Microchip Masters 2009

Лабораторная работа

# Интеграция пользовательских приложений в Microchip TCP/IP Stack

**Отладочные средства**:

- Explorer16 (*DV240001*) и процессорный модуль с контроллером PIC24FJ128GA010
- Плата под разъем Pic-tail с ENC28J60 (*AC164123*)

**Программное обеспечение**:

- Среда разработки MPLAB IDE
- Microchip TCP/IP Stack с набором утилит
- HTML-редактор
- Монитор сетевого трафика

# Шаг 1

- Проверка работоспособности оборудования
- Настройка и сборка демонстрационного проекта
- Загрузка в EEPROM примера веб-странички

# Подключение Explorer16

• Подсоедините плату к питанию и подключите к компьютеру по USB. Убедитесь, что переключатель S2 находится в положении PIM

• Запустите MPLAB IDE

• Откройте проект, находящийся по пути *C:\MASTERS\COM4201\TCPIP Demo App\TCPIP Demo App-C30.mcp*

• В качестве программатора выберите PicKit 2

> На плате Explorer16 имеется запаянный контроллер PIC18F4550. Изначально в нем зашит только бутлоадер, но в нашем случае там находится адаптированная прошивка PicKit 2. Как самостоятельно прошить ей этот контроллер – описано в  даташите на Explorer 16 – Appendix B

• Убедитесь, что PicKit 2 распознал контроллер PIC24FJ128GA010 (в окне Output)

• Соберите (make) проект и прошейте контроллер. Подайте высокий уровень на MCLR цели. Сигнал о корректном выполнении кода — мигание крайнего правого светодиода на плате.

# Структура проекта:
# *TCPIPConfig.h*

Основной конфигурационный файл библиотеки.

Отвечает за:

- Подключаемые модули (Application Options)
- Место хранения данных (Data Storage Options) – образа файлов для веб-сервера и сетевых настроек. Три варианта:
  - внутренный флэш контроллера
  - внешняя SPI-флэш-память
  - внешняя SPI-EEPROM (в нашем случае)
- Сетевые настройки (Network Addressing Options) — имя хоста, MAC, IP, маска подсети, шлюз, DNS.
  Данные настройки находятся в особом массиве — AppConfig, который хранится в EEPROMe (или во флэше, в зависимости от пункта выше)
- Настройка транспортного уровня (Transport Layer Options)
- Настройка приложений (Application-Specific Options)

Начинающим может быть полезна утилита TCPIPConfig.exe:
C:\MASTERS\COM4201\Microchip Solutions\Microchip\TCPIP Stack\Utilities

# Структура проекта:
# *HardwareProfile.h*

Выбор и описание платформы.

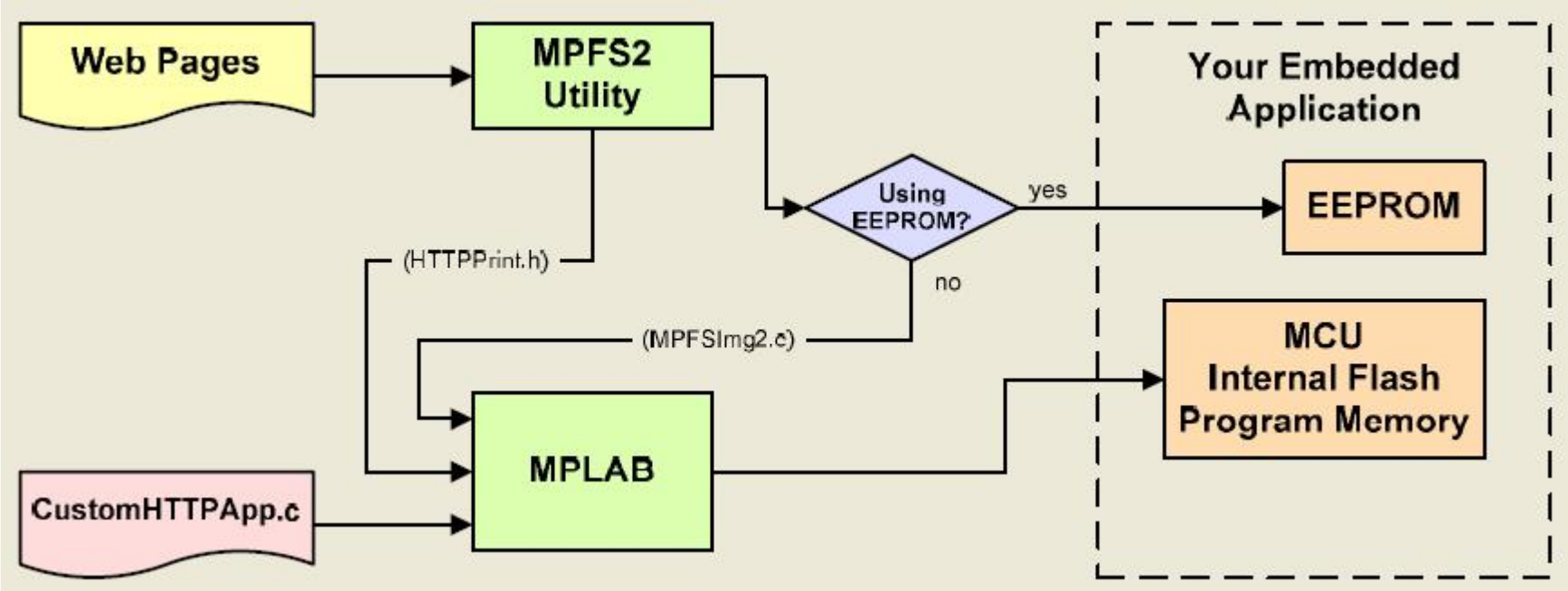
TCP/IP Stack — кроссплатформенная библиотека, совместимая с семействами PIC18, PIC24, dsPIC, PIC32. В данном файле осуществляется выбор контроллера, а также настройка под конкретную отладочную плату (в нашем случае – Explorer 16)

При подключении этой библиотеки к собственному проекту файл *HardwareProfile.h* должен быть переписан под конкретную топологию.

# Структура проекта:
# *MPFS*

Для хранения файлов веб-страниц в библиотеке применена своеобразная файловая система MPFS. Для загрузки в проект исходные файлы веб-сраницы должны быть прекомпилированы специальной утилитой MPFS.exe:

C:\MASTERS\COM4201\Microchip Solutions\Microchip\TCPIP Stack\Utilities



# Структура проекта:
## *HTTP-сервер*

Пользователь может выбирать между старой (HTTP) и новой (HTTP2) версиями веб-сервера. Каждый из них использует свою файловую систему и имеет ряд важных отличий:

### HTTP

Простой и легко поддается модификации

Файловая система: **MPFS**

Просто совмещается с MDD File System

Методы обмена информацией:

%01
параметры в адресной строке

### HTTP2

Поддерживает современные функции (cookies, аутентификация, POST и т.д.)

Файловая система: **MPFS2**

Осуществляет сжатие файлов, ускоряет доступ за счет хэширования

Методы обмена информацией:

~variable~
параметры в адресной строке
POST

# Загрузка в EEPROM примера веб-странички

• Подключите плату к компьютеру по Ethernet-кабелю. При успешном подключении в области уведомлений Windows появится значок «Подключение установлено»

• Адресоваться к плате можно двумя путями: непосредственно по IP-адресу, или по NetBIOS-имени.

> Актуальный IP отображается на дисплее платы; он может отличаться от заданного в TCPIPConfig.h, если включено динамическое автоопределение IP (DHCP)

• Открыв браузер, вбейте в адресной строке
http://MCHPBOARD/mpfsupload

• Выберите прекомпилированный образ демонстрационной страницы:
C:\MASTERS\COM4201\TCPIP Demo App\MPFSImg2.bin

• После загрузки впишите в адресной строке
http://MCHPBOARD

# Шаг 2

- Азы HTML
- CSS

# Создание веб-страницы при помощи визуального редактора

- Запустите Dreamweaver

  Использовать визуальные HTML-редакторы при создании WEB-интерфейсов для контроллеров, в принципе, не рекомендуется, так как они генерируют избыточный код. Однако, для наглядности и ускорения процесса, мы решили использовать Dreamweaver

- Выберите Create New -> HTML
- Перейдите в режим Code

  Редактор самостоятельно создал базовую разметку страницы. Как видно, ее структура состоит из тегов, большинство из которых являются парными: открывающими и закрывающими: <html>…</html>, <head>…</head>, <body>…</body>

# Создание веб-страницы при помощи визуального редактора

• В тэге <title>…</title> введите заголовок страницы. Он будет отображаться в заголовке окна браузера.

• В тэге <body>…</body> разместите произвольный текст, чтобы ознакомиться с тэгами разметки.

```
<body>
<h2> Заголовок выделяется увеличенным шрифтом</h2>
<p align="center">Тэг абзаца позволяет применять форматирование к тексту </p>
<p align="left">
        <b>Жирный текст</b> <br />
        <em>Курсив</em> <br />
        <a href="http://www.microchip.com"> Ссылка на Микрочип</a>
        <input type="button" name="Бесполезная пока кнопка" />
</p>
</body>
```

•Выберите режим отображения Design, чтобы посмотреть, как действуют тэги. В этом режиме можно редактировать текст, как в обычных редакторах.

# Создание каскадной таблицы стилей (CSS)

• Создайте новый файл. Теперь это будет css-таблица.

CSS - технология описания внешнего вида документа, написанного языком разметки. Преимущественно используется как средство оформления веб-страниц в формате HTML и XHTML, но может применяться с любыми видами документов в формате XML.

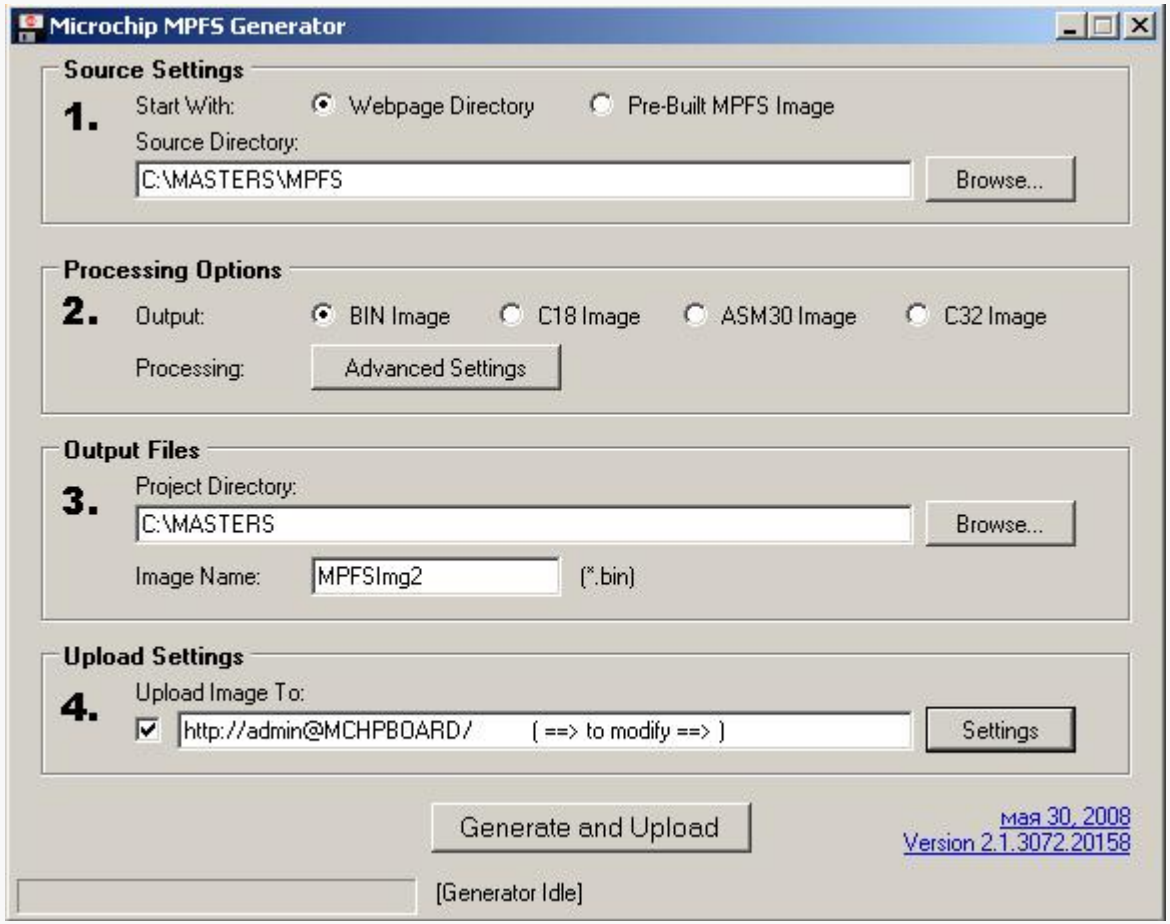
• Опишите стили для тела, заголовка, ссылки и пользовательского стиля style1.
Выглядеть это должно примерно так:

```
body {
        background: #f0f0f0;
        font-family: Verdana, Arial, sans-serif;
        color: #444444;
}
a {color:#0066FF};
h2 {color:#0066FF; text-decoration:blink};
p.style1 {
        color:#FFCC00
        font-size: 36px
}
```

# Прекомпиляция и загрузка страниц с помощью утилиты MPFS2

- Сохраните созданные файлы.
- Запустите утилиту MPFS2 и загрузите страницу в Explorer16

C:\MASTERS\COM4201\Microchip Solutions\Microchip\TCPIP Stack\Utilities



# Шаг 3

- Интеграция проекта в TCP/IP Stack

# Интеграция проекта в TCP/IP Stack

• Откройте, соберите и прошейте в Explorer 16 проект Vending Machine:
C:\MASTERS\COM4201\Vending Machine

Это простая программа, эмулирующая работу автомата продажи напитков.
Наша задача — встроить ее в уже знакомый нам TCP/IP Stack

• Скопируйте в директорию Lab3 файлы VendingMachine.c и VendingMachine.h

• Откройте проект TCP/IP Stack:
C:\MASTERS\COM4201\Lab3\COM4201_Lab3-C30.mcp

•Добавьте скопированные файлы в проект:
Project ► Add Files ► VendingMachine.c и VendingMachine.h

# Интеграция проекта в TCP/IP Stack

- Отредактируйте файл VendingMachine.h:
  Закомментируйте описанные там подключения хэдеров;
  Добавьте хэдер TCPIP Stack/TCPIP.h:

```
53
54
55 /*********************************************************************
56   Section:
57     Required Headers
58   ********************************************************************/
59 //#include "Compiler.h"
60 //#include "GenericTypeDefs.h"
61 //#include "HardwareProfile.h"
62 //#include "Delay.h"
63 //#include "LCDBlocking.h"
64 #include "TCPIP Stack/TCPIP.h"
65
66
```

Так как оба проекта делались под Explorer16, большинство их определений одинаковы. Привнесенный VendingMachine.c «найдет» все необходимое в файлах определений TCP/IP стэка.

# Интеграция проекта в TCP/IP Stack

• Отредактируйте файл MainDemo.c:
Добавьте #include “VendingMachine.h”:

```
88
89 // Include functions specific to this stack application
90 #include "MainDemo.h"
91 #include "VendingMachine.h"
92
93 // Declare AppConfig structure and some other supporting stack va
```

• В функции Main разместите вызов функции инициализации:

```
196     #endif
197
198     // Initialize Stack and application related NV variables into App
199     InitAppConfig();
200     InitializeVend();
201
202     // Initiates board setup process if button is depressed
203     // on startup
```

# Интеграция проекта в TCP/IP Stack

• Далее, в том же файле замените содержимое функции ProcessIO. Вместо обработки АЦП разместите там вызов функции VendingMachine():

```
379     #endif
380 }
381
382 // Processes A/D data from the potentiometer
383 static void ProcessIO(void)
384 {
385     VendingMachine();
386 }
387
388 /*****************************************************************************
```

# Интеграция проекта в TCP/IP Stack

• Осталось избавиться от конфликтов между нашими двумя проектами:
 - Закомментируем вызов функции PingDemo(). Это предоставит самую правую кнопку платы в полное распоряжение программы Vending Machine.
 - Закомментируем DisplayIPValues(), чтобы освободить индикатор.

```
299         #if defined(STACK_USE_ICMP_CLIENT)
300         //PingDemo();
301         #endif
302
303         #if defined(STACK_USE_SNMP_SERVER) && !defined(SNMP_TRAP_DISABLED)
304         SNMPTrapDemo();
305         #endif
306
307         #if defined(STACK_USE_BERKELEY_API)
308         BerkeleyTCPClientDemo();
309         BerkeleyTCPServerDemo();
310         BerkeleyUDPClientDemo();
311         #endif
312
313         ProcessIO();
314
315         // If the DHCP lease has changed recently, write the new
316         // IP address to the LCD display, UART, and Announce service
317         if(DHCPBindCount != myDHCPBindCount)
318         {
319             myDHCPBindCount = DHCPBindCount;
320
321             #if defined(STACK_USE_UART)
322                 putrsUART((ROM char*)"\r\nNew IP Address: ");
323             #endif
324
325             //DisplayIPValue(AppConfig.MyIPAddr);
326
327             #if defined(STACK_USE_ANNOUNCE)
```

# Интеграция проекта в TCP/IP Stack

• Соберите и прошейте проект.

• Убедитесь, что программа Vending Machine выполняет свои функции, и при этом поддерживается связь с компьютером по Ethernet’у.

Мы не привнесли никаких новых функциональных возможностей: оба проекта работают независимо друг от друга. Однако их взаимная работа — уже существенный шаг в интеграции проектов.

# Шаг 4

- Избавляемся от длительных задержек

# Что такое стопорящие циклы?

Проведем небольшой эксперимент с предыдущим проектом: нажимая на левую кнопку, заполним импровизированный автомат в общей сумме пятью долларами.

Если попытаться положить еще, программа выведет на экран предупреждение о невозможности операции.

А теперь обратим внимание на светодиод, мигание которого отражает регулярное исполнение основного цикла. Пока высвечивается предупреждение, он не мигает.

Это происходит из-за того, что все задержки в проекте Vending Machine реализованы с помощью стопорящих циклов.

# Модуль Tick

В условиях работающего TCP/IP-соединения длительные задержки нежелательны (а зачастую — недопустимы). Вместо того, чтобы считать до 1 000 000, лучше потратить это время на что-нибудь осмысленное (выполнение фоновых подпрограмм стека, например).

В библиотеке TCP/IP stack за реализацию не блокирующих код задержек создан целый модуль — Tick.c. Он основан на 48-битном счетчике, который инкрементируется по прерыванию. Доступ к нему осуществляется функциями:

```
TICK TickGet();              // младшие 32 бита (от микросекунд до часов)
TICK TickGetDiv256();        // средние 32 бита (от минут до месяцев)
TICK TickGetDiv64K();        // старшие 32 бита (от дней до лет)
```

Алгоритм использования такой:

- получаем текущее состояние счетчика
- прибавляем к этому значению требуемую задержку
- возвращаемся к выполнению других задач, периодически сравнивая полученное значение со счетчиком.

# Модуль Tick

Для удобства (чтобы не мучиться с герцами и машинными циклами) введены временнЫе константы, которые рассчитываются при компиляции по заданной частоте осциллятора в HardwareProfile.h:

- TICK_SECOND
- TICK_MINUTE
- TICK_HOUR

# Избавляемся от длительных задержек

- Откройте проект TCP/IP Stack:
  C:\MASTERS\COM4201\Lab4\COM4201_Lab4-C30.mcp
- В нем откройте файл VendingMachine.c
- Найдите в коде состояние case SM_DISPLAY_WAIT:
- Замените циклическую задержку на неблокирующую:

```
169             break;
170
171         case SM_DISPLAY_WAIT:
172             // Wait for the timout to occur before continuing
173             //for(displayTimeout *= 1000; displayTimeout > 0; displayT
174             //  DelayMs(1);
175             if((LONG)(TickGet() - displayTimeout) > (LONG)0)
176                 smVend = SM_SHOW_MENU;
177             break;
178
179         case SM_SHOW_MENU:
```

Преобразование в LONG (из UNSIGNED LONG) осуществляется, чтобы избежать вероятной ошибки переполнения счетчика.

# Избавляемся от длительных задержек

• Найдите определение переменной displayTimeout поменяйте ее тип на Tick:

```
85      Section:
86        Vending Machine Implementation
87      ******************************************************************************
88      void VendingMachine(void)
89      {
90          static TICK displayTimeout = 2 * TICK_SECOND;
91
92          // Vending Machine State Machine
93          switch(smVend)
94          {
```

• Найдите все присвоения этой переменной и переделайте их в соответствии с идеологией модуля Tick.

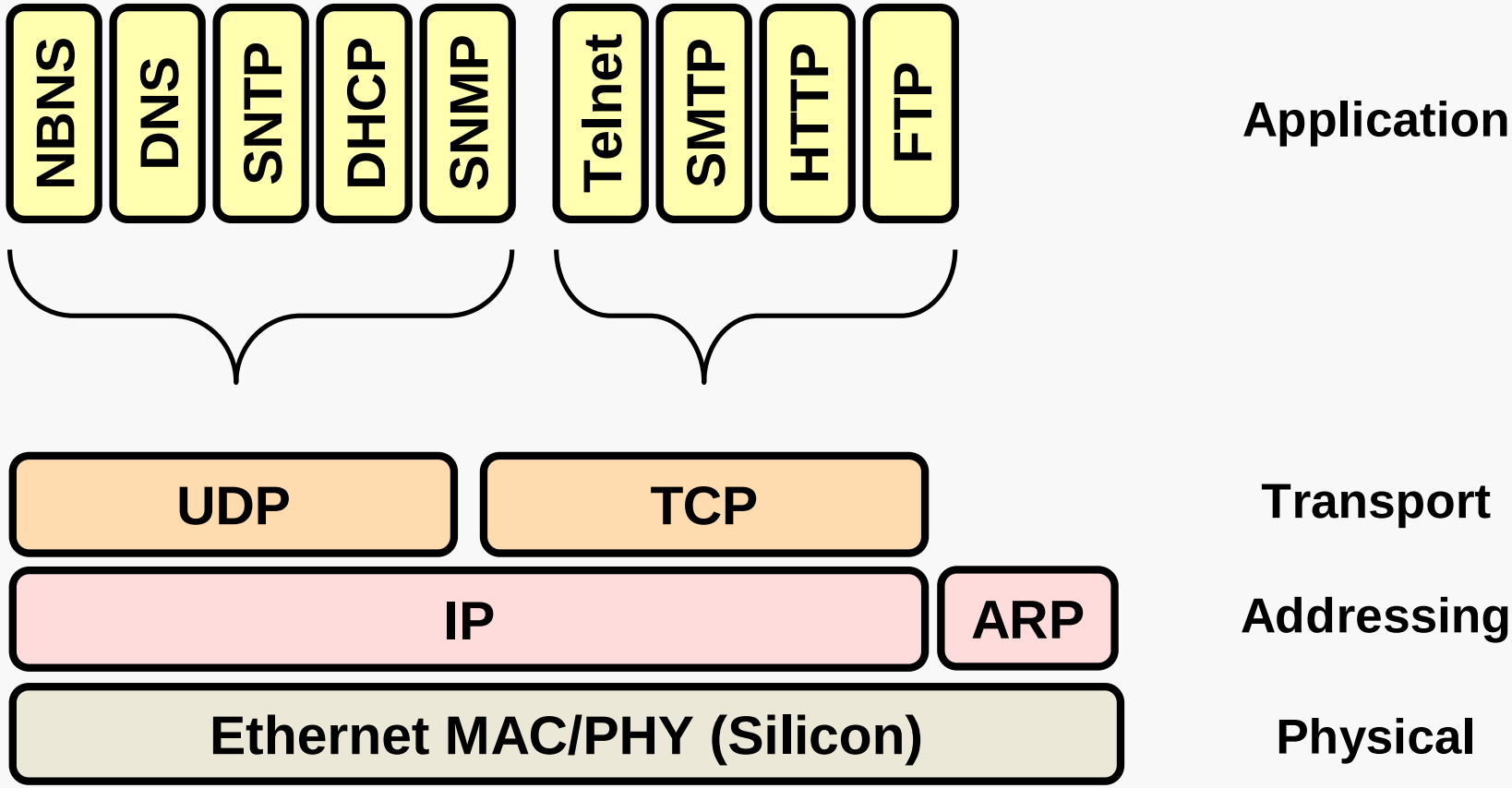
• Соберите, прошейте и запустите проект.

• Повторите эксперимент и убедитесь, что диод продолжает мигать при любых сообщениях на индикаторе.

# Шаг 5

- Основы стандарта IEEE 802.3
- Динамические переменные

# «Слои» стандарта IEEE 802.3



# Транспортный слой

## UDP

Быстрый

Недостоверный

Передача сообщений (датаграмм) без установки соединения

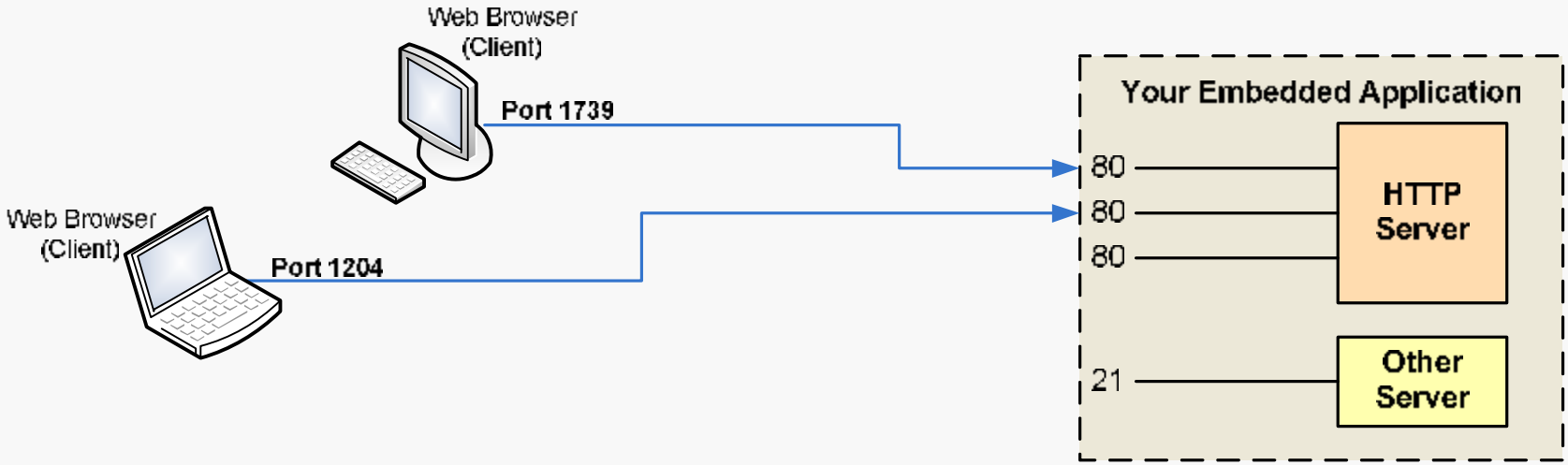
## TCP

Медленнее

Достоверный

Передача потока данных по установленному соединению

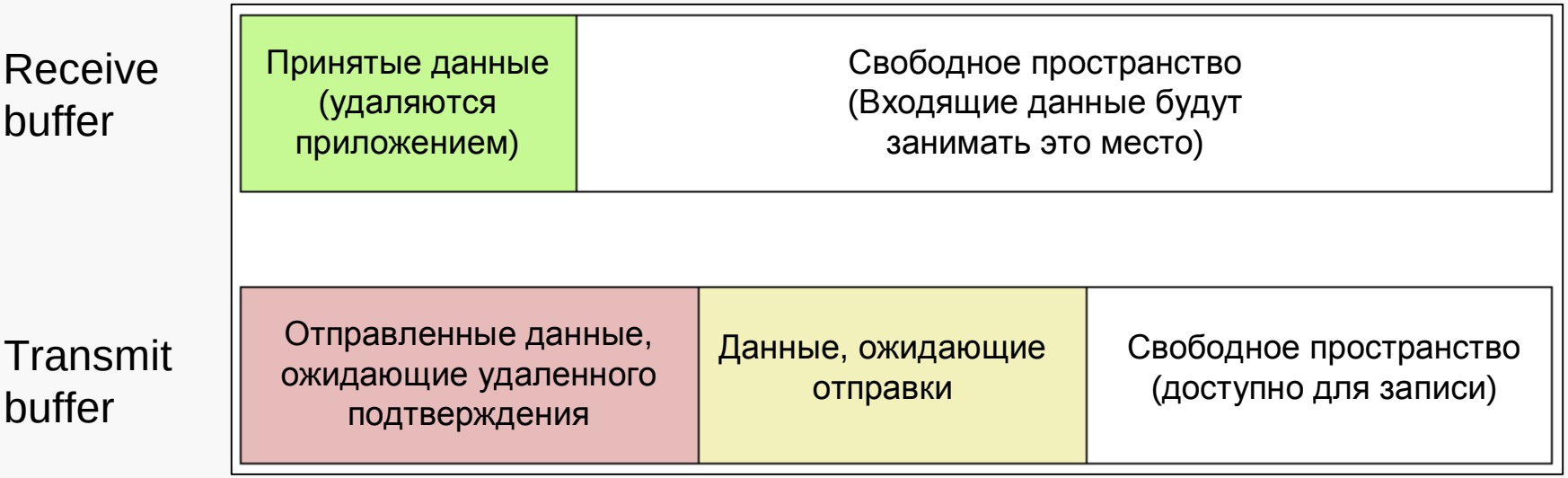
# Основные понятия TCP (на примере HTTP)




- Сервер: прослушивает эфир на предмет запросов
- Клиент: формирует запросы
- Порт — 16-битное число, обозначающее принадлежность данных той или иной службе (например, 80 – HTTP, 20 и 21 – FTP)
- Сокет — абстрактный объект, представляющий конечную точку соединения. Один порт может иметь несколько сокетов

# TCP буфер

• Каждый сокет буферизован




• Для каждого сокета есть хэндл (указатель на буфер):

```
TCP_SOCKET    hTCP;
```

• У HTTP-сервера несколько сокетов. Текущий доступен по хэндлу

```
TCP_SOCKET    sktTCP;
```

# Передача данных в TCP буфер

Передача осуществляется с помощью функций с вполне говорящими названиями:

• Отправка байта (возвращает 1, если буфер принял)

BOOL TCPPut(hTCP, BYTE data);

• Отправка массива из памяти программ/данных (возвращает количество принятых байтов):

WORD TCPPutArray(hTCP, data, len);

WORD TCPPutROMArray(hTCP, data, len);

• Отправка строки из памяти программ/данных (возвращает указатель на следующий байт):

BYTE* TCPPutString(hTCP, data);

BYTE* TCPPutROMString(hTCP, data);

# Чтение данных из TCP буфера

Прием реализован следующими функциями:

• Прием байта(возвращает 1, если байт прочитан):
BOOL TCPGet(hTCP,data)

•Прием массива длиной len (возвращает число прочитанных байтов):
WORD TCPGetArray(hTCP,data,len)

Принятые данные размещаются в массиве data

# Динамические переменные

• В папке Lab5\WebPages находится заготовка страницы для интерфейса автомата продажи напитков. Наша задача сделать ее динамической. Откройте файл index.htm с помощью Dreamweaver'a или блокнота.

• Замените статический текст HOSTNAME на динамическую переменную ~hostname~

• Аналогично замените Machine Location / Description на ~machineDesc~

```
3  <b>Status</b>  | 
4  <a href="lights.htm">Lights</a>  | 
5  <a href="products.htm">Products</a>
6  </p>
7
8  <div id="location">Machine ~hostname~ :: ~machineDesc~ </div>
9
10 <div id="bargraph">
11
12     <div class="productname">~name(0)~</div>
```

# Динамические переменные

- Сохраните страницу, скомпилируйте ее вместе с остальными файлами в папке Lab5\WebPages при помощи утилиты MPFS2.exe. Загрузите образ страниц в EEPROM.
- В данной работе мы создали 2 новые динамические переменные (~hostname~ и ~machineDesc~), поэтому необходимо убедиться, к проекту подключен свежесгенерированный файл HTTPprint.h

# Динамические переменные

• Откройте проект TCP/IP Stack:
C:\MASTERS\COM4201\Lab5\COM4201_Lab5-C30.mcp

• В нем откройте файл CustomHTTPApp.c. Как мы увидим далее, в этом файле описываются все переменные и события, связанные с веб-страницами.

• Для начала удалим функции, начинающиеся на HTTPPrint, кроме двух:
HTTPPrint_version();
HTTPPrint_builddate();

Эти функции — т.н. Callback-функции — основа связи между страницей и программой микроконтроллера. Если модуль HTML2 выдает клиенту по запросу файл и натыкается в нем на строку вида ~variable~, то он передает управление функции HTTPPrint_variable, позволяя отправить клиенту любые данные.

# Динамические переменные

- Добавьте Callback-функции для двух наших новых переменных.

- Так как ~machineDesc~ будет выводить данные из модуля VendingMachine, для доступа к ним нужно подключить хэдер VendingMachine.h:

```
54 #define __CUSTOMHTTPAPP_C
55
56 #include "TCPIP Stack/TCPIP.h"
57 #include "VendingMachine.h"
58
```

```
1335     TCPPutROMString(sktHTTP, (ROM void*)VERSION);
1336 }
1337
1338 void HTTPPrint_hostname(void)
1339 {
1340     TCPPutString(sktHTTP, AppConfig.NetBIOSName);
1341 }
1342
1343 void HTTPPrint_machineDesc(void)
1344 {
1345     if(TCPIsPutReady(sktHTTP) < 32)
1346     {
1347         curHTTP.callbackPos = 0x01;
1348         return;
1349     }
1350
1351     curHTTP.callbackPos = 0x00;
1352     TCPPutString(sktHTTP, machineDesc);
1353 }
```

# Динамические переменные

При попадании в Callback-функцию, мы гарантированно имеем 16 байт свободного пространства в исходящем TCP-буфере. Если же нам требуется передать больший объем информации, можно воспользоваться переменной curHTTP.callbackPos

Ее основное свойство: если при выходе из callback-функции эта переменная имеет ненулевое значение, то эта же callback-функция будет вызвана еще раз (с большим свободным пространством буфера).

Переменную curHTTP.callbackPos можно использовать как счетчик байтов, оставшихся для отправки, или как флаг, как в примере выше.

# Динамические переменные

• Соберите, прошейте и запустите проект.

• Зайдите на нашу страницу. Убедитесь, что в том месте, где были ~hostname~ и ~machineDesc~ сейчас находятся значения внутренних переменных NetBIOSName и machineDesc.

• Осталось сделать динамическими названия и количества продуктов автомата продажи напитков и заодно познакомиться с динамическим массивом.
Снова откройте файл index.htm для редактирования.

# Динамические переменные

• Замените в файле index.htm все слова Product на динамический массив ~name(1)~, ~name(2)~ ... ~name(6)~

• Аналогично замените числа, показывающие количество каждого напитка в аппарате, на массивы ~stock(1)~…~stock(6)~

• Замените классы “ok” и “low” на ~status(1)~…~status(6)~

```
 8 <div id="location">Machine ~hostname~ :: ~machineDesc~ </div>
 9
10 <div id="bargraph">
11
12     <div class="productname">~name(0)~</div>
13     <div class="bar" style="width: ~stock(0)~em">
14       <div class="~status(0)~">~stock(0)~</div>
15     </div>
16
17     <div class="productname">~name(1)~</div>
18     <div class="bar" style="width: ~stock(1)~em">
19       <div class="~status(1)~">~stock(1)~</div>
20     </div>
21
22     <div class="productname">~name(2)~</div>
```

# Динамические переменные

• Сохраняем index.htm, компилируем образ страниц и заливаем его в EEPROM.

• Возвращаемся в наш проект, к файлу CustomHTTPApp.c и добавляем в него новые callback-функции HTTPPrint_name(WORD item) и HTTPPrint_stock(WORD item)

Теперь у callback-функций появились параметры. В локальной переменной item в наши функции будет передаваться номер продукта.

```
1350
1351    curHTTP.callbackPos = 0x00;
1352    TCPPutString(sktHTTP, machineDesc);
1353}
1354
1355void HTTPPrint_name(WORD item)
1356{
1357    TCPPutString(sktHTTP, Products[item].name);
1358}
1359
1360void HTTPPrint_stock(WORD item)
1361{
1362    BYTE buf[4];
1363
1364    uitoa(Products[item].stock, buf);
1365    TCPPutString(sktHTTP, buf);
```

# Динамические переменные

• Опишите еще одну callback-функцию — HTTPPrint_status(WORD item)
Эта функция должна отправлять в буфер строку “ok”, если напитка под номером item осталось больше 8 штук, и “low” — если меньше.

```
1367
1368 void HTTPPrint_status(WORD item)
1369 {
1370     if(Products[item].stock < 8)
1371         TCPPutROMString(sktHTTP, (ROM BYTE*)"low");
1372     else
1373         TCPPutROMString(sktHTTP, (ROM BYTE*)"ok");
1374 }
1375
1376 #endif
```

• Соберите, прошейте и запустите проект.

• Зайдите на страницу через браузер.

• «Купите» несколько напитков у платы Explorer16 и, обновив страницу (Refresh в браузере), посмотрите, изменились ли показатели на странице.

# Шаг 6

- Методы POST и GET
- Использование метода GET для управления светодиодами

# Методы POST и GET

## POST

Данные помещаются в тело запроса

Нет ограничений на объем посылаемых данных

Сложность обработки

## GET

Данные добавляются к URL

Ограничение на объем данных: ~100 байт

Простота обработки

# Использование метода GET для управления светодиодами

• Откройте в HTML-редакторе файл lights.htm:
C:\MASTERS\COM4201\Lab6\WebPages2\

• Обратите внимание на тег формы:
<form method="get" action="lights.htm">

Данная строка означает, что будет сформирован запрос GET по файлу lights.htm с параметрами, определяемыми в теле тега.

Следующими строками, несущими функциональную нагрузку, будут
<input type="radio" name="lights" value="1" /> On
<input type="radio" name="lights" value="0" /> Off

Эти тэги создают 2 радио-кнопки. lights — это и есть имя параметра, которое будет добавлено к URL при формировании запроса GET. Значением же этого параметра будет 1 или 0, в зависимости от кнопки, которая будет нажата в момент submit'а

# Использование метода GET для управления светодиодами

```
<input type="submit" class="btn" value="Set"/>
```

Завершается форма сбора параметров кнопкой типа submit. При ее нажатии формируется запрос GET в форме

GET /lights.htm?lights=1

при условии, что была нажата кнопка 1. Именно в такой форме запрос поступает в контроллер.

Для упрощения обработки такого запроса существуют функции:

```
BYTE*    HTTPGetArg(BYTE* haystack, BYTE* needle)
BYTE*    HTTPGetROMArg(BYTE * haystack, ROM BYTE* needle)
```

где haystack – указатель на область поиска параметров (curHTTP.data)

needle – указатель на строку, содержащую имя параметра (у нас – lights)

Функции возвращают указатель на значение искомого параметра (в нашем случае “1” или “0” – в символьном представлении)

# Использование метода GET для управления светодиодами

• Воплотим все вышесказанное в проекте:
C:\MASTERS\COM4201\Lab6\COM4201_Lab6-C30.mcp

• В файле CustomHTTPApp.c найдем функцию HTTPExecuteGet. Уберем из нее все лишнее, описав только обработку нашего метода GET:

```
169   Function:
170     HTTP_IO_RESULT HTTPExecuteGet(void)
171
172   Internal:
173     See documentation in the TCP/IP Stack API or HTTP2.h for details.
174   ***************************************************************************/
175 HTTP_IO_RESULT HTTPExecuteGet(void)
176 {
177     BYTE *ptr;
178     BYTE filename[20];
179
180     // Load the file name
181     // Make sure BYTE filename[] above is large enough for your longest name
182     MPFSGetFilename(curHTTP.file, filename, 20);
183
184     // If its the lights.htm page
185     if(!memcmppgm2ram(filename, "lights.htm", 10))
186     {
187         // Find the lights parameter
188         ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"lights");
189         if(ptr)
190             LED4_IO = (*ptr == '1');
191     }
192
193     return HTTP_IO_DONE;
194 }
```

# Использование метода GET для управления светодиодами

- Соберите, прошейте и запустите проект.
- При помощи утилиты MPFS2.exe залейте страницы в EEPROM
- Зайдите на страницу ligths.htm и убедитесь, что можете контролировать состояние светодиода при помощи веб-формы.

Поздравляем, теперь вы знаете, как осуществлять обоюдный обмен информацией между веб-страницей и контроллером!

# Шаг 7

- Использование метода POST для отправки больших объемов данных

# Использование метода POST для отправки больших объемов данных

• Откройте в HTML-редакторе файл lights.htm:
C:\MASTERS\COM4202\TCP4-LAB1\WebPages2

Структура формы, осуществляющая запрос POST, не отличается от рассмотренной выше (с запросом GET):

```
<form action="index.htm" method="post">
        <h2>Lab 1</h2>
        <b>Enter your nickname and a brief fact about yourself below:</b><br><br>
        <input type="text" name="nickname" value="Nickname"><br><br>
        <input type="text" name="factoid" value="Enter a Factoid"><br><br>
        <input type="submit" value="Submit">
</form>
```

Только теперь method="post"

# Использование метода POST для отправки больших объемов данных

- Откройте проект TCPIP Demo App-C30.mcp
  C:\MASTERS\COM4202\TCP4-LAB1\

- Найдите в файле CustomHTTPApp.c функцию HTTPExecutePost

В данной функции уже присутствует основные этапы обработки запроса POST.

Так как полученные данные могут находиться в нескольких пакетах и могут даже превышать размер входного буфера, функция HTTPExecutePost вызывается несколько раз и управляться с помощью переменной состояния curHTTP.smPost

У нее 4 состояния:

```
#define SM_POST_INDEX_NAME          (0u)
#define SM_POST_INDEX_NAME_VALUE    (1u)
#define SM_POST_INDEX_FACT          (2u)
#define SM_POST_INDEX_FACT_VALUE    (3u)
```