Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Автоматизация проектирования микропроцессорных систем»

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

Тема «USB- порт» »

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /П.С. Кондратьев /

подпись инициалы, фамилия

Курс 4 Группа ИВТАПбд-41

Направление/ специальность 09.03.01 «Информатика и вычислительная техника»

Руководитель доцент Игонин А.Г.

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2020 г.

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Автоматизация проектирования микропроцессорных систем»

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

студенту ИВТАПбд-41 Кондратьев П.С.

группа фамилия, инициалы

Тема проекта (работы) «USB- порт»

Срок сдачи законченного проекта (работы) «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Исходные данные к проекту (работе) Proteus, USBCONN

Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Пояснительная записка должна содержать 25 – более страниц текста, распределенного по разделам следующим образом (указывается минимальное количество страниц в разделе):

1. Титульный лист – 3 стр.

2. Введение – 1 – 2 стр.

3. Глава I – 4 стр.

4. Глава II – 2 – 5 стр.

5. Глава III – 17 стр.

\* Функциональная спецификация – 1 стр.

\* Описание интерфейса между системой и пользователем – 1 – 2 стр.

\* Проектирование системы – 1 – 2 стр.

\* Проектирование аппаратной части устройства – 4 – 5 стр.

\* Алгоритм работы – 1 – 2 стр.

\* Дополнения и улучшения – 1 – 2 стр.

6. Заключение (подведение итогов) – 1 – 2 стр.

7. Список литературы – 1 – 2 стр.

8. Приложение (листинг программы, изображение модели домофона).

Перечень графического материала (с точным указанием обязательных чертежей)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель доцент / Игонин А.Г. /

должность подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ П.С. Кондратьев /

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ  
руководителя на курсовой проект (работу)**

Студента Кондратьева Павла Сергеевича

фамилия, имя и отчество

Факультет ФИСТ группа ИВТАПбд-41 курс 4

Дисциплина «Автоматизация проектирования микропроцессорных систем»

Тема проекта (работы) «USB- порт»

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ / Игонин А.Г. /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

**Оглавление**

[Введение 6](#_Toc36653284)

[Техническое задание 7](#_Toc36653285)

[Задача разработки системы 7](#_Toc36653286)

[Функционал разрабатываемой системы 7](#_Toc36653287)

[Обзор USB 7](#_Toc36653288)

[Проектирование USB-устройства 9](#_Toc36653289)

[Состав и конструктивные решения схемы USB-устройства. 9](#_Toc36653290)

[Питание микроконтроллера 9](#_Toc36653291)

[Кабель USB 10](#_Toc36653292)

[Регулирование уровня напряжения 10](#_Toc36653293)

[Проектирование в системах автоматизированного проектирования 11](#_Toc36653294)

[Программная реализация USB-устройства 12](#_Toc36653295)

[HID Configuration 12](#_Toc36653296)

[USB Library 17](#_Toc36653297)

[Реализация хоста 19](#_Toc36653298)

[Дополнения и улучшения 28](#_Toc36653299)

[Заключение 30](#_Toc36653300)

[Список литературы. 31](#_Toc36653301)

[Приложение. Листинг программы. 32](#_Toc36653302)

# Введение

В настоящее время микроконтроллеры активно используются в промышленном оборудовании и бытовых устройствах, они работают в огромном числе самых разнообразных любительских микроконтроллерных проектов и устройств.

Микроконтроллеры PIC, на котором и будет стояться будущий проект, являются одними из самых популярных среди микрочипов. В данной работе будет использоваться микроконтроллер PIC18F4550 для разработки продукта на основе USB.

Одним из средств визуализации процессов в схемах является пакет «Proteus», который снабжен необходимым функционалом для разработки проекта.

USB на сегодняшний день, пожалуй, самый распространенный тип интерфейса передачи данных. Все мобильные устройства и настольные электрические приборы имеют в своем арсенале данные порты для передачи данных. Современные гаджеты могут не только обмениваться информацией через USB или micro-USB, но и осуществлять зарядку аккумуляторов. Постоянно появляются все новые версии этого интерфейса, совершенствование продолжается и его популярность только растет.

# Техническое задание

## Задача разработки системы

Задача, которую необходимо решить в ходе выполнения работы, заключается в проектировании и реализации программной части USB-устройства, а также элементов для визуализации процесса поступления сигналов и передачи данных.

Данная задача разделяется на следующие подзадачи:

* Анализ предметной области и формирование технического задания.
* Проектирование схемы алгоритма работы контроллера (программных модулей управления)
* Разработка схемы, определяющей состав и назначение составляющих.

## Функционал разрабатываемой системы

Система должна корректно отображать следующий функционал:

* Корректная работа передачи данных от хоста к USB;
* Корректная работа передачи данных от USB к хосту;
* Индикация отображения передачи данных;

Так же текущая задача будет разделена на две части, в первой части будет рассмотрено, то как использовать компилятор mikroC для разработки простого проекта USB HID, а в другой части, как использовать USB Library Function для работы с USB.

## Обзор USB

Программное обеспечение хоста делится на два отдельных типа. Программное обеспечение инициализации канала связи и программное обеспечение поддержки рабочего режима обмена данными. Программное обеспечение инициализации начинает работать при подключении к хосту нового USB устройства. Происходит обмен служебной информацией между хостом и USB устройством. В результате обменов служебной информацией, хост определяет: тип устройства, его требования к энергопотреблению, возможность поддержки «спящего режима», тип драйверов для правильной работы USB устройства, и даже возможна загрузка необходимых прикладных программ для работы. Эти новые веяния в духе спецификации PNP (plug and play). Устройства могут подключаться и отключаться в горячем режиме. При подключении и отключении происходит автоматическое переконфигурирование программного обеспечения хоста. Процесс настройки хоста на обмен данными, напоминает процесс раскрутки. Первоначально обмениваются простейшими сигналами по шине, затем сообщения усложняются и наконец, выход на рабочий режим.

Программное обеспечение рабочего режима поддерживает обмен данными, когда хост соответственно сконфигурирован и USB устройство вышло на рабочий режим обмена. В спецификации USB этот начальный процесс называется энумерацией.

В последнее время имеется тенденция к унификации не только протоколов обмена, но и унификация устройств, взаимодействующих с персональным компьютером. Точнее, унификация требований к каналу связи. Создается универсальная шина для любых устройств, которые можно подключить, которая сложно устроена, она многоуровневая, гибкая и адаптируемая для разных конфигураций устройств. Унифицируются и драйвера операционной системы персонального компьютера, который взаимодействует с подключаемым устройством. Преимущество данного подхода - отпадает необходимость в разработке драйвера для ОС, разработчиками USB устройства. Это должно повышать надёжность ОС, так как разработкой драйверов могут заняться разработчики ОС, а не разработчики устройств. Минус данного подхода - очевидная сложность и связанная с ней избыточность, громоздкость технических решений.

В связи с выше сказанным, в спецификации USB вводится понятие класса устройств. Все электронные устройства, подключаемые к персональному компьютеру, по своим функциональным качествам очень схожи. Например, звуковые платы предоставляют сервис приблизительно одного уровня. Поэтому устройства стали делить на унифицированные классы. Класс - это группа устройств, объединённых общими характеристиками и способных управляться общим для них программным драйвером операционной системы. Отдельное устройство может объединять функциональность сразу нескольких устройств, принадлежащих разным классам. Если функциональность вашего устройства подходит к некоторому классу, и оно поддерживает спецификацию USB для устройств класса, то нет необходимости писать драйвер для ОС. Вероятнее всего, драйвер уже имеется в ОС. Функциональность устройства, подпадающего под определённый класс, может быть расширена разработчиком устройства добавлением отдельных команд. В стандарте USB предусмотрена возможность некоторого расширения функциональности. В стандарте предусмотрено множество возможностей, которые разработчик устройства может использовать для своих целей, добавляя к базовой функциональности, функциональность расширенную. В спецификации USB, есть USB базового уровня и протоколы устройств классов. Протоколы устройств классов – это некоторая надстройка над протоколами нижнего, базового уровня USB.

# Проектирование USB-устройства

## Состав и конструктивные решения схемы USB-устройства.

Рассмотрев существующее решения, мы можем прийти к выводу, что USB-устройство должно состоять из кабеля USB, светодиодов (LED), резисторов, линейного стабилизатора, микроконтроллера PIC.

## Питание микроконтроллера

Для того чтобы организовать питание для создаваемого устройства USB понадобится:

* Кабель USB и линейка из 4 контактов с шагом 2.54 мм.
* Светодиод (LED) и резистор на 330 Ом.
* Микросхема линейного стабилизатора с низким падением напряжения на 3.3V (Low dropout voltage regulator), LD1086V33.

## Кабель USB



Рис. 1. Внешний вид коннектора

|  |  |  |
| --- | --- | --- |
| **№ контакта** | **Цвет провода** | **Назначение** |
| 1 | Красный | VCC (+5V) |
| 2 | Белый | D- |
| 3 | Зеленый | D+ |
| 4 | Черный | Земля (общий провод, GND) |

Таблица 1. Назначение проводов USB

## Регулирование уровня напряжения

В то время как питание USB осуществляется напряжением 5V, сигналы данных шины USB (D+, D-) требуют логических уровней 3.3V. Если подать питание 5V на микроконтроллер, то он будет работать с логическими уровнями 5V, что будет несовместимым с уровнями сигналов шины USB. В общем случае, как это рекомендуется в руководстве библиотеки V-USB, можно решить проблему согласований уровней сигналов следующими способами:

* Ограничить напряжение питания, поступающее от USB, уровнем 3.3V.
* Подать питание на схему внешнего источника постоянного напряжения 3.3V.
* Использовать резисторы, диоды или стабилитроны (диоды Зенера), чтобы преобразовать уровни сигналов 5V к сигналам уровня 3.3V.

В данной работе я выбрал первый вариант – с помощью использования микросхемы линейного регулятора LD1086V33. Как можно узнать из руководства на эту микросхему, вывод 1 должен быть подключен к «земле» (GND, шасси компьютера), вывод 2 будет выходом стабилизатора (выдает напряжение 3.3V) и вывод 3 - вход (на него должно быть подано напряжение +5V от интерфейса USB).

Для снижения помех (пульсаций напряжения питания) на входе и выходе регулятора можно подключить фильтрующие (блокировочные) конденсаторы номиналом 10 мкф.

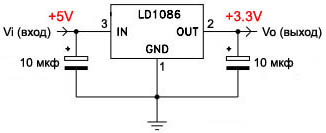


Рис. 2. Пример установки конденсаторов для снижения помех

Для создания устройства USB был использован PIC18F4550-микроконтроллер.

## Проектирование в системах автоматизированного проектирования

Для проектирования системы будем использовать пакет программ для автоматизированного проектирования электронных схем «Proteus».

Соединения на плате должны быть выполнены по данной схеме:

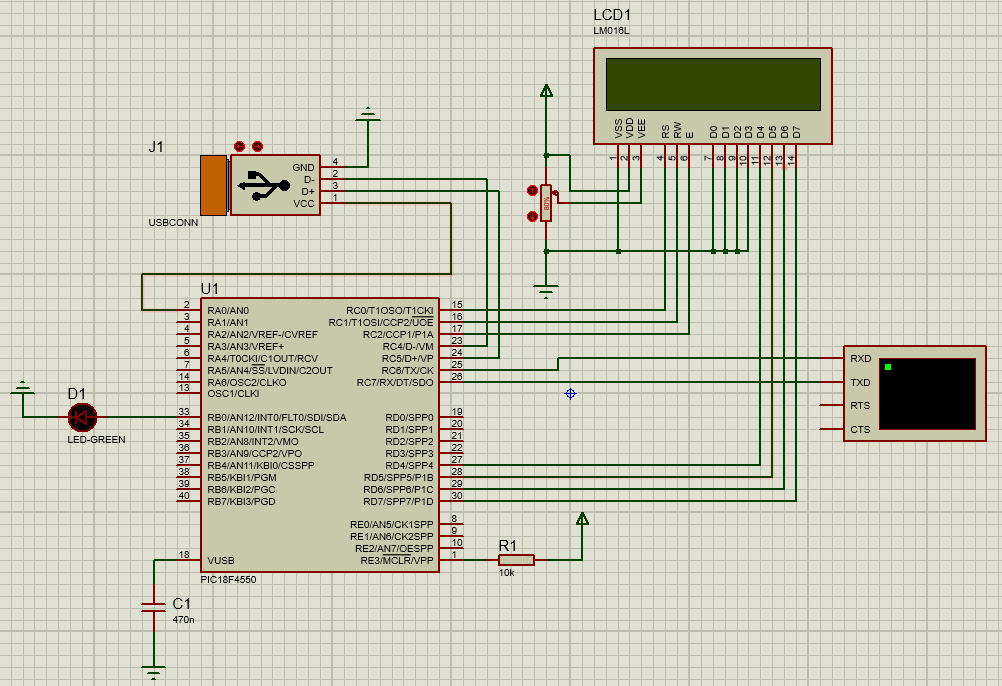


Рис. 3. Принципиальная схема - демонстрации на плате разработки Labcenter Electronics Proteus и PIC18F4550 USB

# Программная реализация USB-устройства

Программную реализацию системы управления будем осуществлять в интегрированной среде разработки для микроконтроллеров семейства PIC «mikroC PRO for PIC».

Для подключения программной реализации в схемную, будем использовать код, подключаемый в среде разработки «Proteus».

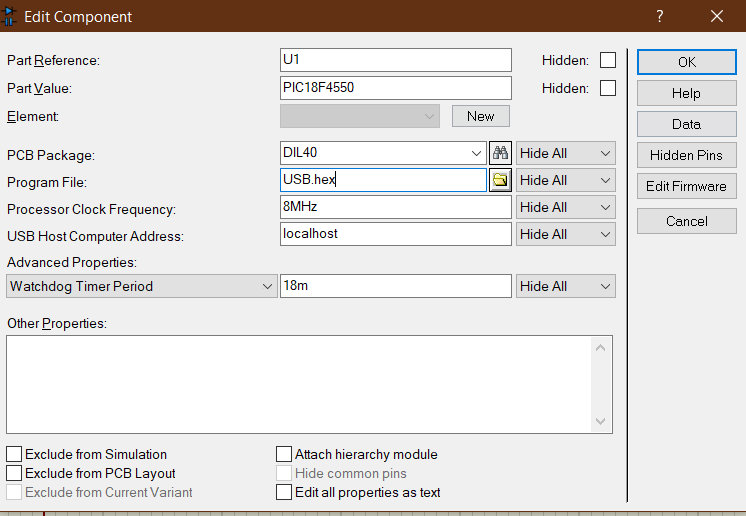


Рис. 4 Добавление файла программы в микроконтроллер

Код программы делится на 3 части: HID Configuration, функции USB Library и код хоста для обеспечения обмена информацией.

## HID Configuration

Дескриптор устройства (USB\_DEVICE\_DESCRIPTOR) является корнем дерева дескрипторов и содержит основную информацию об устройстве. Уникальные номера, idVendor и idProduct, идентифицируют подключенное устройство. Операционная система Windows использует эти цифры для определения того, какой драйвер устройства должен быть загружен.

idVendor-это номер, присвоенный каждой компании, производящей устройства на базе USB. Форум разработчиков USB отвечает за администрирование назначения идентификаторов поставщиков.

IdProduct - это еще одно 16-разрядное поле, содержащее номер, присвоенный производителем для идентификации конкретного продукта.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Поле** | **Тип** | **Размер** | **Значение** | **Описание** |
| 0 | bLength | uint8\_t | 1 | Number | Размер этого дескриптора в байтах.. |
| 1 | bDescriptorType | uint8\_t | 1 | Constant | Тип Дескриптора Устройства = 1. |
| 2 | bcdUSB | uint16\_t | 2 | BCD | USB Номер выпуска спецификации в двоично-десятичном коде (т. е. 2.10 - это 210h).  Это поле определяет выпуск спецификации USB, с которой устройство и его дескрипторы совместимы. |
| 4 | bDeviceClass | uint8\_t | 1 | Class | Код класса (присваивается USB-IF).  Если это поле является:   * при сбросе до нуля каждый интерфейс в конфигурации задает свою собственную информацию о классе, и различные интерфейсы работают независимо. * если задано значение от 1 до FEh, устройство поддерживает различные спецификации классов на разных интерфейсах, и интерфейсы не могут работать независимо друг от друга. Это значение определяет определение класса, используемое для агрегатных интерфейсов. * установите значение FFh, класс устройства зависит от поставщика. |
| 5 | bDeviceSubClass | uint8\_t | 1 | SubClass | Код подкласса (присваивается USB-IF).  Эти коды определяются значением поля bDeviceClass.  Если bDeviceClass это:   * сброс к нулю, это поле также должно быть сброшено к нулю. * не установлено значение FFh, все значения зарезервированы для присвоения USB-IF. |
| 6 | bDeviceProtocol | uint8\_t | 1 | Protocol | Код протокола (присваивается USB-IF).  Эти коды определяются значением полей bDeviceClass и bDeviceSubClass. Если устройство поддерживает специфические для класса протоколы на основе устройства, а не на основе интерфейса, этот код идентифицирует протоколы, используемые устройством, как определено спецификацией класса устройства.  Если это поле является:   * сброшенный до нуля, устройство не использует протоколы класса specific на основе устройства. Однако он может использовать специфические для класса протоколы на основе интерфейса. * если задано значение FFh, устройство использует определенный поставщиком протокол на основе устройства. |
| 7 | bMaxPacketSize0 | uint8\_t | 1 | Number | Максимальный размер пакета для конечной точки ноль (допустимы только 8, 16, 32 или 64). |
| 8 | idVendor | uint16\_t | 2 | ID | Идентификатор поставщика (присваивается USB-IF). |
| 10 | idProduct | uint16\_t | 2 | ID | Идентификатор продукта (присваивается производителем). |
| 12 | bcdDevice | uint16\_t | 2 | BCD | Номер выпуска устройства в двоично-десятичном коде. |
| 14 | iManufacturer | uint8\_t | 1 | Index | Индекс строкового дескриптора, описывающего производителя. |
| 15 | iProduct | uint8\_t | 1 | Index | Индекс строкового дескриптора, описывающего продукт. |
| 16 | iSerialNumber | uint8\_t | 1 | Index | Индекс строкового дескриптора, описывающего серийный номер устройства. |
| 17 | bNumConfigurations | uint8\_t | 1 | Number | Количество возможных конфигураций. |

Пример описания имени USB:

Product string descriptor

const struct{

char bLength;

char bDscType;

unsigned int string[8];

}strd3={

18, //sizeof this descriptor string

0x03,

{'U','S','B','\_','T','e','s','t'}

};

Пример описание девайса USB:

const struct {

char bLength; // bLength - Descriptor size in bytes (12h)

char bDescriptorType; // bDescriptorType - The constant DEVICE (01h)

unsigned int bcdUSB; // bcdUSB - USB specification release number (BCD)

char bDeviceClass; // bDeviceClass - Class Code

char bDeviceSubClass; // bDeviceSubClass - Subclass code

char bDeviceProtocol; // bDeviceProtocol - Protocol code

char bMaxPacketSize0; // bMaxPacketSize0 - Maximum packet size for endpoint 0

unsigned int idVendor; // idVendor - Vendor ID

unsigned int idProduct; // idProduct - Product ID

unsigned int bcdDevice; // bcdDevice - Device release number (BCD)

char iManufacturer; // iManufacturer - Index of string descriptor for the manufacturer

char iProduct; // iProduct - Index of string descriptor for the product.

char iSerialNumber; // iSerialNumber - Index of string descriptor for the serial number.

char bNumConfigurations; // bNumConfigurations - Number of possible configurations

} device\_dsc = {

0x12, // bLength

0x01, // bDescriptorType

0x0200, // bcdUSB

0x00, // bDeviceClass

0x00, // bDeviceSubClass

0x00, // bDeviceProtocol

8, // bMaxPacketSize0

USB\_VENDOR\_ID, // idVendor

USB\_PRODUCT\_ID, // idProduct

0x0001, // bcdDevice

0x01, // iManufacturer

0x02, // iProduct

0x00, // iSerialNumber

0x01 // bNumConfigurations

};

Дескриптор конфигурации (USB\_CONFIGURATION\_DESCRIPTOR) содержит информацию о требованиях к питанию устройства и количестве интерфейсов, которые оно может поддерживать. Устройство может иметь несколько конфигураций. Хост может выбрать конфигурацию, которая наилучшим образом соответствует требованиям прикладного программного обеспечения.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Поле** | **Тип** | **Размер** | **Значение** | **Описание** |
| 0 | bLength | uint8\_t | 1 | Number | Размер этого дескриптора в байтах. |
| 1 | bDescriptorType | uint8\_t | 1 | Constant | Тип Дескриптора Конфигурации = 2. |
| 2 | wTotalLength | uint16\_t | 2 | Number | Общая длина данных, возвращаемых для этой конфигурации. Включает в себя совокупную длину всех дескрипторов (конфигурация, интерфейс, конечная точка и класс или поставщик), возвращаемых для этой конфигурации. |
| 4 | bNumInterfaces | uint8\_t | 1 | Number | Количество интерфейсов, поддерживаемых этой конфигурацией. |
| 5 | bConfigurationValue | uint8\_t | 1 | Number | Значение для выбора этой конфигурации с помощью *SetConfiguration()*. |
| 6 | iConfiguration | uint8\_t | 1 | Index | Индекс строкового дескриптора, описывающего данную конфигурацию. |
| 7 | bmAttributes | uint8\_t | 1 | Bitmap | Характеристики конфигурации   * D7: Зарезервировано (должно быть установлено значение один по историческим причинам) * D6: Без батарейный * D5: Дистанционное Управление * D4...0: Зарезервировано (сброс до нуля)   Конфигурация устройства, использующего питание от шины и локального источника, сообщает ненулевое значение в *bMaxPower* для указания количества требуемой мощности Шины и наборов D6. Фактический источник питания во время выполнения может быть определен с помощью GetStatus(DEVICE) запроса. Если конфигурация устройства поддерживает удаленное пробуждение, D5 имеет значение 1. |
| 8 | bMaxPower | uint8\_t | 1 | mA | Максимальная потребляемая мощность USB устройства от шины в этой конкретной конфигурации, когда устройство полностью работоспособно. Выражается в 2mA (то есть, 50 = 100mA). |

## USB Library

Первым делом нужно оптимизировать работу продуктов разработки проектов Proteus (т.е. установить USB Virtual Drivers), это позволит использовать виртуальные USB-накопители. Пакет содержит набор драйверов, необходимых для моделирования USB-накопителей, работающих с поддерживаемыми моделями микроконтроллеров. Это нужно для Proteus, его виртуальных драйверов USB чтобы создать симуляцию USB с поддерживаемых микроконтроллеров. Также нужно установить Драйвер виртуального последовательного порта - Драйвер виртуального последовательного порта от Eltima Software необходим для создания и эмуляции соединений виртуальных COM-портов. Оно создает виртуальные последовательные порты и соединяет их попарно виртуальным нуль-модемным кабелем. Приложения с обеих концов пары смогут обмениваться данными так, что все написанное первому порту появится во втором и наоборот.

Все виртуальные последовательные порты работают и ведут себя в точности как аппаратные, эмулируя все их настройки. Вы можете создать столько пар виртуальных портов, сколько Вам угодно, и при этом не будет ни нехватки последовательных портов, ни нагромождения дополнительного оборудования на Вашем столе.

Теперь кратко разберем USB Library. Ниже приводится краткое описание некоторых функций библиотеки USB.

Библиотека USB содержит процедуры HID, которые поддерживают устройства класса HID при использовании микроконтроллера PIC со встроенным USB (например, PIC18F4550) и выводами портов RC4 и RC5 подключаются к D+ и D - выводам USB-разъема соответственно.

Библиотека поддерживает также общие процедуры, которые могут использоваться с драйверами, указанными поставщиком.

**Hid\_Enable**

Эта функция обеспечивает связь по USB и требует наличия двух аргументов: адреса буфера чтения и адреса буфера записи.

Прежде чем вы сможете использовать какие-либо другие функции, эта функция должна быть вызвана первой.

**Example:**

unsigned char readbuff[64] absolute 0x500; // Buffers should be in USB RAM, please consult datasheet

unsigned char writebuff[64] absolute 0x540;

HID\_Enable(&readbuff,&writebuff);           // Enable HID communication

**Hid\_Read**

Эта функция получает данные от шины USB и сохраняет их в буфере чтения. Он не имеет аргументов, но возвращает количество полученных символов.

**Example:**

// keep trying to receive until success

while(!HID\_Read());

**Hid\_Write**

Эта функция отправляет данные из буфера записи на шину USB. Имя буфера (тот же буфер, что и при инициализации) и длина передаваемых данных должны быть указаны в качестве аргументов функции. Если передача данных завершилась неудачно, функция возвращает 0. В противном случае он возвращает количество переданных байт и всегда обязательно вызывает эту функцию повторно до тех пор, пока данные не будут успешно отправлены.

**Example:**

// keep trying to send until success

while(!HID\_Write(&writebuff,64));

**Hid\_Disable**

Эта функция отключает передачу данных по USB. Он не имеет аргументов и не возвращает никаких данных.

**Example:**

//Disable the USB data transfer

HID\_Disable();

**USB\_Interrupt\_Proc**

Эта процедура используется для обслуживания различных событий шины USB. Должен быть вызван внутри процедуры прерывания USB.

**Example:**

void interrupt()

{

USB\_Interrupt\_Proc();

}

Заметка: Не используйте эту функцию с USB\_Polling\_Proc, следует использовать только одну. Чтобы включить обслуживание через прерывание, необходимо установить константу USB\_INTERRUPT (она задается по умолчанию в файле дескриптора).

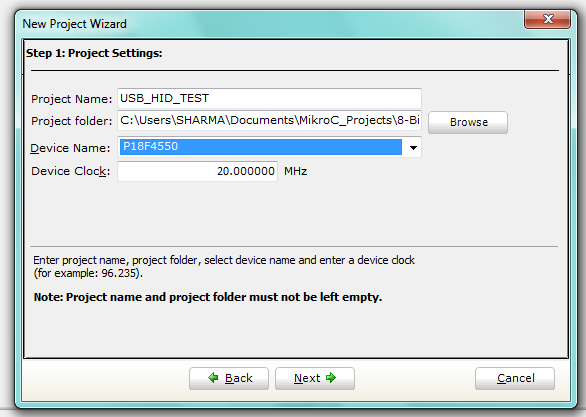
## Реализация хоста

ПО хоста - это программа, работающая на хосте USB. Под хостом USB подразумевается компьютер (на шине USB всегда есть 1 хост и может быть одно или большее количество устройств USB), так что ПО хоста это компьютерная программа, которая управляет устройством USB.

Необходимо написать вспомогательную функцию, чтобы расшифровать строки дескрипторов USB, используемые для имени производителя (vendor name) и имени устройства USB (device name). Для этой цели и впоследствии для обмена с устройством USB будем использовать функцию USB\_Interrupt\_Proc () подключенную из файла USBdsc.c(HID Configuration).

USB-это очень сложное устройство для обработки, но у mikroC есть встроенная функция по работе с USB.

Создаем новый проект в mikroC и выберите mico-контроллер PIC18F4550 со значением тактовой частоты устройства 20 МГц.



После чего создаем файл USB.c, в котором и будет наш код программы.

Для начала нужно перечислить все пины для LCD и вызвать USB\_Interrupt\_Proc().

unsigned char Read\_Buffer[64] absolute 0x500;

unsigned char Write\_Buffer[64]absolute 0x510;

unsigned char num,flag;

void interrupt()

{

USB\_Interrupt\_Proc();

TMR0L = 100; //Reload Value

INTCON.TMR0IF = 0; //Re-Enable Timer-0 Interrupt

}

//LCD 8-bit Mode Connection

sbit LCD\_RS at RC0\_bit;

sbit LCD\_RW at RC1\_bit;

sbit LCD\_EN at RC2\_bit;

sbit LCD\_D7 at RD7\_bit;

sbit LCD\_D6 at RD6\_bit;

sbit LCD\_D5 at RD5\_bit;

sbit LCD\_D4 at RD4\_bit;

sbit LCD\_D3 at RD3\_bit;

sbit LCD\_RS\_Direction at TRISC0\_bit;

sbit LCD\_RW\_Direction at TRISC1\_bit;

sbit LCD\_EN\_Direction at TRISC2\_bit;

sbit LCD\_D7\_Direction at TRISD7\_bit;

sbit LCD\_D6\_Direction at TRISD6\_bit;

sbit LCD\_D5\_Direction at TRISD5\_bit;

sbit LCD\_D4\_Direction at TRISD4\_bit;

sbit LCD\_D3\_Direction at TRISD3\_bit;

// End Lcd module connections

Следующим шагом является инициализация дисплея (создание функций вывода, очистки экрана), настройка пин-кода как цифрового, отключить компараторы, задать начальные значения для пинов (0 для лампы, т.к. USB не запущено).

После всех приготовлений и инициализаций, следует описать работу с USB:

* Подключение USB
* Проверка Идентификация
* Отображение шагов действий (USB и Lcd)
* Обработчик на полученные сведения на USB

**Подключение:**

if(PORTA.B0 == 1)

{

UART1\_Write\_Text\_Newline("Data is Ready to be Received from the PC");

Hid\_Enable(&Read\_Buffer,&Write\_Buffer);

UART1\_Write\_Text\_Newline("USB connected, waiting for enumeration...");

Delay\_ms(2000);

UART1\_Write\_Text\_Newline("OK");

PORTB.B0 = 1;

}

**Идентификация**

if(strncmp(Read\_Buffer,"S",1) == 0)

{

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Out(1,2,"Authentication");

Lcd\_Out(2,8,"OK");

UART1\_Write\_Text\_Newline("USB Authentication - OK");

UART1\_Write\_Text\_Newline("USB enumaration by PC/HOST");

goto loop;

}

else

{

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Out(1,2,"Authentication");

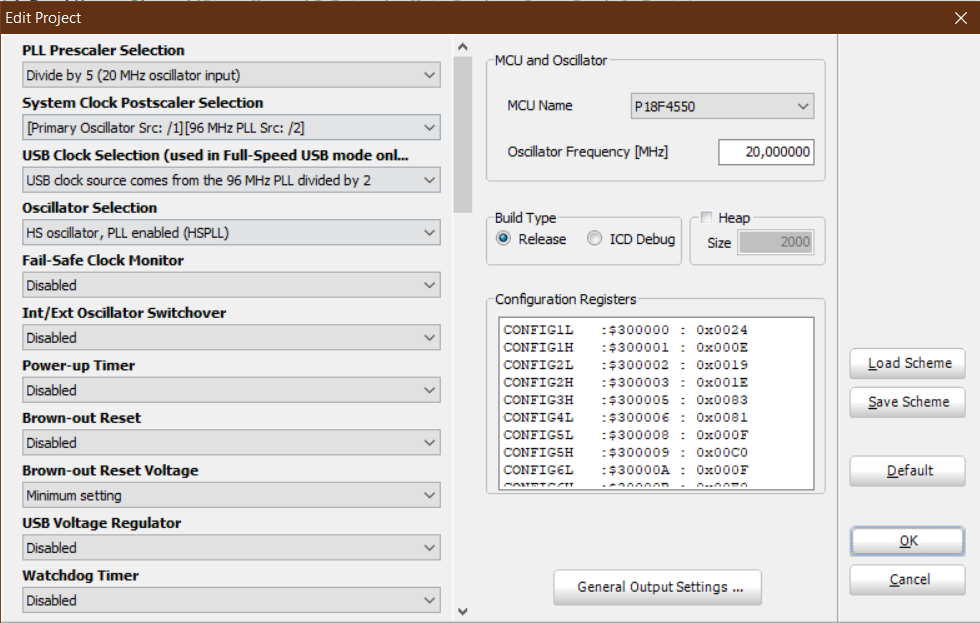
Lcd\_Out(2,5,"Fails!");

UART1\_Write\_Text\_Newline("USB Authentication - Fails!");

goto start;

}

Теперь пришло время отредактировать бит конфигурации, так как они играют решающую роль при работе с PIC Micro.

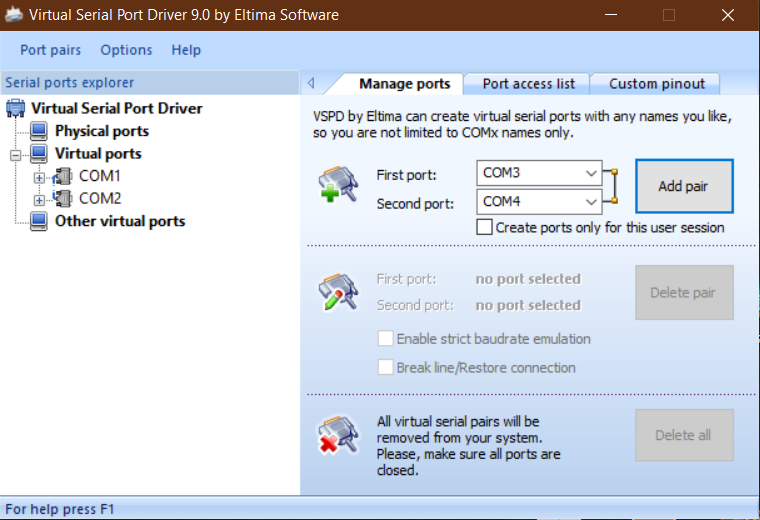


Теперь мы находимся в состоянии проверить соединение USB HID.

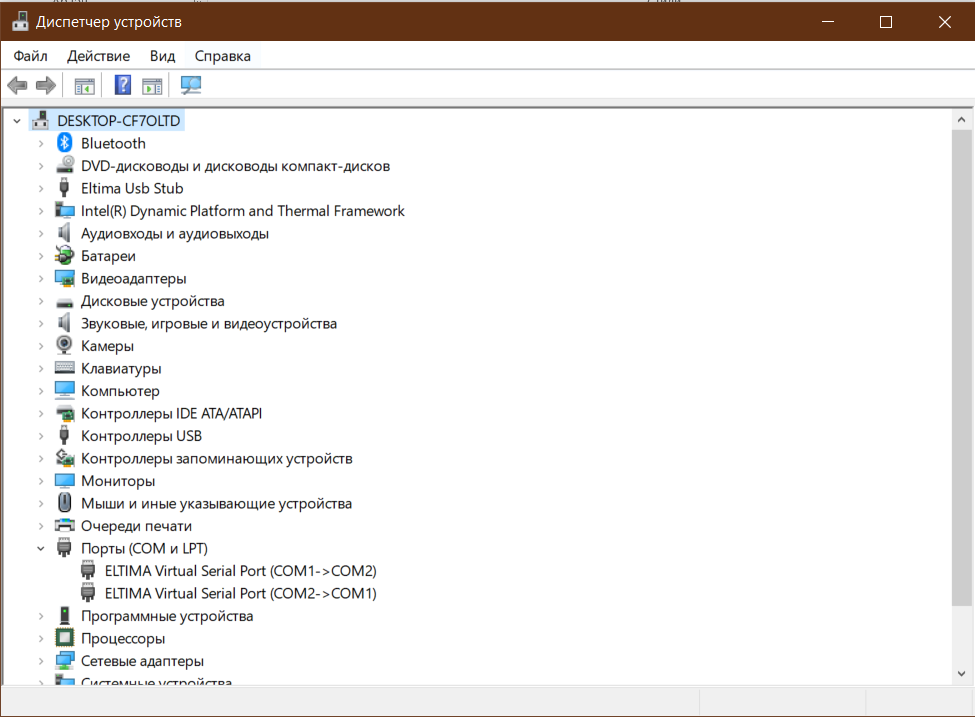
Перед подключением платы к USB-порту или запуском моделирования в Proteus откройте HID Termal и USART-терминал, присутствующие в меню mikroC tools.

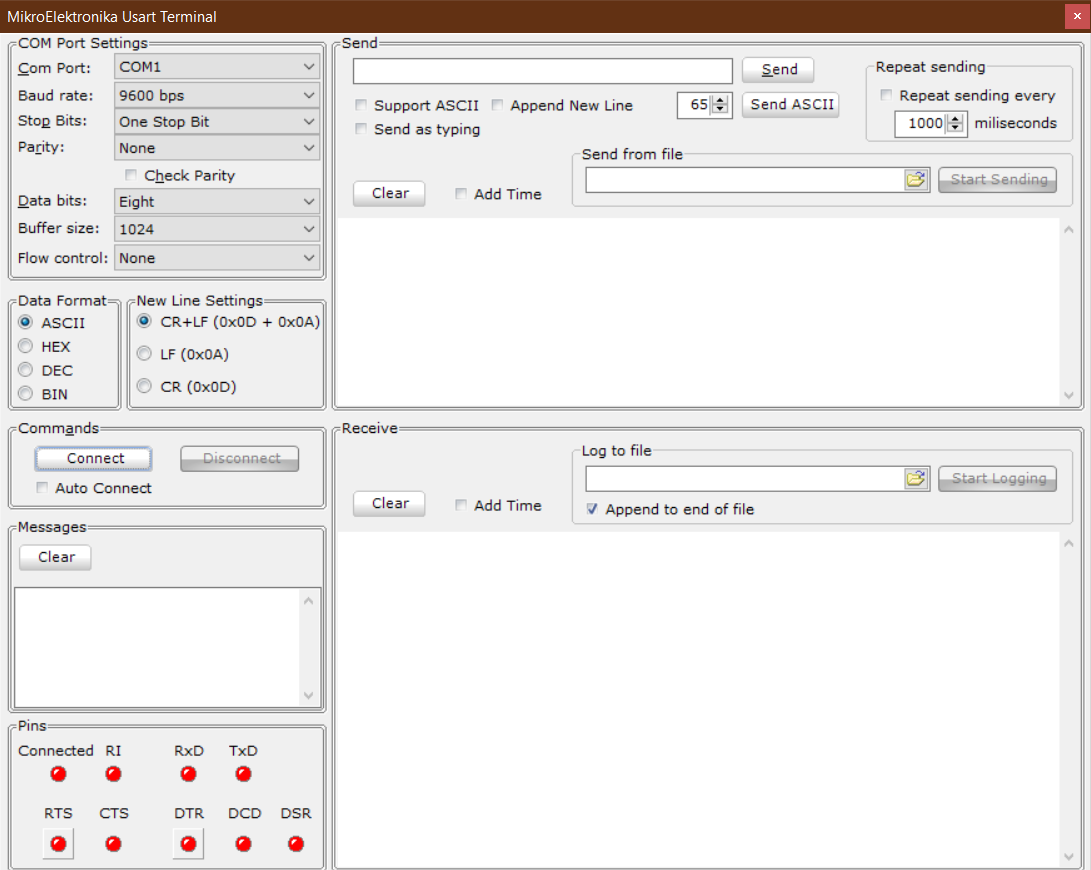
Настройте терминал USART на скорость передачи данных 9600 бод и выберите свой COM-порт, который у меня есть COM1 в моей системе. Нажмите на кнопку Connect, вы получите статус на терминале USART.

Также через Virtual Serial Port Driver создаем 2 виртуальных COM порта.

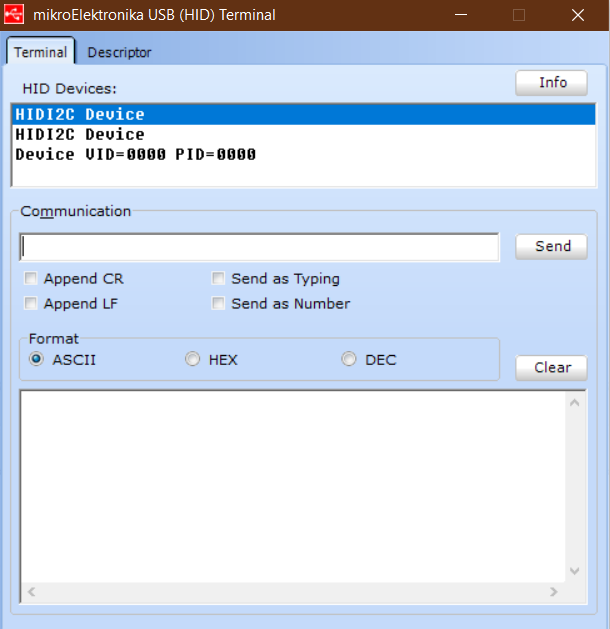


После этого можно их посмотреть в диспетчере устройств.

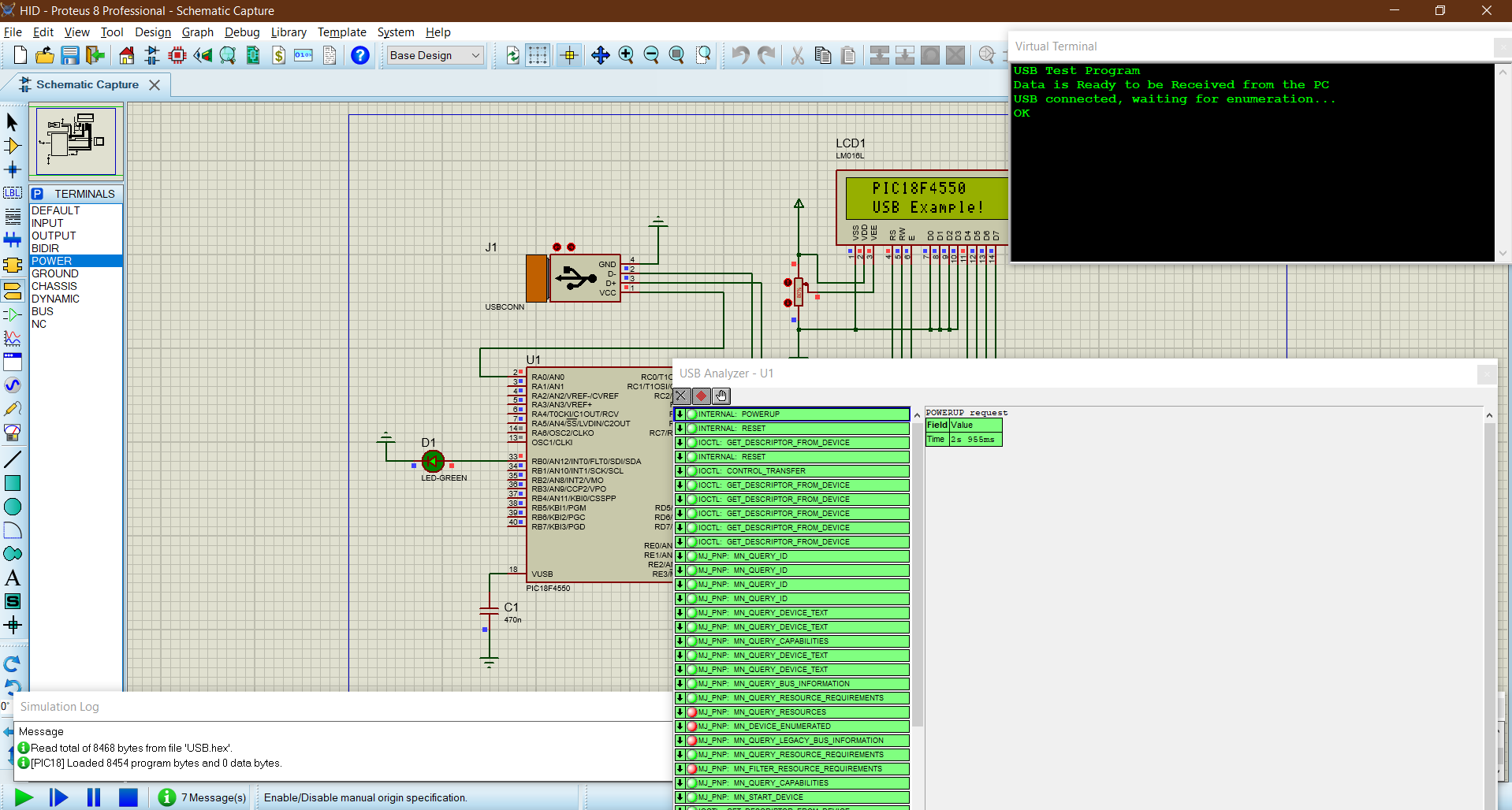




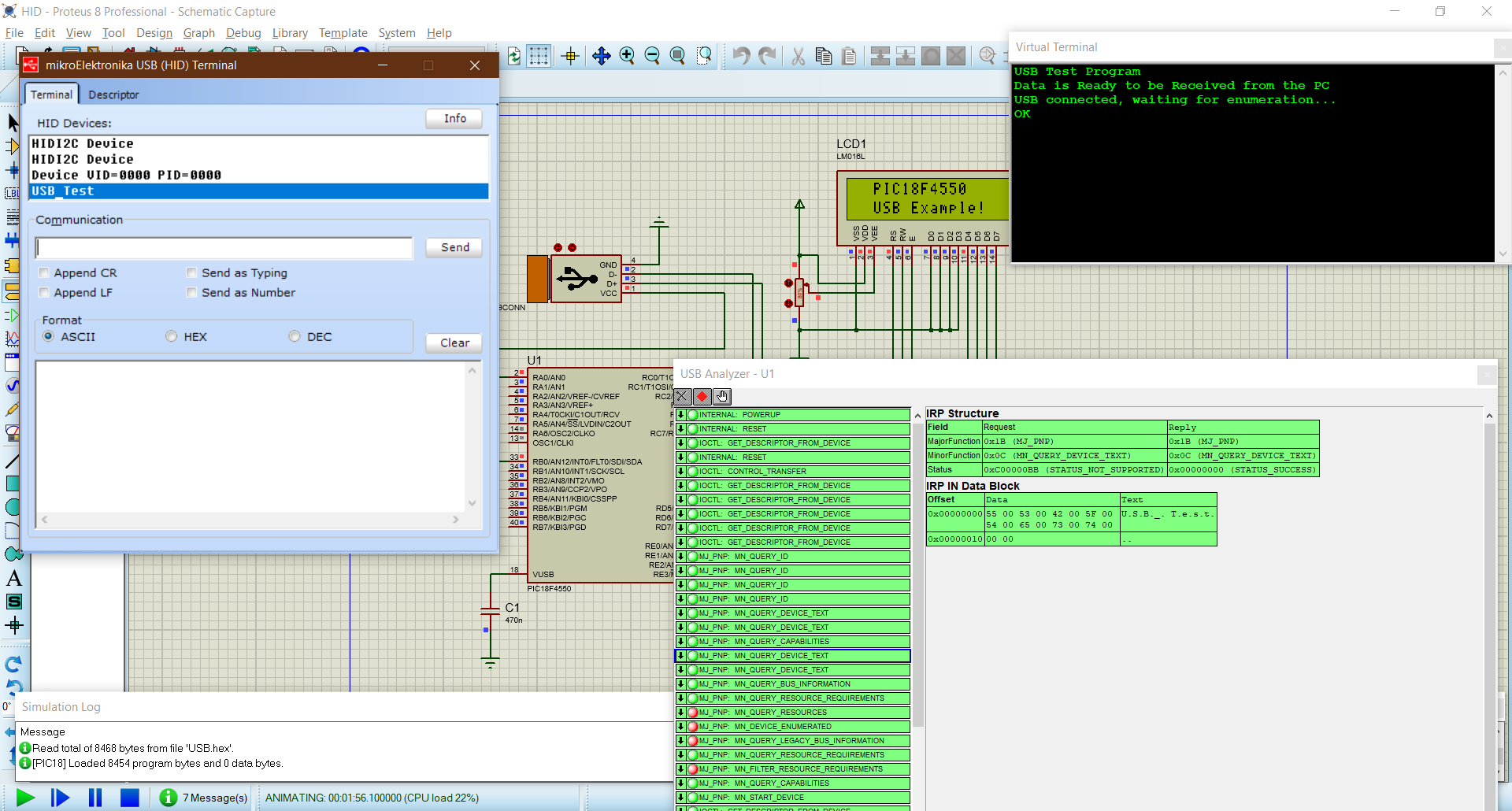
Теперь откройте HID-терминал mikroC, это программное обеспечение перечислит все USB-устройства HID, подключенные к ПК, как показано ниже.



Теперь подключите ваше оборудование к ПК, через моделирование, которое запускается на программном обеспечении Proteus.

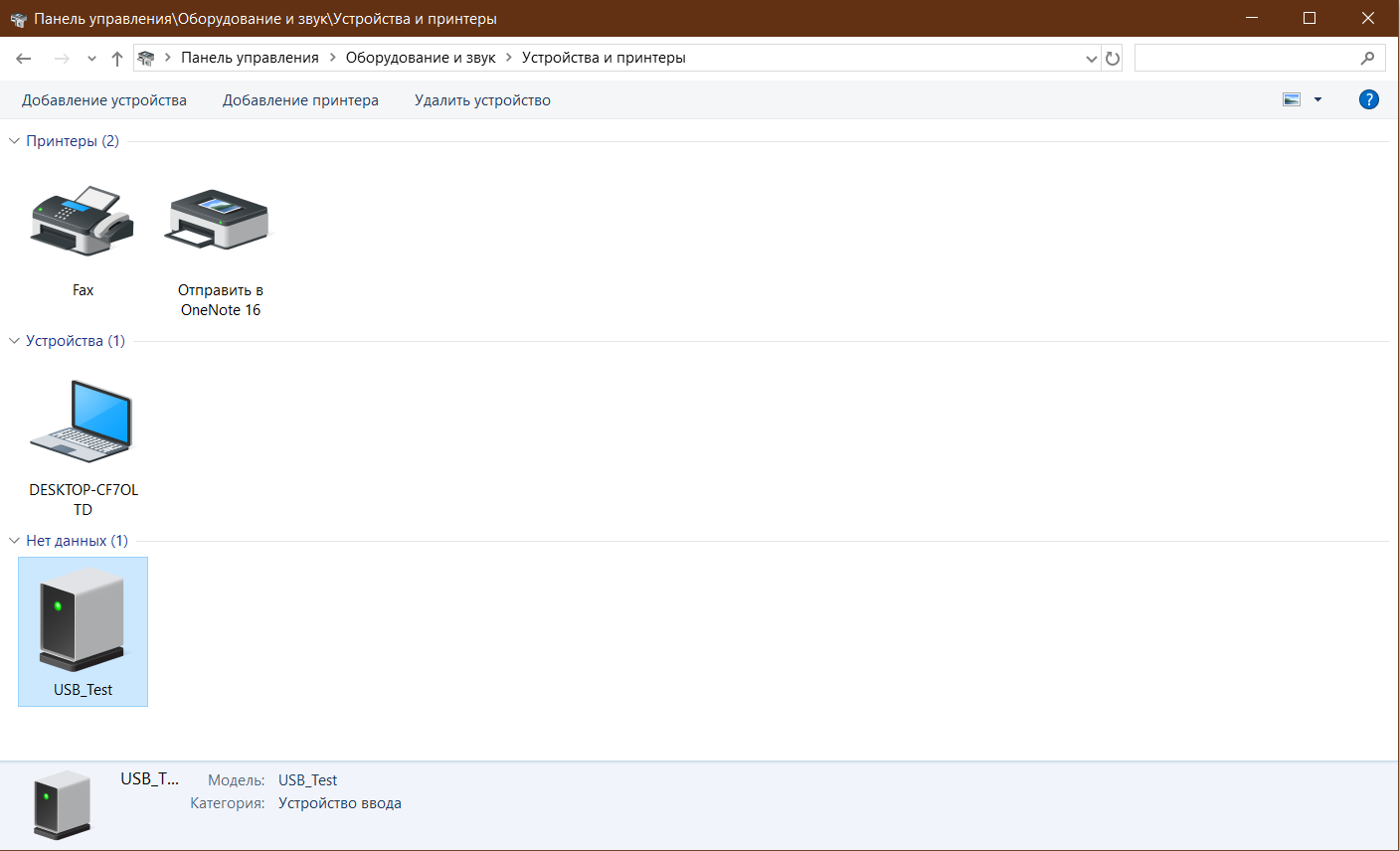


Начинается перечисление USB



Теперь наше устройство обнаруживается терминалом mikroC HID

Так же мы должны проверить USB в Диспетчере устройств Windows.

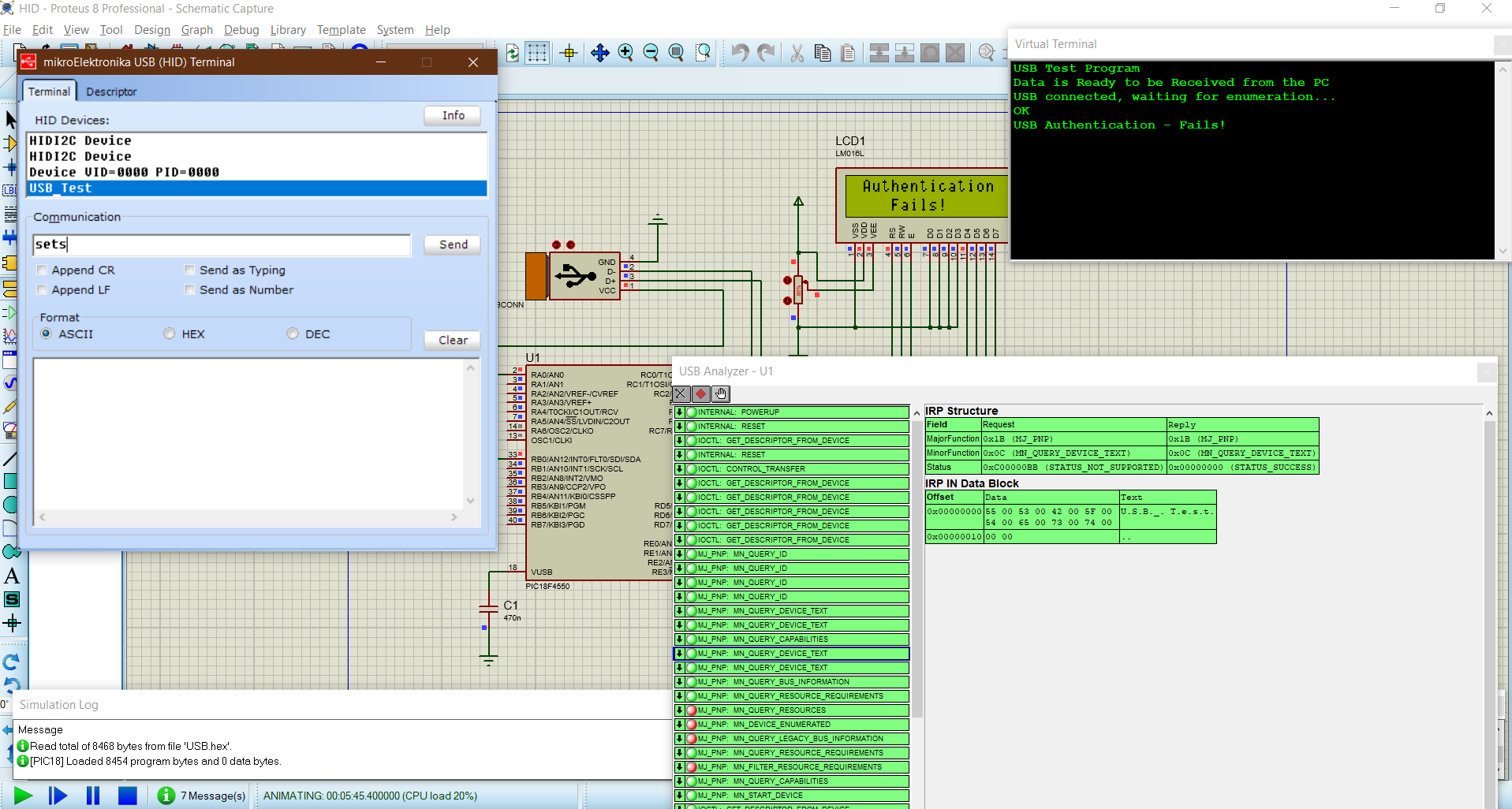


Теперь пришло время отправить некоторые данные с ПК на PIC18F4550. Сначала я должен отправить "S" через USB, это просто как проверка безопасности, если кто-то не отправит этот символ, то мы постоянно будем получать сбой аутентификации.

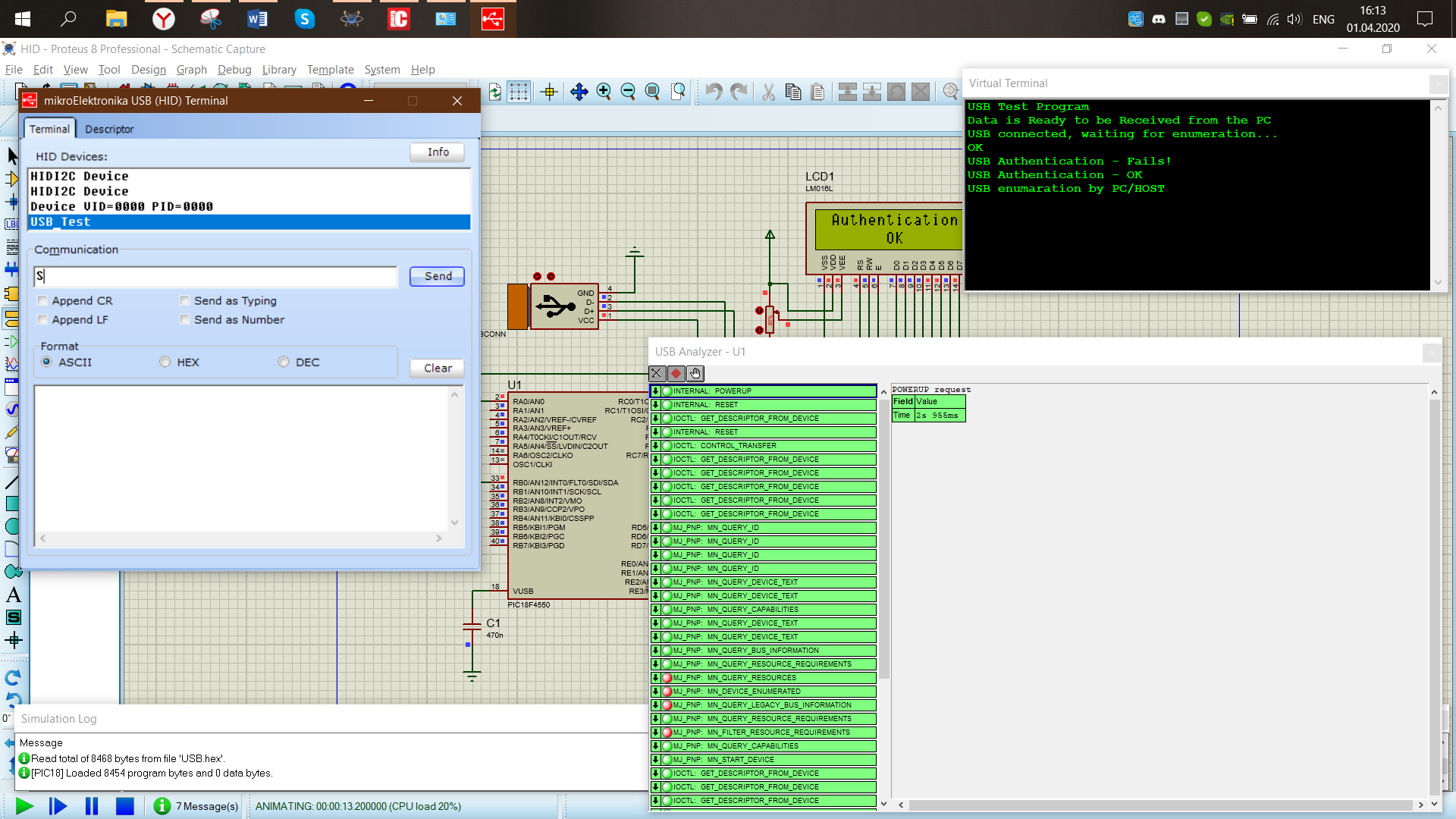
Для того, чтобы действовать правильно, вы должны сначала отправить "S", а затем продолжить дальше.

Прошивка написана таким образом, что строка, отправленная Hid-терминалом, будет отображаться на жидкокристаллическом дисплее.

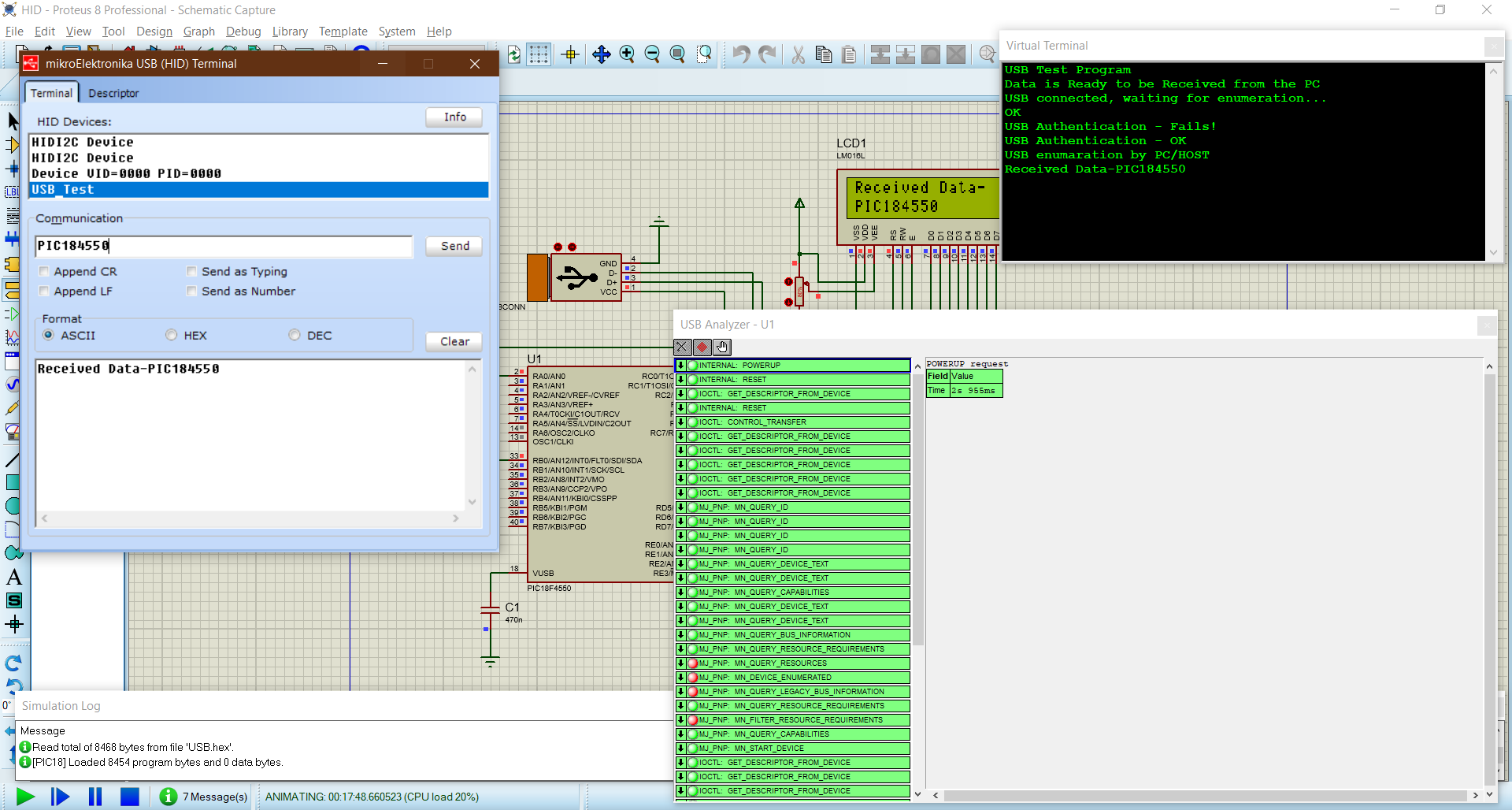
Сначала посмотрим, не посылается ли один неверный символ, для проверки провальной аутентификации.



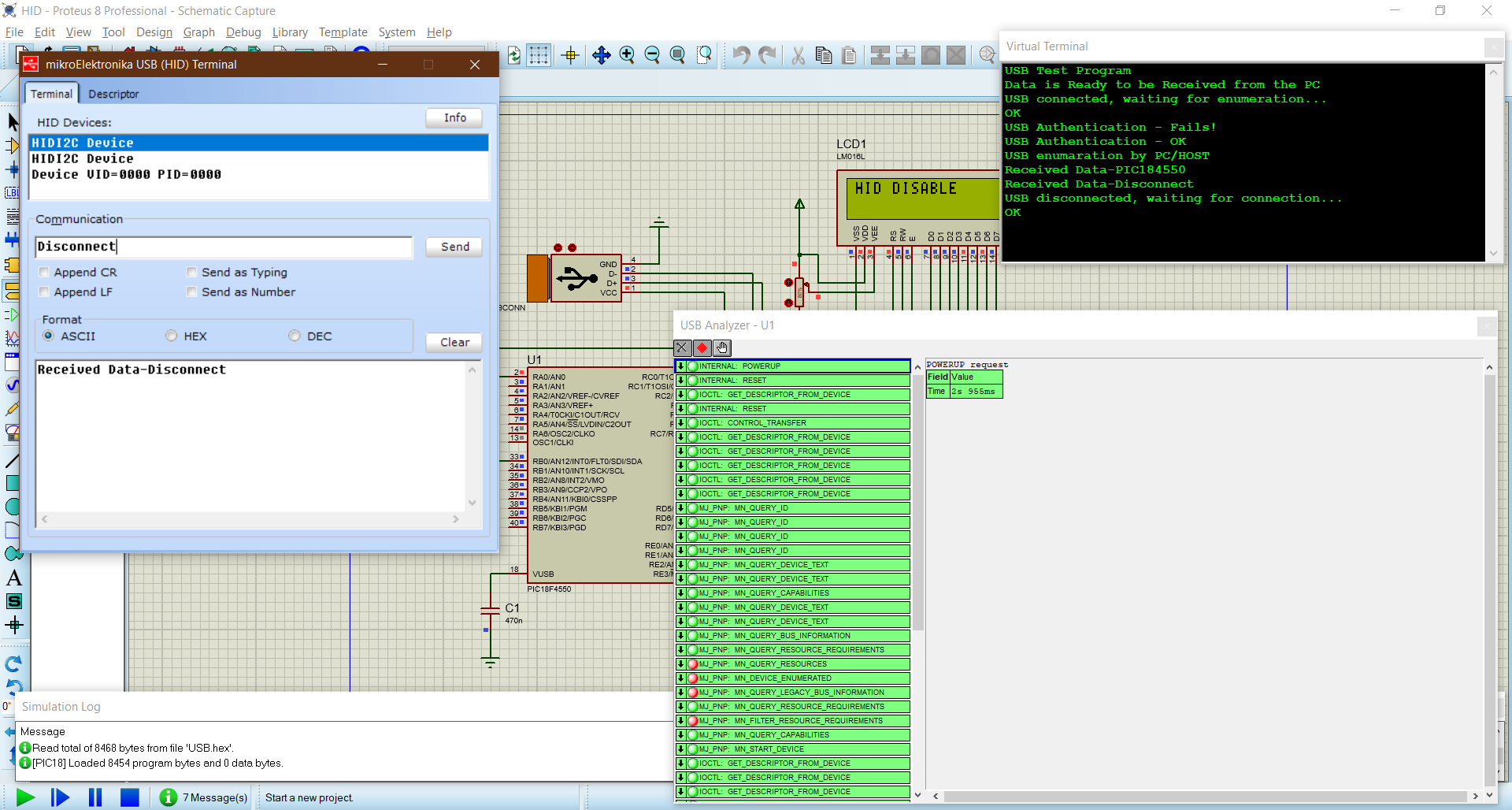
Теперь отправить правильный код, спрятанный от терминала "S".



Теперь мы можем проверить, все, что мы отправим с HID-терминала, будет отображаться на ЖК-дисплее, также и в терминале. Также мы можем увидеть обратный ответ в HID-терминал mikroC от USB (будут присылаться тоже данные что и отправились).



Так же можно отправить Disconnect и USB отключиться.

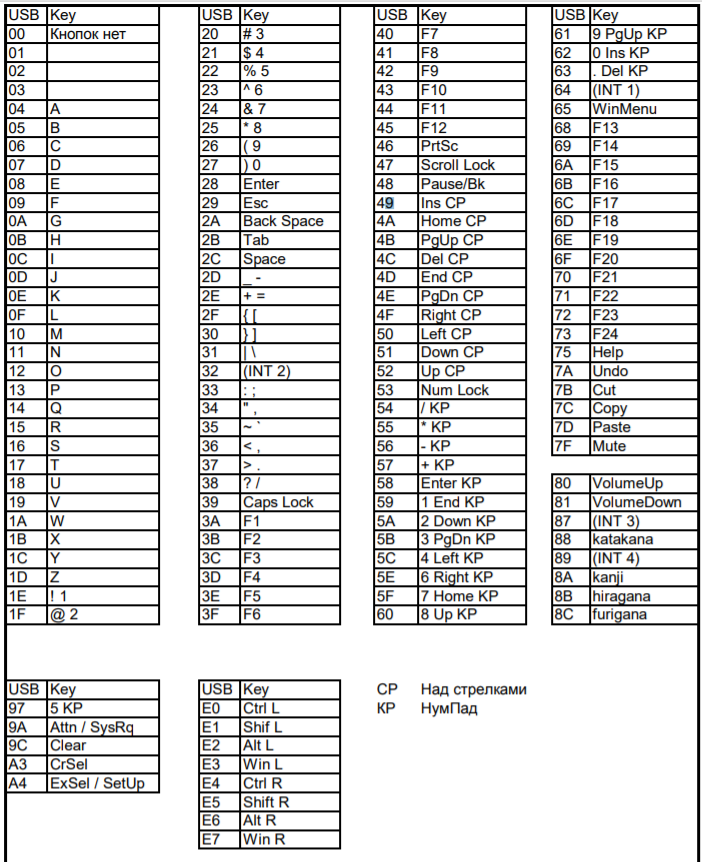


## Дополнения и улучшения

Код, представленный выше, также необходимо модифицировать для возможности передачи большего количества данных с помощью увеличенного буфера памяти.

Передача данных от устройства USB к хосту. По сути вся работа заключается в заполнении соответствующего буфера и в отравке его в ПК. Не сложнее, чем с UART. Вся работа выполняется в подпрограмме прерывания. В комплекте с IDE уже идут готовые библиотеки работы с HID.

Тут следует пояснить, что scan коды клавиатуры отличаются от ASCII.



Также приведенный выше код можно усовершенствовать и подключить HID устройство, например, мышь.

# Заключение

В данной работе было реализовано USB-устройство с помощью систем проектирования «Proteus» и «MikroC».

В качестве микроконтроллера использовался PIC микроконтроллер PIC18F4550, а программа реализовывалась на языке C. В процессе программирования была активно использована библиотека V-USB, а также программирование портов и пинов микроконтроллера, работа с буфером, прерывания и сторожевой таймер.

В данной работе спроектировано и запрограммировано USB-устройство, тем самым были приобретены теоретические и практические навыки работы с PIC микроконтроллерами, логическими схемами и программированием в среде «MikroC», отправка и чтение данных хоста и usb.

# Список литературы.

1. USB Component [https://www.keil.com/pack/Doc/MW/USB/html/\_u\_s\_b\_\_ device\_\_descriptor.html](https://www.keil.com/pack/Doc/MW/USB/html/_u_s_b__%20device__descriptor.html) (Дата обращения 28.03.20)
2. Белов А.В. Конструирование устройств на микроконтроллерах. – СПб.: Наука и техника, 2005. — 256 с.
3. Virtual Serial Port Driver <https://www.eltima.com/ru/products/vspdxp/> (Дата обращения 28.03.20)
4. USB Library <https://download.mikroe.com/documents/compilers/mikroc/pic/help/usb_hid_library.htm> (Дата обращения 28.03.20)
5. PIC18F4550 <http://kazus.ru/datasheets/pdf-data/3692418/MICROCHIP/PIC18F4550.html> (Дата обращения 28.03.20)
6. Proteus и два с половиной hello world для UART и USB на микроконтроллере https://habr.com/ru/post/206034/ (Дата обращения 28.03.20)
7. Цифровые интегральные микросхемы М.И. Богданович 1996.
8. Микропроцессоры и машинное проектирование микропроцессорных систем М. Рафикумазан 1988.
9. Стрыкин В.В, Щарев Л.С. Основы вычислительной, микропроцессорной техники и программирования: Учебник для учащихся техникумов. – 2-е изд, перераб. и доп. – М.: Высшая школа, 1989.
10. Использование САПР PROTEUS https://articlekz.com/article/22138, свободный – (09.03.2020)

# Приложение. Листинг программы.

USB.c

unsigned char Read\_Buffer[64] absolute 0x500;

unsigned char Write\_Buffer[64]absolute 0x510;

unsigned char num,flag;

void interrupt()

{

USB\_Interrupt\_Proc();

TMR0L = 100; //Reload Value

INTCON.TMR0IF = 0; //Re-Enable Timer-0 Interrupt

}

void clrUSB()

{

memset(Write\_Buffer, 0, sizeof Write\_Buffer);

while(!HID\_Write(&Write\_Buffer, sizeof Write\_Buffer));

}

//LCD 8-bit Mode Connection

sbit LCD\_RS at RC0\_bit;

sbit LCD\_RW at RC1\_bit;

sbit LCD\_EN at RC2\_bit;

sbit LCD\_D7 at RD7\_bit;

sbit LCD\_D6 at RD6\_bit;

sbit LCD\_D5 at RD5\_bit;

sbit LCD\_D4 at RD4\_bit;

sbit LCD\_D3 at RD3\_bit;

sbit LCD\_RS\_Direction at TRISC0\_bit;

sbit LCD\_RW\_Direction at TRISC1\_bit;

sbit LCD\_EN\_Direction at TRISC2\_bit;

sbit LCD\_D7\_Direction at TRISD7\_bit;

sbit LCD\_D6\_Direction at TRISD6\_bit;

sbit LCD\_D5\_Direction at TRISD5\_bit;

sbit LCD\_D4\_Direction at TRISD4\_bit;

sbit LCD\_D3\_Direction at TRISD3\_bit;

// End Lcd module connections

char i; // Loop variable

void UART1\_Write\_Text\_Newline(unsigned char msg[])

{

UART1\_Write\_Text(msg);

UART1\_Write(10);

UART1\_Write(13);

}

void clear\_buffer(unsigned char buffer[])

{

unsigned int i = 0;

while(buffer[i] != '\0')

{

buffer[i] = '\0';

i++;

}

}

void main()

{

UART1\_Init(9600);

Delay\_ms(100);

UART1\_Write\_Text\_Newline("USB Test Program");

ADCON1 |= 0x0F; // Configure AN pins as digital

CMCON |= 7; // Disable comparators

TRISA = 0x00;

TRISB = 0x00;

TRISC = 0x80;

PORTB.B0 = 0;

Lcd\_Init(); // Initialize Lcd8

Delay\_ms(100);

Lcd\_Cmd(\_LCD\_CLEAR); // Clear display

Delay\_ms(100);

Lcd\_Cmd(\_LCD\_CURSOR\_OFF); // Cursor off

Delay\_ms(100);

Lcd\_Out(1,3,"PIC18F4550"); // Write text in first row

Delay\_ms(100);

Lcd\_Out(2,3,"USB Example!"); // Write text in second row

Delay\_ms(2000);

INTCON = 0;

INTCON2 = 0xF5;

INTCON3 = 0xC0;

RCON.IPEN = 0;

PIE1 = 0;

PIE2 = 0;

PIR1 = 0;

PIR2 = 0;

//

// Configure TIMER 0 for 3.3ms interrupts. Set prescaler to 256

// and load TMR0L to 100 so that the time interval for timer

// interrupts at 48MHz is 256.(256-100).0.083 = 3.3ms

//

// The timer is in 8-bit mode by default

T0CON = 0x47; // Prescaler = 256

TMR0L = 100; // Timer count is 256-156 = 100

INTCON.TMR0IE = 1; // Enable T0IE

T0CON.TMR0ON = 1; // Turn Timer 0 ON

INTCON = 0xE0; // Enable interrupts

//

// Enable USB port

//

if(PORTA.B0 == 1)

{

UART1\_Write\_Text\_Newline("Data is Ready to be Received from the PC");

Hid\_Enable(&Read\_Buffer,&Write\_Buffer);

UART1\_Write\_Text\_Newline("USB connected, waiting for enumeration...");

Delay\_ms(2000);

UART1\_Write\_Text\_Newline("OK");

PORTB.B0 = 1;

}

// Read from the USB port. Number of bytes read is in num

start:

while(Hid\_Read() == 0); //Stay Here if Data is Not Coming from Serial Port

//If Some Data is Coming then move forward and check whether the keyword start is coming or not

if(strncmp(Read\_Buffer,"S",1) == 0)

{

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Out(1,2,"Authentication");

Lcd\_Out(2,8,"OK");

UART1\_Write\_Text\_Newline("USB Authentication - OK");

UART1\_Write\_Text\_Newline("USB enumaration by PC/HOST");

goto loop;

}

else

{

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Out(1,2,"Authentication");

Lcd\_Out(2,5,"Fails!");

UART1\_Write\_Text\_Newline("USB Authentication - Fails!");

goto start;

}

loop:

//Now Authentication is Successfull Lets Try Something else

//Lets Display the Data Coming from the USB HID Port to the LCD

Delay\_ms(1000);

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Out(1,1,"Received Data-");

flag = 0;

loop\_second:

clear\_buffer(Read\_Buffer);

while(Hid\_Read() == 0)

{

if(flag == 0)

{

Lcd\_Out(2,1,"No Data");

flag = 1;

}

}

Lcd\_Cmd(\_LCD\_CLEAR);

Lcd\_Out(1,1,"Received Data-");

Lcd\_Out(2,1,Read\_Buffer);

strcpy(Write\_Buffer, "Received Data-");

strcat(Write\_Buffer, Read\_Buffer);

while(!HID\_Write(&Write\_Buffer, 64)) ;

UART1\_Write\_Text\_Newline(Write\_Buffer);

clrUSB();

if(strncmp(Read\_Buffer,"Disable",1) == 0)

{

Lcd\_Cmd(\_LCD\_CLEAR);

goto end;

}

goto loop\_second;

end:

UART1\_Write\_Text\_Newline("USB disconnected, waiting for connection...");

Delay\_ms(1000);

Hid\_Disable();

Lcd\_Out(1,1,"HID DISABLE");

UART1\_Write\_Text\_Newline("OK");

PORTB.B0 = 0;

}

USBdsc.c

const unsigned int USB\_VENDOR\_ID = 0x1234;

const unsigned int USB\_PRODUCT\_ID = 0x0001;

const char USB\_SELF\_POWER = 0x80; // Self powered 0xC0, 0x80 bus powered

const char USB\_MAX\_POWER = 50; // Bus power required in units of 2 mA

const char HID\_INPUT\_REPORT\_BYTES = 64;

const char HID\_OUTPUT\_REPORT\_BYTES = 64;

const char USB\_TRANSFER\_TYPE = 0x03; //0x03 Interrupt

const char EP\_IN\_INTERVAL = 1;

const char EP\_OUT\_INTERVAL = 1;

const char USB\_INTERRUPT = 1;

const char USB\_HID\_EP = 1;

const char USB\_HID\_RPT\_SIZE = 33;

/\* Device Descriptor \*/

const struct {

char bLength; // bLength - Descriptor size in bytes (12h)

char bDescriptorType; // bDescriptorType - The constant DEVICE (01h)

unsigned int bcdUSB; // bcdUSB - USB specification release number (BCD)

char bDeviceClass; // bDeviceClass - Class Code

char bDeviceSubClass; // bDeviceSubClass - Subclass code

char bDeviceProtocol; // bDeviceProtocol - Protocol code

char bMaxPacketSize0; // bMaxPacketSize0 - Maximum packet size for endpoint 0

unsigned int idVendor; // idVendor - Vendor ID

unsigned int idProduct; // idProduct - Product ID

unsigned int bcdDevice; // bcdDevice - Device release number (BCD)

char iManufacturer; // iManufacturer - Index of string descriptor for the manufacturer

char iProduct; // iProduct - Index of string descriptor for the product.

char iSerialNumber; // iSerialNumber - Index of string descriptor for the serial number.

char bNumConfigurations; // bNumConfigurations - Number of possible configurations

} device\_dsc = {

0x12, // bLength

0x01, // bDescriptorType

0x0200, // bcdUSB

0x00, // bDeviceClass

0x00, // bDeviceSubClass

0x00, // bDeviceProtocol

8, // bMaxPacketSize0

USB\_VENDOR\_ID, // idVendor

USB\_PRODUCT\_ID, // idProduct

0x0001, // bcdDevice

0x01, // iManufacturer

0x02, // iProduct

0x00, // iSerialNumber

0x01 // bNumConfigurations

};

/\* Configuration 1 Descriptor \*/

const char configDescriptor1[]= {

// Configuration Descriptor

0x09, // bLength - Descriptor size in bytes

0x02, // bDescriptorType - The constant CONFIGURATION (02h)

0x29,0x00, // wTotalLength - The number of bytes in the configuration descriptor and all of its subordinate descriptors

1, // bNumInterfaces - Number of interfaces in the configuration

1, // bConfigurationValue - Identifier for Set Configuration and Get Configuration requests

0, // iConfiguration - Index of string descriptor for the configuration

USB\_SELF\_POWER, // bmAttributes - Self/bus power and remote wakeup settings

USB\_MAX\_POWER, // bMaxPower - Bus power required in units of 2 mA

// Interface Descriptor

0x09, // bLength - Descriptor size in bytes (09h)

0x04, // bDescriptorType - The constant Interface (04h)

0, // bInterfaceNumber - Number identifying this interface

0, // bAlternateSetting - A number that identifies a descriptor with alternate settings for this bInterfaceNumber.

2, // bNumEndpoint - Number of endpoints supported not counting endpoint zero

0x03, // bInterfaceClass - Class code

0, // bInterfaceSubclass - Subclass code

0, // bInterfaceProtocol - Protocol code

0, // iInterface - Interface string index

// HID Class-Specific Descriptor

0x09, // bLength - Descriptor size in bytes.

0x21, // bDescriptorType - This descriptor's type: 21h to indicate the HID class.

0x01,0x01, // bcdHID - HID specification release number (BCD).

0x00, // bCountryCode - Numeric expression identifying the country for localized hardware (BCD) or 00h.

1, // bNumDescriptors - Number of subordinate report and physical descriptors.

0x22, // bDescriptorType - The type of a class-specific descriptor that follows

USB\_HID\_RPT\_SIZE,0x00, // wDescriptorLength - Total length of the descriptor identified above.

// Endpoint Descriptor

0x07, // bLength - Descriptor size in bytes (07h)

0x05, // bDescriptorType - The constant Endpoint (05h)

USB\_HID\_EP | 0x80, // bEndpointAddress - Endpoint number and direction

USB\_TRANSFER\_TYPE, // bmAttributes - Transfer type and supplementary information

0x40,0x00, // wMaxPacketSize - Maximum packet size supported

EP\_IN\_INTERVAL, // bInterval - Service interval or NAK rate

// Endpoint Descriptor

0x07, // bLength - Descriptor size in bytes (07h)

0x05, // bDescriptorType - The constant Endpoint (05h)

USB\_HID\_EP, // bEndpointAddress - Endpoint number and direction

USB\_TRANSFER\_TYPE, // bmAttributes - Transfer type and supplementary information

0x40,0x00, // wMaxPacketSize - Maximum packet size supported

EP\_OUT\_INTERVAL // bInterval - Service interval or NAK rate

};

const struct {

char report[USB\_HID\_RPT\_SIZE];

}hid\_rpt\_desc =

{

{0x06, 0x00, 0xFF, // Usage Page = 0xFF00 (Vendor Defined Page 1)

0x09, 0x01, // Usage (Vendor Usage 1)

0xA1, 0x01, // Collection (Application)

// Input report

0x19, 0x01, // Usage Minimum

0x29, 0x40, // Usage Maximum

0x15, 0x00, // Logical Minimum (data bytes in the report may have minimum value = 0x00)

0x26, 0xFF, 0x00, // Logical Maximum (data bytes in the report may have maximum value = 0x00FF = unsigned 255)

0x75, 0x08, // Report Size: 8-bit field size

0x95, HID\_INPUT\_REPORT\_BYTES,// Report Count

0x81, 0x02, // Input (Data, Array, Abs)

// Output report

0x19, 0x01, // Usage Minimum

0x29, 0x40, // Usage Maximum

0x75, 0x08, // Report Size: 8-bit field size

0x95, HID\_OUTPUT\_REPORT\_BYTES,// Report Count

0x91, 0x02, // Output (Data, Array, Abs)

0xC0} // End Collection

};

//Language code string descriptor

const struct {

char bLength;

char bDscType;

unsigned int string[1];

} strd1 = {

4,

0x03,

{0x0409}

};

//Manufacturer string descriptor

const struct{

char bLength;

char bDscType;

unsigned int string[16];

}strd2={

34, //sizeof this descriptor string

0x03,

{'M','i','k','r','o','e','l','e','k','t','r','o','n','i','k','a'}

};

//Product string descriptor

const struct{

char bLength;

char bDscType;

unsigned int string[8];

}strd3={

18, //sizeof this descriptor string

0x03,

{'U','S','B','\_','T','e','s','t'}

};

//Array of configuration descriptors

const char\* USB\_config\_dsc\_ptr[1];

//Array of string descriptors

const char\* USB\_string\_dsc\_ptr[3];

void USB\_Init\_Desc(){

USB\_config\_dsc\_ptr[0] = &configDescriptor1;

USB\_string\_dsc\_ptr[0] = (const char\*)&strd1;

USB\_string\_dsc\_ptr[1] = (const char\*)&strd2;

USB\_string\_dsc\_ptr[2] = (const char\*)&strd3;

}