Міністерство освіти і науки України

Черкаський національній університет імені Богдана Хмельницького

Кафедра програмного забезпечення автоматизованих систем

Курсова робота на тему:

"Створення вимірювального апаратно – програмного комплексу термометра на основі мікро контролерів сім'ї ATMEGA"

Виконав: Перевірив:

Студент групи КС-061 Доцент к. н. т.

Голубченко Юрій Сергійович Хамід

Черкаси 2008р.

Задание:

Создать измерительный аппаратно – программный комплекс термометра на основе микроконтроллеров семейства ATMEGA.

## 1)Схемотехническое решение поставленной задачи:

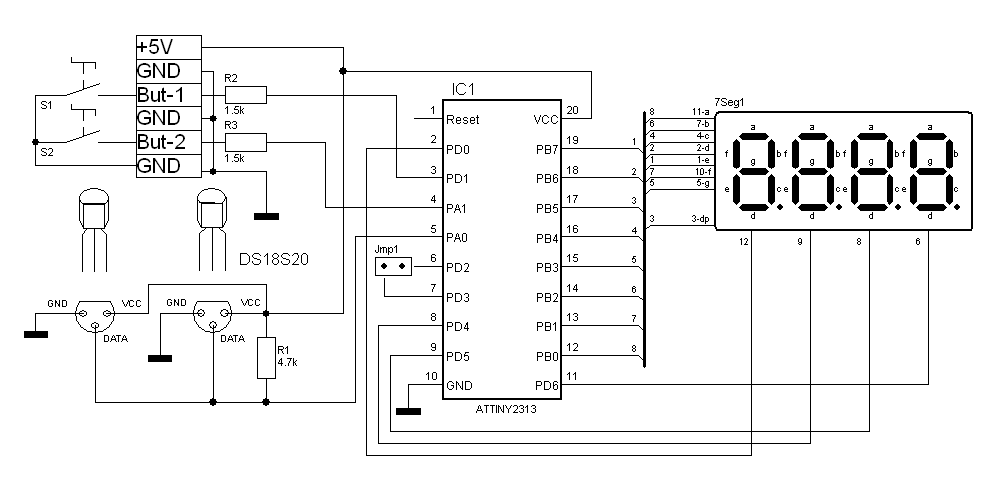


Схема включает в себя:

1) Микроконтроллер ATTINY2313;

2) 2 датчика температуры DS18S20;

3) Индикатором служит 3-хзначный, общий анод, динамическая индикация, зеленый;

4) Два дискретных вывода переведены на вход для кнопок для управления индикацией;

5) Определение до двух датчиков на одной шине 1-Wire;

6) Вывод значение первого, второго, разности первого из второго, разности второго из первого на дисплей с помощью двух кнопок. Так же возможно подключение одного датчика;

## Применение термометра для регулировки температуры внутри корпуса компьютера:

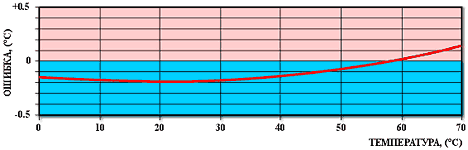
В корпусе современного компьютера сконцентрировано большое количество элементов, выделяющих тепло. По большому счету, тепло выделяет практически всё, так как любая работающая электронная схема рассеивает некоторую мощность. Однако есть элементы, которые являются весьма интенсивными источниками тепла. Это процессор, чипы на материнской плате и на видеокарте, элементы на плате жесткого диска, элементы блока питания и т.д. Давно прошли те времена, когда процессор мог работать без принудительного охлаждения. Уже стал привычным кулер на видеокарте, иногда он устанавливается также на северный мост чипсета и на жесткий диск. Современный корпус обычно имеет места для установки дополнительных вентиляторов, которые призваны продувать весь внутренний объем корпуса компьютера. Особенно остро проблема охлаждения стоит для компьютеров, насыщенных платами расширения, а также для компьютеров с «разогнанными» процессорами.

Надежность полупроводниковых приборов при повышении рабочей температуры падает, не говоря уже о надежности и долговечности жесткого диска. Однако повышенная температура внутри корпуса компьютера ведет не только к сокращению срока службы компонентов, но и к неустойчивой работе, если какой-то компонент перегревается.

В свете сказанного очень важно обеспечить должное охлаждение компонентов и правильную вентиляцию корпуса. Правильно выбрать количество и тип вентиляторов, а также правильно организовать воздушные потоки является весьма сложным делом, так как свободный объем внутри корпуса имеет сложную конфигурацию, и потокам воздуха мешают различные предметы, в том числе провода. Иногда применение более мощного вентилятора даёт худший эффект, чем правильное распределение воздушных потоков от маломощного вентилятора. К тому же мощный вентилятор обычно имеет высокий уровень шума. Теоретически рассчитать потоки не представляется возможным, поэтому действовать приходится интуитивно, методом проб и ошибок. Основная трудность заключается в том, что очень трудно оценить эффективность той или иной принятой меры ввиду отсутствия средств контроля температуры. Имеющиеся на некоторых материнских платах и жестких дисках термодатчики позволяют судить лишь о температуре в нескольких точках. Поэтому приходится замерять температуру компонентов «на ощупь», что нельзя назвать точным и повторяемым методом.

При экспериментах с охлаждением компонентов компьютера неоценимую помощь может оказать независимый термометр. Такой термометр должен иметь небольшой по размерам датчик для его легкого размещения на различных компонентах, датчик должен иметь маленькую инерционность для возможности быстрого проведения измерений, ну и, конечно, достаточно высокую точность измерений. К тому же, термометр должен быть не дорогим. Всем этим требованиям удовлетворяют цифровые датчики температуры, выпускаемые фирмой DALLAS Semiconductor (теперь это уже MAXIM), которые могут быть подключены к последовательному порту компьютера.

С помощью дополнительных вычислений дискретность представления температуры можно уменьшить, в нашем случае она равна 0.1°C. Самым привлекательным является то, что такой термометр уже откалиброван на заводе, гарантированная точность составляет ±0.5°C в диапазоне –10..+85°C и ±2°C во всем диапазоне рабочих температур. Типичная кривая ошибки измерения температуры приведена на рис



Несмотря на ограниченную абсолютную точность, малая дискретность представления температуры является весьма желательной, так как очень часто на практике требуются относительные измерения.

DS18S20 допускает напряжение питания от +3 до +5.5В. В режиме ожидания потребляемый ток близок к нулю (менее 1мкА), а во время преобразования температуры он равен примерно 1мА. Процесс преобразования длится максимум 750мс.

Принцип действия цифровых датчиков температуры фирмы DALLAS основан на подсчете количества импульсов, вырабатываемых генератором с низким температурным коэффициентом во временном интервале, который формируется генератором с большим температурным коэффициентом. Счетчик инициализируется значением, соответствующим -55°C (минимальной измеряемой температуре). Если счетчик достигает нуля перед тем, как заканчивается временной интервал (это означает, что температура больше -55°C), то регистр температуры, который также инициализирован значением -55°C, инкрементируется. Одновременно счетчик предустанавливается новым значением, которое задается схемой формирования наклона характеристики. Эта схема нужна для компенсации параболической зависимости частот генераторов от температуры. Счетчик снова начинает работать, и если он опять достигает нуля, когда интервал еще не закончен, процесс повторяется снова. Схема формирования наклона загружает счетчик значениями, которые соответствуют количеству импульсов генератора на один градус Цельсия для каждого конкретного значения температуры. По окончанию процесса преобразования регистр температуры будет содержать значение температуры.

Для DS18S20 температура представляется в виде 9-битного значения в дополнительном коде. Поскольку это значение занимает 2 байта, все разряды старшего байта равны знаковому разряду. Дискретность представления температуры составляет 0.5°C. Зависимость выходного кода от температуры приведена в таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| Температура | Выходной код (Binary) | | Выходной код (Hex) |
| Ст. байт | Мл. байт |
| +125°C | 0000 0000 | 1111 1010 | 00FAh |
| +25°C | 0000 0000 | 0011 0010 | 0032h |
| +0.5°C | 0000 0000 | 0000 0001 | 0001h |
| 0°C | 0000 0000 | 0000 0000 | 0000h |
| -0.5°C | 1111 1111 | 1111 1111 | FFFFh |
| -25°C | 1111 1111 | 1100 1110 | FFCEh |
| -55°C | 1111 1111 | 1001 0010 | FF92h |

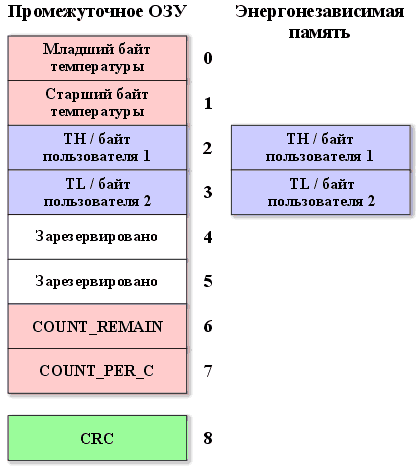
Более высокая разрешающая способность может быть получена, если произвести дополнительные вычисления на основе значений COUNT\_REMAIN (значение, оставшееся в счетчике в конце измерения) и COUNT\_PER\_C (количество импульсов на один градус для данной температуры), которые доступны. Для вычислений требуется взять считанное значение температуры и отбросить младший бит. Полученное значение назовём TEMP\_READ. Теперь действительное значение температуры может быть вычислено по формуле:

TEMPERATURE=TEMP\_READ-0.25+(COUNT\_PER\_C - COUNT\_REMAIN)/COUNT\_PER\_C

В нашем случае такой расчет позволяет получить дискретность представления температуры 0.1°C.

Каждый экземпляр DS18S20 имеет уникальный 48-битный номер, записанный с помощью лазера в ПЗУ в процессе производства. Этот номер используется для адресации устройств. Кроме серийного номера в ПЗУ содержится код семейства (для DS18S20 это 10h) и контрольная сумма.

Кроме ПЗУ DS18S20 имеет промежуточное ОЗУ объемом 8 байт, плюс два байта энергонезависимой памяти. Карта памяти DS18S20 показана на рисунке:

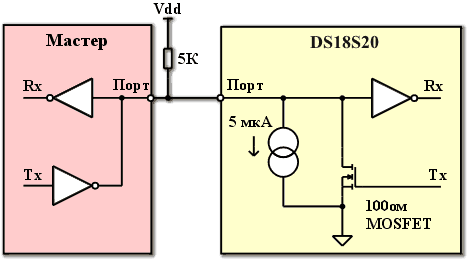


Байты TH и TL представляют собой температурные пороги, с которыми сравниваются 8 бит каждого измеренного значения температуры (младший бит отбрасывается). С помощью специальной команды можно организовать сигнализацию выхода температуры за пределы этих порогов. Если такая функция не нужна, байты TH и TL можно использовать для хранения любых данных пользователя.

Считывание значения измеренной температуры, а также передача команды начала преобразования и других команд производится с помощью 1-проводного интерфейса (1-WireTM) фирмы DALLAS. На основе этого интерфейса фирма DALLAS даже создала сеть, называемую microLAN (или µLAN). Для работы в этой сети выпускается целый ряд устройств, таких как адресуемые ключи, АЦП, термометры, часы реального времени, цифровые потенциометры. Кстати, такой же протокол обмена имеют и цифровые ключи IButton (или Touch Memory), которые сейчас широко используются в системах ограничения доступа.

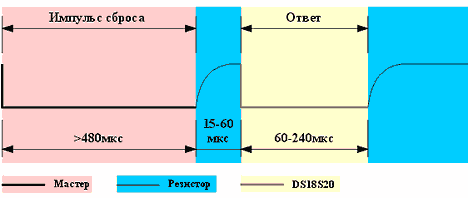
Протокол, который используется 1-проводным интерфейсом, достаточно прост. В любой момент времени на 1-проводной шине можно выделить устройство-мастер, которым может быть микропроцессор или компьютер, и подчиненное устройство, в нашем случае это микросхема термометра. Так как у нас на шине присутствуют только мастер и всего одно подчиненное устройство, можно опустить всё то, что связано с адресацией устройств. В результате требуется знать лишь протокол передачи байтов, которые могут являться командами или данными.

Вначале рассмотрим аппаратную конфигурацию. 1-проводная шина является двунаправленной. На рис. 4 показана аппаратная конфигурация интерфейсной части DS18S20 и мастера шины. У каждого 1-проводного устройства к шине подключен вход приемника и выход передатчика с открытым стоком. Открытый сток позволяет подключать к шине множество устройств, обеспечивая логику «монтажное или». Генератор тока 5мкА обеспечивает на входе 1-проводного устройства низкий логический уровень, когда шина не подключена. Так как линия тактового сигнала отсутствует, обмен является синхронным. Это означает, что в процессе обмена нужно достаточно точно выдерживать требуемые временные соотношения.



1-проводная шина оперирует с TTL-уровнями, т.е. логическая единица представлена уровнем напряжения около 5В, а логический ноль – напряжением вблизи 0В. В исходном состоянии на линии присутствует уровень логической единицы, который обеспечивается подтягивающим резистором номиналом около 5Ком.

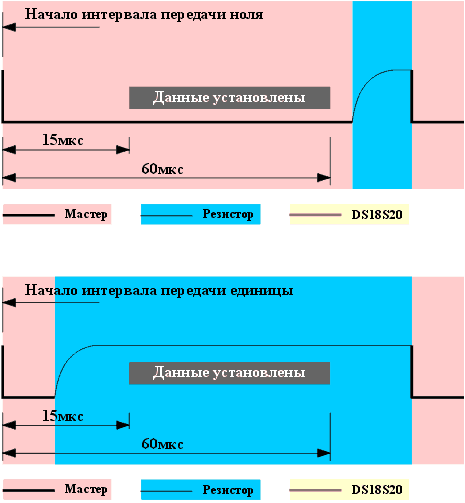
Инициатором обмена по 1-проводной шине всегда выступает мастер. Все пересылки начинаются с процесса инициализации. Инициализация производится в следующей последовательности



Мастер посылает импульс сброса (reset pulse) - сигнал низкого уровня длительностью не менее 480 мкс.

За импульсом сброса следует ответ подчиненного устройства (presence pulse) - сигнал низкого уровня длительностью 60 - 240 мкс, который генерируется через 15 - 60 мкс после завершения импульса сброса.

Ответ подчиненного устройства даёт мастеру понять, что на шине присутствует термометр и он готов к обмену. После того, как мастер обнаружил ответ, он может передать термометру одну из команд. Передача ведётся путём формирования мастером специальных временных интервалов (time slots). Каждый временной интервал служит для передачи одного бита. Первым передаётся младший бит. Интервал начинается импульсом низкого уровня, длительность которого лежит в пределах 1 - 15 мкс. Поскольку переход из единицы в ноль менее чувствителен к ёмкости шины (он формируется открытым транзистором, в то время как переход из ноля в единицу формируется подтягивающим резистором), именно этот переход используют 1-проводные устройства для синхронизации с мастером. В подчиненном устройстве запускается схема временной задержки, которая определяет момент считывания данных. Номинальное значение задержки равно 30 мкс, однако, оно может колебаться в пределах 15 - 60 мкс. За импульсом низкого уровня следует передаваемый бит. Он должен удерживаться мастером на шине в течение 60 - 120 мкс от начала интервала. Временной интервал завершается переводом шины в состояние высокого уровня на время не менее 1 мкс. Нужно отметить, что ограничение на это время сверху не накладывается. Аналогичным образом формируются временные интервалы для всех передаваемых битов



Первой командой, которую должен передать мастер для DS18S20 после инициализации, является одна из команд функций ПЗУ. Всего DS18S20 имеет 5 команд функций ПЗУ:

Read ROM [33h]. Эта команда позволяет прочитать содержимое ПЗУ. В ответ на эту команду DS18S20 передает 8-битный код семейства (10h), затем 48-битный серийный номер, а затем 8-битную CRC для проверки правильности принятой информации.

Match ROM [55h]. Эта команда позволяет адресовать на шине конкретный термометр. После этой команды мастер должен передать нужный 64-битный код, и только тот термометр, который имеет такой код, будет «откликаться» до следующего импульса сброса.

Skip ROM [CCh]. Эта команда позволяет пропустить процедуру сравнения серийного номера и тем самым сэкономить время в системах, где на шине имеется всего одно устройство.

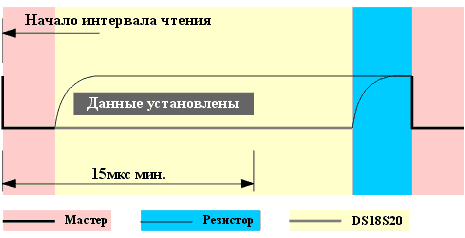
Search ROM [F0h]. Эта довольно сложная в использовании команда позволяет определить серийные номера всех термометров, присутствующих на шине.

Alarm Search [ECh]. Эта команда аналогична предыдущей, но «откликаться» будут только те термометры, у которых результат последнего измерения температуры выходит за предустановленные пределы TH и TL.

Поскольку у нас всего одно устройство, наиболее подходящей для нас функцией является функция Skip ROM. Кроме неё ещё может быть полезной функция Read ROM, которая позволяет идентифицировать подключенное на шину устройство по его коду семейства и серийному номеру.

Приняв команду Read ROM, DS18S20 будет готов передать 64-битный код, который мастер должен принять.

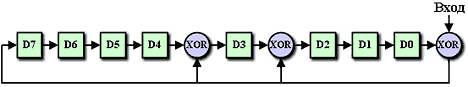
При приеме данных от подчиненного устройства временные интервалы для принимаемых битов тоже формирует мастер. Интервал начинается импульсом низкого уровня длительностью 1 - 15 мкс. Затем мастер должен освободить шину, чтобы дать возможность термометру вывести бит данных. По переходу из единицы в ноль DS18S20 выводит на шину бит данных и запускает схему временной задержки, которая определяет, как долго бит данных будет присутствовать на шине. Это время лежит в пределах 15 - 60 мкс. Для того чтобы данные на шине, которая всегда обладает некоторой ёмкостью, гарантированно установились, требуется некоторое время. Поэтому момент считывания данных мастером должен отстоять как можно дальше, но не более чем на 15 мкс от начала временного интервала



Прием байта начинается с младшего бита. Вначале идет байт кода семейства. За кодом семейства идет 6 байт серийного номера, начиная с младшего. Затем идет байт контрольной суммы (CRC). В вычислении байта контрольной суммы принимают участие первые 7 байт, или 56 передаваемых бит. Для вычисления используется следующий полином:

CRC = X8+X5+X4+1

После приема данных мастер должен вычислить контрольную сумму и сравнить получившееся значение с переданной CRC. Если эти значения совпадают, значит, прием данных прошел без ошибок. Можно также вычислить контрольную сумму для всех 64 принятых бит, которая в этом случае должна быть равна нулю. Блок-схема алгоритма вычисления контрольной суммы показана на рис. 8. Алгоритм использует операции сдвига и «исключающего или». Квадратиками показаны биты переменной, которая используется для вычисления CRC. Перед вычислением её необходимо обнулить, а затем на вход алгоритма нужно последовательно подать 56 принятых бит в том порядке, в котором они были приняты. В результате переменная будет содержать значение CRC.



Такой же алгоритм вычисления контрольной суммы используется и в случае чтения промежуточного ОЗУ, только там считанная из термометра CRC (9-й байт) рассчитана для 8-ми байтов данных.

После обработки одной из команд функций ПЗУ, DS18S20 способен воспринимать еще несколько команд:

Write Scratchpad [4Eh]. Эта команда позволяет записать данные в промежуточное ОЗУ DS18S20.

Read Scratchpad [BEh]. Эта команда позволяет считать данные из промежуточного ОЗУ.

Copy Scratchpad [48h]. Эта команда копирует байты TH и TL из промежуточного ОЗУ в энергонезависимую память. Эта операция требует около 10мс.

Convert T [44h]. Эта команда запускает процесс преобразования температуры.

Recall E2 [B8h]. Эта команда действует обратным образом по отношению к команде Copy Scratchpad, т.е. она позволяет считать байты TH и TL из энергонезависимой памяти в промежуточное ОЗУ. При включении питания эта команда выполняется автоматически.

Read Power Supply [B4h]. Эта команда позволяет проверить, использует ли DS18S20 паразитное питание. Дело в том, что DS18S20 можно подключать всего с помощью двух проводов, в этом случае для питания используется линия данных. Особенности этого режима мы здесь рассматривать не будем.

При использовании DS18S20 только для измерения температуры нужны всего две из этих команд: Convert T и Read Scratchpad.

Последовательность действий при измерении температуры должна быть следующей:

Посылаем импульс сброса и принимаем ответ термометра.

Посылаем команду Skip ROM [CCh].

Посылаем команду Convert T [44h].

Формируем задержку минимум 750мс.

Посылаем импульс сброса и принимаем ответ термометра.

Посылаем команду Skip ROM [CCh].

Посылаем команду Read Scratchpad [BEh].

Читаем данные из промежуточного ОЗУ (8 байт) и CRC.

Проверяем CRC, и если данные считаны верно, вычисляем температуру.

Для подключения DS18S20 к COM-порту компьютера используется адаптер, схема которого приведена на рисунке, где показано окно помощи программы.

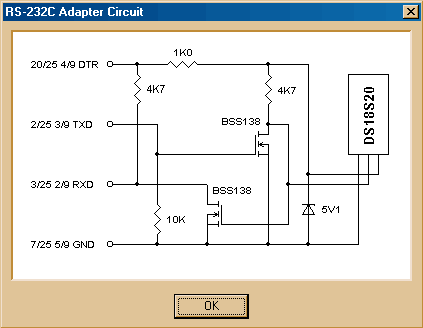
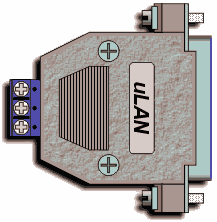
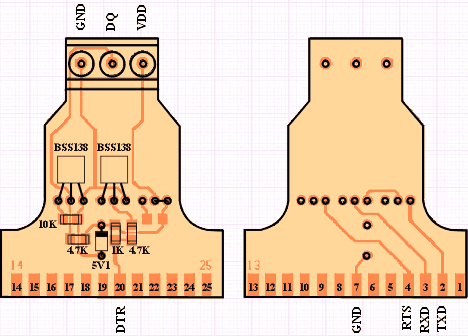


Схема этого адаптера не так проста, как, например, схема адаптера DS9097 фирмы Dallas. Это связано в первую очередь с тем, что хотелось иметь общую «землю» компьютера и 1-проводной шины. Для питания DS18S20 используется линия DTR последовательного порта. Адаптер обеспечивает на входе RXD порта компьютера лишь однополярные уровни, что, строго говоря, не соответствует спецификации RS-232C. Однако большинство портов с такими уровнями работают нормально. Вместо указанных на схеме n-канальных МОП-транзисторов можно применить близкие по параметрам транзисторы других типов, например, 2N7000. Подойдут также отечественные транзисторы КП501 или КП505. В принципе, можно применить и биполярные транзисторы, добавив в базы ограничительные резисторы. Конструктивно адаптер выполнен в корпусе разъема D-SUB-25 :



Разъем паяется прямо на плату, которая входит между рядами контактов. Другая сторона платы выполнена удлиненной и выходит за пределы корпуса разъема. В эту часть платы впаян 3-х контактный винтовой терминал



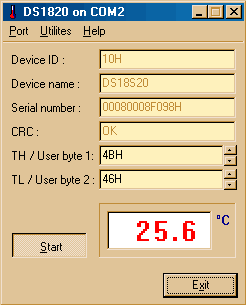
Можно вовсе обойтись без печатной платы, а выполнить монтаж адаптера навесным способом на контактах разъема.

К винтовому терминалу подключается микросхема термометра с помощью проводов, длина которых может составлять до нескольких метров. Если датчик термометра используется в комнатных условиях, то никаких мер по его защите применять не нужно, необходимо только заизолировать выводы. Если же предполагается измерять температуру наружного воздуха или каких-то агрессивных сред, датчик термометра необходимо упаковать. Например, можно взять алюминиевый корпус электролитического конденсатора подходящего диаметра и поместить туда датчик, заполнив весь свободный объем теплопроводящей пастой. Сверху такой стакан необходимо загерметизировать.

Описанный адаптер также подходит для считывания электронных ключей IButton и для подключения других однопроводных устройств.

Программа ds1820.exe, работающая под Win95/98/ME/NT, позволяет считывать и отображать показания термометра, а также считывать серийный номер и программировать два пользовательских байта. Эта программа кроме термометра DS18S20 поддерживает и его предшественника DS1820.

Вид главного окна программы показан на рисунке



Окно имеет следующие элементы управления:

Поле Device ID, куда выводится код семейства 1-проводного устройства. Для DS1820 и DS18S20 он равен 10h.

Поле Device Name, где приводится расшифровка типа устройства.

Поле Serial number, куда выводится серийный номер, записанный в ПЗУ.

Поле CRC, где отображается результат проверки контрольной суммы (OK или FAIL).

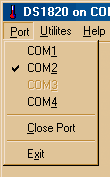
Поле TH / User byte 1, куда можно ввести значение в шестнадцатеричной форме, которое будет записано в регистр TH.

Поле TL / User byte 2, куда можно ввести значение в шестнадцатеричной форме, которое будет записано в регистр TL.

Кнопка Start запускает процесс измерения температуры. Преобразования выполняются периодически, а измеренная величина выводится в поле температуры. Дискретность представления составляет 0.1°C, что достигается дополнительными вычислениями.

Кнопка Exit позволяет выйти из программы.

Кроме того, вверху окна имеется меню, состоящее из трех пунктов: Port, Utilites и Help.



Меню Port в развернутом виде показано на рис. 13. Это меню позволяет открыть один из четырех портов COM1 – COM4. Кроме того, меню позволяет закрыть порт и выйти из программы. В списке активны только доступные порты (т.е. те, которые физически присутствуют и не заняты в данный момент другими приложениями). Когда порт открывается, проверяется наличие на этом порту адаптера (достаточно соединения TXD – RXD). Если адаптер не обнаружен, выводится соответствующее сообщение



Меню Help содержит рисунок принципиальной схемы адаптера и сведения о разработчике программы.

Для хранения установок программа использует ini-файл, который создается автоматически в том же директории, где расположен exe-файл. В ini-файле содержится информация о положении окна программы и номер используемого COM-порта:

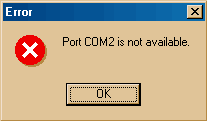
[General]

Left=427

Top=295

COM port=2

Если в существующем ini-файле указан номер COM-порта, который на момент запуска программы занят или отсутствует, выводится специальное сообщение

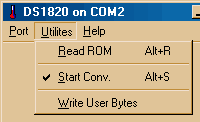


Меню Utilites содержит три пункта:

Read ROM – считывание содержимого ПЗУ (код семейства, серийный номер), а также считывание TH и TL.

Start Conv. – дублирует кнопку Start основного окна.

Write User Bytes – записывает значения TH и TL, которые введены в соответствующих полях.



Обмен, который производится по 1-проводной шине, требует довольно точного соблюдения временных соотношений микросекундного диапазона. Работая под Windows, точно сформировать такие интервалы программным способом невозможно. Поэтому необходимые временные интервалы формируются аппаратно микросхемой приемопередатчика последовательного порта, который используется не совсем обычно.

Для генерации импульса сброса и приема ответа порт настраивается на скорость 9600 бод, длина символа 8 бит, и передается число F0h. Это приводит к формированию импульса сброса низкого уровня (с учетом инверсии адаптера) длительностью примерно 520 мкс (стартовый бит + 4 передаваемых бита). За ним следует импульс высокого уровня такой же длительности (4 передаваемых бита + стоповый бит), в течение которого ожидается ответ термометра. Если термометр не подключен, то порт примет число F0h не искаженным. Но если термометр сформировал импульс ответа, то принятое число будет содержать большее количество единиц, чем четыре. Таким образом определяется наличие ответа.

Для генерации интервалов приема и передачи битов, порт настраивается на скорость 115200 бод. Стартовый бит начинает временной интервал, затем идут 8 единичных или нулевых бит, в зависимости от значения бита, который нужно передать. Заканчивается последовательность стоповым битом, который на некоторое время устанавливает на 1-проводной шине высокий уровень. Вид этой последовательности полностью соответствует требуемому виду временного интервала передачи бита, за исключением того, что обмен является более медленным, чем способна обеспечить 1-проводная шина: на один бит затрачивается примерно 87 мкс.

Прием со стороны передатчика последовательного порта ничем не отличается от передачи единичного бита. Проанализировав принятый последовательным портом символ можно установить, какой бит передал термометр. В случае нулевого бита один или несколько битов принятого портом байта будут единичными (с учетом инверсии адаптера).

Для работы с COM-портом программа пользуется функциями API через специальную «оберточную» динамическую библиотеку comapi32.dll. Однако скорость обмена получается гораздо ниже расчетной из-за того, что переключение скорости COM-порта (вызов функции SetCommState с изменённым значением поля BaudRate структуры DCB) идёт удивительно долго (порядка 200 миллисекунд!). Это весьма неприятное свойство API.

Программа DS1820.zip вместе с исходным текстом на Delphi 5 source20.zip распространяется бесплатно.

При использовании термометра необходимо учитывать, что корпус DS18S20 сделан из материала с относительно плохой теплопроводностью. Поэтому утечка тепла через выводы и подключенные к ним провода может быть весьма ощутимой. Для уменьшения ошибки измерения следует использовать провода малого сечения.

Кроме измерения температуры компонентов внутри корпуса компьютера, этот термометр можно использовать и для других целей, например, как комнатный или как наружный термометр.

3) Робота с COM-портом в среде ОС Windows:

Вопрос «как работать с COM-портами?» стал классическим на многих конференциях по языкам программирования. Рано или поздно чуть не каждому программисту приходится работать с портами ввода/вывода. Сегодня я хочу рассказать про работу с последовательным портом из-под самой распространенной на сегодняшний день 32-разрядной операционной системы — Windows. К статье прилагается пример программы, работающей с COM-портом, написанной на Borland Delphi 7.

Статья построена по принципу «от простого к сложному». Сначала будут изложены основы работы с портами из-под Win32 с описанием необходимых функций. Затем рассмотрим применение этих функций на примере Delphi-программы. Конечным результатом будет класс, предназначенный для работы с COM-портом, и пример использующей его программы.

Очень часто программисту приходится управлять с помощью компьютера каким-либо внешним устройством, или просто анализировать состояние этого устройства. Порты ввода/вывода — самый распространенный способ сопряжения компьютера и внешнего устройства. Давным-давно уже написано множество классов, библиотек и компонент для работы с портами, поэтому можно, конечно, воспользоваться уже готовым и к тому же бесплатным решением. Именно так я и поступил лет семь назад, при этом потеряв самое главное — своевременное понимание того, как все-таки работать с портами из-под Win32. Незнание внутренних механизмов — это, во-первых, пробел в стройном ряду знаний, а во-вторых, актуальная возможность ошибок в работе программы.

С портами из-под Win32 работают так же, как и с обычными файлами, используя при этом всего несколько специфичных функций WinAPI. Однако коммуникационный порт — это не совсем обычный файл. Для него, например, нельзя выполнить позиционирование файлового указателя, или же создать порт, если таковой отсутствует. Любая работа с портом начинается с его открытия. Для этого используется файловая функция WinAPI (описания WinAPI-функций взяты из MSDN (Microsoft Developer Network), следовательно, приводятся в синтаксисе C):

HANDLE CreateFile(

LPCTSTR lpFileName,

DWORD dwDesiredAccess,

DWORD dwShareMode,

LPSECURITY\_ATTRIBUTES lpSecurityAttributes,

DWORD dwCreationDistribution,

DWORD dwFlagsAndAttributes,

HANDLE hTemplateFile

);

lpFileName — указатель на строку с нулевым завершающим символом. Обычно это имя открываемого файла, но в нашем случае это должно быть название порта (COM1, COM2, …).

dwDesiredAccess — тип доступа. В нашем случае должен быть равен GENERIC\_READ|GENERIC\_WRITE.

dwShareMode — параметр совместного доступа. Для коммуникационных портов всегда равен 0.

lpSecurityAttributes — атрибут защиты. Для коммуникационных портов всегда равен NULL.

dwCreationDistribution — режим автосоздания. Для коммуникационных портов всегда равен OPEN\_EXESTING.

dwFlagsAndAttributes — атрибут режима обработки. Для коммуникационных портов должен быть равен 0 или FILE\_FLAG\_OVERLAPPED.

hTemplateFile — описатель файла-шаблона. Для коммуникационных портов должен быть равен NULL.

При успешном открытии порта функция возвращает его описатель, а в случае ошибки возвращает INVALID\_HANDLE\_VALUE.

Сразу оговорюсь: все недостающие описания можно найти на http://msdn.microsoft.comи еще по ряду адресов, которые вам подскажет поисковый сервер.

Из всех параметров функции CreateFile() особого пояснения требует dwFlagsAndAttributes. Работа с портом может быть организована в синхронном (nonoverlapped) или асинхронном (overlapped) режимах обработки, что и задается этим флагом. При синхронном режиме (когда параметр dwFlagsAndAttributes = 0) только один поток приложения может либо читать, либо писать в порт. Помните переговорное устройство в лифте? Нажали кнопку — можем только говорить, отпустили кнопку — можем только слушать.

Синхронный режим обработки прост в реализации. Если надо записать данные в порт, то вызываем функцию записи и ожидаем, пока она не завершится. Если же надо читать данные, то вызываем функцию чтения и ждем, пока она не отработает. Для простых задач синхронный режим обработки вполне подходит, однако в мире Windows он почти всегда обречен на неудачу. Ожидание операции чтения или записи воспринимается пользователем программы как «зависание».

Асинхронный режим (когда параметр dwFlagsAndAttributes = FILE\_FLAG\_OVERLAPPED) позволяет производить операции чтения и записи в порт параллельно из разных потоков. В то время, пока один поток приложения принимает данные, другой поток может параллельно с первым передавать данные — как при разговоре по телефону, когда вы можете слушать и говорить одновременно. Данный режим обработки больше импонирует идее многозадачности Windows. Но за все надо платить: для реализации этого режима обработки требуется в два раза больше написанного кода, вдобавок, умения писать многопоточные программы. Какой режим выбрать — решайте сами. Но если уж разбираться в работе порта, то разбираться «по-взрослому», до конца, а потому и рассмотрим более сложный вариант — асинхронную обработку.

На практике открытие порта для асинхронного режима обработки из программы на Delphi выглядит примерно так:

OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, 0);

if hPort = INVALID\_HANDLE\_VALUE then

raise Exception.Create('Error opening port');

Функция возвращает описатель порта (hPort), который нам потом пригодится для вызова других функций работы с портом. Если в результате открытия порта описатель не получен, то возбуждается исключение с соответствующим текстом ошибки. Открыв порт, мы получаем его в свое распоряжение. Теперь с этим портом может работать только наша программа (точнее, только наш процесс). По окончании работы с портом его следует закрыть, вызвав функцию:

BOOL CloseHandle(

HANDLE hObject

);

В качестве единственного параметра надо передать полученный ранее описатель порта (hPort).

Хоть система при завершении выполнения программы и освобождает все выделенные ей ресурсы (в том числе и порты), хорошим тоном программирования считается собственноручное закрытие портов. Открывать/закрывать порт как будто несложно. Кроме того, нам потребуется программная настройка порта. Думаю, все видели диалог настройки последовательного порта в диспетчере устройств системы. Все эти настройки мы можем произвести программно. Для этих целей используется функция WinAPI:

BOOL SetCommState(

HANDLE hFile,

LPDCB lpDCB

);

hFile — описатель открытого порта.

lpDCB — указатель на структуру DCB.

Основные параметры последовательного порта описываются структурой DCB. Она содержит массу полей, каждое из которых соответствует определенному параметру настройки порта. Мы рассмотрим несколько полей, которые нам нужны:

BaudRate — скорость передачи данных. Возможно указание констант —CBR\_100, CBR\_300, CBR\_600, CBR\_1200, …, CBR\_256000.

Parity — схема контроля четности. Может содержать одно из следующих значений: EVENPARITY, MARKPARITY, NOPARITY, ODDPARITY, SPACEPARITY.

ByteSize — число информационных бит в передаваемых и принимаемых байтах.

StopBits — количество стоповых бит. Может быть ONESTOPBIT, ONE5STOPBIT, TWOSTOPBIT.

Чтобы не заполнять структуру DCB вручную, ее можно заполнить информацией о текущем состоянии порта вызовом функции GetCommState(), затем изменить необходимые поля и установить настройки вызовом функции SetCommState(). Настройку порта желательно производить сразу после его открытия. На Delphi это выглядит так:

var

Dcb: TDcb;

…

if not GetCommState(hPort, Dcb) then

raise Exception.Create('Error setting port state');

Dcb.BaudRate := CBR\_9600;

Dcb.Parity := NOPARITY;

Dcb.ByteSize := 8;

Dcb.StopBits := ONESTOPBIT;

if not SetCommState(hPort, Dcb) then

raise Exception.Create('Error setting port state');

Еще одна операция, которая нам понадобится сразу после открытия порта — его сброс.

BOOL PurgeComm(

HANDLE hFile,

DWORD dwFlags

);

Вызов этой функции очищает очередь приема/передачи и завершает все находящиеся в ожидании запросы ввода/вывода.

hFile — описатель открытого порта.

dwFlags — производимые действия в виде набора флагов PURGE\_TXABORT, PURGE\_RXABORT, PURGE\_TXCLEAR, PURGE\_RXCLEAR.

Пример на Delphi:

if not PurgeComm(hPort, PURGE\_TXCLEAR or PURGE\_RXCLEAR) then

raise Exception.Create('Error purging port');

На этом подготовительная фаза заканчивается, и можно приступать непосредственно к приему/передаче данных. Прием данных у нас будет происходить по событийной схеме; программа будет ожидать прием одного или нескольких символов (байт). Для перевода порта в этот режим необходимо вызвать функцию SetCommMask() с флагом EV\_RXCHAR:

if not SetCommMask(hPort, EV\_RXCHAR) then

raise Exception.Create('Error setting port mask');

Прием и передача данных выполняется функциями ReadFile() и WriteFile(), то есть теми же самыми функциями, которые используются для работы с дисковыми файлами. Вот их описание:

BOOL ReadFile(

HANDLE hFile,

LPVOID lpBuffer,

DWORD nNumberOfBytesToRead,

LPDWORD lpNumberOfBytesRead,

LPOVERLAPPED lpOverlapped

);

BOOL WriteFile(

HANDLE hFile,

LPCVOID lpBuffer,

DWORD nNumberOfBytesToWrite,

LPDWORD lpNumberOfBytesWritten,

LPOVERLAPPED lpOverlapped

);

hFile — описатель открытого порта.

lpBuffer — адрес буфера.

nNumberOfBytesToRead/nNumberOfBytesToWrite — число ожидаемых к приему или предназначенных для передачи байт.

lpNumberOfBytesRead/lpNumberOfBytesWritten — число фактически принятых или переданных байт.

lpOverlapped — адрес структуры OVERLAPPED, используемой для асинхронных операций.

Передача данных является довольно быстрой операцией, поэтому как правило ее выполняют из главного потока приложения. На Delphi это выглядит так:

var

dwWrite: DWORD;

OverWrite: TOverlapped;

WriteBytes: array of Byte;

…

begin

OverWrite.hEvent := CreateEvent(nil, True, False, nil);

if OverWrite.hEvent = Null then

raise Exception.Create('Error creating write event');

…

if (not WriteFile(hPort, WriteBytes, SizeOf(WriteBytes),

dwWrite, @OverWrite))

and (GetLastError <> ERROR\_IO\_PENDING) then

raise Exception.Create('Error writing port');

end;

В данном примере функция WriteFile() выполняет асинхронную запись массива байтов WriteBytes в порт. Она сразу возвращает управление, и запись в порт происходит параллельно с выполнением основного кода потока. Если результат WriteFile() равен False, то это значит, что на момент возврата управления передача массива байтов еще не закончилась. Поэтому код ошибки выполнения WriteFile() в данном случае должен быть равен ERROR\_IO\_PENDING. Переменная OverWrite — overlapped-структура, необходимая для асинхронных операций.

В принципе, вас не должно волновать, когда закончится передача массива байтов. Зато момент приема одного или нескольких символов действительно важен. Поэтому его можно разбить на две части: инициирование приема и определение момента приема с последующим чтением символов. Поскольку при этом приходится считаться с фактором ожидания приема символа, рекомендуется функции приема данных вынести в отдельный поток. Передавать данные можно и из основного потока приложения, поскольку это происходит довольно быстро. А вот событие приема символа будем ожидать в отдельном потоке.

Рассмотрение работы с потоками в Windows, в частности того, как это реализовано в Delphi, выходит за рамки данной статьи. Предполагаю, что читатель встречался или по крайней мере знаком с этим. Скажу лишь, что у любого потока есть главная функция, которая начинает выполняться после его создания. В Delphi для потоков существует класс TThread, а его главная процедура называется TThread.Execute().

Вот так выглядит главная процедура отдельного потока, которая ожидает появление одного или нескольких символов и считывает их:

procedure TReadThread.Execute;

var

ComStat: TComStat;

dwMask, dwError: DWORD;

OverRead: TOverlapped;

Buf: array[0..$FF] of Byte;

dwRead: DWORD;

begin

OverRead.hEvent := CreateEvent(nil, True, False, nil);

if OverRead.hEvent = Null then

raise Exception.Create('Error creating read event');

FreeOnTerminate := True;

while not Terminated do

begin

if not WaitCommEvent(hPort, dwMask, @OverRead) then

begin

if GetLastError = ERROR\_IO\_PENDING then

WaitForSingleObject(OverRead.hEvent, INFINITE)

else

raise Exception.Create('Error waiting port event');

end;

if not ClearCommError(hPort, dwError, @ComStat) then

raise Exception.Create('Error clearing port');

dwRead := ComStat.cbInQue;

if dwRead > 0 then

begin

if not ReadFile(hPort, Buf, dwRead, dwRead, @OverRead) then

raise Exception.Create('Error reading port');

// В Buf находятся прочитанные байты

// Далее идет обработка принятых байтов

end;

end; {while}

end;

В приведенном примере в потоке крутится цикл, тем самым инициируется ожидание события порта вызовом функции WaitCommEvent(), ожидание же самого этого события задается функцией WaitForSingleObject(). Для определения количества принятых символов используется функция ClearCommError(). Когда количество принятых символов (dwRead) известно, непосредственное чтение символов выполняется функцией ReadFile().

Используя вышеописанные выкладки, я написал на Borland Delphi 7 класс TComPort для работы с COM-портами. К классу прилагается пример приложения, использующий его. Для проверки работоспособности программы я просто соединил нуль-модемным кабелем два COM-порта на своем компьютере и запустил два экземпляра программы для каждого порта. Данные передаются через один порт и одновременно принимаются через другой.

Для передачи и приема данных предусмотрены отдельные окна. Формат передаваемых данных — строка. Принимаемые данные представляются в виде массива байт.