"Интерфейс PIC-контроллера с компьютером"

При разработке устройств на микроконтроллере часто возникает проблема связи его с персональным компьютером. Одно из решений этой серьезной проблемы предлагается в этой статье.

Нередко при разработке устройства на микроконтроллере приходится думать о его связи с компьютером для обмена информацией. В большинстве случаев требуется двунаправленный режим при сравнительно невысокой скорости обмена.

В лучшем случае микроконтроллер может иметь последовательный интерфейс, но чаще всего приходится выбирать среди дешевых контроллеров, не имеющих такого интерфейса. Например, очень популярный в последнее время микроконтроллер производства фирмы Microchip PIC16F84A не имеет последовательного интерфейса [1].

В этой статье рассматривается вариант программной реализации последовательного интерфейса, как со стороны микроконтроллера, так и со стороны компьютера.

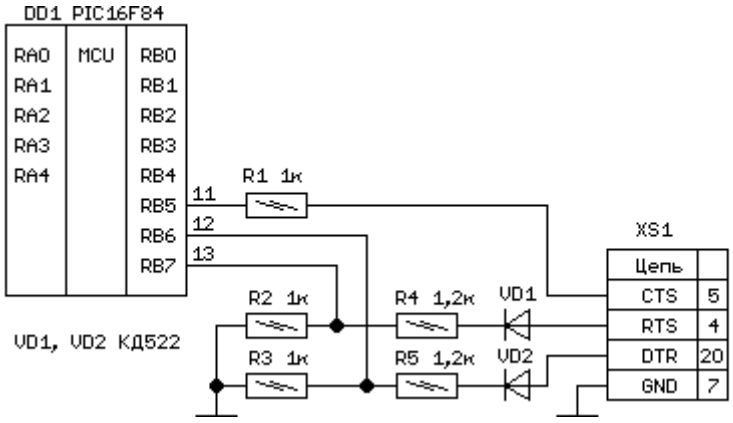
Для связи с устройством можно использовать параллельный порт LPT или последовательный COM порт. Использование LPT порта проще, в нем можно использовать относительно большее число входных и выходных сигналов, уровень сигналов совместим с ТТЛ. Недостатком LPT порта является то, что если под DOS или Linux для его использования достаточно простых операций ввода/вывода, то для корректной работы под Windows требуется строго соблюдать протокол передачи данных, который при работе с микроконтроллером был бы не эффективным. Возможно также и прямое управление отдельными линиями LPT порта, но для этого требуется установка специального драйвера. Также к недостаткам LPT порта относится и то, что на большинстве компьютеров он присутствует в единственном экземпляре и занят принтером.

Главным преимуществом COM порта является то, что стандартный программный интерфейс Windows (API) позволяет непосредственное управление некоторыми выходными линиями и непосредственный контроль входных линий, а также имеет функцию ожидания некоторого события,

связанного с COM портом. Достоинство COM порта также в том, что стандарт RS-232 по которому выполнены COM порты. допускает подключение и отключение кабелей во время работы устройств (hot plug). Также, почти всегда, в компьютере имеется свободный COM порт. Недостатком COM порта является отличный от ТТЛ уровень сигналов, в котором низкому логическому уровню соответствует напряжение -12В, а высокому логическому уровню - напряжение +12В.

Реализация стандартного интерфейса RS-232 потребовала бы от микроконтроллера точного соблюдения временных интервалов между выдаваемыми сигналами. В реальной ситуации кварцевый резонатор микроконтроллера может не соответствовать частоте передачи данных, а сам микроконтроллер обычно занят чем-то более важным, чем формирование точных временных интервалов. В результате оказывается проще программно реализовать последовательный синхронный вариант обмена, когда каждый бит данных подтверждается импульсом синхронизации.

Принципиальная схема интерфейса показана на рис. 1.

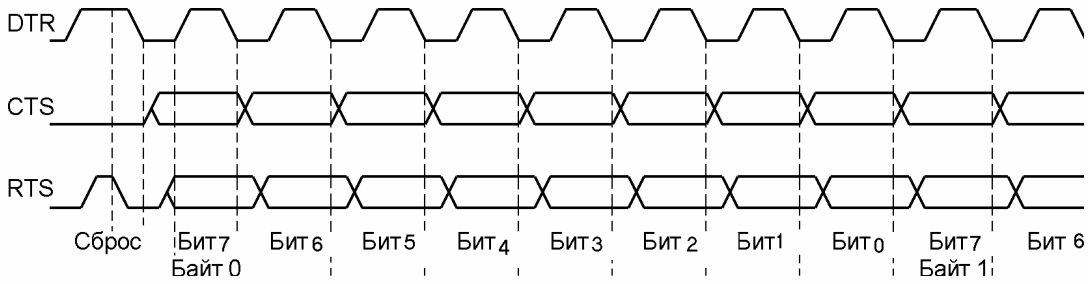


Для преобразования уровней RS-232 в ТТЛ используются резистивные делители R2R4 и R3R5. Диоды VD1 и VD2 необходимы для того, чтобы не пропускать отрицательное напряжение, соответствующее логическому нулю. Выходной ТТЛ сигнал микроконтроллера в преобразовании не нуждается и может быть подключен к входным линиям COM порта непосредственно. Резистор R1 служит для ограничения тока с выхода микроконтроллера при возможном случайном коротком замыкании.

Как видно из схемы, для связи с компьютером требуются 4 провода. Компьютер инициирует обмен данными, выдавая синхронизирующие

импульсы на линии DTR, выставляя при этом на линии RTS передаваемые данные и получая по линии CTS принимаемые. Изменять данные компьютер и микроконтроллер могут только при низком логическом уровне сигнала синхронизации. Такой вариант реализации интерфейса позволяет реализовать дуплексный режим передачи данных.

Скорость следования синхронизирующих импульсов должна выбираться такой, чтобы микроконтроллер гарантированно успевал обрабатывать данные от компьютера, реагируя на каждый синхронизационный импульс. Информационные биты передаются последовательно. После окончания передачи битов одного байта непосредственно следует передача битов следующего байта, при этом первым передается старший информационный бит. Для приведения интерфейса в исходное состояние (установка номера передаваемого байта в 0) компьютер должен при логической единице линии синхронизации изменить состояние линии данных. Микроконтроллер выдает новый бит данных на линии CTS по спаду импульсов положительной полярности на входе синхронизации DTR, а считывает данные на линии RTS по фронту импульсов положительной полярности. Обмен можно в любой момент прервать, прекратив подавать синхронизационные импульсы. Временная диаграмма обмена данными приведена на рис. 2.



При реализации интерфейса рекомендуется в некоторых байтах передавать контрольные значения для проверки правильности передаваемых данных.

Исходный код процедуры для микроконтроллера PIC16F84 на языке C, реализующий предлагаемый интерфейс приведен в табл. 1. Вызов процедуры link(), должен находиться в основном цикле программы и при работе микроконтроллера вызывается постоянно для того, чтобы контролировать состояние интерфейса. Все переменные, используемые процедурой, должны

быть объявлены как глобальные. Процедура при каждом вызове считывает состояние входных линий интерфейса (RB6 и RB7) и сравнивает с их состоянием при предыдущем вызове. При некоторых условиях (спад синхронизации, фронт синхронизации, сброс интерфейса) выполняются действия согласно логике работы интерфейса.

Исходный код процедуры для компьютера на языке Pascal (Delphi) приведен в табл. 2. Процедура link однократно вызывается для проведения акта обмена информацией с микроконтроллером. Перед вызовом процедуры необходимо заполнить передаваемый буфер obuf. По окончании работы процедуры принятые данные будут находиться в массиве ibuf. Процедура открывает указанный COM порт компьютера и с помощью функций Windows API [2] управляет состоянием выходных линий и опрашивает входные линии. После завершения обмена информацией порт закрывается. В процедуре link временные задержки реализованы с помощью функции sleep(). Значения задержек рассчитывается или подбирается экспериментально по отсутствию потери бит при обмене данными между микроконтроллером и компьютером. В примере указаны задержки для обмена с PIC контроллером с кварцевым резонатором на 4 МГц, который кроме обмена с компьютером совершает и другую полезную работу. Если процедура обмена выполняется слишком долго, то допускается выносить ее в отдельную нить выполнения операционной системы, чтобы она выполнялась параллельно основной программе [2].

Если при обмене информацией требуется отдельно чтение и запись, то можно разнести по различным адресам массивы передаваемых и принимаемых данных, как показано на рис 2.

В микроконтроллере формирование передаваемых данных и использование принимаемых удобно построить в виде процедур upload() и download(). Эти процедуры вызываются перед передачей и при приеме очередного байта соответственно. Процедура upload() должна возвращать значение передаваемого байта по его номеру в передаваемом пакете информации. Процедура download() получает значение принятого байта и его номер в пакете и должна использовать эти значения для изменения регистров микроконтроллера, записи в EEPROM и пр.

Реализация этих процедур для обработки информационного пакета размером в 4 байта, показанного в табл. 3 приведена в табл. 4.

Номера контактов разъема XS1 на рис. 1 соответствуют разъему DB-25F и использованию стандартного модемного кабеля. Номера контактов для других разъемов и при использовании нуль-модемного кабеля приведены в табл. 5.

Пример программы для микроконтроллера приведен для компилятора C2C [3]. Процедура для компьютера может быть использована в программе написанной на Borland Delphi 3 и выше.

Литература.

1. http://www.microchip.com/download/lit/pline/picmicro/families/16f8x/30430c.pdf
2. Microsoft Developer Network, Technical Articles
3. http://www.picant.com/c2c/c.html

Таблица 1.

```
char new6,new7,old6,old7;     //Глобальные
char ibuf,obuf,cbit,cbyte;    //переменные

void link()
      {
      new7=input_pin_port_b(7);      //Читаем состояние RB7 (DTR)
      new6=input_pin_port_b(6);      //Читаем состояние RB6 (RTS)
      if ((new6!=old6)&&(new7))      //Изменение данных при
            {                        // импульсе синхронизации
            cbit=0;cbyte=0;          //Обнуление счетчиков
            ibuf=0;                  //Обнуление входного буфера
            }
      else if ((new7)&&(!old7))      //Фронт синхронизации
            {
            ibuf<<=1;                //Сдвигаем буфер на бит влево
            if (input_pin_port_b(6)) ibuf++; // и читаем данные
            if (cbit==0)             //Если 0 бит (байт получен),
                  {                  //то записываем буфер
                  download(cbyte-1,ibuf);
                  ibuf=0;
                  }
            }
      if ((!new7)&&(old7))           //Спад синхронизации
            {
            if (cbit==0)             //Если 0 бит (новый байт),
                  obuf=upload(cbyte);     //то заполняем буфер
            if (obuf&0x80) output_high_port_b(5); else
            output_low_port_b(5);    //Выводим данные
            obuf<<=1;                //Сдвигаем буфер на бит влево
            cbit++;                  //Увеличиваем на 1 счетчик битов
            if (cbit==8)             // Если счетчик битов равен 8,
                  {
                  cbit=0;            //то обнуляем счетчик битов
                  cbyte++;           //увеличиваем на 1 счетчик байтов
                  }
            }

      old7=new7;                     //Запоминаем старые RB7
```

```
      old6=new6;                    //и RB6
      }
```

Таблица 2.

```
var ibuf, obuf: array [0..3] of byte; //Входной и выходной буферы

procedure Link;   //Процедура обмена данными с устройством
var cbyte, cbit: integer;
    modemstat: dword;
    comh: hFile;
begin
//открываем порт COM2
comh:=CreateFile('\\.\COM2', GENERIC_READ or GENERIC_WRITE, 0, nil,
                 OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
//Переводим DTR и RTS в состояние логического нуля
EscapeCommFunction(comh, CLRDTR);
EscapeCommFunction(comh, CLRRTS);
Sleep(2);   //Пауза 2 мс
//Сброс интерфейса (изменение данных во время синхронизации)
EscapeCommFunction(comh, SETDTR); Sleep(2);
EscapeCommFunction(comh, SETRTS); Sleep(2);
EscapeCommFunction(comh, CLRRTS); Sleep(2);
EscapeCommFunction(comh, CLRDTR); Sleep(2);

//Непосредственно обмен в дуплексном режиме
for cbyte:=0 to 3 do
    begin
    for cbit:=0 to 7 do
        begin
//Проверяем старший (передаваемый) бит и устанавливаем RTS
        if (obuf[cbyte] shl cbit) and $80>0 then
           EscapeCommFunction(comh, SETRTS) else
           EscapeCommFunction(comh, CLRRTS);
        Sleep(2);
//Получаем состояние входных сигналов COM порта
        GetCommModemStatus(comh, ModemStat);
//Проверяем CTS и записываем соответствующий бит в буфер
        if (ModemStat and MS_CTS_ON<>0) then
           ibuf[cbyte]:=ibuf[cbyte]+($80 shr cbit);
//Формируем импульс синхронизации
        EscapeCommFunction(comh, SETDTR); Sleep(4);
        EscapeCommFunction(comh, CLRDTR); Sleep(2);
        end;
    end;
//Закрываем COM порт
CloseHandle(comh)
end;
```

Таблица 3.

Направление	Номер байта			
	0	1	2	3
MCU → PC	Переменная CTIME	Переменная CDATE	Переменная CSTATE	Контрольный байт 55h
PC → MCU	Переменная COUNT	Контрольный байт 0AAh	Контрольный байт 0AAh	Переменная CONTROL

Таблица 4.

```
void download(char num, char data)  //полученный байт
      {
      switch (num)                  //действие зависит от
            {                       //номера байта
            //устанавливаем некоторые переменные
            case 0: count=data; break;
            //проверка контрольных байтов и вызов error() при ошибке
            case 1: if (data!=0xAA) error(); break;
            case 2: if (data!=0xAA) error(); break;
            case 3: control=data; break;
            }
      }

char upload(char num)               //передаваемый байт
      {
      switch (num)
            {
            //читаем некоторые переменные
            case 0: return ctime; break;
            case 1: return cdate; break;
            case 2: return cstate; break;
            //передаем контрольный байт
            case 3: return 0x55; break;
            }
      }
```

Таблица 5.

Сигнал	DB-25F	DB-9F	DB-25M (нуль-модем)	DB-9M (нуль-модем)
CTS	5	8	4	7
RTS	4	7	5	8
DTR	20	4	6	6
GND	7	5	7	5