66 26 67 8B 0D 66 |
00000020: 81 E1 FF 7F B6 03 B2
FE|EC B2 FB B0 47 EE B2 FE |
00000030: EC A8 02 74 FB 33 C9 B1|14
EC E2 FD B2 FB B0 07
00000040: EE B2 FD EC A8 01 74 FB|B2
F8 EC 67 88 04 66 46 |
00000050: E2 D2 1F 07 66 5D | |
```

Видно, что в подпрограмме p86rd.com присутствует кодовая последовательность (выделена шрифтом), которая в точности совпадает с кодами выделенных операций в файле-листинге. Эта кодовая последовательность и представляет собой необходимые коды для вставки в подпрограмму на C++ Inbasm.cpp. Если теперь текстовым редактором убрать весь текст слева от выделенного в файле-листинге, снабдить каждый код операции символом «0x» и запятыми, а текст справа от выделенного отделить знаками комментариев «//», то мы получим код подключаемой подпрограммы для Inbasm.cpp.

Фрагмент файла-листинга P86WR.LST (подпрограмма p86wr.asm) и сама подпрограмма p86wr.com в HEX-формате приведены на сайте журнала.

В подпрограмме p86wr.com кодовая последовательность, соответствующая кодовой последовательности в файле-листинге, встречается три раза (выделено). Поскольку вывод байта (OUTBYTE) является макросом, а не подпрограммой, первые два обращения к этому макросу соответствуют выводу младшего и старшего байтов длины строки, а третье – выводу самой строки. Кодовая последовательность файла-листинга и является подключаемой вставкой в подпрограмму Outbasm.cpp.

Программа, написанная на языке Clarion с подключаемыми вставками, предназначена для работы в Windows XP, т.е. в 32-разрядном режиме, а программа на TB – для DOS, т.е. для 16-разрядного режима. Коды некоторых операций в 16- и 32-разрядном режимах могут различаться между собой. Кроме того, кодовая последовательность в файле-листинге (точнее, порядок следования байт) также может отличаться от соответствующей ей кодовой последовательности в оттранслированном файле в *.com-формате (т.е. от объектных кодов).

Для иллюстрации рассмотрим простейшую процедуру ввода байта из порта компьютера: загрузим адрес порта ввода (например, 3feh) в регистр dx процессора, затем дадим команду ввода из этого порта в младший байт аккумулятора al:

```
mov dx,03feh
in al,dx
```

Рассмотрим команду mov dx,03feh. Ниже приведены два варианта трансляции этой команды: первый – в 16-разрядном режиме, когда параметр «.386» закомментирован, второй – в 32-разрядном режиме:

```
;.386
BA 03 FE mov dx,03feh - в файле-листинге
BA FE 03 - в файле *.com - формата
--------------------------------
.386
66 BA 03 FE mov dx,03feh - в файле-листинге
66 BA FE 03 - в файле *.com -формата
```

Видно, что коды операций в 16- и 32-разрядном режиме различаются между собой (в 32-разрядном режиме транслятор вставил дополнительный код 66h – префикс размера операнда, указывающий, что это edx, а не dx, и что адрес также 32-разрядный). Кроме того, порядок следования байт кодов операций в файле-листинге и в файле в *.com-формате в каждом из режимов также отличается. Разумеется, в этом случае коды файла-листинга для вставки использовать нельзя.

С другой стороны, загрузить число 03feh в регистр dx можно иначе: вначале загрузить старший байт числа 03feh (03) в старший байт регистра dx (это регистр dh), а в младший байт dl загрузить младший байт числа 03feh (0feh). Ниже приведён вариант такой загрузки, который даёт одинаковые коды в 16- и 32-разрядном режимах, и кроме того, коды файла-листинга полностью совпадают с кодами файла в *.com-формате.

```
.386/;.386
B6 03 mov dh,03h ; Итого:
B2 FE mov dl,0feh; dx=3feh - в файле-листинге
B6 03 B2 FE - в файле *.com - формата
```

Таким свойством обладают коды операций, длина которых не превышает двух байт. Кстати, если для ввода/вывода используется порт COM1, у которого старший адрес (dh) равен 03h, то, загрузив однажды по этому адресу число 03h, для дальнейшего ввода/вывода с порта COM1 достаточно изменять только младший байт dl. Поэтому необходимо программу составить так, чтобы в ней использовались двухбайтовые команды.

Вторым важным моментом является правильное использование оператора цикла loop. Этот оператор работает аналогично операторам цикла Бейсика: for ... next, C++: for (...) {...}, Clariona: loop ... end.

При использовании оператора loop, в счётчик циклов – регистр cx (ecx – в 32-разрядном режиме) – необходимо загрузить их число. Ниже приведены два варианта загрузки числа 20 в счётчик циклов в 32-разрядном и 16-разрядном режимах и показаны кодовые последовательности в файлах-листингах и в объектных кодах файлов в *.com-формате.

```
.386
B9 00 00 00 14 mov ecx,20 - в файле-листинге
B9 14 00 00 00 - в файле *.com - формата
------------
;.386
B9 00 14 mov cx,20 - в файле-листинге
B9 14 00 - в файле *.com - формата
```

Видно, что коды операций в 16- и 32-разрядном режиме различаются

между собой. Кроме того, порядок следования байт кодов операций в файле-листинге и в файле в *.com-формате в каждом из режимов также отличается. Значит, коды файла-листинга для вставки использовать нельзя. Если последовательно загрузить в cx сначала старший байт – ch (нулевым значением), затем младший байт – cl (числом 20) – как в первом примере при загрузке регистра dx, – это не решит проблему. Дело в том, что в 32-разрядном режиме работы оператор loop работает с 32-разрядным регистром ecx. Если загрузить числом 20 только его младшее слово (регистр cx), без изменения старшего слова, то после 20 циклов работы в регистр cx загрузится число 65535 (ffffh), из старшего слова вычтется единица, и цикл будет продолжаться до обнуления регистра ecx, что приведёт к сбою. С другой стороны, в старшее слово загрузить нулевое значение невозможно, т.к. старшее слово ecx недоступно для загрузки.

Однако решить задачу загрузки ecx числом 20, при которой сгенерируются двухбайтные коды, достаточно просто, если вначале обнулить весь регистр ecx, а потом загрузить в самый младший его регистр cl число 20. Для обнуления регистра ecx можно применить 2-байтную операцию исключающего ИЛИ (xor) над этим регистром:

```
.386
33 C9 xor ecx,ecx
B1 14 mov cl,20 - в файле-листинге
33 C9 B1 14 - в файле *.com - формата
-------------
;.386
33 C9 xor cx,cx
B1 14 mov cl,20 - в файле-листинге
33 C9 B1 14 - в файле *.com - формата
```

Как можно убедиться, такой вариант загрузки даёт одинаковые коды в 16- и 32-разрядном режиме, и кроме того, коды файла-листинга полностью совпадают с кодами файла в *.com-формате.

Второй вывод, который следует из вышеприведенного примера, – загрузку регистра ecx необходимо производить в соответствии с вышеприведённым вариантом (или придумать свой, дающий 2-байтные коды операций). В противном случае следует использовать объектные коды из файла в *.com-формате.

Заключение

Использование низкоуровневых команд процессора компьютера позволяет обеспечить более быструю и точную обработку сигналов интерфейса RS-232 из языка высокого уровня. В результате повышается надёжность обмена информацией между компьютером и микроконтроллером на максимальной скорости интерфейса.

Литература

1. *Кузьминов А.* Современные программные средства связи микроконтроллера с компьютером по интерфейсу RS-232. Компоненты и технологии. 2006. №№ 6–11.
2. *Кузьминов А.* Современные аппаратные средства связи микроконтроллера с компьютером по интерфейсу RS-232. Компоненты и технологии. 2006. № 3–5.
3. www.microcompsys.narod.ru.