ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Томский политехнический университет

ПРОЕКТИРОВАНИЕ CDC – УСТРОЙСТВ НА МИКРОКОНТРОЛЛЕРАХ СО ВСТРОЕННЫМ USB – МОДУЛЕМ

Учебное – методическое пособие

Издательство ТПУ
Томск – 2010

УДК 681.325.5 – 181.48(075.8)
ББК 32.973.26 – 04я73
В751

Авторы
Г.С. Воробьева, В.А. Юрченков, С.М. Мартемьянов

Проектирование CDC – устройств на микроконтроллерах со встроенным USB – модулем: Учебно-методическое пособие / Г.С. Воробьева, В.А Юрченков, С.М. Мартемьянов. – Томск:, 2010. – 72 с.

В пособии кратко рассмотрены физическая и логическая организация одной из наиболее сложной, высокоскоростной современной USB – шины, а также приемы проектирования CDC - устройств на ее основе. Дано описание средств программной разработки в среде Keil и Delphi включая отладку программ как в симуляторах, так и с использованием стандартной инструментальной программы гипертерминал. Пособие содержит примеры рабочих программ, а также вопросы и задания для самопроверки.

Предназначено для бакалавров, магистрантов, аспирантов и инженеров, занимающихся проектированием микропроцессорных систем передачи информации по USB – шине.

УДК 681.325.5 – 181.48(075.8)
ББК 32.973.26 – 04я73

Рецензенты
Доктор технических наук, профессор, зав.кафедрой медицинской и биологической кибернетики
ГОУВПО СибГМУ РОСЗДРАВа
Я.С. Пеккер

Зам.нач.СКБ ОАО «Манотомь», к.т.н.
В.В. Бычков

© Авторы, 2010
© Томский политехнический университет, 2010
© Оформление. Издательство Томского политехнического университета, 2010

Глава 1. **Введение в USB**

Увеличение числа устройств, подключаемых к персональному компьютеру, и, соответственно, развитие внешних интерфейсов привело к довольно неприятной ситуации: с одной стороны, компьютер должен иметь множество различных разъемов, а с другой — большая часть из них не используется. Такая ситуация определяется историческим развитием интерфейсов ПК — каждый интерфейс имел свой специализированный разъем. Например, к последовательному порту можно подключить мышь или модем, к параллельному — принтер или сканер, для клавиатуры стало необходимо иметь два порта — старый клавиатурный и PS/2 и т. д. Более того, к одному порту можно подключить только одно устройство (если не считать подключение "прозрачных" ключей защиты, но это, скорее, исключение). Кроме этой проблемы, многочисленность разнообразных подключений добавляет и другие "радости":

- практически для каждого из устройств необходимо выделение аппаратного прерывания (IRQ);
- большая часть устройств требует наличия внешнего блока питания;
- каждое устройство имеет свой, придуманный разработчиком, протокол обмена, многократно увеличивая необходимое количество драйверов, как в памяти, так и в инсталляции операционной системы;
- конфигурирование огромного числа устройств, многие из которых не поддерживают спецификации Plug and Play, — практически невыполнимая работа для обычного пользователя;
- огромное число разнокалиберных шлейфов, тянущихся от компьютера, превращает его перестановку в сложную проблему.

Естественно, что производители компьютерного "железа" задумались о создании единого и универсального интерфейса. В начале 1996 года была опубликована версия 1.0 нового интерфейса, названного USB (Universal Serial Bus, универсальная последовательная шина), а осенью 1998 — спецификация 1.1, исправляющая проблемы, обнаруженные в первой редакции. Весной 2000 года была опубликована версия 2.0, в которой предусматривалось 40-кратное повышение пропускной способности шины. Так, спецификации 1.0 и 1.1 обеспечивают работу на скоростях 12 Мбит/с и 1,5 Мбит/с, а спецификация 2.0 — на скорости 480 Мбит/с. При этом предусматривается обратная совместимость USB 2.0 с USB 1.x, т. е. "старые" USB 1.x устройства будут работать с USB 2.0

контроллерами, правда, на скорости 12 Мбит/с. Скорость 480 Мбит/с достигается только при одновременном использовании USB 2.0 контроллера и USB 2.0 периферии.

Физическая и логическая организация шины

Обычная архитектура USB подразумевает подключение одного или нескольких USB-устройств к компьютеру, который в такой конфигурации является главным управляющим устройством и называется хостом. Подключение USB-устройств к хосту производится с помощью кабелей. Для соединения компьютера и USB- устройства используется хаб. Компьютер имеет встроенный хаб, называемый корневым хабом (рис. 1)

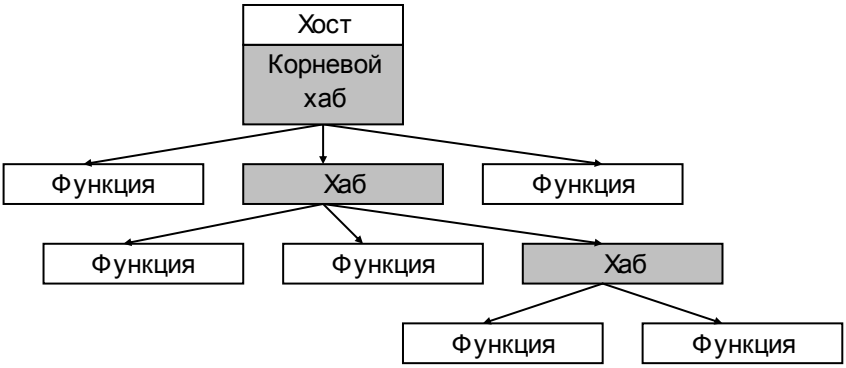


Рис.1. Физическая архитектура USB

Физическая архитектура USB определяется следующими правилами:

- Все USB-устройства подключаются к хосту;
- Физическое соединение устройств между собой осуществляется по топологии многоярусной звезды, вершиной которой является корневой хаб;
- Центром каждой звезды является хаб;
- каждый кабельный сегмент соединяет между собой две точки: хост с хабом или функцией, хаб с функцией или другим хабом;

- к каждому порту хаба может подключаться периферийное USB-устройство или другой хаб, при этом допускается до 5 уровней каскадирования хабов, не считая корневого

Составляющие USB

Для того, чтобы разобраться как работает USB-шина, необходимо определиться с терминологией. Ниже представлены основные составляющие USB-шины.

Хост-контроллер(Host) – это главный контроллер, который входит в состав системного блока компьютера и управляет работой всех устройств на шине USB. На шине USB допускается наличие только одного хоста. Системный блок персонального компьютера содержит один или несколько хостов, каждый из которых, управляет отдельной шиной USB.

Устройство(Device) – может представлять собой хаб, функцию или их комбинацию.

Порт(Port) – точка подключения

Хаб(Hub) – устройство, которое обеспечивает дополнительные порты на шине USB . Другими словами, хаб преобразует один порт во множество портов. Архитектура допускает соединение нескольких хабов(не более 5). Хаб распознает подключение и отключение устройств к портам и может управлять подачей питания на порты. Каждый из портов может быть разрешен или запрещен и сконфигурирован на полную или ограниченную скорость обмена. Хаб обеспечивает изоляцию сегментов с низкой скоростью от высокоскоростных. Хаб может ограничивать ток, потребляемый каждым портом;

Корневой хаб(Root Hub) – это хаб, входящий в состав хоста;

Функция(Function) – это периферийное USB- устройство или его отдельный блок, способный передавать и принимать информацию по шине USB. Каждая функция представляет собой конфигурационную информацию, описывающую возможности периферийного устройства и требования к ресурсам. Перед использованием функция

должна быть сконфигурирована хостом – ей должна быть выделена полоса в канале и выбраны опции конфигурации;

Логическое USB – устройство – устройство представляющее собой набор конечных точек, через которые происходит обмен данными. В качестве USB – устройств выступают хорошо знакомые вам приборы (принтеры, сканеры, плееры, флешки и.т.д.).

Конечные точки(End Point) **–** это буферы типа FIFO в которых хранится вся передаваемая по шине USB информация.

Свойства USB-устройств

Спецификация USB достаточно жестко определяет набор свойств, которые должно поддерживать любое USB устройство:

Адресация – устройство должно отзываться на назначенный ему уникальный адрес и только на него;

Конфигурирование – после включения или сброса устройство должно предоставлять нулевой адрес для возможности конфигурирования его портов;

Передача данных – устройство имеет набор конечных точек для обмена данными с хостом. Для конечных точек, допускающих разные типы передач, после конфигурирования доступен только один из них;

Управление энергопотреблением – любое устройство при подключении не должно потреблять от шины ток, превышающий 100мА. При конфигурировании устройство заявляет свои потребности тока, но не более 500мА. Если хаб не может обеспечить устройству заявленный ток, устройство не будет использоваться;

Приостановка(Suspend) – USB устройство должно поддерживать приостановку, при которой его потребляемый ток не превышает 500мкА. USB – устройство должно автоматически приостанавливаться при прекращении активности шины;

Принципы передачи данных

Механизм передачи данных является асинхронным и блочным. Блок передаваемых данных называется USB-фреймом или USB-кадром и передается за фиксированный временной интервал. Оперирование командами и блоками данных реализуется при помощи логической абстракции, называемой каналом. Внешнее устройство также делится на логические абстракции, называемые конечными точками. Таким образом, канал является логической связкой между хост-контроллером и конечной точкой внешнего устройства. Канал можно сравнить с открытым файлом.

Для передачи команд (и данных, входящих в состав команд) используется канал по умолчанию, а для передачи данных открываются либо потоковые каналы, либо каналы сообщений.

Механизм прерываний

Для шины USB настоящего механизма прерываний (как, например, для последовательного порта) не существует. Вместо этого хост-контроллер опрашивает подключенные устройства на предмет наличия данных о прерывании. Опрос происходит в фиксированные интервалы времени, обычно каждые 1—32 мс. Устройству разрешается посылать до 64 байт данных.

С точки зрения драйвера, возможности работы с прерываниями фактически определяются хост-контроллером, который и обеспечивает поддержку физической реализации USB-интерфейса.

Режимы передачи данных

Пропускная способность шины USB, соответствующей спецификации 1.1, составляет 12Мбит/с. Спецификация 2.0 определяет шину с пропускной способностью 400Мбит/с. Полоса пропускания делится между всеми устройствами, подключенными к шине.

Шина USB имеет три режима передачи данных:

- Низкоскоростной(LS, Low-speed);
- полноскоростной(FS, Full-speed);
- высокоскоростной(HS, High-speed, только для USB2.0).

Логические уровни обмена данными

Спецификация USB определяет три логических уровня с определенными правилами взаимодействия. USB-устройство содержит интерфейсную, логическую и функциональную части. Хост тоже делится на три части: интерфейсную, системную и ПО. Каждая часть отвечает только за определенный круг задач. Логическое и реальное взаимодействие между ними показано на рисунке 2.

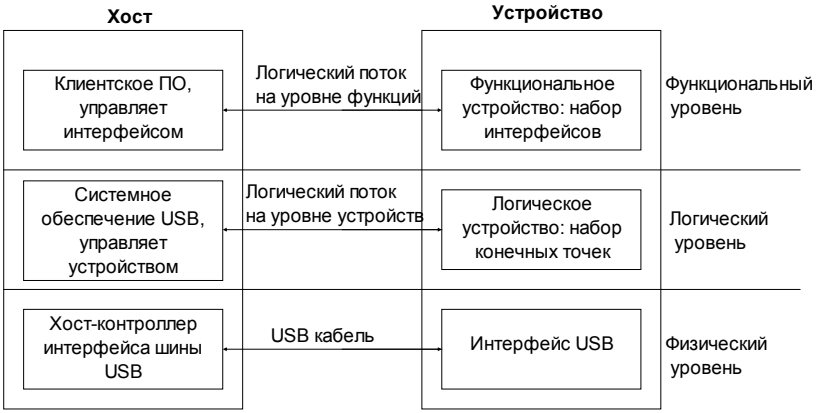


Рис.2. Взаимодействие компонентов USB

Таким образом, операция обмена данными между прикладной программой и шиной USB выполняется путем передачи буферов памяти через следующие уровни:

- *уровень клиентского ПО в хосте:*
    1. Обычно представляется драйвером USB-устройства.
    2. Обеспечивает взаимодействие пользователя с операционной системой с одной стороны и системным драйвером с другой.
- *уровень системного драйвера USB в хосте:*
    1. Управляет нумерацией устройств на шине.
    2. Управляет распределением пропускной способности шины и мощности питания.
    3. Обрабатывает запросы пользовательских драйверов.
- уровень *хост-контроллера интерфейса шины USB:*

1. Преобразует запросы ввода/вывода в структуры данных, по которым выполняются физические транзакции.
2. Работает с регистрами хоста.

Передача данных по уровням

Логическая передача данных между конечной точкой и ПО производится с помощью выделения канала и обмена данными по этому каналу, а с точки зрения представленных уровней передача данных выглядит следующим образом.

1. Клиентское ПО посылает IRP-запросы на уровень системного драйвера USB.
2. Системный драйвер USB разбивает запросы на транзакции по следующим правилам:
    - выполнение запроса считается законченным, когда успешно завершены все транзакции, его составляющие;
    - все подробности обработки транзакций до клиентского ПО не доводятся;
    - ПО может только запустить запрос и ожидать или выполнения запроса, или выхода по тайм-ауту;
    - устройство может сигнализировать о серьезных ошибках, что приводит к аварийному завершению запроса, о чем уведомляется источник запроса.
3. Драйвер контроллера хоста принимает от системного драйвера USB перечень транзакций и выполняет следующие действия:
    - планирует исполнение полученных транзакций, добавляя их к списку транзакций;
    - извлекает из списка очередную транзакцию и передает ее уровню хост-контроллера интерфейса шины USB;
    - отслеживает состояние каждой транзакции вплоть до ее завершения.
4. Хост-контроллер интерфейса шины USB формирует кадры.
5. Кадры передаются последовательной передачей бит по методу, называемому NRZI(метод возврата к нулю с инвертированием единиц).

Таким образом, можно сформировать следующую упрощенную схему (Рис.3):

- каждый кадр состоит из наиболее приоритетных посылок, состав которых формирует драйвер контроллера хоста;
- каждая передача состоит из одной или нескольких транзакций;
- каждая транзакция состоит из пакетов;
- каждый пакет состоит из пакетов;
- каждый пакет состоит из идентификатора пакета, данных и контрольной суммы.

Типы передачи данных

Спецификация шины определяет четыре различных типа передачи данных для конечных точек (табл.1):

- ***управляющие передачи*** – используются хостом для конфигурирования устройства во время подключения для управления устройством и получения статусной информации в процессе работы. Протокол обеспечивает гарантированную доставку таких посылок. Длина поля данных управляющей посылки не может превышать 64 байта на полной скорости и 8 на байтов на низкой. Для таких посылок хост гарантированно выделяет 10% полосы пропускания;
- ***передача массивов данных*** – применяется при необходимости обеспечения гарантированной доставки данных от хоста к функции или от функции к хосту, но время доставки не ограничено. Такая передача занимает всю доступную полосу пропускания шины. Пакеты имеют поле данных размером 8, 16, 32 или 64 байт. Приоритет у таких передач самый низкий, они могут приостанавливаться при большой загрузке шины. Допускаются только на полной скорости передачи. Такие посылки используются, например, принтерами или сканерами;
- ***передача по прерываниям*** – используются в том случае, когда требуется передавать одиночные пакеты данных небольшого размера. Каждый пакет требуется передать за ограниченное время. Операции передачи носят спонтанный характер и должны обслуживаться не медленнее, чем того

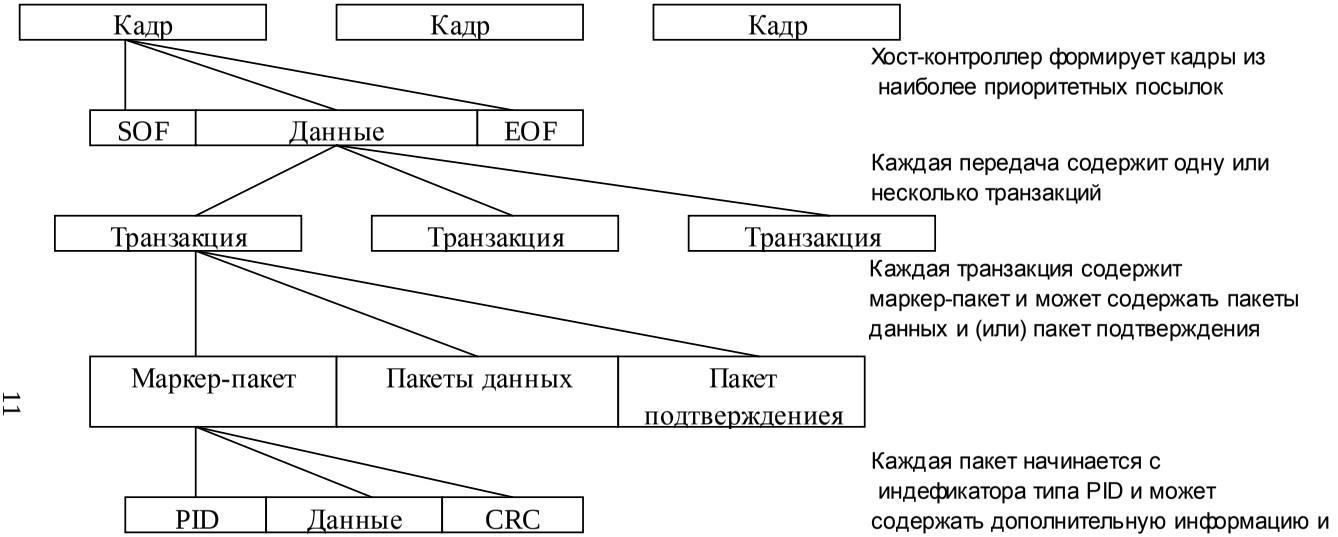


Рис.3. Общая схема составляющих USB – протокола

требует устройство. Поле данных может содержать до 64 байтов при передаче на полной скорости и до 8 байтов на низкой. Предел времени обслуживания устанавливается в диапазоне 1-255мс для полной скорости и 10-255мс – для низкой. Такие передачи используются в устройствах ввода например, как мышь и клавиатура. ***Именно этот тип передачи мы будем использовать в нашем проекте;***

- ***Изохронные передачи*** – применяются для обмена данными в «реальном времени», когда на каждом временном интервале требуется передавать строго определенное количество данных, но доставка данных не гарантирована (передача данных ведется без повторения при сбоях, допускается потеря пакетов). Такие передачи занимают предварительно согласованную часть пропускной способности шины и имеют заданную задержку доставки. Изохронные передачи обычно используются в мультимедийных устройствах для передачи аудио – и видеоданных. например, цифровая передача голоса. Изохронные передачи разделяются по способности синхронизации конечных точек – источников или получателей данных – с системой: различают асинхронный, синхронный и адаптивный классы устройств, каждому из которых соответствует свой тип канала USB.

Таблица 1

Типы передач по шине USB

Тип передачи	**Направление**	**Частота запуска**	**Гарантия доставки**
Управляющие посылки	Двунаправленные	Не гарантируется	Есть
Изохронные	Однонаправленные	Каждые 1мс	Нет
Передача по прерываниям	Однонаправленные	Определяется частотой опроса	Есть
Передача массивов данных	Двунаправленные	Не гарантируется	Есть

Все операции по передаче данных инициируются только хостом независимо от того, принимает ли он данные или пересылает в

периферийное USB –устройство. Все невыполненные операции хранятся в виде четырех списков по типам передач. Списки постоянно обновляются новыми запросами. Планирование операций по передаче информации в соответствии с упорядоченными в виде списков запросами выполняется хостом с интервалом в один кадр. Обслуживание запросов выполняется по следующим правилам:

- наивысший приоритет имеют изохронные передачи;
- после обработки всех изохронных передач система переходит к обслуживанию передач прерываний;
- в последнюю очередь обслуживаются запросы на передачу массивов данных;
- по истечении 90% указанного интервала хост автоматически переходит к обслуживанию запросов на передачу управляющих команд независимо от того, успел ли он полностью другие три списка или нет.

Выполнение этих правил гарантирует, что управляющим передачам всегда будет выделено не менее 10% пропускной способности шины USB. Если передача всех управляющих пакетов будет завершена до истечения выделенной для них доли интервала планирования, то оставшееся время будет использовано хостом для передачи массивов данных. Таким образом:

- изохронные передачи гарантированно получают 90% пропускной способности шины;
- передачи прерываний занимают оставшуюся часть этой доли;
- под передачу данных большого объема выделяется все время, оставшееся после изохронных передач и передач прерываний (по-прежнему в рамках 90% пропускной способности);
- управляющим передачам гарантируется 10% пропускной способности шины;
- если передача всех управляющих пакетов будет завершена до завершения выделенного для них 10 – процентного интервала, то оставшееся время будет использовано для передачи файлов большого объема.

Кадры

Любой обмен по шине USB инициируется хостом. Он организует обмены с устройствами согласно своему плану распределения ресурсов.
Контроллер циклически (с периодом 1,0 +/-0,0005см) формирует кадры, в которые укладываются все запланированные передачи (рис.3). Каждый кадр начинается с посылки маркер пакета SOF (Start Of Frame, начало кадра), который является синхронизирующим сигналом для всех устройств, включая хабы. В конце каждого кадра выделяется интервал времени EOF (End Of Frame, конец кадра) на время которого хабы запрещают передачу по направлению к контроллеру. Если хаб обнаружит, что с какого-то порта в это время ведется передача данных, то он отключит этот порт.
В режиме высокоскоростной передачи пакеты SOF передаются в начале каждого микрокадра (период 125+/-0,0625мкс). Хост планирует загрузку кадров так, чтобы в них всегда находилось место для наиболее приоритетных передач, а свободное место кадров заполняется низкоприоритетными передачами больших объемов данных. Спецификация USB позволяет занимать под периодические транзакции (изохронные и прерывания) до 90% пропускной способности шины.

Каждый кадр имеет свой номер. Хост оперирует 32-битным счетчиком, но в маркере SOF передает только младшие 11 бит. Номер кадра циклически увеличивается во время EOF.

Для изохронной передачи важна синхронизация устройств и контроллера. Есть три варианта синхронизации:

- синхронизация внутреннего генератора устройства с маркерами SOF;
- подстройка частоты кадров под частоту устройства;
- согласование скорости передачи (приема) устройства с частотой кадров.

В каждом кадре может быть выполнено несколько транзакций, их допустимое число зависит от скорости, длины поля данных каждой из них, а также от задержек, вносимых кабелями, хабами и USB – устройствами. Все транзакции кадров должны быть завершены до момента времени EOF. Частота генерации кадров может немного варьироваться с помощью специального регистра хоста, что позволяет подстраивать частоту для изохронных передач.

Подстройка частоты кадров контроллера возможна под частоту внутренней синхронизации только одного USB – устройства.

Конечные точки

Конечная точка (endpoint) – это часть USB – устройства, которая имеет уникальный идентификатор и является получателем или отправителем информации, передаваемой по шине USB. Проще говоря, это буфер, сохраняющий несколько байт. Обычно это блок данных в памяти или регистр микроконтроллера. Данные, хранящиеся в конечной точке, могут быть либо принятыми данными, либо данными, ожидающими передачу. В хосте также присутствует буфер для приема и передачи данных, но отсутствуют конечные точки. Конечная точка имеет следующие основные параметры:

- частота доступа к шине;
- допустимая величина задержки обслуживания;
- требуемая ширина пропускания канала;
- номер;
- способ обработки ошибок;
- максимальный размер пакета, который может быть принят или отправлен;
- используемый тип посылок;
- направление передачи данных.

Любое USB – устройство имеет конечную точку с нулевым номером или нулевую точку (endpoint zero). Эта точка позволяет хосту опрашивать USB – устройство с целью определения его типа и параметров, а также выполнять его инициализацию и конфигурирование.

Кроме нулевой точки, USB – устройства обычно имеют дополнительные конечные точки, которые используются для обмена данными с хостом. Дополнительные точки могут работать либо только на прием данных от хоста (входные точки, IN), либо только на передачу данных хосту (выходные точки, OUT). Число дополнительных конечных точек определяется режимом передачи. Для низкоскоростных USB – устройств допускается наличие одной или двух дополнительных конечных точек, а для высокоскоростных – до 15 входных и 15 выходных дополнительных точек.
Нулевая точка становится доступна после того, как USB – устройство подключено к шине, включено и получило сигнал сброса по шине

(bus reset). Все остальные конечные точки после включение питания или сброса находятся в неопределенном состоянии и недоступны для работы до тех пор, пока хост не выполнит процедуру конфигурирования.

В нашем проекте мы будем использовать две дополнительные конечные точки: одну – как буфер приемника, в который будут поступать данные, а другую – как буфер передатчика, в который мы будем записывать данные для дальнейшей их передачи в компьютер.

Практическая реализация CDC – устройств

При разработке USB – устройств, в начале необходимо решить ряд задач:

- выбор микроконтроллера со встроенным USB – модулем;
- выбор типа реализуемого устройства в соответствии со спецификацией USB;
- выбор среды разработки ПО для микроконтроллера;
- выбор среды разработки ПО для компьютера.

На сегодняшний день существует довольно много фирм производящих микроконтроллеры со встроенным модулем USB. Мы будем использовать микроконтроллер фирмы Atmel (AT89C5131), который имеет встроенный, полноценный модуль, поддерживающий стандарт USB2.0. Также, немаловажным фактором является то, что, данный микроконтроллер построен на хорошо известном вам ядре MCS-51.

Все устройства которые подключаются к USB – шине делятся на классы которые объединяют в себе устройства имеющие схожее функциональное назначение. Например, устройства хранения данных, к которым относятся большинство аудио плееров, флеш–карт, всевозможных жестких дисков и фотоаппаратов. Для реализации нашего USB устройства мы будем использовать специальный класс коммуникационных устройств – Communication Device Class(CDC - класс). Данный класс позволяет работать с USB – устройством как с обычным COM – портом. Что позволяет использовать разработанные ранее программы для работы с USB – устройством. Таким образом, получается, что для прикладной программы пользователя вообще нет разницы с каким портом

работать с обычным или виртуальным, иными словами прикладная программа даже не знает, что она работает с виртуальным COM – портом, а не с настоящим.

В качестве среды разработки была выбрана среда Keil UV3. Данная среда позволяет создавать и компилировать проекты, к которым можно подключать дополнительные заголовочные файлы, файлы уже готовых библиотек, а также самому создавать библиотеки. Помимо всего прочего при помощи данной программы можно сразу же загружать в микроконтроллер прокомпилированный проект. Демонстрационная версия программы всегда доступна на официальном сайте фирмы Keil (www.keil.com). Демонстрационная версия программы компилирует файлы объемом не более 2Кб, однако, этого вполне достаточно, для того чтобы создавать небольшие программы для обучающих целей.

Однако для работы с USB – устройством не достаточно написать программу только для микроконтроллера. Необходимо также создать пользовательскую программу на компьютере, которая будет управлять устройством, отображать принимаемые данные с устройства такие, например, как давление, температура, скорость, значение напряжения, тока и т. д. В качестве такой среды была выбрана среда разработки Delphi 7. При помощи которой можно довольно легко и быстро разрабатывать визуальные приложения для компьютера. Огромный набор встроенных объектов освобождает разработчика от рутинной работы по созданию меню, графиков, баз данный, всплывающих окон и многих других вещей.

Таким образом, для практической реализации CDC – устройства нам понадобятся ряд инструментальных программ, которые мы и рассмотрим в следующей главе.

Глава 2. Разработка и отладка программ

Целью данной главы является ознакомление со средой разработки Keil UV3 и программой внутрисистемного программирования FLIP. Изучение основных функций и назначения основных пунктов меню. Получение практических навыков по работе с инструментальными средствами отладки микропроцессорных систем.

Общие сведения

Существует множество инструментальных средств, предназначенных для облегчения и интенсификации труда разработчиков. Основное назначение любого отладчика – помочь разработчику разобраться на программно – аппаратном уровне в процессах происходящих в устройстве, а затем добиться желаемых характеристик функционирования. Кроме этого инструментальные средства могут обладать дополнительными возможностями, которые избавляют разработчика от множества утомительных процедур. В данном пособии рассмотрена интегрированная среда разработки Keil UV3. Тестовая версия данной программы была взята с официального сайта фирмы Keil: www.keil.com.

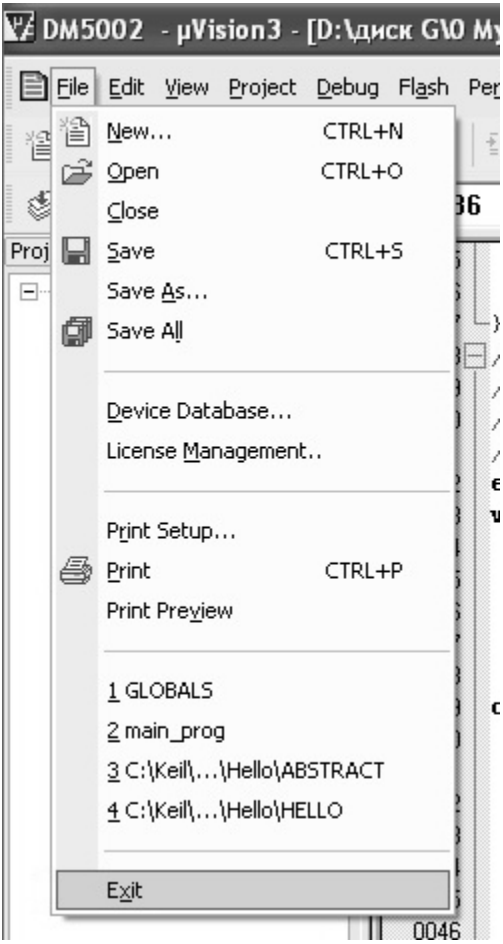
Общее описание внутрисхемного отладчика и Keil UV3

Keil UV3 – интегрированная среда разработки, представляет собой программный продукт, работающий под управлением операционной системы Windows и предназначенный для написания, отладки и оптимизации программ для контроллеров MCS-51.

Keil UV3 включает в себя:

Project Manager – менеджер проекта, позволяет создавать, сохранять текущие проекты;
Editor – текстовый редактор для написания программы;
Simulator – симулятор для проверки функционирования программы;
Keil Assembler – язык программирования;
Linker – утилита объединения всех файлов проекта;
Compilar– утилита преобразования программы в машинные коды.

Всплывающее меню File



Во всплывающем меню File(рис.4) доступны следующие опции: New, Open, Close, Save, Save As, Save All, Print Setup, Print, Device Database, License Manager. Эти опции предназначены для создания, сохранения, открытия и закрытия файлов.

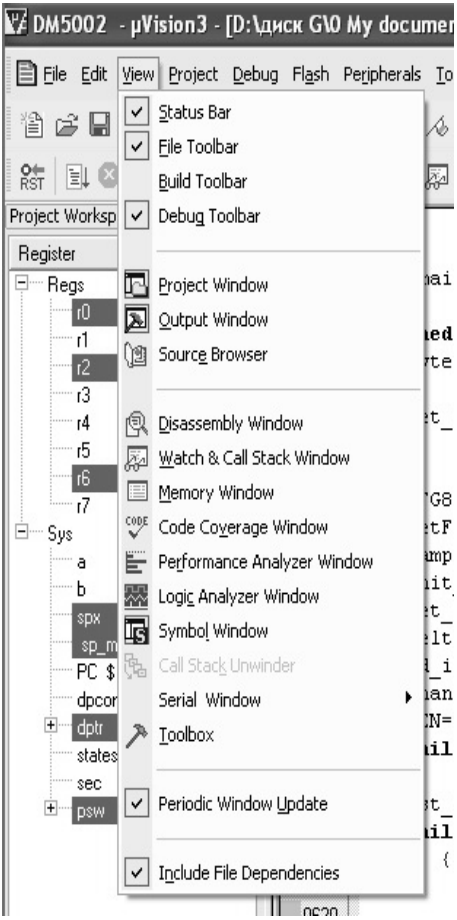
Рис.4. Всплывающее меню File

Всплывающее меню View

Во всплывающем меню View(рис.5) доступны следующие опции: Status Bar, File Toolbar, Build Toolbar, Debug Toolbar, Project Window, Output Window, Source Browser. Остальные пункты данного меню доступны только в режиме симулятора, который включается

следующим образом: Debug>Start/Stop Debug session, либо сочетанием клавиш CTRL+F5.

- Disassembly Window – окно, отображающее команды на языке ассемблер.



- Watch & Call Stack Window – окно для просмотра текущего значения локальных и глобальных переменных.

- Memory Window – окно просмотра содержимого ячеек оперативной памяти.

- Code Coverage – окно для просмотра текущего состояния выполняемого кода. Здесь можно посмотреть сколько инструкций занимает каждая подпрограмма, какая подпрограмма или прерывание сейчас выполняются и сколько процентов текущей подпрограммы выполнено.

- Serial Window – окно для работы с UART

Рис.5. Всплывающее меню View

Создание нового проекта

1. Выберете из меню Project>New uVision Project.

2. Затем, в появившемся меню (рис.6), следует указать имя файла (латинскими буквами), и нажать кнопку «Сохранить». После чего файл проекта будет сохранен в указанном месте.

Для сохранения файла проекта следует завести отдельную папку (***название папки тоже не должно содержать русских символов***). В этой папке будут сохраняться все рабочие файлы проекта, в том числе и сам HEX – файл.

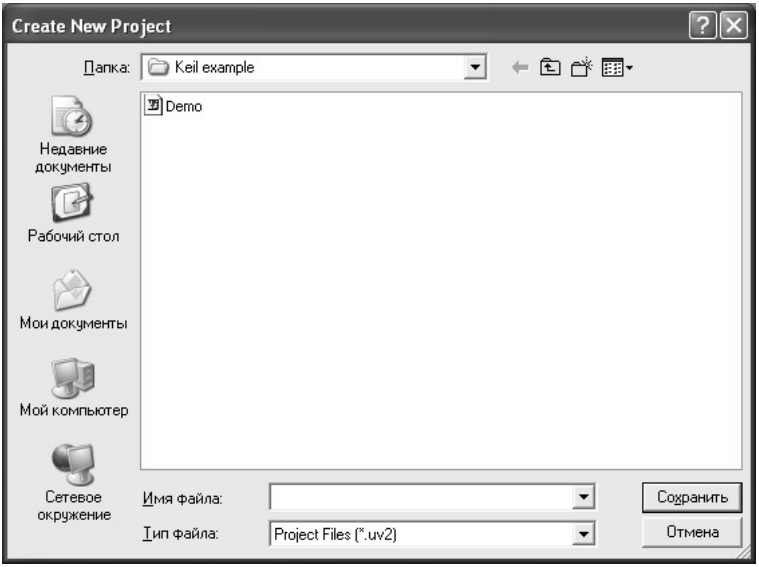


Рис.6. Окно сохранения проекта

3. После этого появится окно для выбора используемого в проекте микроконтроллера (рис.7). Сначала необходимо выбрать фирму изготовитель, а за тем и сам тип микроконтроллера. После чего нажать клавишу ОК.

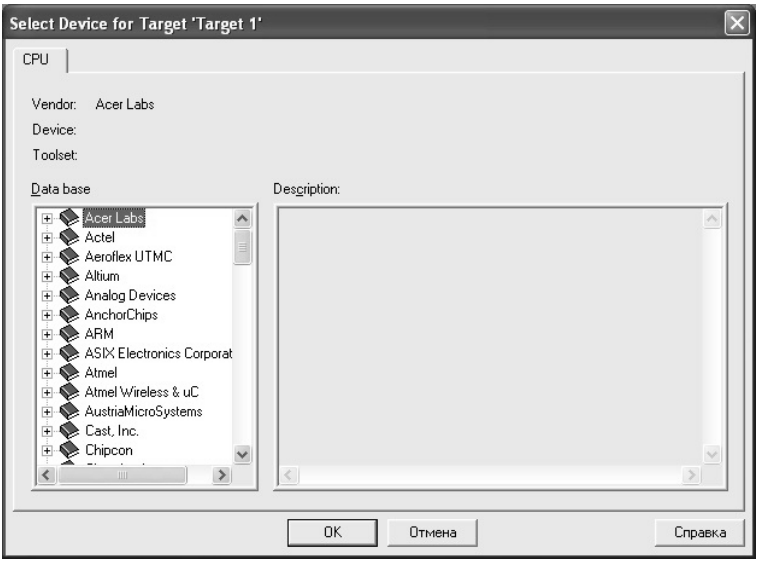


Рис.7. Окно выбора используемого в проекте микроконтроллера

4. Следующим шагом программа предложит скопировать стандартный Startup код для выбранного микроконтроллера в папку с проектом и добавить данный файл к проекту. ***Следует нажать кнопку «Да».***

5. После этого в окне Project Workspace(данное окно находится в левой части монитора, сразу под панелями инструментов см. рис. 8) появится папочка с названием *target1*. Чтобы раскрыть все дерево каталогов следует нажать на плюсик слева от папки.

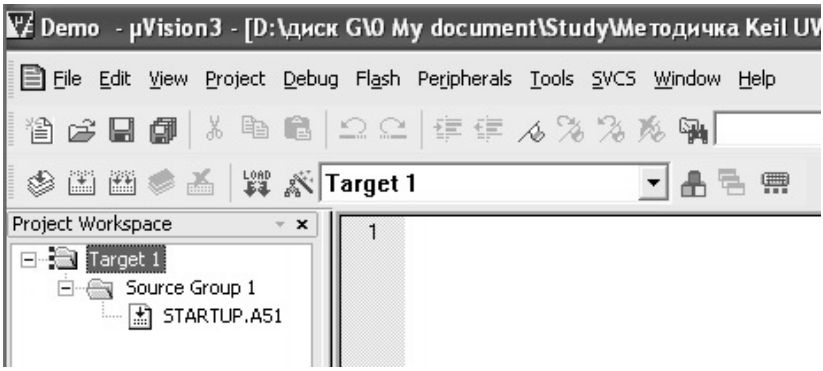


Рис.8. Окно работы с проектом

6. Следующим шагом необходимо создать файл, где и будет находиться основная программа, написанная на языке С. Для этого необходимо зайти в меню File и выбрать пункт New. После чего в окне программы появится пустая белая форма. Для того чтобы ее сохранить необходимо снова зайти в меню File и выбрать пункт Save us. В открывшемся окне следует указать имя файла с расширением ***.c (например main.c), и затем нажать кнопку “Сохранить” (***Важно!!!. Чтобы сохраняемый файл находился в той же папке, что и проект***).

7. Для того чтобы программа смогла откомпилировать данный файл, необходимо его добавить к проекту. Для этого необходимо в окне Project Workspace выделить папку с названием “Source Group 1” (рис. 9), нажать на правую клавишу мышки и в появившемся меню выбрать Add Files to Group “Source Group 1”

8. Затем следует выбрать файл, сохраненный на этапе 6 и нажать кнопку Add. (рис.10). Теперь данный файл добавлен в проект, и с ним можно работать.

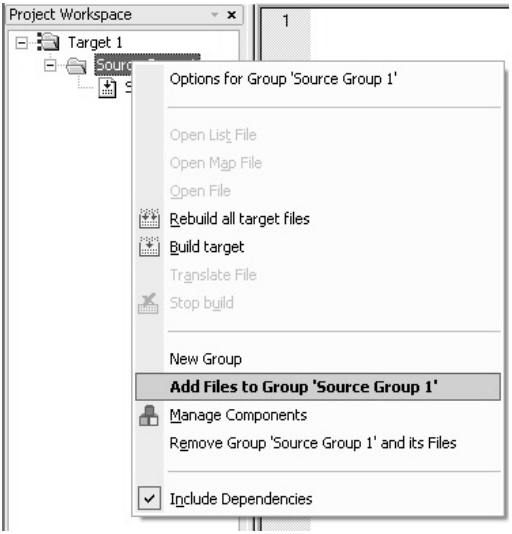


Рис.9. Добавление файлов к проекту

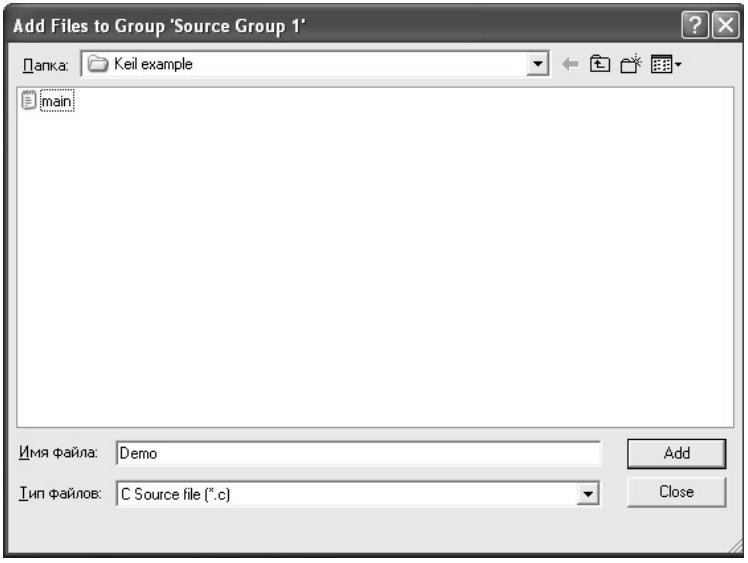


Рис.10. Выбор файлов для добавления к проекту

Написание первой программы на С и компиляция ее в Hex-файл

К данному этапу работы следует приступать только после того, как были выполнены все изложенные выше действия или, другими словами, был создан проект в среде Keil UV3.

Перед тем как приступить к написанию программы, сначала разберемся, что и зачем мы будем писать. Любая программа сколь сложной или простой она не была бы, обязательно имеет основную функцию или основной поток. Эта программа, которая замкнута в “вечный цикл” (т.е. она выполняется все время, пока работает микроконтроллер). Самый простой пример такой программы приведен ниже:

```
void main(void)
{
	for(;;)
	{
	}
}
```

В данной программе слово Main указывает на то, что это основная программа. Любая основная программа обязательно должна содержать вечный цикл. Если вся система работает по прерываниям, то данный цикл допускается делать пустым, как приведено выше.
Какой бы короткой не была эта программа, но это уже программа и ее можно загрузить в контроллер. Но так как контроллер понимает только шестнадцатеричные коды нашу программу необходимо откомпилировать и получить HEX-файл, который и будет загружаться в микроконтроллер.

Для компиляции проекта необходимо нажать Project>Rebuild all targets files. Или нажать на соответствующую иконку, расположенную на панели инструментов над окном Project Workspace.

После выполнения компиляции в нижней части экрана появится сообщение о результатах компиляции (рис.11). В случае успешной компиляции сообщение должно содержать нулевое количество ошибок ("Demo" - 0 Error(s), 0 Warning(s)), как показано на рисунке. Если в проекте имеются предупреждения (Warning(s)), Hex-файл все равно будет создан. Однако, в этом случае, необходимо обратить внимание на данные предупреждения, они будут выведены в этом же окне, и проанализировать их (возможно в программе есть некоторые

неточности, которые могут привести к неправильной работе программы).

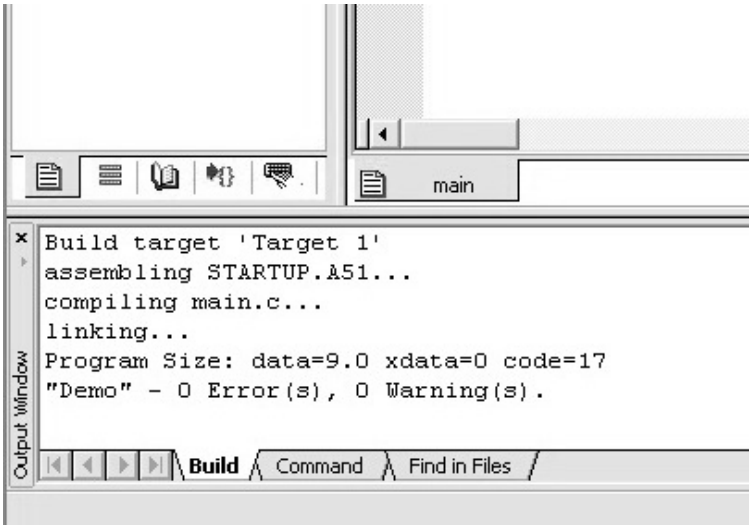


Рис.11.Окно состояния компиляции проекта

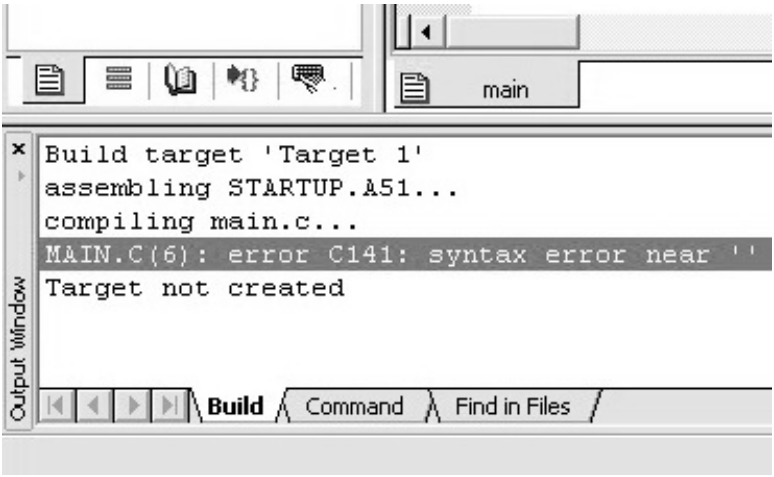


Рис.12.Окно состояния компиляции проекта

В случае, когда в исходном коде допущены серьезные ошибки, то компилятор выдаст следующее сообщение (рис. 12). В данном сообщении будет указана ошибка (ошибки). Для того чтобы, посмотреть в каком месте исходного кода они произошли необходимо щелкнуть два раза левой кнопкой мышки по строчке с ошибкой, в окне (Output Window). После чего, мигающий курсор переместится в то место, где была допущена ошибка.

Работа с симулятором Keil UV3

Для работы с симулятором необходимо написать программу, которая будет выполнять какие-нибудь операции. Допустим, нам необходимо сложить два числа находящихся в оперативной памяти, и результат сохранить в другой ячейке оперативной памяти. Для этого нам необходимо объявить три глобальные переменные, находящиеся в оперативной памяти. Для того чтобы более четко структурировать нашу программу создадим еще один файл (по методике изложенной в пункте 6, с той лишь разницей, что расширение создаваемого файла необходимо указать ***.h). Это так называемый заголовочный файл. По сути дела это тоже программа, точнее ее часть, в которой, как правило, прописываются все подпрограммы, объявляются переменные и т. д. Данные файлы служат для более наглядного представления программного кода.

Для того чтобы подключить данный файл к проекту необходимо прописать его в окне основной программы следующим образом (#include <global.h>), где global – это название созданного на предыдущем этапе файла. В данном файле мы будем прописывать все глобальные переменные, используемые в нашей программе.

И так, после создания файла займемся непосредственно объявлением переменных. Любые переменные объявляются следующим образом. Сначала указывается тип памяти, в которой будет создана переменная, затем указывается тип переменной и только после этого указывается имя самой переменной. В скобках приведен пример объявления трех переменных, расположенных в расширенной оперативной памяти. (*xdata unsigned char a,b,c;)*
xdata – указывает на тип памяти.
unsigned char – указывает на тип данных. В данном случае это беззнаковое однобайтное число.
a,b,c – имена самих переменных.

После объявления переменных займемся написанием самой программы. Вся программа будет расположена в вечном цикле for(;;). Для наглядности выполнения программы все переменные в конце цикла будем обнулять.

```
#include <global.h>
void main(void)
{
	for(;;)
	{
	a=10; b=54; c=a+b; a=0; b=0; c=0;
	}
}
```

После написания программы, ее нужно откомпилировать, для чего снова выбираем Project>Rebuild all targets files. После успешной компиляции запускаем симулятор(Debug>Start/Stop debug session).

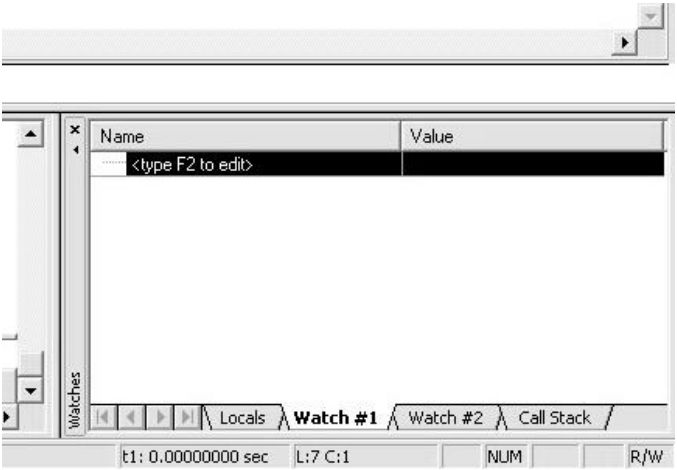


Рис. 12.Окно просмотра переменных

Для того чтобы посмотреть текущее значение наших переменных необходимо выбрать (view> Watch & Call Stack Window). Это окно появится в нижней правой части экрана(рис. 12). Далее, в этом окне необходимо выбрать вкладку Watch #1. Для того чтобы посмотреть значения интересующих нас переменных необходимо нажать на клавишу F2 и добавить необходимые переменные (как показано на рисунке 13).

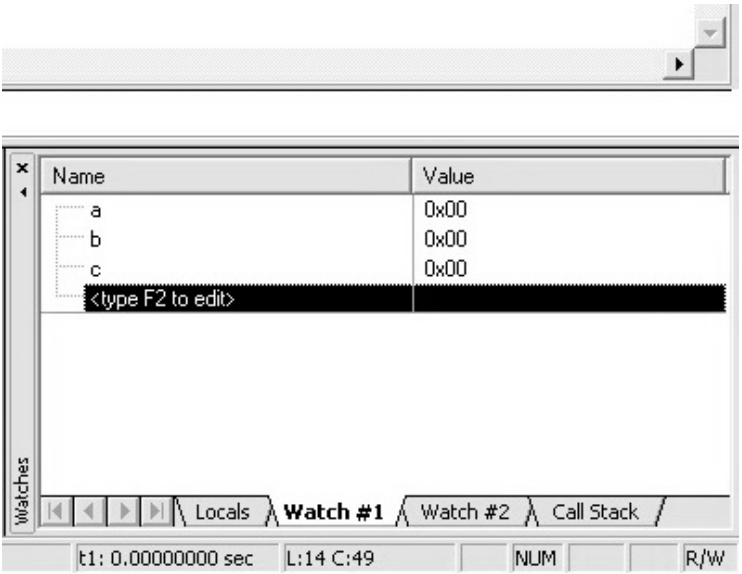


Рис.13.Окно просмотра переменных

Программное обеспечение внутрисистемного программирования FLIP

Программа для внутрисистемного программирования FLIP работает под управлением операционной системы Windows 98/2000, NT или XP. Программа FLIP позволяет вести внутрисистемное программирование MCS-51 Flash микроконтроллеров через RS-232 интерфейс, либо USB. Регулярно выходят обновленные версии данной программы, которые доступны на официальном сайте компании Atmel www.atmel.com. Программа поддерживает большинство современных микроконтроллеров с архитектурой MCS-51.

В настоящее время доступна версия программы FLIP, предназначенная для работы под управлением ОС Linux, а также остальных операционных систем.

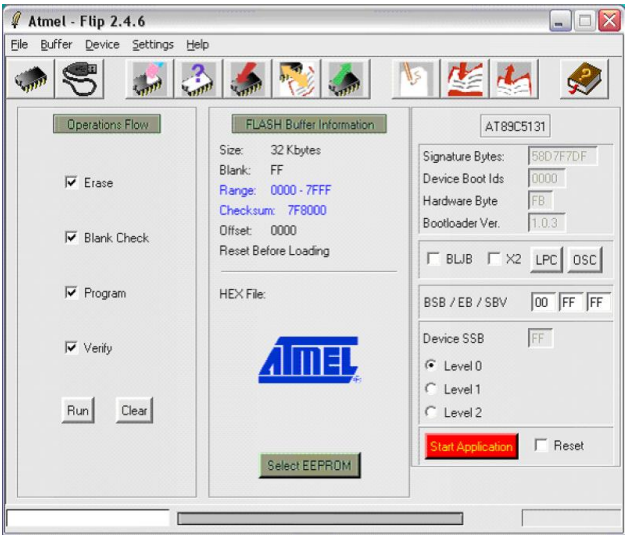


Рис.16. Основное окно программы

Работа с программным обеспечением Atmel FLIP

Для начала вам необходимо настроить программатор для работы с вашим микроконтроллером. Выберете меню Device, затем подменю Select (рис. 15).

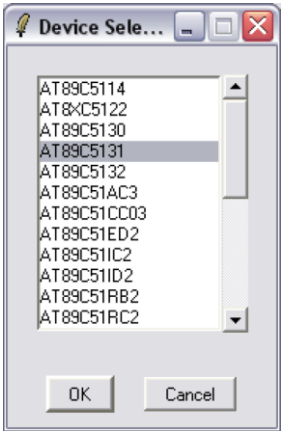


Рис.17. Выбор контроллера

У Вас станет, доступна кнопка и меню установки соединения Communication» (горячая клавиша F3). Выберете в нём порт USB, появится следующее окно (рис. 17):

Рис.18. Окно подключения

Затем выполните следующую последовательность действий в указанном порядке:

1. Отсоединить от устройства кабель USB.
2. При нажатой кнопке S2 подключите кабель USB.
3. Отпустите кнопку S2.

Устройство входит в режим программирования, если при сбросе или включении питания нажата кнопка S2 или установлена перемычка J2. После включения кнопку нужно отпустить, так как вывод, к которому она подключена, является выходом микроконтроллера.

Выберете “Open”(рис.18), т.е. открыть, при правильно подключенной плате, у Вас станут доступными меню управления программированием микроконтроллера. Теперь Вы можете приступать к программированию микроконтроллера.

Если у Вас появится следующее сообщение, то проверьте подключение платы к USB порту и правильную установку перемычек на плате. Так же не забудьте, что не всегда программа правильно работает при подключенной USB мышке или клавиатуре.

Рис.19. Окно предупреждения

После корректного подключения станут доступными следующие кнопки управления:

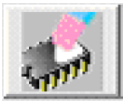
- Запись в буфер HEX – файла.


- Считывание из буфера HEX – файла.


- Ручная корректировка HEX – файла.


- Чтение памяти микроконтроллера.


- Запись в память микроконтроллера.


- Стирание памяти микроконтроллера.

Все кнопки дублируются в меню.

- Далее выберете в меню File – Open File и загрузите файл программы для микроконтроллера с расширением HEX.
- Снимите флажок с позиции BLJB и нажмите RUN
- После прохождения процедуры программирования микроконтроллера и завершения проверки, отключите кабель USB.
- После подключения кабеля USB устройство будет готово к работе.

Настройка и конфигурирование USB-модуля

Для работы с USB модулем, необходимо открыть проект(example-usb), который находится на диске. Для этого необходимо запустить программу KeilUV3, ярлык которой находится на рабочем столе. После запуска программы заходим во вкладку Project, и выбираем OpenProject. Данный проект поможет вам разобраться, каким образом отправлять и принимать данные по USB интерфейсу.

И так, перед тем как приступить непосредственно к изучению кода программы, разберемся, каким образом происходит обмен данными между нашим устройством и компьютером. Любое USB устройство содержит внутри себя так называемые конечные точки, которые представляют собой буферы типа FIFO (первый вошел – первый вышел) это означает, что данные загружаются в данный тип памяти последовательно и выгружаются в той же последовательности что и были загружены. Одним словом это просто буферы для хранения всей информации, которая будет передаваться по шине USB. Каждая конечная точка (буфер) является либо входящей, либо исходящей (конечная точка 0 используется для конфигурирования шины и обмена служебной информацией, она может работать в обоих направлениях).

В отличие от шины RS232, где устройство само может передавать какие-либо данные, шина USB имеет строгую иерархию, на вершине которой стоит хост-контроллер (рис. 1). Поэтому если устройству требуется передать какие-либо данные хост-контроллеру или иными словами нашему компьютеру, то ему необходимо занести соответствующие данные в конечную точку, предназначенную для вывода данных, и подождать пока компьютер не заберет эти данные. Время ожидания зависит от того, с какой периодичностью компьютер опрашивает устройства на USB шине, это время, как правило, составляет 1 миллисекунду.

Ниже представлена структура программы и алгоритм работы микроконтроллера (рис. 14) или точнее его главной функции main, о которой упоминалось ранее. В самом начале текста программы идет подключение заголовочных файлов в которых прописаны все переменные и функции для работы нашего USB – устройства (такие как «usb_init(), cdc_init() ит.д.», затем производится инициализация USB-модуля, для этого в начале работы программы вызывается функция UsbInit.

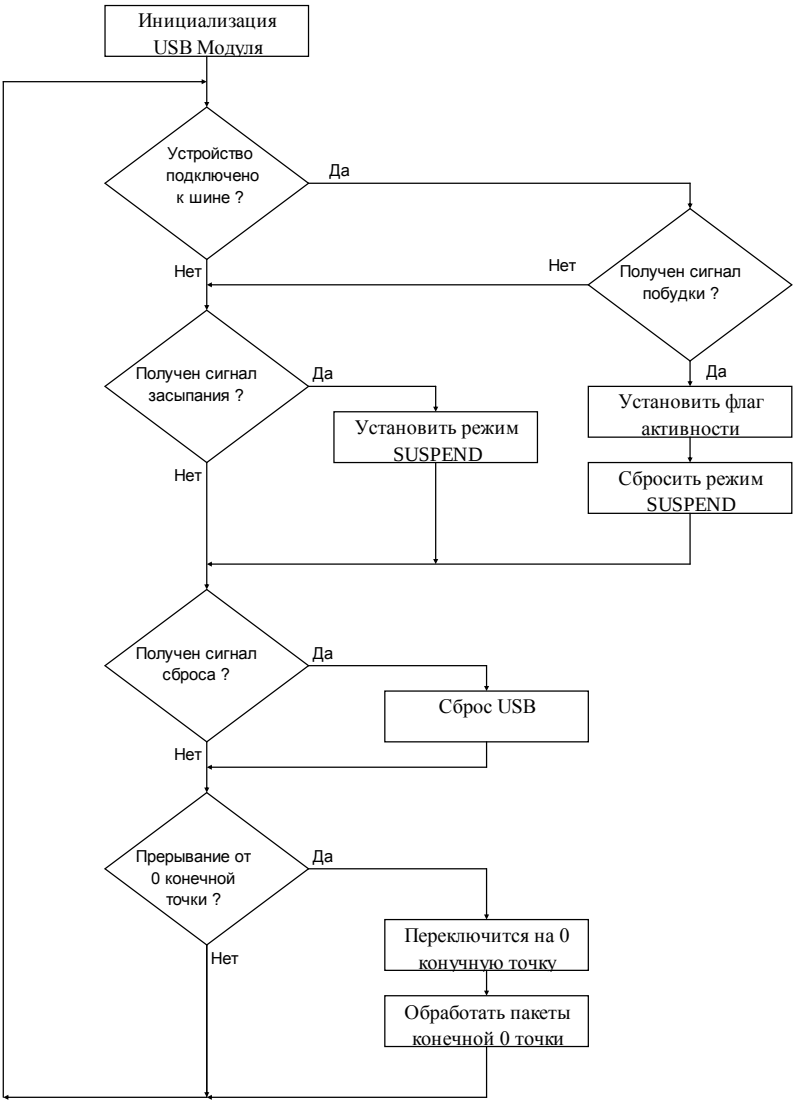
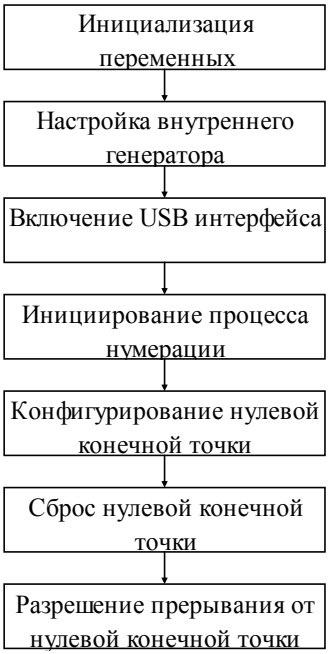


Рис.14. Алгоритм основной функции Main

Эта функция достаточно проста и заключается в выполнении следующей последовательности действий:



1) инициализации переменных;
2) инициализации внутреннего генератора (его функции описаны в документации на микросхему);
3) включении USB- интерфейса;
4) инициировании процесса нумерации (для этого USB – устройство отключается от шины и не ранее чем через 5мс подключается);
5) конфигурировании нулевой конечной точки (разумеется, нулевая точка конфигурируется как управляющая);
6) сбросе нулевой конечной точки(reset);
7*) разрешении прерываний от нулевой конечной точки* (при этом прерывание как таковое не вызывается, т.к. бит IE сброшен, но соответствующие биты в регистре UEPINT выставляются).

Рис.15. Инициализация USB-модуля.

После выполнения этих операций USB – устройство готово к конфигурированию через канал нулевой конечной точки. Через данную конечную точку или проще говоря буфер компьютер получает от нашего устройства все необходимы сведения(название, устройства, класс устройства, скорость работы устройства число конфигураций и т.д.).

После инициализации программа циклически выполняет обработку сигналов побудки, сброса и прерывания от нулевой конечной точки. На этом конфигурирование USB-модуля завершается. Далее программа переходит в режим ожидания прерываний от периферийных устройств. (Приложение D)

```
/* описание регистров AT89C5131 */
//#include "i5131.h"
/* описание констант для доступа к регистрам */
#include "ext_5131.h"
#include "types.h"
#include "const.h"
#include "reg_macr.h"
#include "descript.h"
#include "test.h"
#include "at89c5131.h"
/* функции usb */
#include "usb_util.h"
/* описание USB дескрипторов */
#include "usb_enum.h"
/* функции cdc */
#include "usb_cdc.h"
#include "usb_func.h"
```

MAIN - основная функция программы

```
void main()
{
usb_init();              // выполнить инициализацию USB
cdc_init();              // выполнить инициализацию CDC
Enable_interrupt();      // Разрешить прерывания
usb_connected = FALSE;   // Сброс флага подключения устройства к
                         шине

for (;;)                 // основной цикл программы
{

   if (!usb_connected)   // если устройство отключено от шины
   {
      if (Usb_resume())  // если получен сигнал побудки
      {
      usb_connected = TRUE;  // установить флаг активности

      Usb_clear_suspend_clock();  // сброс режима SUSPEND
      Usb_clear_suspend();
      Usb_clear_resume();
      Usb_clear_sof();
      }
   }
```

```
   else                    // если устройство подключено к шине
   {
      if (Usb_suspend())          // если получен сигнал "засыпания"
      {
      usb_connected = FALSE;      // сбросить флаг подключения устройства
      Usb_clear_suspend();        // сброс режима SUSPEND
      Usb_set_suspend_clock();
      }

      if (Usb_reset())            // если получен сигнал сброса
      {
      Usb_clear_reset();
      }
      if (Usb_sof())              //  если получен сигнал начала кадра
      {
      Usb_clear_sof();
      }

      if (Usb_endpoint_interrupt()) // если обнаружено прерывание от конечной
                                    точки
      {
      Usb_select_ep(0);           // переключиться на 0 конечную точку
         if (Usb_setup_received())  // если получен пакет SETUP
         {
         usb_control_packet_processed();
         }
      }
   }                  // end (else)
}                     // end (for(;;))
}                     // end (Main)
```

Приведенная выше тестовая программа осуществляет прием нескольких байт и отправку их обратно. Для удобства работа программы организована по прерываниям. В самом начале основного тела программы (Main) осуществляется инициализация USB – модуля, и обработка пакетов нулевой конечной точки, после чего контроллер переходит в сконфигурированное состояние. Прием и отправка данных происходит в обработчике прерывания от USB – модуля. (Вектор прерывания 13 см. приложение D) В данном модуле происходит прием данных от компьютера, которые находятся во 2 конечной точке. После чего эти данные сохраняются в массиве info.

(Данный массив прописан в заголовочном файле “ test.h ”) Затем, для того чтобы отправить данные обратно в компьютер, мы переключаемся на первую конечную точку и заполняем ее буфер данными из массива info.

Обработчик прерывания от USB модуля.

```
void  USB_inter(void) interrupt 13              // указываем номер вектора
                                                //прерывания
 {
  Disable_interrupt( );              // запрещаем прерывания


   if (Usb_endpoint_interrupt())       //если обнаружено прерывание от конечной
точки
   {
    Usb_select_ep(0);           // переключиться на 0 конечную точку
    if (Usb_setup_received())     // если получен пакет SETUP
    {
        usb_control_packet_processed(); // обработка пакета setup
    }
                                        // получение данных со 2 конечной точки,
                                        прием //данных от компьютра
    Usb_select_ep(2);          // переключение на 2 конечную точку

 bcount = UBYCTLX;                  // считывание числа принятых байт

    if (bcount > 0)              // если число принятых байт больше нуля
    {
     for (i=0; i<bcount;i++)
     {
      Usb_read(i);                          // считываем данные с буфера конечной
                                            точки 2 и //сохраняем в массиве info
      Usb_clear_rx();           // сбрасываем флаг приема
     }
      Usb_select_ep(1);         // переключаемся на первую конечную точку
        for (i=0; i<bcount;i++)
     {
      Usb_write_byte(info[i]);            // заполняем буфер конечной точки один
                                          принятыми //ранее данными
     }
 Usb_set_tx_ready();          // устанавливаем флаг готовности данных для передачи
```

```
 while (!Usb_tx_complete()) // дожидаемся, пока компьютер заберет данные
    {
    }
Usb_clear_tx_complete();      // очищаем флаг завершения передачи

      } /* if (bcount > 0) */
}/* if endpoint interrupt */

Enable_interrupt();          // Разрешение глобальных прерываний
}                        // end if interrupt
```

Глава 3. **Инструментальные средства работы с виртуальным COM – портом.**

В данном разделе будет рассмотрена среда визуального программирования Delphi 7 и стандартная программа Windows – HyperTerminal, которая позволяет очень легко и просто принимать и получать данные с COM – порта, в том числе и виртуального. Данная программа будет весьма полезна на начальном этапе работы с вашим USB – устройством.

Среда Программирования Borland Delphi

Целью данного раздела является приобретение навыков работы со средой программирования Borland Delphi;

Распространение USB постепенно приводит к вытеснению последовательного порта (RS-232). Так, например, последовательный порт уже давно отсутствует на всех современных ноутбуках. Тем не менее, многие приложения продолжают его использовать. Причин для этого много. Часто решающим фактором становится невозможность переработки программы, нежелание дополнительных трудозатрат или некоторая сложность работы с USB относительно COM-порта. В таких случаях бывает удобным использование преобразователя интерфейсов.

Основная задача преобразования интерфейса RS-232 в USB — получить со стороны программы обычный последовательный порт, позволяющий передавать информацию через USB привычным и более простым способом.

Существует несколько способов решения этой проблемы. Первый и самый простой — приобретение специальных переходников USB-to-COM. Второй — использование микросхем преобразователей интерфейсов FTDI. И, наконец, можно использовать драйвер виртуального COM-порта usbser.sys. Особенностью этого способа является отсутствие «лишних» аппаратных средств, как в двух предыдущих случаях. Сам драйвер распространяется бесплатно, его можно найти на сайте компании Atmel.

Borland Delphi — это среда визуального программирования. Создание программ в ней строится на тесном взаимодействии двух процессов: конструировании визуального проявления программы (т.е. ее Windows-окна) и написании кода, придающего элементам этого окна и программе в целом необходимую функциональность. Между содержимым окна формы и кодом существует неразрывная связь — изменение какого-либо компонента на форме приводит к автоматическому изменению кода программы, а удаление тех или иных фрагментов кода может привести к удалению соответствующих компонентов. Как правило, программисты сначала конструируют форму, размещая на ней очередной компонент, а после этого переходят к написанию фрагмента кода, обеспечивающего требуемое поведение компонента в программе. Программный код в Delphi пишется на языке Object Pascal.

Среда разработчика (рис. 20) представляет собой полнофункциональный инструмент, объединяющий все средства, необходимые для создания приложений: редактор исходных текстов, менеджер проектов, палитру компонентов, инспектор объектов и ряд дополнительных утилит.

Палитра компонентов

Палитра компонентов позволяет выбрать визуальные и другие компоненты, которые будут присутствовать в приложении. Компонентами могут быть как визуальные компоненты (кнопки, списки и т.д.), так и невизуальные компоненты (например, таблицы для доступа к базам данных).

Дерево объектов

Это окно предназначено для наглядного отображения связей между отдельными компонентами, размещенными на активном модуле.

Редактор кода

Используется для непосредственного написания кода. Редактор позволяет осуществлять выделение синтаксиса цветом, имеет неограниченные возможности отмены действий и возможность переключения между всеми исходными файлами, входящими в проект.

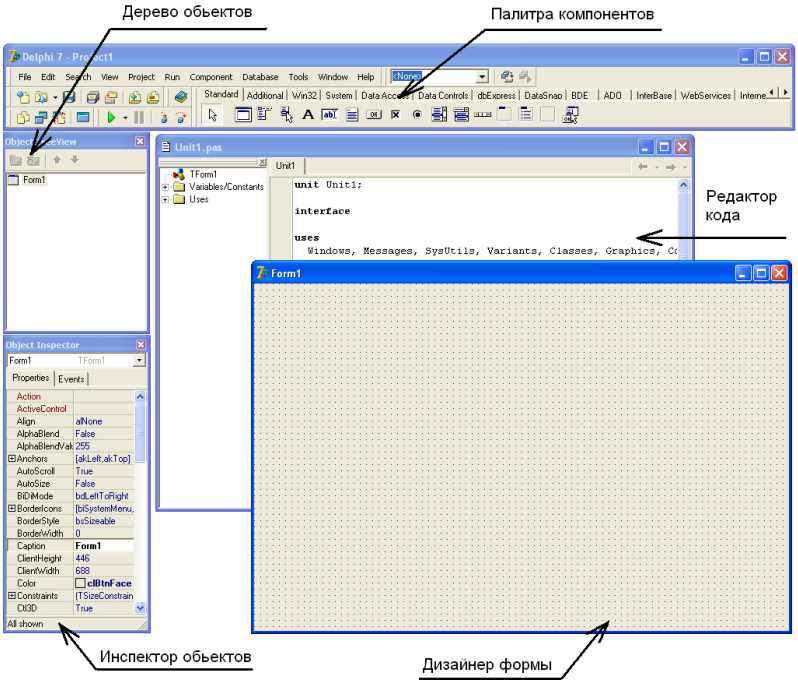


Рис.20. Интегрированная среда разработчика

Инспектор объектов

Инспектор объектов позволяет устанавливать свойства объектов и назначать методы-обработчики событий. Окно инспектора объектов содержит две страницы – Properties (Свойства) и Events (События). Страница Properties служит для установки нужных свойств компонента, страница Events позволяет определить реакцию компонента на то или иное событие.

Дизайнер формы

Дизайнер формы представляет собой чистую форму, на которой разработчик устанавливает необходимые ему компоненты. Вся рабочая область окна формы заполнена точками координатной сетки, служащей для упорядочения размещаемых на форме компонентов. Приложение может включать в себя несколько форм.

Программирование COM-порта

Работа с USB через драйвер виртуального COM-порта ведется так же, как с обычным COMом, правда, так как этот порт виртуальный, некоторые функции для него не могут быть реализованы.

Любая работа с портом начинается с его открытия. Для этого используется файловая функция CreateFile:

```
HANDLE CreateFile(
 LPCTSTR lpFileName,
 DWORD dwDesiredAccess,
 DWORD dwShareMode,
 LPSECURITY_ATTRIBUTES lpSecurityAttributes,
 DWORD dwCreationDistribution,
 DWORD dwFlagsAndAttributes,
 HANDLE hTemplateFile
);
```

Параметры вызова этой функции:

- lpFileName — имя открываемого файла, т.е. название порта (COM1, COM2, …). В нашем случае это должен быть виртуальный COM;
- 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.

В зависимости от значения параметра dwFlagsAndAttributes работа с портом может быть организована в *синхронном* (dwFlagsAndAttributes = 0) или *асинхронном*

(dwFlagsAndAttributes = FILE_FLAG_OVERLAPPED) режимах обработки.

При синхронном режиме только один поток приложения может либо читать, либо писать в порт. 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.

Асинхронный режим 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.

После открытия порта его необходимо настроить. Основные параметры последовательного порта описываются *структурой DCB*. Она содержит следующие поля:

- BaudRate — скорость передачи данных (в бодах). Задается указанием 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():

```
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');
```

Одновременно выполняется проверка на ошибки. Параметры вызова функций:

- hPort — описатель открытого порта;
- Dcb — указатель на структуру DCB.

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

```
BOOL PurgeComm(
 HANDLE hPort,
 DWORD dwFlags
);
```

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

- hPort — описатель открытого порта;
- dwFlags — производимые действия в виде набора флагов PURGE_TXABORT, PURGE_RXABORT, PURGE_TXCLEAR, PURGE_RXCLEAR.

Прием и передача данных выполняются соответственно функциями ReadFile() и WriteFile():

```
BOOL ReadFile(
 HANDLE hPort,
 LPVOID lpBuffer,
 DWORD nNumberOfBytesToRead,
 LPDWORD lpNumberOfBytesRead,
 LPOVERLAPPED lpOverlapped
);

BOOL WriteFile(
 HANDLE hPort,
 LPCVOID lpBuffer,
 DWORD nNumberOfBytesToWrite,
 LPDWORD lpNumberOfBytesWritten,
 LPOVERLAPPED lpOverlapped
);
```

Параметры вызова функций:

- hPort — 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.

По окончании работы с портом его следует закрыть, вызвав функцию CloseHandle:

```
BOOL CloseHandle(
 HANDLE hObject
);
```

В качестве единственного параметра надо передать полученный ранее описатель порта.

Пример приема/передачи с использованием отладочного модуля для микроконтроллера AT89C5131

1. Программа для микроконтроллера

Тестовая программа для AT89C5131 на языке С приведена в папке *Программа для микроконтроллера*:

- test.c51 – исходный текст программы;

ПРИМЕЧАНИЕ: Редактировать файл лучше с помощью Блокнота, хотя читать удобнее в Word'е.

- test.hex – скомпилированный HEX-файл;
- err.txt и errs.txt – после компиляции будут содержать информацию об ошибках;
- make.exe – программа компилятора. При запуске программы производится компиляция программы в файле test.c51; скомпилированный код помещается в файл test.hex, информация об ошибках будет находиться в файлах err.txt и errs.txt.

ПРИМЕЧАНИЕ: При наличии фатальных ошибок HEX-файл не создается.

Для зашивки программы в контроллер используется программатор FLIP. Для программирования контроллера:

- запустите FLIP;
- зайдите в меню Device – Select…, в появившемся списке выберите тип контроллера – AT89C5131;
- подключите микроконтроллер в режиме программирования;

ПРИМЕЧАНИЕ: Для программирования микроконтроллера его необходимо подключать к порту при нажатой кнопке S2.

- откройте порт для программирования: Settings – Communication – USB;
- скопируйте HEX-файл в директорию Program Files/Atmel/FLIP;
- выберите HEX-файл: File – Load HEX File. Укажите: Program Files/ Atmel/FLIP/test.hex;

ПРИМЕЧАНИЕ: FLIP открывает только HEX-файлы, расположенные в директории программатора.

- для программирования контроллера выберите меню Device – Program.

2. Программа для компьютера

Тестовая программа для компьютера, написанная в Delphi, приведена в папке *Программа для компьютера*:

- Project1.dpr – файл проекта Delphi;
- Project1.exe – скомпилированное приложение;

ПРИМЕЧАНИЕ: Для компиляции приложения используется меню Project – Compile Project1.

- Unit1.pas, ComPortUnit.pas – файлы с исходным текстом программы;
- Unit1.dfm – файл графической формы приложения.

3. Прием/передача данных.

Для приема/передачи данных:

- подключите отладочный модуль с загруженной в него программой test.hex к USB-порту компьютера;
- запустите приложение Project1.exe;
- в группе переключателей «Порт» установите флажок возле виртуального порта;
- нажмите кнопку «Подключить», после этого установится соединение;
- введите в поле «Передача данных» число от 0 до 255;
- нажмите кнопку «Передать». Введенное число будет передано в микроконтроллер. Полученный байт воспримется программой микроконтроллера как код символа для LCD-дисплея. Символ,

соответствующий переданному байту, будет выведен в первую позицию дисплея.

ПРИМЕЧАНИЕ: Таблицу кодов для LCD-дисплея можно найти в заголовочном файле *Программа для микроконтроллера/display.h.*

Далее микроконтроллер передаст число назат в компьютер. Полученный байт с некоторой задержкой отобразится в поле «Прием данных»;

- измените число в поле «Передача данных», нажмите кнопку «Передать». Символ, соответствующий этому коду, будет выведен на дисплей в следующую позицию.

ПРИМЕЧАНИЕ: В файле display.h код символов приведен в шестнадцатеричном формате, а вводить его в поле «Передача данных» следует в десятичном.

Основные конструкции языка Object Pascal

Составной оператор – это последовательность произвольных операторов программы, заключенная в операторные скобки – зарезервированные слова begin … end. Составные операторы – важнейший инструмент Object Pascal, дающий возможность писать программы по современной технологии структурного программирования. Object Pascal не накладывает никаких ограничений на характер операторов, входящих в составной оператор. Среди них могут быть и другие составные операторы – Object Pascal допускает произвольную глубину их вложенности.

Условный оператор – позволяет проверить некоторое условие и, в зависимости от результатов проверки, выполнить то или иное действие. Таким образом, условный оператор – это средство ветвления вычислительного процесса. Структура условного оператора имеет вид:

```
if <условие> then <оператор1> else <оператор2>;
```

Вычисляется <условие>, если результат – True (истина), выполняется <оператор1>, а <оператор2> пропускается; если результат – False (ложь), наоборот, <оператор1> пропускается, а <оператор2> выполняется.

Счетный оператор цикла имеет структуру:

```
for <параметр> := <нач_знач> to <конеч_знач> do <оператор>;
```

При выполнении этого оператора сначала вычисляется выражение <нач_знач> и осуществляется присвоение результата в переменную <параметр>. Затем циклически повторяется:

- проверка условия <параметр> <= <конеч_знач>; если условие не выполнено, оператор завершает свою работу;
- выполнение оператора <оператор>;
- инкремент переменной <параметр>.

Если значение <нач_знач> больше значения <конеч_знач>, вместо to в структуре оператора ставится downto.

Оператор цикла с предусловием имеет структуру:

```
while <условие> do <оператор>;
```

Если выражение <условие> имеет значение True, то выполняется <оператор>, после чего вычисление выражения <условие> и его повторяются. Если <условие> имеет значение False, оператор прекращает работу.

Оператор цикла с постусловием имеет структуру:

```
repeat <оператор> until <условие>;
```

Оператор <оператор> выполняется как минимум один раз. После первого выполнения вычисляется выражение <условие>: если результат False, цикл повторяется, в противном случае оператор завершает свою работу.

Для гибкого управления операторами цикла в состав Object Pascal включены две процедуры без параметров:

break – реализует немедленный выход из цикла; действие процедуры заключается в передаче управления оператору, стоящему сразу за концом циклического оператора;

continue – обеспечивает досрочное завершение очередного прохода цикла; действие процедуры заключается в передаче управления в самый конец циклического оператора.

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит ключ выбора – выражение любого порядкового типа. Структура оператора выбора:

```
case <ключ_выбора> of <список выбора> else <оператор> end;
```

Здесь <ключ выбора> – выражение порядкового типа; <список_выбора> – одна или более конструкций следующего вида:

```
<константа_выбора>: <оператор>;
```

где <константа_выбора> – константа того же типа, что и <ключ_выбора>; <оператор> – произвольный оператор Object Pascal. Оператор выбора работает следующим образом. Сначала вычисляется выражение <ключ_выбора>, затем в последовательности операторов <список_выбора> отыскивается такой, которому предшествует <константа_выбора>, равная вычисленному значению. Найденный оператор выполняется, после чего оператор выбора завершает свою работу. Если в списке выбора не будет найдена

константа, соответствующая вычисленному значению ключа выбора, управление передается оператору, стоящему за словом else. Любому из операторов списка выбора может предшествовать не одна, а несколько констант выбора, разделенных запятыми.

В некоторых случаях бывает удобным использование ***оператора перехода***. В общем виде оператор перехода имеет вид:

```
goto <метка>;
```

Метка в Object Pascal – это произвольный идентификатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него. В Object Pascal допускается использование в качестве меток также целых чисел баз знака. Метка располагается непосредственно перед оператором и отделяется от него двоеточием.

Процедурой в Object Pascal называется особым образом оформленный фрагмент программы, имеющий собственное имя. Упоминание этого имени в тексте программы приводит к активизации процедуры – начинают выполняться входящие в нее операторы. После выполнения последнего из них управление передается обратно в основную программу, и выполняются операторы, стоящие непосредственно за оператором вызова процедуры. *Параметры вызова* – переменные, используемые для обмена информацией между основной программой и процедурой – перечисляются в круглых скобках за именем процедуры и вместе с ним образуют *оператор вызова процедуры*.

Функция отличается от процедуры тем, что результат ее работы возвращается в виде значения этой функции, и, следовательно, вызов функции может использоваться наряду с другими операндами в выражениях.

Программа работы с COM-портом Hyperterminal

Данная программа позволяет очень легко и просто работать с COM – портом. программа находится в: меню пуск – программы – стандартные – связь – HyperTerminal. После запуска программы появится окно (рис. 21).

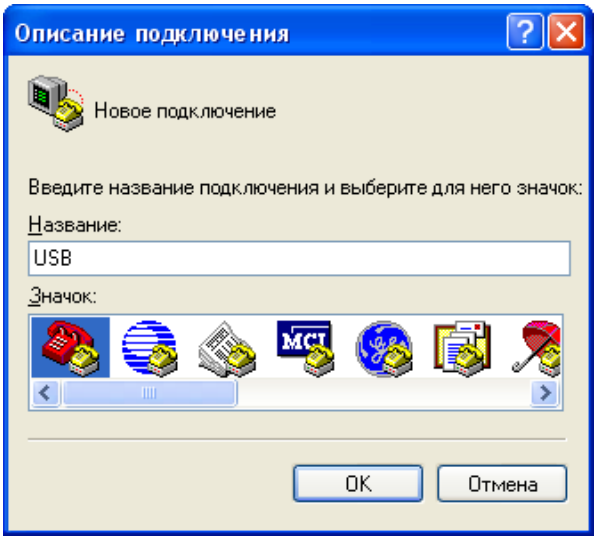


Рис.21. Окно HyperTerminal(описание подключения)

В данном окне необходимо ввести название соединения. Название может быть любым. В следующем окне (рис. 22) необходимо выбрать требуемый Com – порт. После чего появится окно настройки Com – порта.
В данном окне необходимо ввести соответствующие настройки и нажать кнопки: «применить» и «ОК».

В конце всех действий должно появиться окно, показанное на рисунке 24. Данное окно отображает в ASCI – коде, все данные, принимаемые с выбранного COM – порта. Для того чтобы предать что-либо необходимо просто нажать любую букву на клавиатуре и соответствующий символ будет передан.

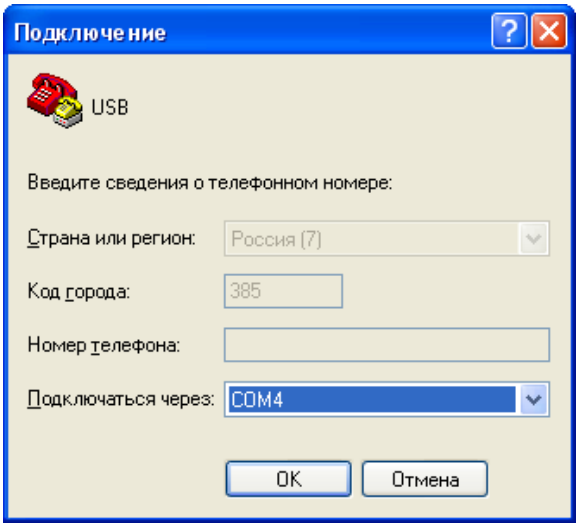


Рис.22. Окно выбора COM – порта

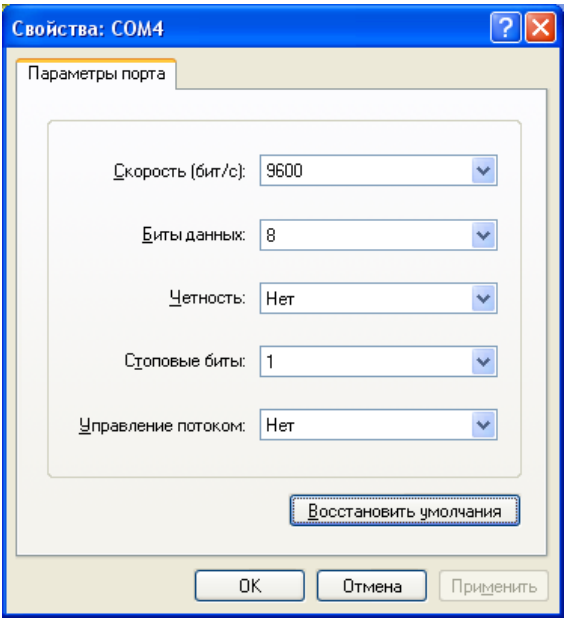


Рис.23. Окно настройки COM – порта

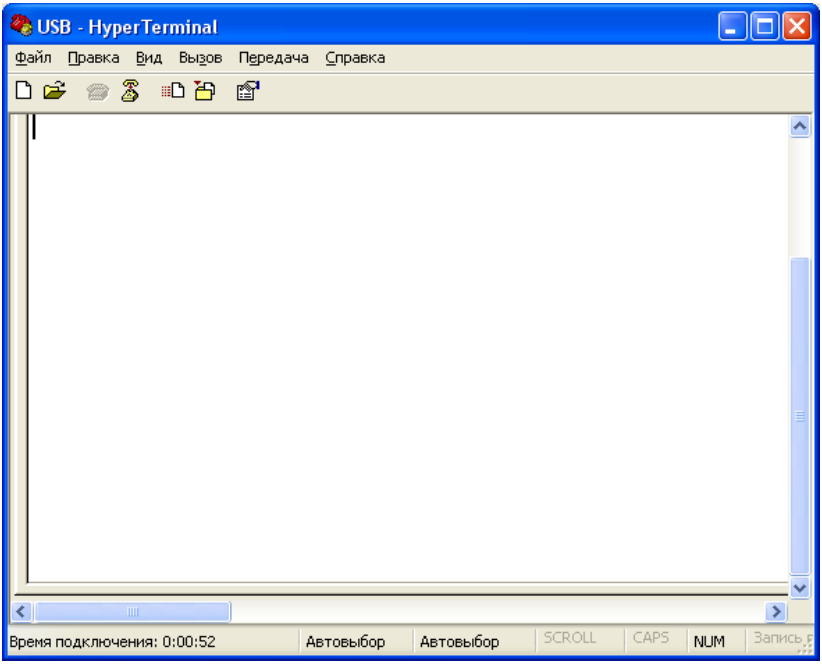


Рис.24. Окно HyperTerminal

Глава 4. **Лабораторный практикум**

Целью данного лабораторного практикума является получение практических навыков в разработке алгоритмов программного обеспечения, как для микроконтроллера, так и для компьютера на базе отладочного модуля AT89C5131.

Оборудование:

1. Персональный компьютер.
2. Отладочный модуль для микроконтроллера AT89C5131 с двухстрочным ЖК-дисплеем.
3. Среда программирования Borland Delphi 7.0 Enterprise Edition.
4. Программатор FLIP.
5. Драйвер usbser.sys.
6. Стандартная программа windows – HyperTerminal.
7. Для соединения контроллера с компьютером через RS232 использовать стандартный нуль-модемный кабель.
8.

Варианты заданий

1. Вывести на дисплей свою фамилию, имя и номер группы.
2. Изменить яркость подсветки.
3. Организовать обмен данными между приложением и микроконтроллером с созданием окна для подсчета количества принятых байтов.
4. Организовать обмен данными между приложением и микроконтроллером с созданием окна для подсчета количества переданных байтов.
5. Передать в микроконтроллер и принять назад числа от 0 до 255.
6. Написать программу, выводящую на дисплей отладочного модуля при нажатии кнопки «Передать» ваши фамилию и инициалы.
7. Написать программу для микроконтроллера, которая будет осуществлять прием строчных символов в ASCII – коде и передачу этих же символов, но уже прописных. (В качестве программы верхнего уровня использовать HyperTerminal).
8. Написать программу для микроконтроллера, которая будет осуществлять прием ASCII – символов, и выводить их на

ЖК-дисплей платы микроконтроллера. (В качестве программы верхнего уровня использовать HyperTerminal).

9. Разработать систему фильтрации входящих данных. Пропускать только строчные символы английского алфавита и отправлять их обратно на компьютер. (В качестве программы верхнего уровня использовать HyperTerminal).
10. Разработать систему управления подсветкой дисплея. Обеспечить возможность включения и выключения подсветки, а также возможность регулирования яркости подсветки. (В качестве программы верхнего уровня использовать HyperTerminal).
11. Организовать связь между двумя компьютерами через USB интерфейс. У одного компьютера использовать RS232. В качестве программы использовать гипертерминал. Данные должны передаваться в обе стороны.
12. В дополнение к 11 заданию. Отображать на дисплее отладочной платы всю передаваемую по шине USB информацию. В зависимости от направления передачи, данные должны выводиться либо в первую, либо во вторую строки индикатора.
13. Разработать программу, позволяющую при помощи компьютера выводить на ЖК – экран отладочной платы бегущую строку, состоящую из введенного на компьютере текста. Предусмотреть возможность изменения скорости движения бегущей строки и направления ее движения, а также плавное изменение яркости дисплея.
14. Разработать систему ввода аналоговой информации в компьютер через USB - интерфейс с последующим сохранением ее на жестком диске в виде текстового файла. Для преобразования можно использовать любую плату с АЦП на борту.

Контрольные вопросы

1. Для чего используется драйвер виртуального COM-порта?
2. В чем заключается принцип визуального программирования?
3. Что отображает окно «Дерево объектов»?
4. Что отображает окно «Инспектор объектов»?
5. Для чего используется «Палитра компонентов»?
6. Для чего используется «Редактор кода»?
7. Для чего используется «Дизайнер формы»?

8. Какие функции используются в Delphi для работы с COM-портом, что они делают?
9. Области применения ЖКИ-модуля.
10. По какому сочетанию и каких сигналов осуществляется запись команды?
11. По какому сочетанию и каких сигналов осуществляется запись данных?
12. Какая основная отличительная особенность работы с 8-и и 4-х разрядной шиной данных?
13. По каким разряда и какого порта осуществляется регулировка подсветки дисплея?
14. Поясните принцип работы подпрограммы задержки.
15. Приведите функциональную схему подключения ЖКИ к микроконтроллеру MCS-51.
16. Поясните, что такое флаг занятости BF и для чего он нужен.
17. Приведите функциональную схему подключения ЖКИ к питанию.
18. Назовите основные свойства шины USB.
19. Какие ограничения накладываются на устройства, подключаемые к шине USB.
20. Какие существуют классы USB устройств.
21. К каким классам относятся следующие USB устройства: клавиатура, мышь, веб-камера, флешка, внешний HDD винчестер.
22. Какие устройства используют CDC – класс.
23. В чем особенность механизма прерываний USB шины.
24. На каких скоростях работает USB шина.
25. Какие типы передачи данных определены стандартом USB 2.0.
26. Можно ли запитать от USB шины устройство, которое потребляет 600мА,50мА,450мА?
27. Для чего необходим сброс шины перед работой устройства.

Содержание отчета

1. Цель работы.
2. Используемые устройства и программы.
3. Задание.
4. Программа, написанная вами при выполнении задания с пояснениями (без подпрограмм).
5. Вывод по проделанной работе.

ПРИЛОЖЕНИЕ А

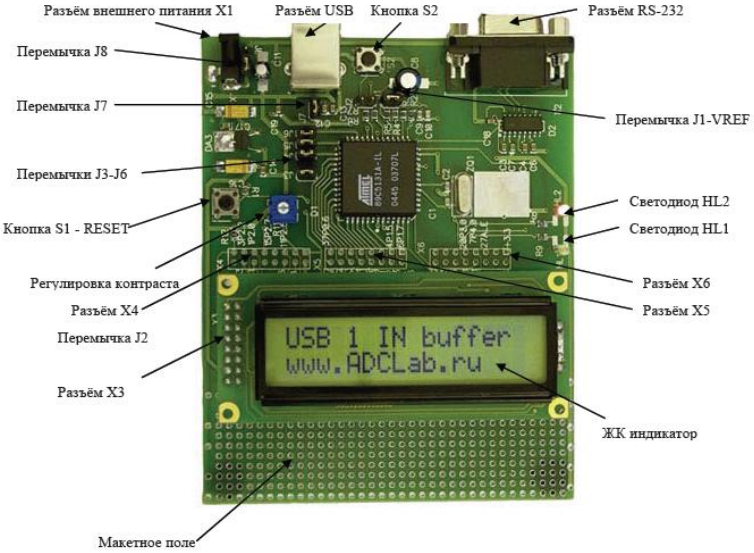


Рис. 25. Внешний вид платы

Назначение перемычек плат

J1 - отключает сигнал VREF от USB - это необходимо для того, чтобы при подключенном USB кабеля к плате, отключить постоянное напряжение 3,3В, тогда операционная система воспримет это действие как отключение устройства от USB

J2 - дублируется кнопкой S2 - Вы можете воспользоваться перемычкой или кнопкой, как Вам удобно, но необходимо помнить, что нельзя оставлять перемычку J2 замкнутой на продолжительное время (запрещается разработчиком микроконтроллера) Замыкание данных контактов перед подключением платы к USB порту компьютера позволяет войти в режим In-System программирование (читайте ниже)

J3-J6 - при замкнутых перемычках порты P3.3, P3.5, P3.6, P3.7 подключены к X4 и используются для питания подсветки индикатора. Управление данными портами позволяет регулировать яркость подсветки.

J7 -подключает питание устройства от USB порта, при установленной перемычке плата подключена шиной питания к USB. Перемычка необходима для отключения нагрузки смонтированными устройствами платы от USB порта и подключения внешнего источника питания.

J8 - подключает питание +5В от внешнего источника через разъём X1

Для индикации работы микроконтроллера дополнительно предусмотрены два светодиода, которые подключены к портам P4.0 и P4.1 через резисторы R8, R9 номиналом 1кОм.

ПРИЛОЖЕНИЕ В

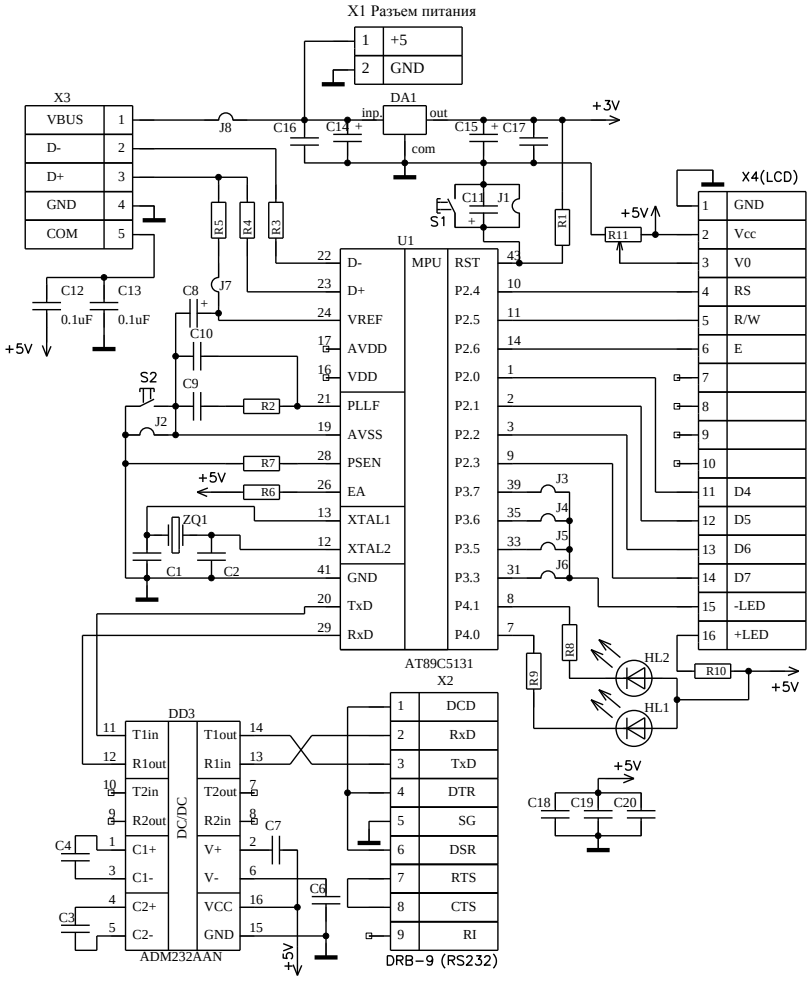
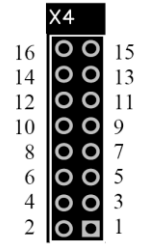


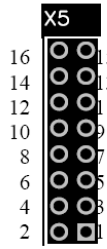
Рис. 26. Принципиальная схема

ПРИЛОЖЕНИЕ С

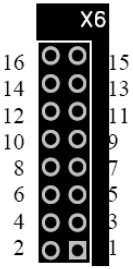
Описание внешних разъемов платы



1 – GND
2 - +5V
3 – Контраст
4 – Порт P2.4 (вывод 10 D1)
5 - Порт P2.5 (вывод 11 D1)
6 - Порт P2.6 (вывод 14 D1)
7 – не подключен
8 - не подключен
9 – Порт P2.7 (вывод 15 D1)
10 – Не подключен
11 – Порт P2.0 (вывод 1 D1)
12 – Порт P2.1 (вывод 2 D1)
13 – Порт P 2.2 (вывод 3 D1)
14 – Порт P2.3 (вывод 9 D1)
15 - +5V
16 – Порт P3.7 (вывод 39 D1)



1 – Порт P1.7 (вывод 6 D1)
2 - Порт P1.6 (вывод 5 D1)
3 – Порт P1.5 (вывод 4 D1)
4 – Порт P1.4 (вывод 51 D1)
5 - Порт P1.3 (вывод 50 D1)
6 - Порт P1.2 (вывод 49 D1)
7 – Порт P1.1 (вывод 48 D1)
8 - Порт P1.0 (вывод 47 D1)
9 – Порт P0.0 (вывод 52 D1)
10 – Порт P0.1 (вывод 45 D1)
11 – Порт P0.2 (вывод 44 D1)
12 – Порт P0.3 (вывод 42 D1)
13 – Порт P 0.4 (вывод 40 D1)
14 – Порт P0.5 (вывод 38 D1)
15 - Порт P0.6 (вывод 37 D1)
16 – Порт P0.7 (вывод 36 D1)



1 – +3V3
2 - GND
3 – не подключен
4 – не подключен
5 - ALE (вывод 27 D1)
6 - не подключен
7 – Порт P4.0 (вывод 7 D1)
8 - Порт P4.1 (вывод 8 D1)
9 – Порт P3.0 (вывод 20 D1)
10 – Порт P3.1 (вывод 29 D1)
11 – Порт P3.2 (вывод 30 D1)
12 – Порт P3.3 (вывод 31 D1)
13 – Порт P 3.4 (вывод 32 D1)
14 – Порт P3.5 (вывод 33 D1)
15 - Порт P3.6 (вывод 35 D1)
16 – Порт P3.7 (вывод 39 D1)

ПРИЛОЖЕНИЕ D

Таблица векторов прерываний микроконтроллера AT89C5131

Номер прерывания	Приоритет	Источник	Флаг	Вектор
-	0	Reset		0000h
0	1	INT0	IE0	0003h
1	2	Timer0	TF0	000Bh
2	3	INT1	IE1	0013h
3	4	Timer1	IF1	001Bh
4	5	UART	RI+TI	0023h
5	6	Timer2	TF2+EXF2	002Bh
6	7	PCA	CF+CCFn	0033h
7	8	Keyboard	KBDIT	003Bh
8	9			0043h
9	10	SPI	SPIIT	004Bh
10	11			0053h
11	12			005Bh
12	13			0063h
13	14	USB	UEPINT+USBINT	006Bh
14	15			0073h

ПРИЛОЖЕНИЕ Е

ЖКИ-модуль
Описание, система команд, программирование

Жидкокристаллические индикаторы в настоящее время наиболее распространенные устройства отображения информации. Алфавитно-цифровые (матричные) и графические модули − устройства отображения информации, при подключении которых необходимо организовать порядка полутора-двух десятков соединений (их количество практически не зависит от информационной емкости модуля), а для управления ими требуются только управляющие и информационные сигналы от микропроцессора. Каждый модуль содержит один или несколько, в зависимости от информационной емкости, контроллеров, которые, принимая данные и управляющие сигналы от микропроцессора, организуют процесс выработки необходимых для работы сигналов и напряжений формируя на ЖКИ-модуле требуемое изображение.

Некоторые характеристики ЖКИ-модулей:

- алфавитно-цифровые и графические
- широкий выбор форматов индикаторов
- встроенная схема инициализации
- большая площадь отображения информации
- несколько вариантов фоновой подсветки
- малые напряжения питания и энергопотребления
- возможность работы с 8 и 4-разрядными микропроцессорами
- различные наборы шрифтов
- возможность программирования символов
- компактность, малые вес и толщина
- широкий угол обзора и высокая контрастность
- различные варианты цвета фоновой подсветки
- широкий диапазон температур
- высокая надежность и качество

Из табл. 1 видно насколько просто подключается дисплей к микроконтроллеру.

Таблица 2

Назначение выводов

Вывод	Обозначение	Назначение вывода
1	GND	Общий вывод (0V)
2	VCC	Напряжение питания (5V)
3	Vo	Управление контрастностью

4	RS	Адресный сигнал – выбор между передачей данных и команд управления
5	R/W	Выбор режима записи или чтения (1 – чтение, 0 – запись)
6	E	Разрешение обращения к модулю (а также строб данных)
7	DB0	Шина данных (8-и битный режим) (младший бит в 8-и битном режиме)
8	DB1	Шина данных (8-и битный режим)
9	DB2	Шина данных (8-и битный режим)
10	DB3	Шина данных (8-и битный режим)
11	DB4	Шина данных (8-и и 4-х битные режимы) (младший бит в 4 –х битном режиме)
12	DB5	Шина данных (8-и и 4-х битные режимы)
13	DB6	Шина данных (8-и и 4-х битные режимы)
14	DB7	Шина данных (8-и и 4-х битные режимы) (старший бит)
15	-LED	- питания подсветки
16	+LED	+ питание подсветки

Система команд

Таблица 3

Система команд

Инструкция	RS	R/W	DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB 0	Время выполнения
Очистка дисплея	0	0	0	0	0	0	0	0	0	1	82 мкс – 1,62 мс
Возврат в начало	0	0	0	0	0	0	0	0	1	*	40 мкс – 1,6 мс
Выбор направления сдвига	0	0	0	0	0	0	0	1	I/D	S	40 мкс
Дисплей, курсор, мерцание курсора	0	0	0	0	0	0	1	D	C	B	40 мкс
Сдвиг курсора и дисплея	0	0	0	0	0	1	S/C	R/L	*	*	40 мкс
ШД, число строк, размер шрифта	0	0	0	0	1	DL	N	F	*	*	40 мкс
Установка адреса CGRAM	0	0	0	1	Acg						40 мкс

Инструкция	RS	R/W	DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB 0	Время выполнения
Установка адреса DDRAM	0	0	1	Acg							40 мкс
Чтение флага BF и счетчика AC	0	1	BF	AC							1 мкс
Запись данных в CG или DDRAM	1	0	ЗАПИСЫВАЕМЫЕ ДАННЫЕ								40 мкс
Чтение данных из CG или DDRAM	1	1	ЧИТАЕМЫЕ ДАННЫЕ								40 мкс

ПРИМЕЧАНИЕ

На практике времена выполнения инструкций, указанные в таблице, недостаточны, в связи с этим применяются временные задержки большей длительности.

***** - *любое значение.*

Очистка дисплея

Записывает код пробела (20h) по всем адресам DD RAM. Устанавливает счетчик адреса DD RAM в 0. Возвращает дисплей в его первоначальное состояние, если содержимое его было сдвинуто. Другими словами, дисплей очищается, курсор сдвигается в левый край дисплея (на первую строчку, если дисплей двухстрочный). Кроме того, I/D устанавливается в 1 (режим автоинкремента). Состояние режима ввода S не меняется.

Возврат на начало

Устанавливает адрес 0 DD RAM в счетчике адреса. Возвращает дисплей в первоначальное состояние, если он был сдвинут.Содержимое DD RAM не изменяется, курсор сдвигается в левый край дисплея (на первую строчку, если дисплей двухстрочный.

Установка режима ввода

I/D: Инкремент (I/D=1) или декремент (I/D=0) счетчика адреса DD RAM на единицу при записи или чтении кода символа в или из DDRAM.

S: S=0 Курсор сдвигается направо на одну позицию при инкременте и налево при декременте. То же самое относится к чтению записи CG RAM.

S: Сдвигает полностью содержимое дисплея на одну позицию вправо или влево при S=1: влево при I/D=1 и вправо при I/D=0. Внешне это выглядит так: курсор остается на месте, в то время как содержимое дисплея сдвигается. Содержимое дисплея не смещается во время считывания из DD RAM, также не происходит смещения при чтении/записи CG RAM.

Дисплей включить/выключить (ON/OFF)

D: Дисплей включен при D=1 и выключен при D=0 .При этом данные сохраняются неизменными в DD RAM и выводятся на дисплей сразу после установки D=1.

C: Курсор включен при C=1 и включен при C=0. Курсор использует 5 точек в 8 линии при шрифте 5 × 7 точек, и 5 точек в 11 линии при шрифте 5 × 10 точек. Даже если курсор не отображается, функция I/D и другие спецификации не будут изменяться во время записи данных дисплея.

B: Мерцание символа расположенного над позицией курсора с частотой 409.6 мс Fosc=270 Кгц при B=1. Мерцание реализуется переключение между матрицей полностью заполненной точками и отображаемым символом. Включение курсора и мерцание могут быть инициализированы одновременно.

Сдвиг курсора или дисплея

Сдвигает курсор или дисплей влево или вправо без чтения/записи данных. В двустрочном дисплее курсор сдвигается на вторую строку при достижении 40 позиции в первой строке Необходимо отметить, что и первая и вторая строка сдвигаются по горизонтали одновременно. При выполнении операции смещения дисплея значение счетчика адреса AC не меняется.

S/C	R/L	
0	0	Сдвиг курсора на позицию влево (Счетчик адреса инкрементируется на единицу)
0	1	Сдвиг курсора на позицию вправо (Счетчик адреса декрементируется на единицу)
1	0	Сдвиг всего экрана на позицию влево, с курсором. (Счетчик адреса не меняется)
0	1	Сдвиг всего экрана на позицию вправо,с курсором. (Счетчик адреса не меняется)

Установка функций

DL: Устанавливает ширину шины данных. Данные посылаются /принимаются по 8-битовой шине, при DL=1 и по 4-битовой шине, при DL=0.

N: Установка количества строк дисплея, N=0 одна строка, N=1 две строки.

F: Установка размера шрифта.

Установка адреса CG RAM

Устанавливает адрес CG RAM в счетчике адреса в двоичном формате. Передаваемые затем данные адресуется для CG RAM.

Установка адреса DD RAM

Устанавливает адрес DD RAM в счетчике адреса в двоичном формате. Передаваемые затем данные адресуется для DD RAM. Необходимо отметить, что при N=0 (однострочный дисплей) адрес DD RAM должен быть в пределах 00H-4FH включительно, при N=1 (двустрочный дисплей) в пределах 00H-27H (первая строка) и 40H-67H (вторая строка) включительно.

Чтение флага занятости и адреса

Читается состояние флага занятости (BF), установка которого означает, что система в данный момент обрабатывает предыдущую инструкцию. Следующая инструкция не может быть выполнена, пока флаг BF не сбросится в 0. Необходимо каждый раз проверять состояние флага BF перед посылкой инструкции.

Этой же инструкцией считывается состояние счетчика адреса выраженное в двоичном формате. Счетчик адреса используется при адресации как DD RAM так и CG RAM.

Запись данных в CG RAM или DD RAM

Записывает байт данных в двоичном формате в CG RAM или DD RAM . Запись происходит по адресу, определяемому счетчиком адреса (установка счетчика адреса выполняется инструкциями (7), (8)). После записи счетчик адреса автоматически инкрементируется или декрементируется на 1, в соответствии с установкой режима ввода (2). Установка режима ввода также определяет сдвиг дисплея.

Чтение данных в CG RAM или DD RAM

Читает байт данных в двоичном формате из CG RAM или DD RAM. Перед посылкой инструкции чтения, необходимо выполнить инструкцию установки счетчика адреса либо для CG RAM (7), либо для DD RAM (8). В противном случае первое считывание данных будет некорректным. Выполнение инструкции установки счетчика адреса не требуется при операциях связанных со сдвигом курсора (когда происходит чтение DD RAM, так как инструкция сдвига

курсора действует подобно инструкции установки адреса DD RAM). После чтения счетчик адреса автоматически инкрементируется или декрементируется на 1, в соответствии с установкой режима ввода (2).

№ Знакоместа		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
АДРЕС	1-я строка	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
	2-я строка	40h	41h	42h	43h	44h	45h	46h	47h	48h	49h	4Ah	4Bh	4Ch	4Dh	4Eh	4Fh

Рис.27. Распределение DDRAM.

Инициализация ЖКИ-модуля

Инициализация внутренней схемой сброса

Контроллер HD44780 автоматически инициализируется (сбрасывается) по включению питания. При инициализации выполняются следующие инструкции. Флаг занятости удерживается до конца инициализации (BF=1).

(1) Очистка дисплея

(2) Набор функций ....

DL=1: разрядность шины 8 бит

N=0: однострочный дисплей

F=0: символы 5 × 7 точек

(3)Включение/выключение дисплея

D=0: дисплей выключен

C=0: курсор выключен

B=0: мерцание выключено

(4)Режим ввода...

I/D=1:+1 (инкремент)

S=0: нет сдвига

(5)Запись DD RAM (ОЗУ знакогенератора)

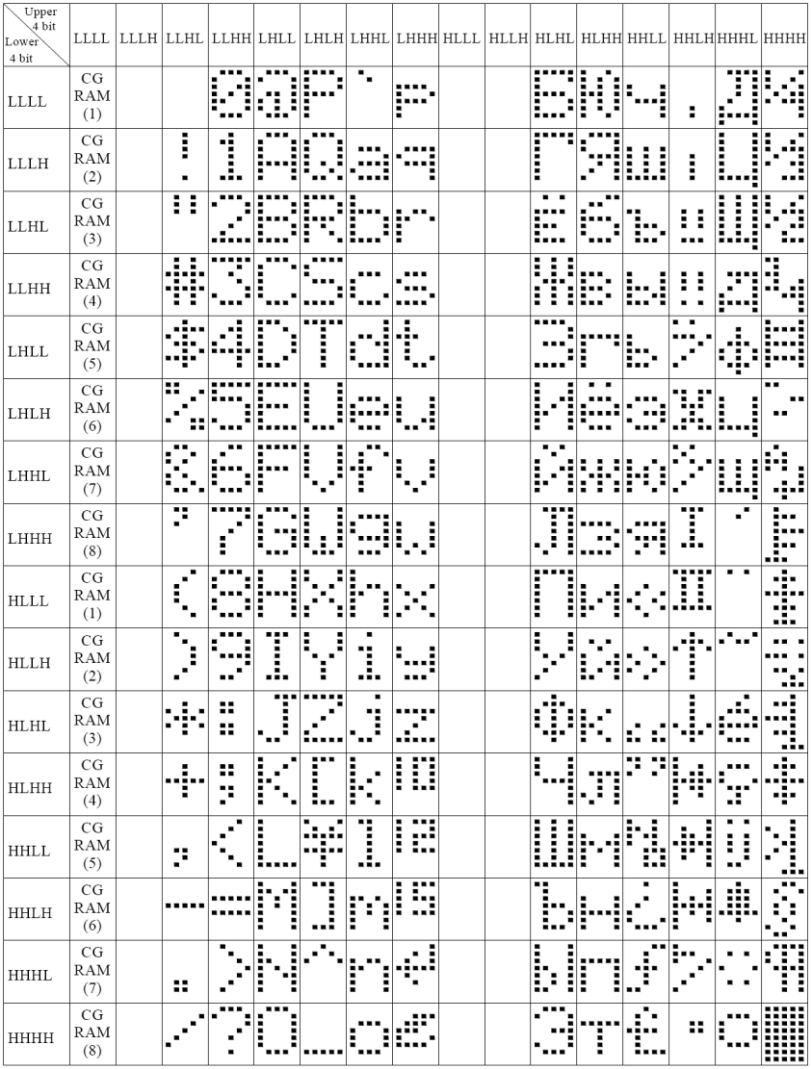
Если время установления напряжения питания (0,2-4,5) больше, чем 0,1 мс 10 мс, то внутренняя схема сброса может сработать некорректно. В этом случае инициализация должна проводится программно микропроцессором.

В нашем случае применяется 4-х разрядная шина данных, поэтому инициализация дисплея осуществляется программным путем по алгоритму, указанному на рисунке 8.

Рис.28. Алгоритм программной инициализации для 4-х разрядной шины данных

Таблица 4

Стандартный знакогенератор



СПИСОК ЛИТЕРАТУРЫ

1. Агуров П.В. Практика программирования USB. — СПб.: БХВ – Петербург, 2007. — 624 с.
2. Рейзлин В.И. Программирование на языке С++. Учебное пособие. Томск: Изд-во ТПУ, 2003. – 179с.
3. Консультация по электронике и электронным компонентам. Язык С для МК, [Электронный ресурс]. – Режим доступа: http://avr123.nm.ru/05a.htm, свободный. – Загл. С экрана.
4. Букреев В.Г., Гусев Н.В. Delphi-6 – среда разработки программного обеспечения для систем промышленной автоматизации: Учебное пособие. – Томск: Изд-во ТПУ, 2004. – 106 с.
5. www.usb.org (международная организация поддержки usb)
6. www.alldatasheet.ru (сайт технического описания микросхем)
7. www.adclab.ru (сайт производителя отладочной платы с микроконтроллером AT89C5131)

СОДЕРЖАНИЕ

Учебное издание

ВОРОБЬЕВА Галина Степановна
ЮРЧЕНКОВ Вячеслав Алексеевич
МАРТЕМЬЯНОВ Сергей Михайлович

ПРОЕКТИРОВАНИЕ CDC – УСТРОЙСТВ НА МИКРОКОНТРОЛЛЕРАХ СО ВСТРОЕННЫМ USB – МОДУЛЕМ

Учебно-методическое пособие

Научный редактор
доктор технических наук,
профессор Г.С. Евтушенко

Редактор Г.С. Воробьева
Верстка В.А. Юрченков
Дизайн обложки В.А. Юрченков

Подписано к печати 21.02.2010. Формат 60x84/16. Бумага «Классика».
Печать RISO. Усл.печ.л. 4,185. Уч.-изд.л. 3,78.
Заказ . Тираж 10 экз.

Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2000

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30.