Задача.

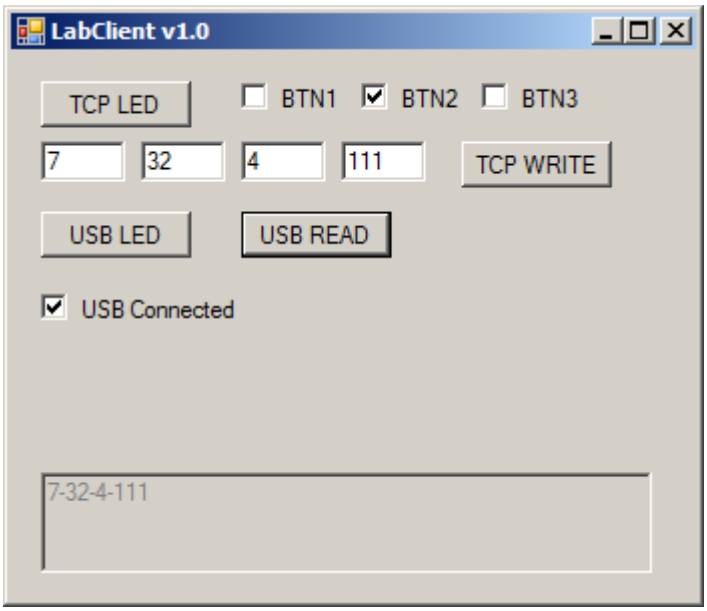
На данной лабораторной работе мы научимся:

- Создавать «с нуля» программу для платы микроконтроллера, которая позволяет работать одновременно с TCP и USB стеком
- Задавать логику приема и передачи данных по интерфейсам USB и Ethernet на обоих сторонах взаимодействия
- Создавать «с нуля» программу для компьютера, позволяющую передавать и принимать данные на плату микроконтроллера через интерфейсы USB и Ethernet
- Создавать пользовательский интерфейс, позволяющий вводить и выводить данные на компьютере

Мы определяем следующую задачу для нашего проекта:

Отладочная плата PIC32 Ethernet Starter Kit должна выступать в роли обработчика команд, поступающих как по интерфейсу USB, так и по интерфейсу Ethernet.

Команды выдаются с компьютера. При этом пользователь «общается» с платой через следующий пользовательский интерфейс:



При нажатии на кнопку “TCP LED” команда должна передаваться через Ethernet на плату. По этой команде должно изменяться состояние светодиода LED2 на плате. По этой же команде в пользовательском интерфейсе должно отображаться состояние кнопок, нажатых на плате в момент команды.

При нажатии на кнопку “USB LED” команда должна передаваться через USB на плату. По этой команде должно изменяться состояние светодиода LED3 на плате.

Четыре байта данных, вводимые в форме должны передаваться в плату через Ethernet и записываться в память платы при нажатии кнопки «TCP WRITE». Эти данные должны считываться из памяти платы через USB и выводится в окне сообщений при нажатии кнопки «USB READ»

В поле «USB Connected» должно отображаться состояние подключения платы к компьютеру по USB.

Мы будем создавать весь проект – как микроконтроллерную часть, так и ответную часть, выполняющуюся на компьютере.

Микроконтроллерная плата при подключении к компьютеру через Ethernet будет выполнять роль TCP сервера, а при подключении через USB интерфейс – будет являться HID устройством.

Соответственно, программа компьютера будет выполнять роль TCP клиента для Ethernet и Host-а для USB. А также обеспечивать пользовательский интерфейс для работы.

Инструменты.

Для решения задачи на компьютере должны быть установлены следующие инструменты:

- Microchip Application Library (www.microchip.com/mal)
- MPLAB IDE (www.microchip.com/mplab)
- MPLAB C Compiler for PIC32 MCUs (www.microchip.com/c32)
- Microsoft Visual C++ 2008 Express Edition (http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express)

Все инструменты являются свободно распространяемыми и могут быть скачаны с соответствующих сайтов

Работа компьютера будет происходить под управлением операционной системы Windows 7 с установленным пакетом .Net версии 4.0 (http://www.microsoft.com/net/download)

Шаг 1. Создаем TCP/IP сервер на плате PIC32 Ethernet Starter Kit

a) **Готовим необходимую структуру для проекта.**

   Проект будем создавать в папке C:\Masters\Lab\MyLab\Embedded.
   В этой папке создаем подпапку Configs для файлов заголовков, описывающих аппаратную конфигурацию.

   В папку C:\Masters\Lab\MyLab\Embedded из папки “C:\Microchip Solutions vxxxx-xx-xx\TCPIP\Demo App” копируем общие файлы заголовков конфигурации стека и аппаратной части :
   TCPIPConfig.h
   HardwareProfile.h

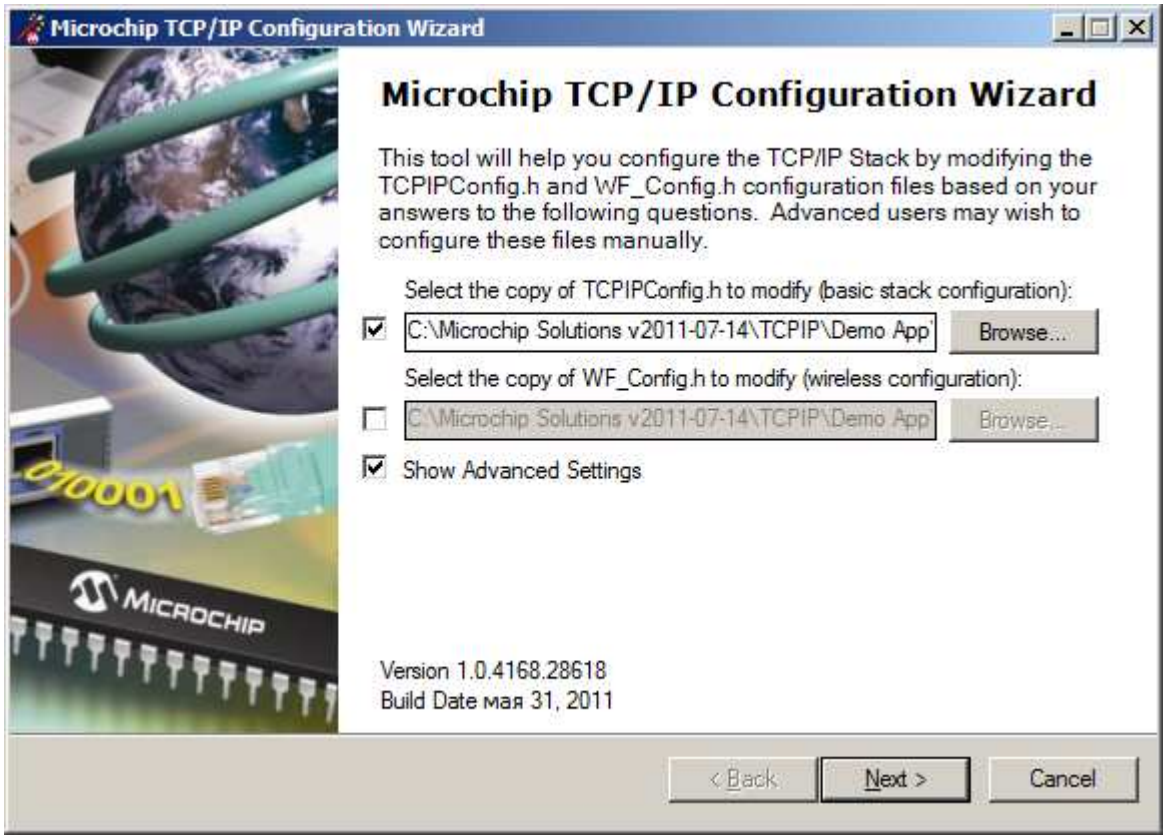
Из библиотеки “C:\Microchip Solutions vxxxx-xx-xx\TCPIP\Demo App\Configs” в созданную папку C:\Masters\Lab\MyLab\Embedded\Configs копируем файлы, описывающие конкретную конфигурацию стека и используемой платы. В нашем случае:
TCPIP ETH795.h
HWP PIC32_ETH_SK_ETH795.h.

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep1a.bat из папки C:\Masters\Lab\Solution\SolutionSteps)

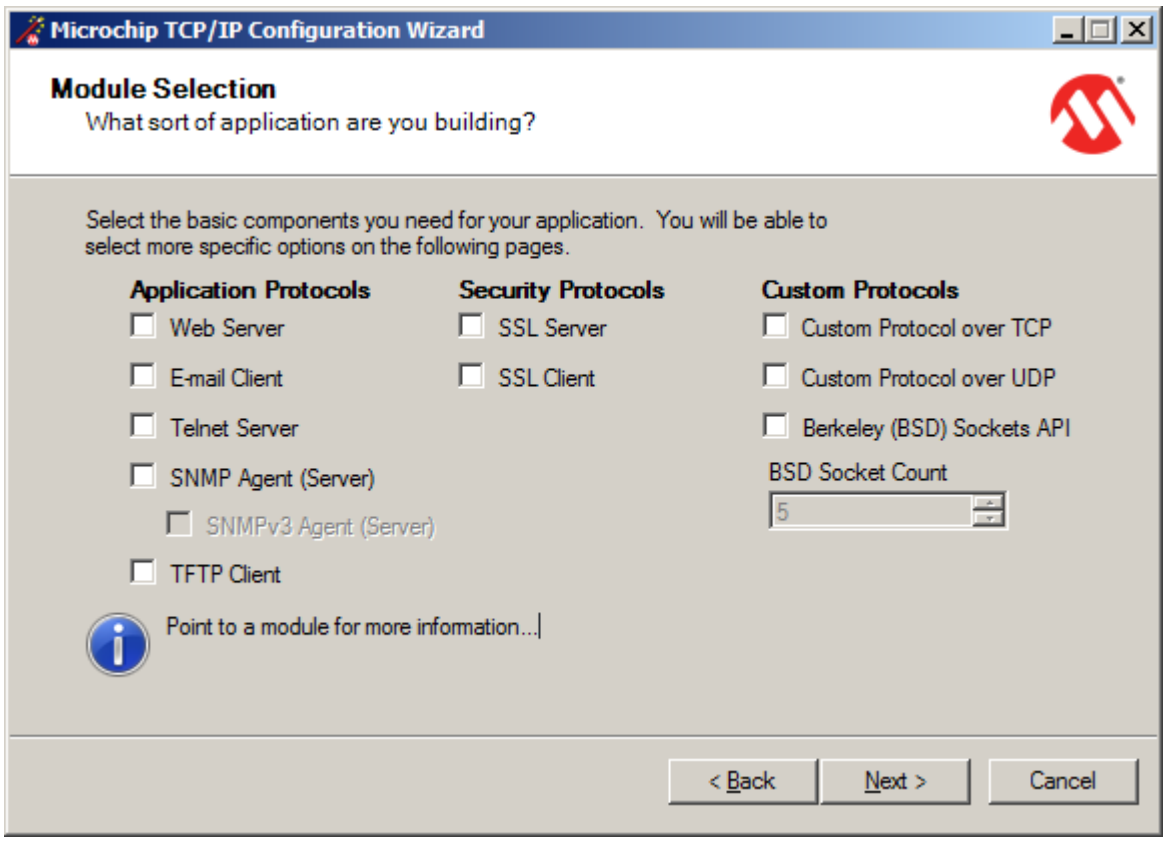
b) Задаем конфигурацию стека для проекта

Задавать конфигурацию стека для проекта можно вручную, исправляя требуемые параметры в файле конфигурации C:\Masters\Lab\Mylab\Embedded\Configs\TCPIP ETH795.h

То же самое, но в более наглядной форме, можно сделать с использованием утилиты TCPIPConfig.exe из библиотеки (C:\Microchip Solutions vxxxx-xx-xx\Microchip\TCPIP Stack\Utilites).

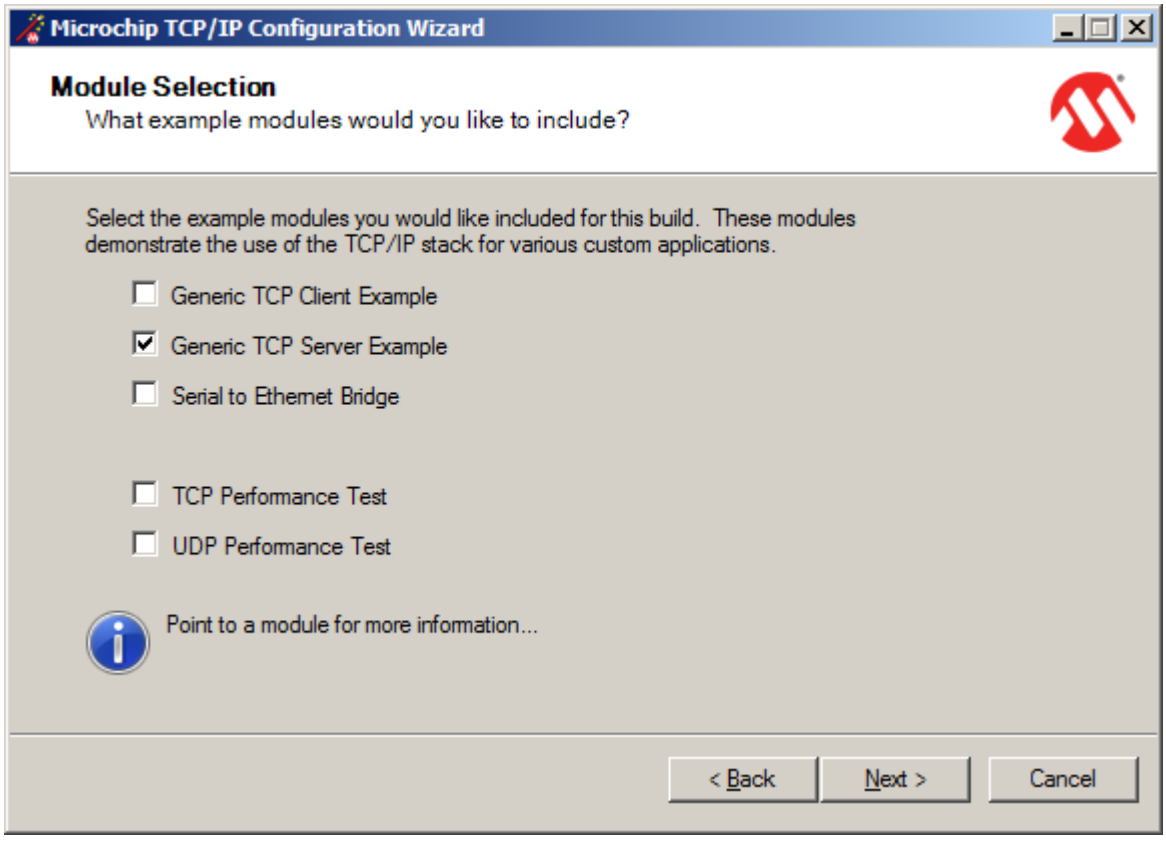


Нажав кнопку “Browse…” указываем на конфигурацию стека нашего проекта:
C:\Masters\Lab\Mylab\Embedded\Configs\TCPIP ETH795.h
Переходим к конфигурации, нажав “Next >”
Выключаем все неиспользуемые в нашем проекте протоколы.



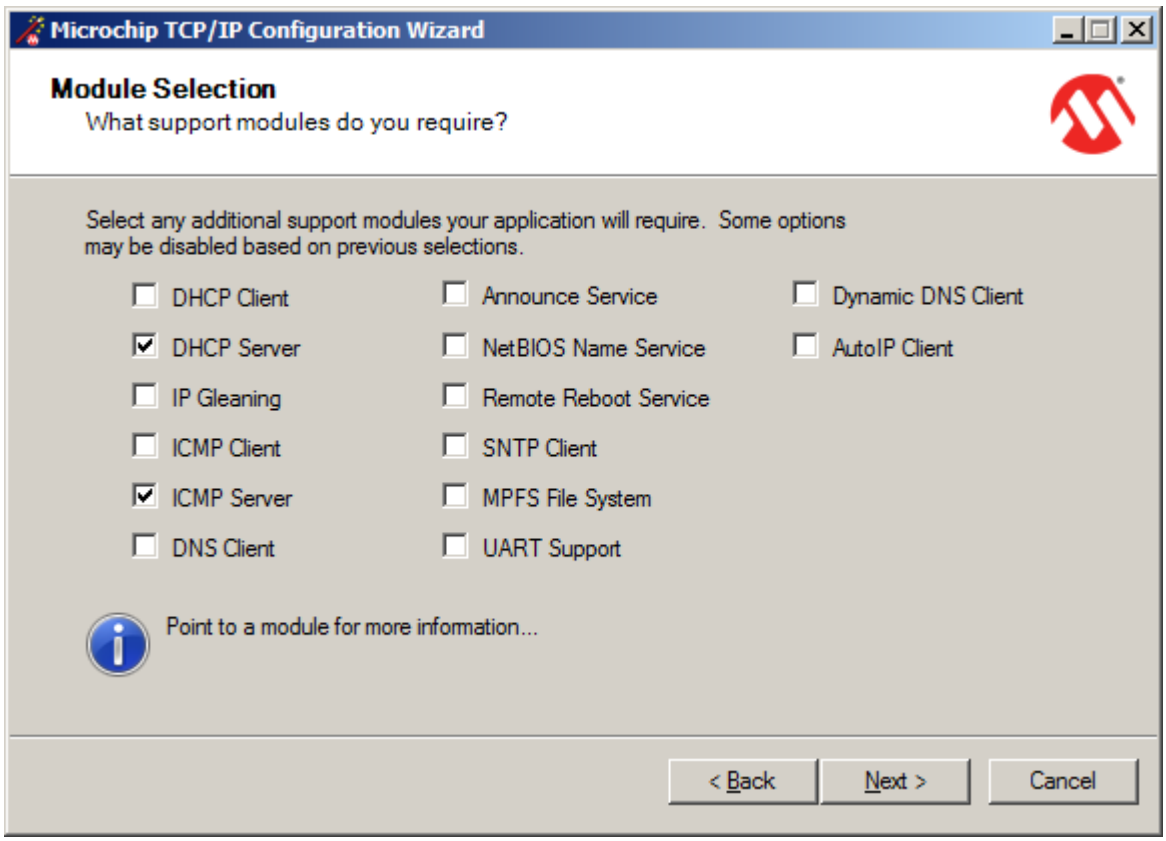
Переходим к следующей странице конфигурации, нажав “Next >”

Мы оставляем только один, необходимый нам сервер – Generic TCP Server Example.



Переходим к странице конфигурации вспомогательных протоколов, нажав “Next >”

Для работы нам понадобятся только DHCP Server и ICMP Server.



Переходим к конфигурации сетевых параметров, нажав “Next >”

В нашем проекте мы будем использовать сетевые параметры, заданные по умолчанию:

Host Name: MCHIPBOARD

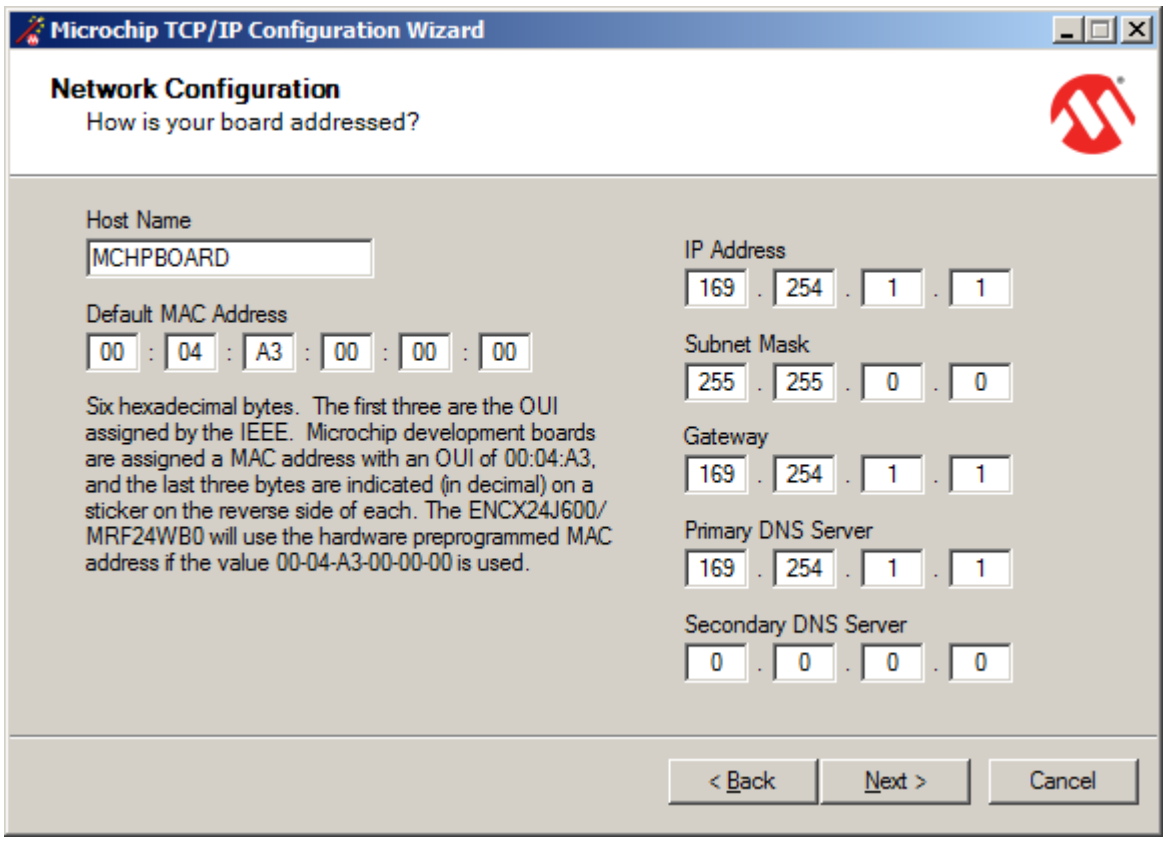
Default MAC Address: 00-04-A3-00-00-00

IP Address: 169.254.1.1

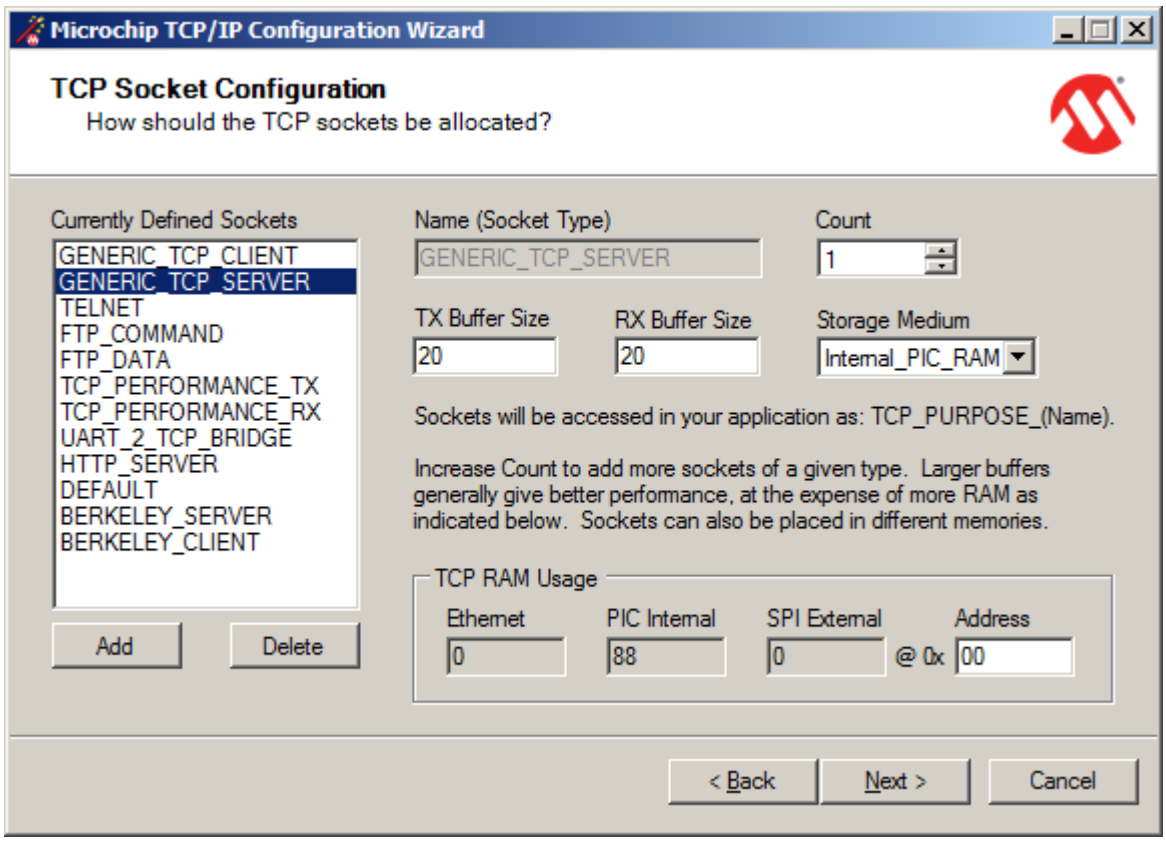
Subnet mask: 255.255.0.0

Gateway: 169.254.1.1

Primary DNS server: 169.254.1.1

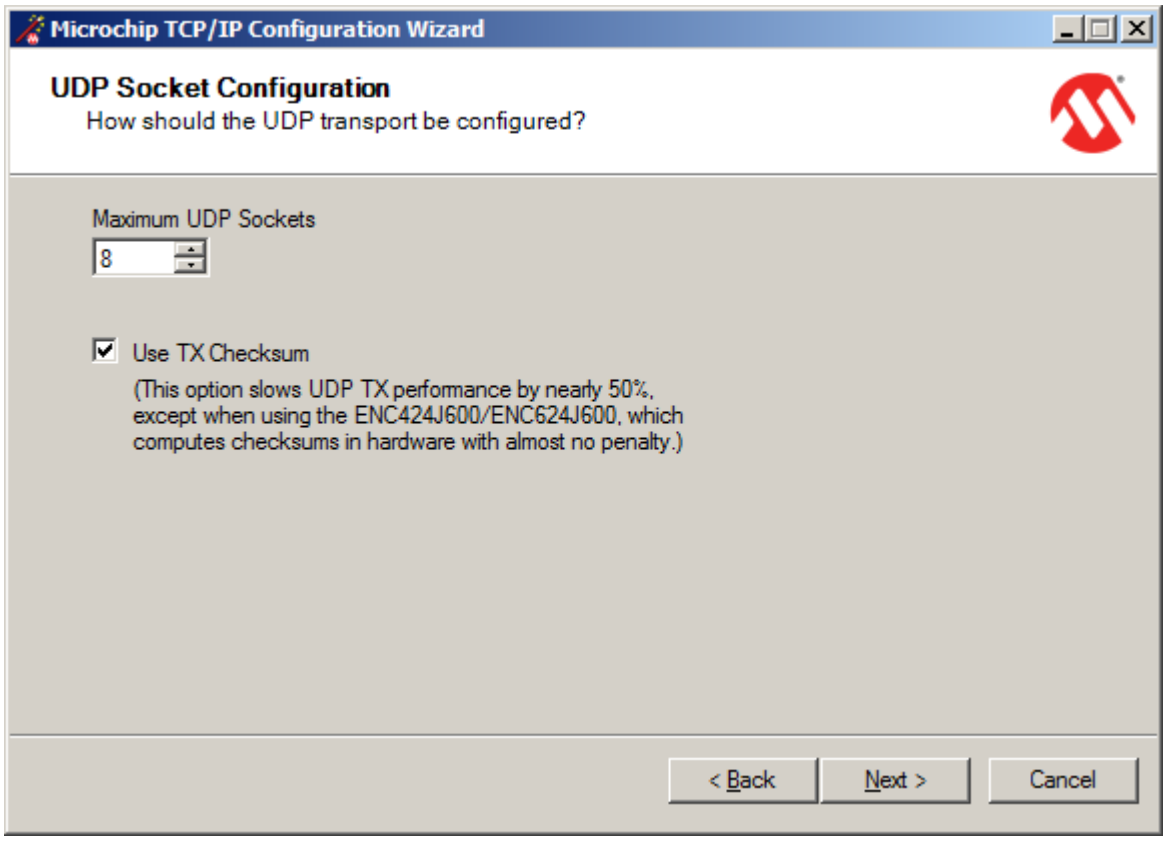


Переходим к конфигурации памяти TCP сокетов, нажав “Next >”
Т.к. в нашем проекте мы используем только один сервис – GENERIC_TCP_SERVER, и нам будет достаточно только одного сокета для этого сервиса, то для всех типов сокетов, перечисленных в таблице Currently Defined Sockets необходимо установить значение 0 в окошке Count.
Для GENERIC_TCP_SERVER в окошке Count устанавливаем 1.
Остальные параметры – по умолчанию.



Переходим к конфигурации памяти UDP сокетов, нажав “Next >”

Оставляем значения по умолчанию.



Завершаем конфигурацию, нажав “Next >” и ”Finish”

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep1b.bat из папки C:\Masters\Lab\Solution\SolutionSteps)

c) Создаем проект MPLAB

Запускаем MPLAB IDE.
Выполняем мастер создания проекта Project->Project Wizard
При прохождении мастера задаем:
- Микроконтроллер PIC32MX795F512L
- Компилятор Microchip PIC32 C-Compiler Toolsuite
- Через кнопку “Browse” выбираем папку C:\Masters\Lab\Mylab\Embedded и имя проекта Lab
Файлы в проект пока не добавляем – нажимаем «Далее» и «Готово»

Добавляем в проект файлы из библиотеки.
Для этого выбираем из меню Project->Add Files to Project
Заходим в папку “C:\Microchip Solutions vxxxx-xx-xx\Microchip\TCPIP Stack” и выбираем следующие файлы для добавления:

ARP.c
DHCPs.c
ETHPIC32ExtPhy.c
ETHPIC32ExtPhyDP83848.c
ETHPIC32IntMac.c
Helpers.c
ICMP.c
IP.c
StackTsk.c
TCP.c
Tick.c
UDP.c

Для правильной компиляции стека необходимо установить некоторые параметры компилятора и линкера. Для этого выбираем пункт меню Project->Build options … ->Project.

На вкладке «Directories» из списка «Show directories for» необходимо выбрать «Include Search Path».
Нажимая кнопку «NEW» необходимо добавить в список следующие паки:
C:\Masters\Lab\Mylab\Embedded
C:\Microchip Solutions vxxxx-xx-xx\Microchip\Include

На вкладке «MPLAB PIC32 C Compiler» необходимо задать «Preprocessor Macros», определяющий аппаратную платформу. Для этого нажимаем “Add” и добавляем следующий макрос:
CFG_INCLUDE_PIC32_ETH_SK_ETH795

На вкладке «MPLAB PIC32 Linker» необходимо установить следующие параметры:
Heap size: 16000 bytes

Min stack size: 2048 bytes

Сохранить полученный проект File->Save Workspace

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep1c.bat из папки C:\Masters\Lab\Solution\SolutionSteps. MPLAB IDE при выполнении должен быть закрыт)

d) Создаем основную программу проекта

Создаем в MPLAB новый файл File->New

Начинаем создавать программу.

Для правильной работы стека определяем биты конфигурации:

```
#pragma config FPLLODIV = DIV_1,FPLLMUL = MUL_20,FPLLIDIV = DIV_2
#pragma config FWDTEN = OFF,FPBDIV = DIV_1,POSCMOD = XT,FNOSC = PRIPLL,CP = OFF
#pragma config FMIIEN = OFF,FETHIO = OFF
#pragma config UPLLEN = ON,UPLLIDIV = DIV_2
```

Включаем файлы заголовков

```
#include "TCPIP Stack/TCPIP.h"
```

Определяем глобальные переменные, необходимые для работы:

```
APP_CONFIG AppConfig; // Переменная конфигурации TCP/IP стека
```

Определяем константы, хранящиеся в памяти программ:

```
ROM BYTE SerializedMACAddress[6] =
      {MY_DEFAULT_MAC_BYTE1,MY_DEFAULT_MAC_BYTE2,MY_DEFAULT_MAC_BYTE3,
      MY_DEFAULT_MAC_BYTE4,MY_DEFAULT_MAC_BYTE5,MY_DEFAULT_MAC_BYTE6};
```

Задаем прототипы функций

```
void InitAppConfig(void);  // Функция инициализации приложения
void InitializeBoard(void); // Функция инициализации платы
```

Создаем функцию main

```
int main(void)
{
      InitializeBoard(); // Инициализируем плату
      TickInit();        // Инициализируем таймер
      InitAppConfig();   // Инициализируем приложение
     StackInit();       // Инициализируем стек

      while(1)                        // бесконечный цикл
      {
            StackTask();              // выполняем задачи стека
            StackApplications();      // выполняем задачи приложения
      }
}
```

Создаем функцию инициализации платы

```
void InitializeBoard(void)
{
      // конфигурация LEDs
      LED0_TRIS = 0;
      LED1_TRIS = 0;
```

```
        LED2_TRIS = 0;
        LED_PUT(0x00);

        //Разрешение прерываний
        INTEnableSystemMultiVectoredInt();

        // Настройка оптимального быстродействия
        SYSTEMConfigPerformance(GetSystemClock());
        mOSCSetPBDIV(OSC_PB_DIV_1);
        CNPUESET = 0x00098000; //Включение pullup резисторов на входах кнопок
}
```

Создаем функцию инициализации приложения – заполняем поля переменной AppConfig. В этой функции мы присваиваем полям структуры AppConfig данные, которые мы сформировали при конфигурации стека (Шаг 1 b).

```
void InitAppConfig(void)
{
        memset((void*)&AppConfig, 0x00, sizeof(AppConfig));
        AppConfig.Flags.bIsDHCPEnabled = TRUE;
        AppConfig.Flags.bInConfigMode = TRUE;
        memcpypgm2ram((void*)&AppConfig.MyMACAddr,
                (ROM void*)SerializedMACAddress, sizeof(AppConfig.MyMACAddr));
        AppConfig.MyIPAddr.Val = MY_DEFAULT_IP_ADDR_BYTE1 |
                MY_DEFAULT_IP_ADDR_BYTE2<<8ul | MY_DEFAULT_IP_ADDR_BYTE3<<16ul |
                MY_DEFAULT_IP_ADDR_BYTE4<<24ul;
        AppConfig.DefaultIPAddr.Val = AppConfig.MyIPAddr.Val;
        AppConfig.MyMask.Val = MY_DEFAULT_MASK_BYTE1 | MY_DEFAULT_MASK_BYTE2<<8ul
                | MY_DEFAULT_MASK_BYTE3<<16ul | MY_DEFAULT_MASK_BYTE4<<24ul;
        AppConfig.DefaultMask.Val = AppConfig.MyMask.Val;
        AppConfig.MyGateway.Val = MY_DEFAULT_GATE_BYTE1 |
                MY_DEFAULT_GATE_BYTE2<<8ul | MY_DEFAULT_GATE_BYTE3<<16ul |
                MY_DEFAULT_GATE_BYTE4<<24ul;
        AppConfig.PrimaryDNSServer.Val = MY_DEFAULT_PRIMARY_DNS_BYTE1 |
                MY_DEFAULT_PRIMARY_DNS_BYTE2<<8ul  |
                MY_DEFAULT_PRIMARY_DNS_BYTE3<<16ul  |
                MY_DEFAULT_PRIMARY_DNS_BYTE4<<24ul;
        AppConfig.SecondaryDNSServer.Val = MY_DEFAULT_SECONDARY_DNS_BYTE1 |
                MY_DEFAULT_SECONDARY_DNS_BYTE2<<8ul  |
                MY_DEFAULT_SECONDARY_DNS_BYTE3<<16ul  |
                MY_DEFAULT_SECONDARY_DNS_BYTE4<<24ul;
}
```

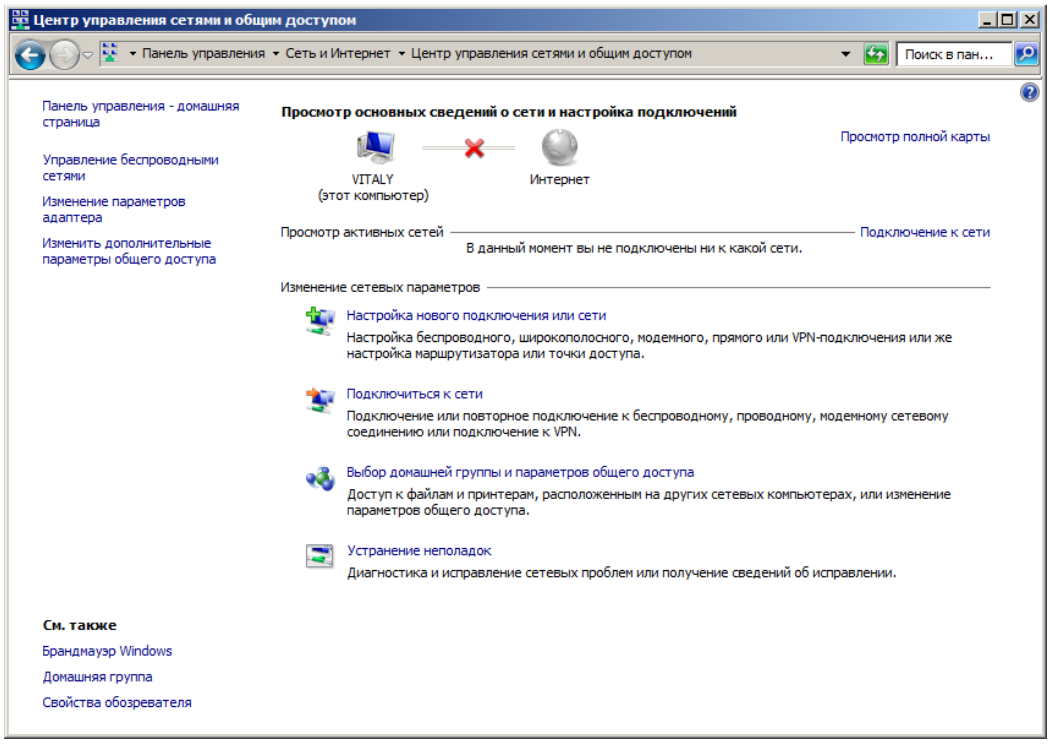
Сохраняем полученный файл под именем main.c в папке C:\Masters\Lab\Mylab\Embedded командой File->Save.
И добавляем этот файл к проекту командой Project->Add Files to Project.
Сохраняем проект File->Save All, File->Save Workspace

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep1d.bat из папки C:\Masters\Lab\Solution\SolutionSteps. MPLAB IDE при выполнении должен быть закрыт)

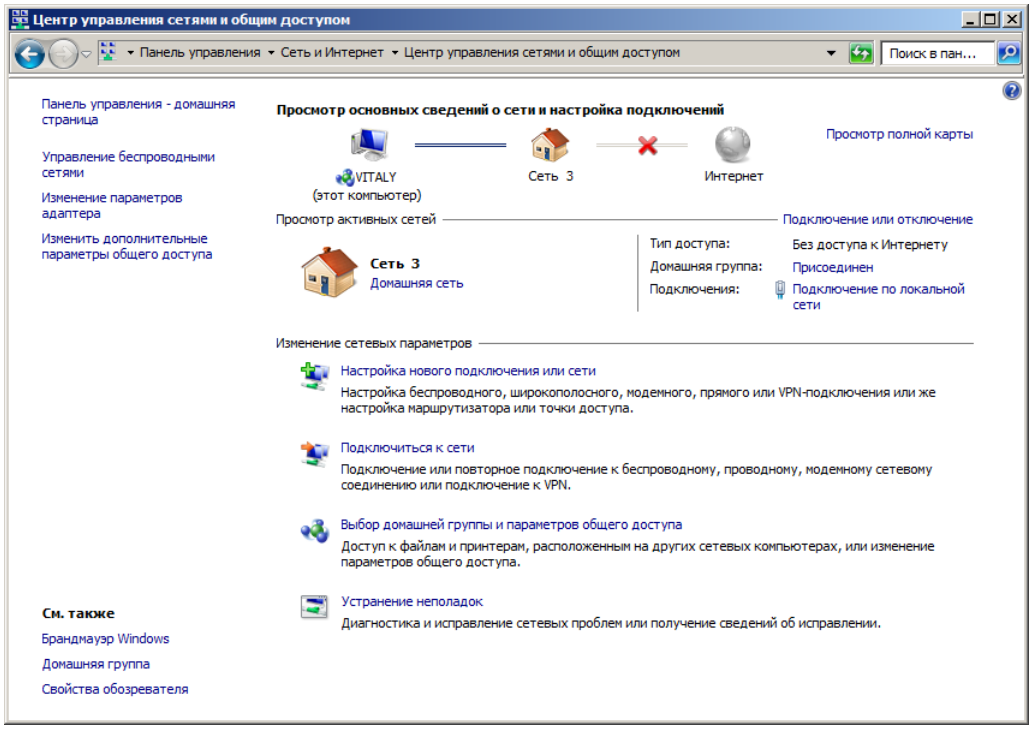
e) Проверка подключения к сети

Подключаем отладочный разъем платы PIC32 Ethernet Starter Kit через USB кабель к компьютеру.

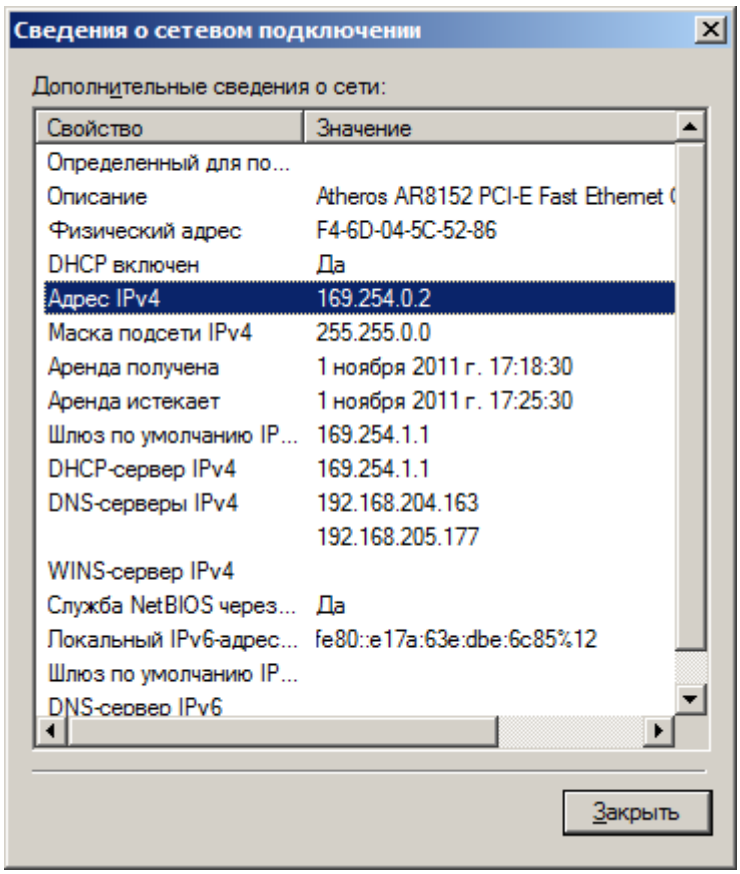
Включаем плату как отладчик (Debugger->Select Tool->PIC32 Starter Kit)
Компилируем проект (Project->Make)
Загружаем программу (Debugger->Programming->Program All Memories). Подтвердить программирование.
Запускаем выполнение программы (Debugger -> Run)
Чтобы увидеть подключение платы к компьютеру необходимо открыть на компьютере окно «Центр управления сетями и общим доступом»
(Пуск->Панель управления->Просмотр состояния сети и задач)



Видно, что исходно компьютер не подключен к сети.
Соединяем плату и компьютер Ethernet кабелем
Компьютер должен увидеть новую TCP/IP сеть, и после инициализации и получения IP адреса от DHCP сервера подключенной платы, показать, что он подключен к этой сети.

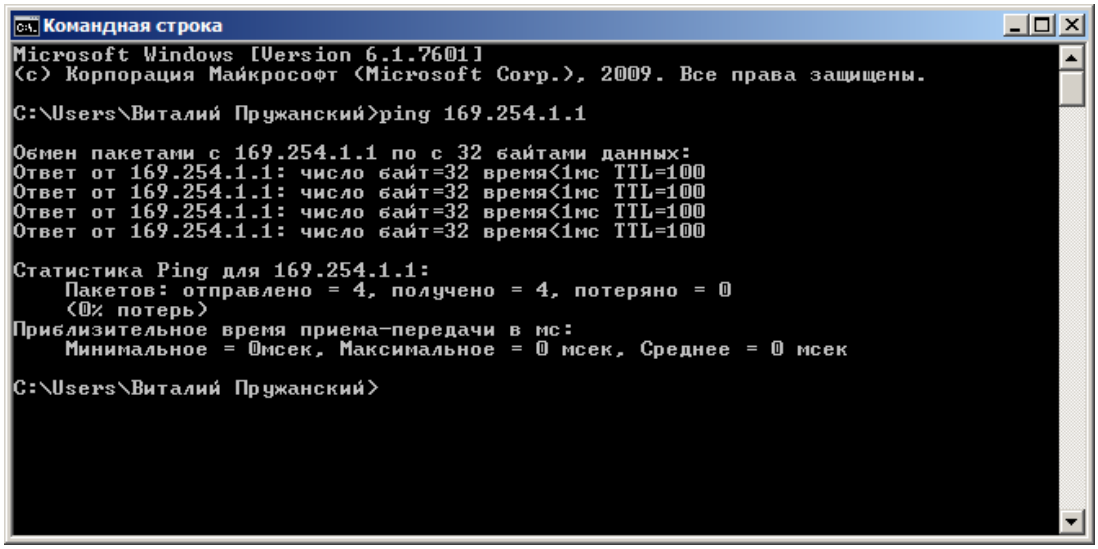


Параметры сети можно посмотреть, кликнув на ссылку «Подключение по локальной сети», и в открывшемся окне нажав кнопку «Сведения…»



Можно видеть, что подключенная плата с IP адресом 169.254.1.1 установлена в качестве шлюза для работы компьютера. И DHCP сервер платы присвоил компьютеру IP адрес 169.254.0.2

Проверяем отклик платы. Для этого запускаем командную строку (Пуск->Все программы->Стандартные->Командная строка) . В открывшемся окне выполняем команду:
ping 169.254.1.1
Видим, что плата откликается.



Сеть работает.

f) Создаем TCP/IP сервер

Открываем файл main.c в окне редактора MPLAB.
В функции main, в теле бесконечного цикла добавляем функцию обработки TCP сервера.
После добавления функция main будет выглядеть следующим образом:

```
int main(void)
{
	InitializeBoard(); // Инициализируем плату
	TickInit();        // Инициализируем таймер
	InitAppConfig();   // Инициализируем приложение
	StackInit();       // Инициализируем стек

	while(1)                        // бесконечный цикл
	{
		StackTask();            // выполняем задачи стека
		StackApplications();    // выполняем задачи приложения
		ServerTask();           // выполняем задачу сервера
	}
}
```

Добавляем прототип функции ServerTask в список прототипов функций:

```
void InitAppConfig(void);   // Функция инициализации приложения
void InitializeBoard(void); // Функция инициализации платы
void ServerTask(void);      // Серверная задача
```

Добавляем глобальную переменную общего буфера для нашей задачи:

```
APP_CONFIG AppConfig; // Переменная конфигурации TCP/IP стека
BYTE CommonBuf[4];    // Общий буфер задачи
```

Определяем макрос, обозначающий порт подключения

```
#define  SERVER_PORT      9760
```

Создаем задачу сервера.

```
void ServerTask(void)
{

  // Статические перменные
  static TCP_SOCKET MySocket; // Сокет соединения
  static BOOL isListening = FALSE; // Состояние сокета (TRUE - открыт)
  // Локальные перменные
  WORD len;  // Длина сообщений
  BYTE AppBuffer[32]; // Временный буфер для сообщений

  // Машина состояний
  if ( !isListening ) {

    // Если сокет еще не открыт - открываем его
    MySocket = TCPOpen(0, TCP_OPEN_SERVER,
        SERVER_PORT,TCP_PURPOSE_GENERIC_TCP_SERVER);
    if(MySocket == INVALID_SOCKET) return; // Выход при неудаче
    isListening = TRUE; //Если сокет открыт-переходим в состояние прослушивания
  } else {

    // Обработка входящих сообщений
    if(!TCPIsConnected(MySocket)) return;

    //Только для установленного соединения
    len = TCPIsGetReady(MySocket); // Провряем наличие сообщения и его длину
    if( len > 0 ) {

      // если данные получены
      TCPGetArray(MySocket, AppBuffer, ( len > sizeof(AppBuffer)) ?
          sizeof(AppBuffer) : len ); //считываем их в AppBuffer

      // Обрабатываем данные
      if (AppBuffer[0] == 'a') {
        // По команде 'a' изменяем состояние светодиода
        // и возвращаем состояние кнопок
        LED1_IO = (LED1_IO) ?   0 : 1;
        AppBuffer[0] = BUTTON0_IO;
        AppBuffer[1] = BUTTON1_IO;
        AppBuffer[2] = BUTTON2_IO;
        len = 3;
      }
      else if ( AppBuffer[0] == 'w' ) {
        // По команде 'w' сохраняем полученные данные в CommonBuf
        // и возвращаем версию ПО

        if ( len == 5) {
          CommonBuf[0] = AppBuffer[1];
          CommonBuf[1] = AppBuffer[2];
          CommonBuf[2] = AppBuffer[3];
          CommonBuf[3] = AppBuffer[4];
          AppBuffer[0] = '1';
          AppBuffer[1] = '.';
          AppBuffer[2] = '0';
        } else {
          // Или возвращаем сообщение об ошибке,
          // если длина полученного сообщения неправильная
          AppBuffer[0] = 'E';
          AppBuffer[1] = 'r';
          AppBuffer[2] = 'r';
```

```
            len = 3;
          }
        }
        // Передаем данные в сокет
        TCPPutArray(MySocket, AppBuffer, len);
      }
    }
}
```

Компилируем проект (Project->Make)
Загружаем программу (Debugger->Programming->Program All Memories). Подтвердить программирование.
Запускаем выполнение программы (Debugger -> Run)

Проверяем работу сервера. Для этого на компьютере запускаем командную строку (Пуск->Все программы->Стандартные->Командная строка) . В открывшемся окне выполняем команду:
telnet 169.254.1.1 9760
При подключении к серверу окно очищается. Теперь, нажимая на клавишу 'a' на клавиатуре компьютера мы посылаем эту команду в TCP сокет. При каждом нажатии мы должны видеть изменение состояния светодиода на плате.
Закрываем окно.

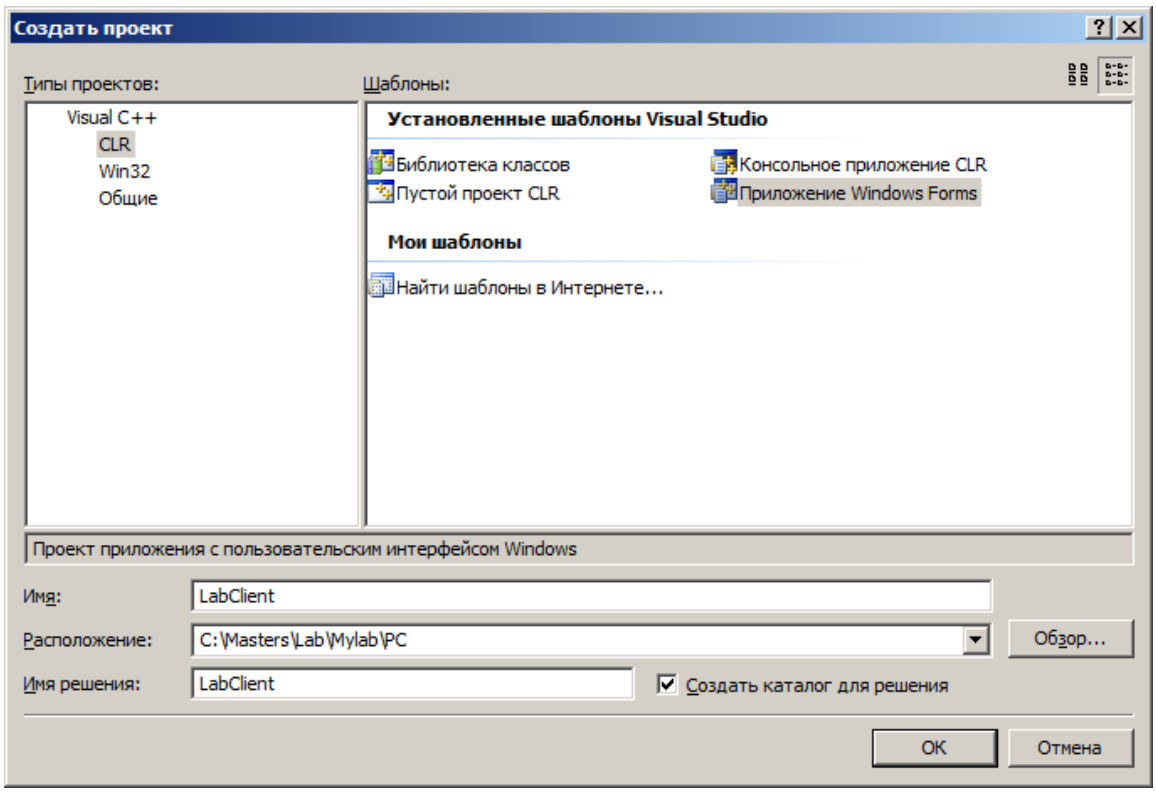
(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep1f.bat из папки C:\Masters\Lab\Solution\SolutionSteps. MPLAB IDE при выполнении должен быть закрыт)

Шаг 2. Создаем на компьютере программу-клиент для работы с сервером

a) Создаем проект и пользовательский интерфейс

Проект будем создавать в папке C:\Masters\Lab\MyLab\PC.
Запускаем Visual C++ (Пуск->Все программы->Visual C++ 9.0 Express Edition->Microsoft Visual C++ 2008 Express Edition).

Запускаем мастер создания проекта (Файл->Создать->Проект).
Выбираем создание CLR приложения Windows Forms.
В качестве папки указываем C:\Masters\Lab\MyLab\PC
Имя задаем – LabClient



Нажимаем OK

Если в появившемся виде отсутствует окно «Свойства», открываем его из меню:
Вид->Другие окна->Окно свойств

Создаем пользовательский интерфейс

Кликаем на форму Form1 и в окне «Свойства» изменяем следующие свойства:
Text : “LabClient v1.0”
Size : 350; 300

Открываем панель элементов (Вид->Панель элементов), добавляем и расставляем по форме следующие элементы:
2 элемента Button
3 элемента CheckBox
5 элементов TextBox

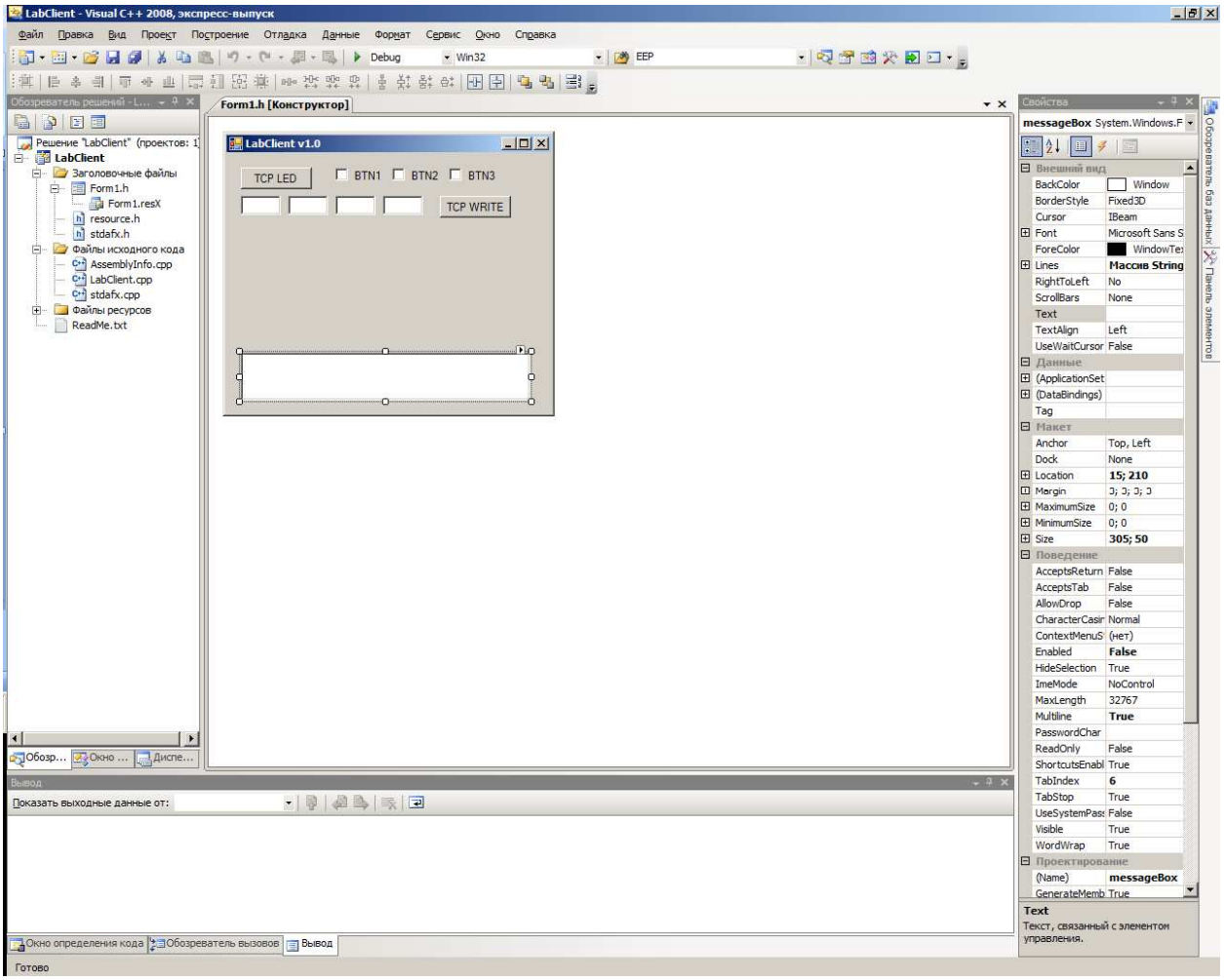
Выбирая по очереди каждый из добавленных элементов, в окне «Свойства» изменяем следующие свойства:

- Для кнопки управления светодиодом платы:
  Text: TCP LED
  Location: 15;15
  (Name): tcpLED
- Для кнопки записи данных в плату:
  Text: TCP WRITE
  Location: 225;45
  (Name): tcpWrite
- Для отображения состояния кнопок платы:

Text: BTN1/BTN2/BTN3
Location: 115;15/175;15/235;15
(Name): btn1/btn2/btn3

- Для полей данных, записываемых в плату:
  Location: 15;45/65;45/115;45/165;45
  Size: 40;20
  (Name): dat1/dat2/dat3/dat4
- Для поля выводимых сообщений:
  Multiline: true
  Enabled: false
  Location: 15;210
  Size: 305;50
  (Name): messageBox

В результате мы должны получить в конструкторе сформированный пользовательский интерфейс:



Сохраняем состояние. (Файл->Сохранить все)
Проверяем результат. Отладка->Начать отладку. Убеждаемся, что получили окно с требуемым пользовательским интерфейсом. Закрываем окно.

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep2a.bat из папки C:\Masters\Lab\Solution\SolutionSteps. Visual C++ при выполнении должен быть закрыт)

b) Создаем обработчик кнопки управления светодиодом

Делаем двойной клик в форме на кнопке управления светодиодом (TCP LED).
По этому действию открывается окно с кодом программы и в этом окне создается заготовка для обработчика нажатия кнопки:

```
private: System::Void tcpLED_Click(System::Object^  sender, System::EventArgs^  e) {
}
```

В эту заготовку мы вписываем код который:
- открывает клиентское соединение на сервер с адресом 169.254.1.1 на порт 9760
- выдает в соединение команду 'a'
- считывает ответ, индицирующий состояние кнопок платы
- отображает состояние кнопок платы в пользовательском интерфейсе
- закрывает клиентское соединение.

```
private: System::Void tcpLED_Click(System::Object^  sender, System::EventArgs^  e) {

         // Блок try/catch позволяет коррректно обработать ошибочные
         // ситуации при работе с сервером
         try {
              // Открываем клиентское соединение на порт 9760 сервера 169.254.1.1
              System::Net::Sockets::TcpClient ^client =
                    gcnew System::Net::Sockets::TcpClient( "169.254.1.1", 9760);

           if ( client->Connected ) {  // Если подключение успешно

              // Создаем передающий буфер
              array<unsigned char, 1> ^outBuf = gcnew array<unsigned char,1>(1);

              // Формируем команду 'a' в передающем буфере
              outBuf[0] = 'a';

              // Создаем приемный буфер
              array<unsigned char, 1> ^inBuf = gcnew array<unsigned char,1>(3);

              // Передаем команду
              client->Client->Send(outBuf);

              // Принимаем ответ
              client->Client->Receive(inBuf);

              // Отображаем состояние кнопок по результатам ответа
              if ( inBuf[0] ) btn1->Checked = false;
              else btn1->Checked = true;

              if ( inBuf[1] ) btn2->Checked = false;
              else btn2->Checked = true;

              if ( inBuf[2] ) btn3->Checked = false;
              else btn3->Checked = true;

              messageBox->Text =   "OK";
           }

           // Закрваем соединение
           client->Close();
         } catch (Exception^ e) {

           // Отображаем ошибку
           messageBox->Text =  e->Message;
```

```
        }
}
```

Проверяем работу программы. Отладка->Начать отладку.
В полученном окне нажать кнопку «TCP LED» и убедиться, что светодиод на плате меняет свое состояние.
Удерживать одну или несколько кнопок платы при нажатии на кнопку «TCP LED» окна.
Убедиться, что состояние кнопок отображается в соответствующих элементах окна.
Закрываем окно.
Сохраняем состояние. (Файл->Сохранить все)

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep2b.bat из папки C:\Masters\Lab\Solution\SolutionSteps. Visual C++ при выполнении должен быть закрыт)

c) Создаем обработчик кнопки записи данных

Обработчик создаем по аналогии с обработчиком кнопки управления светодиодом.
Переходим к окну Form1.h(Конструктор) и делаем двойной клик на кнопке записи данных (TCP WRITE).
В получившуюся заготовку для обработчика вписываем код который:
- открывает клиентское соединение на сервер с адресом 169.254.1.1 на порт 9760
- выдает в соединение команду 'w' с данными, считанными из полей данных
- считывает ответ, показывающий версию ПО
- отображает версию в поле вывода сообщений
- закрывает клиентское соединение.

```
private: System::Void tcpWrite_Click(System::Object^  sender, System::EventArgs^  e) {

         // Блок try/catch позволяет корррректно обработать ошибочные
         // ситуации при работе с сервером
         try {
               // Открываем клиентское соединение на порт 9760 сервера 169.254.1.1
               System::Net::Sockets::TcpClient ^client =
                     gcnew System::Net::Sockets::TcpClient( "169.254.1.1", 9760);

           if ( client->Connected ) {  // Если подключение успешно

               // Создаем передающий буфер
               array<unsigned char, 1> ^outBuf = gcnew array<unsigned char,1>(5);

               // Формируем команду 'w' в передающем буфере
               outBuf[0] = 'w';
               // Записываем в буфер данный из полей ввода данных
               System::Byte::TryParse( dat1->Text, outBuf[1]);
               System::Byte::TryParse( dat2->Text, outBuf[2]);
               System::Byte::TryParse( dat3->Text, outBuf[3]);
               System::Byte::TryParse( dat4->Text, outBuf[4]);

               // Создаем приемный буфер
               array<unsigned char, 1> ^inBuf = gcnew array<unsigned char,1>(3);

               // Передаем команду
               client->Client->Send(outBuf);

               // Принимаем ответ
               client->Client->Receive(inBuf);

               // Отображаем полученную версию ПО
```

```
            System::Text::ASCIIEncoding ^str = gcnew System::Text::ASCIIEncoding();
            messageBox->Text = str->GetString(inBuf);
        }

        // Закрваем соединение
        client->Close();
    } catch (Exception^ e) {

        // Отображаем ошибку
        messageBox->Text =  e->Message;
    }
}
```

Проверяем работу программы. Отладка->Начать отладку.
В полученном окне нажать кнопку «TCP Write» и убедиться, что в поле выводимых сообщений выводится номер версии ПО, передаваемый с платы
Закрываем окно.
Сохраняем состояние. (Файл->Сохранить все)

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep2c.bat из папки C:\Masters\Lab\Solution\SolutionSteps. Visual C++ при выполнении должен быть закрыт)

Шаг 3. Добавляем USB HID device на плате PIC32 Ethernet Starter Kit

a) Добавляем в проект USB стек

В папку проекта C:\Masters\Lab\MyLab\Embedded\ из примера «C:\Microchip Solutions vxxxx-xx-xx\USB\Device - HID - Custom Demos\Firmware» копируем файлы:
usb_descriptors.c
usb_config.h

Если еще не открыт, открываем наш проект в MPLAB IDE.
В MPLAB IDE в наш проект добавляем файлы USB стека (Project->Add Files To Project):
usb_descriptors.c (из папки C:\Masters\Lab\MyLab\Embedded\)
usb_device.c (из библиотеки “C:\Microchip Solutions vxxxx-xx-xx\Microchip\USB”)
usb_function_hid.c (из библиотеки “C:\Microchip Solutions vxxxx-xx-xx\Microchip\USB\HID Device Driver”)

Открываем в MPLAB IDE (File->Open…) файл аппаратной конфигурации HWP PIC32_ETH_SK_ETH795.h из подпапки Configs нашего проекта.
В этом файле определяем макрос, выбирающий вариант питания платы от USB:

```
#define self_power          0
```

Сохраняем проект File->Save All, File->Save Workspace

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep3a.bat из папки C:\Masters\Lab\Solution\SolutionSteps. MPLAB IDE при выполнении должен быть закрыт)

b) Создаем в программе USB HID device

Для добавления USB устройства в файле main.c делаем следующие изменения:

Включаем необходимые файлы заголовков:

```
#include "TCPIP Stack/TCPIP.h"
#include "USB/usb.h"
#include "USB/usb_function_hid.h"
```

Объявляем глобальные переменные для USB дескрипторов и буферов приема и передачи:

```
APP_CONFIG AppConfig; // Переменная конфигурации TCP/IP стека
BYTE CommonBuf[4];    // Общий буфер задачи
unsigned char ReceivedDataBuffer[64];     // Приемный буфер USB
unsigned char ToSendDataBuffer[64];       // Передающий буфер USB
USB_HANDLE USBOutHandle = 0;              // Передающий дескриптор
USB_HANDLE USBInHandle = 0;               // Приемный дескриптор
```

В функцию main добавляем инициализию и подключение USB устройства:

```
InitAppConfig();   // Инициализируем приложение
StackInit();       // Инициализируем стек
USBDeviceInit();   // Инициализируем USB Device
USBDeviceAttach(); // Подключаем USB Device
```

Добавляем обработчик USB событий. В простейшем случае нам надо обрабатывать только два события: EVENT_CONFIGURED и EVENT_EP0_REQUEST

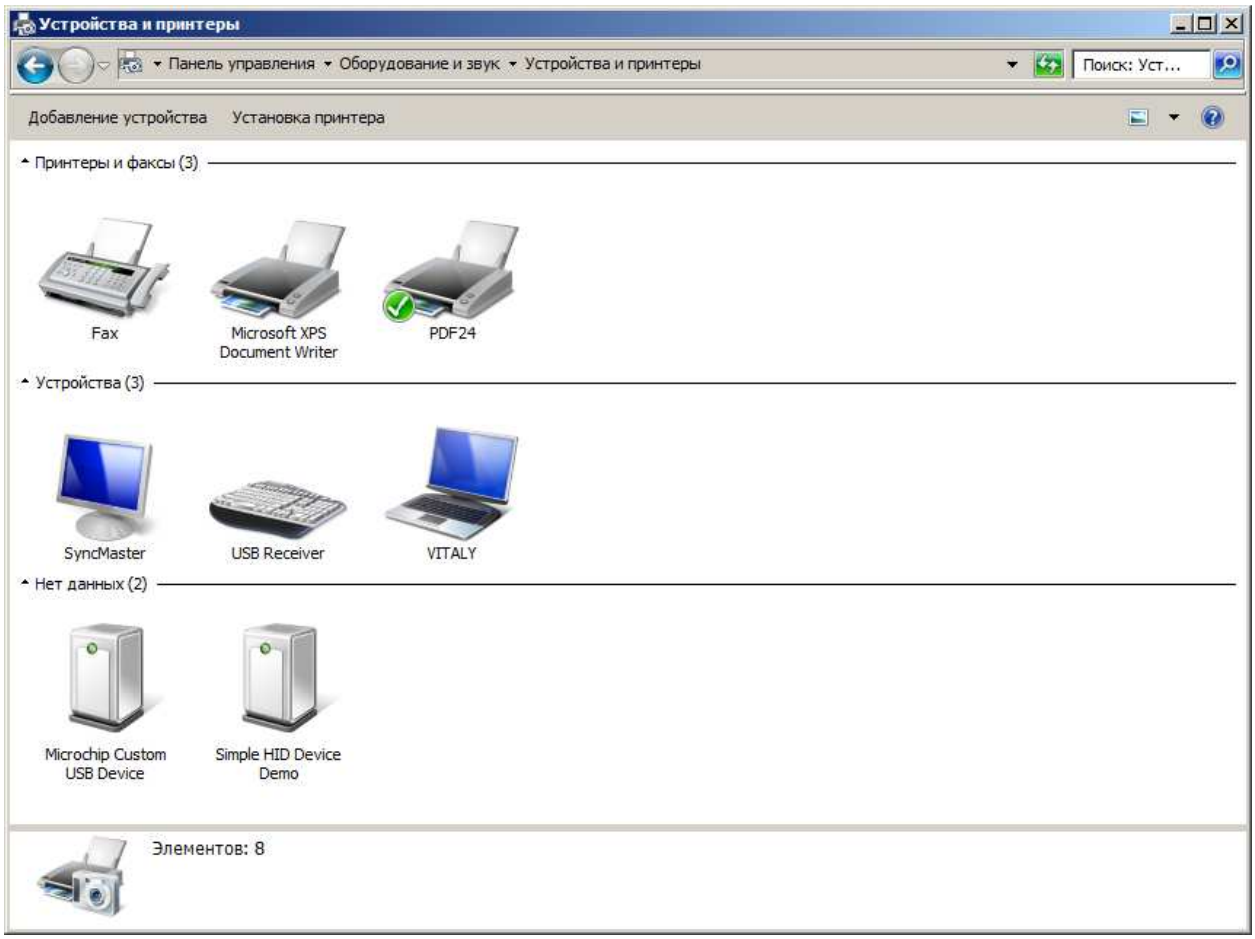
```
BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size)
{
    switch(event)
    {
        case EVENT_CONFIGURED:
             //Разрешаем конечную точку EP0
             USBEnableEndpoint(HID_EP,
             USB_IN_ENABLED|USB_OUT_ENABLED|USB_HANDSHAKE_ENABLED|
             USB_DISALLOW_SETUP);
             // Подготавливаем точку EP0 для приема сообщений
             USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
             break;
        case EVENT_EP0_REQUEST:
             USBCheckHIDRequest(); // Стандартный обработчик HID запросов
             break;
        default:
             break;
    }
    return TRUE;
}
```

Сохраняем проект File->Save All, File->Save Workspace

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep3b.bat из папки C:\Masters\Lab\Solution\SolutionSteps. MPLAB IDE при выполнении должен быть закрыт)

c) Проверяем подключение USB device к компьютеру

Компилируем проект (Project->Build All).
Загружаем программу (Debugger->Programming->Program All Memories). Подтвердить программирование.
Запускаем выполнение программы (Debugger -> Run).
Соединяем кабелем USB device разъем платы с USB разъемом компьютера.
Подключенное USB устройство должно распознаться компьютером как HID устройство (драйвер для HID устройств уже имеется в составе Windows).
Проверить, что устройство подключилось можно через панель управления.
Пуск->Панель управления->Просмотр устройств и принтеров



Наше USB устройство, описанное в файле usb_descriptors.c как “Simple HID Device Demo” , отображается в списке подключенных устройств

d) Добавляем задачу обмена командами по USB

Добавляем прототип функции USBTask в список прототипов функций:

```
void InitAppConfig(void);  // Функция инициализации приложения
void InitializeBoard(void); // Функция инициализации платы
void ServerTask(void);           // Серверная задача
void USBTask(void);       // USB задача
```

Добавляем вызов функции USBTask в теле бесконечного цикла функции main.

```
StackTask();        // выполняем задачи стека
StackApplications();    // выполняем задачи приложения
ServerTask();           // выполняем задачу TCP сервера
USBTask();              // выполняем USB задачу
```

Создаем функцию USBTask:

```
void USBTask(void)
{
    // проверяем, что устройство в работоспособном состоянии
    if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1)) return;


    // Проверяем, получены ли от компьютера новые данные

    if(!HIDRxHandleBusy(USBOutHandle))     {
      // данные приняты
        switch(ReceivedDataBuffer[0]) // Проверяем команду
        {

            case 'a': // По команде ‘a’.
                LED2_IO = (LED2_IO) ?   0 : 1; // меняем состояние светодиода
                break;
            case 'r': //По команде ‘r’.
                // Заполняем передающий буфер данными из общего буфера
                ToSendDataBuffer[0] = CommonBuf[0];
                ToSendDataBuffer[1] = CommonBuf[1];
                ToSendDataBuffer[2] = CommonBuf[2];
                ToSendDataBuffer[3] = CommonBuf[3];
                if(!HIDTxHandleBusy(USBInHandle)) // Если готов к передаче
                {
                    // Передаем пакет
                     USBInHandle =
                           HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],64);
                }
                break;
        }

        //Готовим конечную точку к следующему приему
        USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
    }

}
```

Компилируем проект (Project->Build All).
Загружаем программу (Debugger->Programming->Program All Memories). Подтвердить программирование.
Запускаем выполнение программы (Debugger -> Run).
Сохраняем проект File->Save All, File->Save Workspace

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep3c.bat из папки C:\Masters\Lab\Solution\SolutionSteps. MPLAB IDE при выполнении должен быть закрыт)

Шаг 4. Добавляем в программу-клиент работу с USB

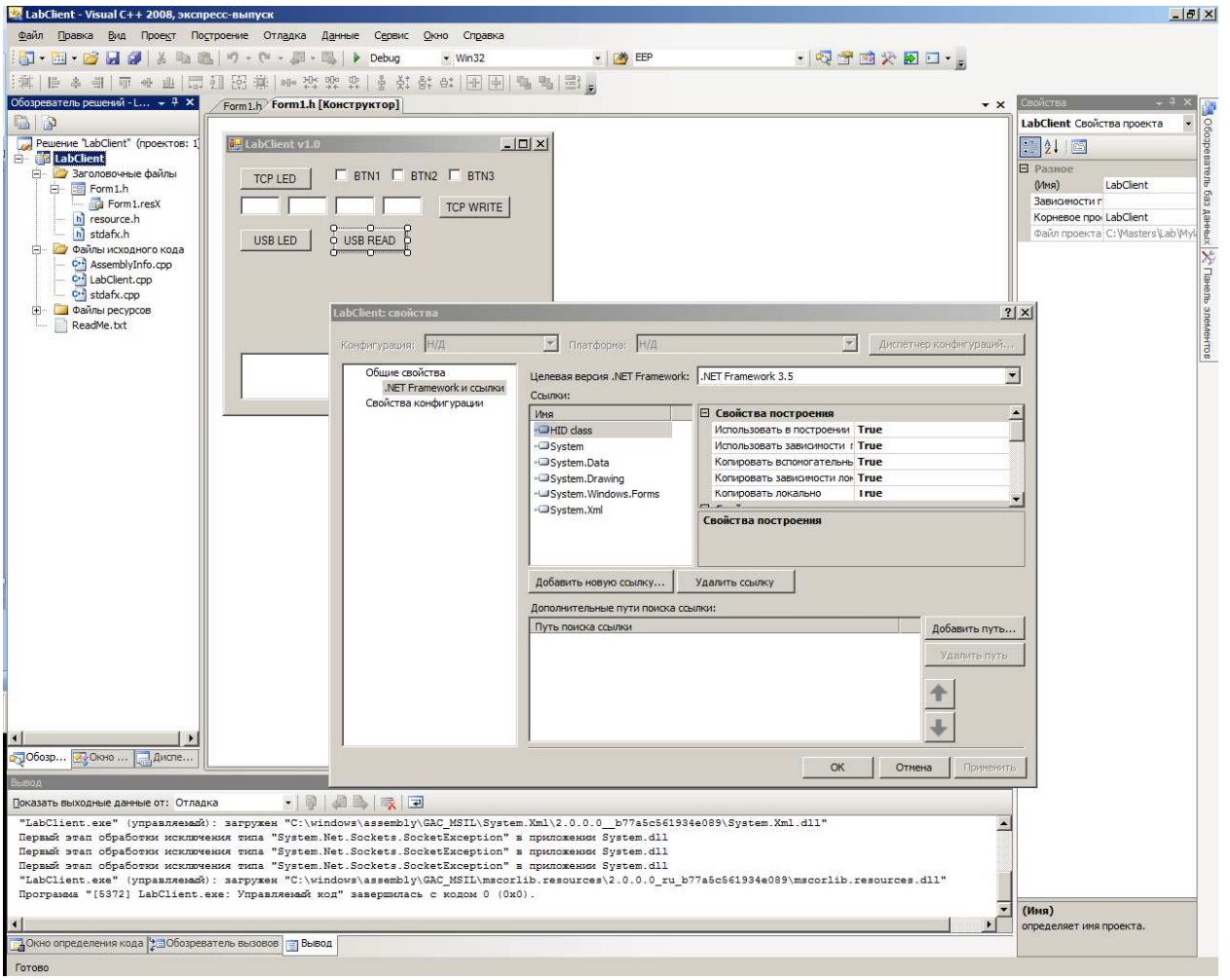
a) **Добавляем элементы пользовательского интерфейса**
Если еще не открыт, то открываем в Visual C++ наше решение C:\Masters\Lab\MyLab\PC\LabClient\LabClient.sln, кликнув по нему мышкой.

В окне конструктора из панели элементов, добавляем и расставляем по форме 2 элемента Button

Выбирая по очереди каждый из добавленных элементов, в окне «Свойства» изменяем следующие свойства:

- Для кнопки управления светодиодом платы:
  Text: USB LED
  Location: 15;80
  (Name): usbLED
- Для кнопки чтения данных из платы:
  Text: USB READ
  Location: 115;80
  (Name): usbRead

b) **Подключаем HID Class**
Для работы с HID драйвером мы будем использовать уже готовый HID Class, имеющийся в библиотеке Microchip.
Ссылку на него надо подключить к проекту.
Для этого в обозревателе решений устанавливаем курсор на проект LabClient и кликаем правой кнопкой мыши. Из выпадающего меню выбираем пункт «Ссылки».



В открывшемся окне кликаем по «Добавить новую ссылку…» .

Выбираем закладку «Обзор» и идем в папку библиотеки “C:\Microchip Solutions vxxxx-xx-xx\USB\Device - HID – Custom Demos\HID DLL – PC Software\Microsoft Visual C++ 2008 Express ”. Выбираем файл “HID Class.dll” и нажимаем ОК.

c) Добавляем инициализацию USB класса

Инициализировать класс будем при загрузке формы. Прототип обработчика этого события создаем из закладки Form1(Конструктор), кликнув на свободном от элементов поле формы.

В получившийся прототип функции Form1_Load вписываем программу инициализации класса.

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {

            // Инициализируем класс
            HIDClass::MCHPHIDClass::HIDClassInit(0x4D8, 0x003F, 64);
            if (HIDClass::MCHPHIDClass::HIDIsConnected() ) {
                  // В случае успеха выводим сообщение
                 messageBox->Text =   "USB Connected";
            } else {
                  // Иначе сообщаем об ошибке
                 messageBox->Text =   "Can’t Connect USB";
            }

}
```

Проверяем работу программы. Отладка->Начать отладку.

Убеждаемся, что окно имеет требуемый нам пользовательский интерфейс и при запуске программы в окне сообщений выводится «USB Connected».
Закрываем окно.
Сохраняем состояние. (Файл->Сохранить все)

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep4c.bat из папки C:\Masters\Lab\Solution\SolutionSteps. Visual C++ при выполнении должен быть закрыт)

d) Создаем обработчики кнопок управления

Прототипы функции обработчиков кнопок управления светодиодом и считывания данных также создаем из закладки Form1(Конструктор), кликнув на соответствующей кнопке.

Обработчик кнопки управления светодиодом должен выдавать команду 'a' в USB.

```
private: System::Void usbLED_Click(System::Object^  sender, System::EventArgs^  e) {

             // Создаем буфер передачи
              unsigned char OutputPacketBuffer[64];
             // Записываем команду
              OutputPacketBuffer[0] = 'a';
             // Передаем буфер в USB
              if (   HIDClass::MCHPHIDClass::HIDWriteReport(OutputPacketBuffer, 1)) {
                    // В случае успеха выводим сообщение
                    messageBox->Text =   "USB is OK ";
              } else {
                    // Иначе сообщаем об ошибке
                    messageBox->Text =   "Cant send USB";
              }

}
```

Обработчик кнопки чтения данных должен выдавать команду 'r' и выводить данные, полученные по USB.

```
private: System::Void usbRead_Click(System::Object^  sender, System::EventArgs^  e) {


       // Создаем буфера приема и передачи
       unsigned char OutputPacketBuffer[64];
       unsigned char InputPacketBuffer[64];

       // Записываем команду
       OutputPacketBuffer[0] = 'r';

       // Передаем буфер в USB
       if (HIDClass::MCHPHIDClass::HIDWriteReport(OutputPacketBuffer, 1)) {
              // Принимаем ответный пакет
              if ( HIDClass::MCHPHIDClass::HIDReadReport(InputPacketBuffer) ) {

                    // Формируем сообщение о принятых данных
                    messageBox->Text =  Convert::ToString(InputPacketBuffer[0])+"-"+
                           Convert::ToString(InputPacketBuffer[1])+"-"+
                           Convert::ToString(InputPacketBuffer[2])+"-"+
                           Convert::ToString(InputPacketBuffer[3]);
                    return;
              }
       }
       // При проблемах сообщаем об ошибке
       messageBox->Text =   "Cant send USB";
}
```

Проверяем работу программы. Отладка->Начать отладку.
В полученном окне нажимать кнопку «USB LED» и убедиться, что светодиод на плате переключается.
Проверяем обмен данными между TCP и USB через плату. Для этого в поля данных формы записываем четыре значения в диапазоне от 0 до 255. Нажимаем кнопку «TCP WRITE», после чего нажимаем кнопку «USB READ». Проверяем, что в окне сообщений выводятся те же данные, что и в полях данных.
Закрываем окно.

Сохраняем состояние. (Файл->Сохранить все)

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep4d.bat из папки C:\Masters\Lab\Solution\SolutionSteps. Visual C++ при выполнении должен быть закрыт)

e) Добавляем автоматическое распознавание подключения USB

В окне конструктора в пользовательский интерфейс добавляем элемент checkbox со следующими свойствами:

Text: USB Connected
Location: 15;120
(Name): connected

Для периодического опроса состояния в пользовательский интерфейс добавляем элемент Timer из раздела «Компоненты» Панели инструментов.

В свойствах таймера устанавливаем:

Enabled: True
Interval: 500

Прототип обработчика создаем, кликая на элемент таймера в форме конструктора.

В обработчик добавляем отображение состояния подключения.

```
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {

      if (HIDClass::MCHPHIDClass::HIDIsConnected()) connected->Checked = true;
      else connected->Checked = false;

}
```

Сохраняем состояние. (Файл->Сохранить все)
Проверяем работу программы. Отладка->Начать отладку.
«Птичка» в элементе «USB Connected» должна изменять свое состояние при подключении и отключении кабеля USB.

Задача решена !!!

(Решение: При необходимости, все действия этого пункта можно выполнить, запустив файл labStep4e.bat из папки C:\Masters\Lab\Solution\SolutionSteps. Visual C++ при выполнении должен быть закрыт)