# Разработка приложений на STM32Cube со стеком LwIP TCP / IP

## Вступление

STMCube ™ – это оригинальная инициатива STMicroelectronics, призванная облегчить жизнь разработчиков за счет сокращения усилий, времени и затрат на разработку. STM32Cube покрывает портфель STM32.

Версия 1.x STM32Cube включает в себя:

• STM32CubeMX, графический инструмент конфигурации программного обеспечения, который позволяет генерировать код инициализации C с использованием графических мастеров.

• Комплексная встроенная программная платформа, поставляемая по сериям (например, STM32CubeF4 для серии STM32F4)

– STM32Cube HAL, встроенное программное обеспечение уровня абстракции STM32, обеспечивающее максимальную переносимость по всему портфелю STM32

– Постоянный набор компонентов промежуточного программного обеспечения, таких как RTOS, USB, TCP / IP, графика

– Все встроенные программные утилиты поставляются с полным набором примеров.

Некоторые микроконтроллеры STM32 оснащены высококачественной периферией Ethernet 10/100 Мбит / с, которая поддерживает как независимый от среды интерфейс (MII), так и независимый от среды уменьшенный интерфейс (RMII) для взаимодействия с физическим уровнем (PHY).

При работе с коммуникационным интерфейсом Ethernet стек TCP / IP чаще всего используется для связи по локальной или глобальной сети.

Данное руководство пользователя предназначено для разработчиков, которые используют микропрограмму STM32Cube на микроконтроллерах STM32. Он содержит полное описание того, как интегрировать бесплатный промежуточный стек TCP / IP с помощью драйверов STM32Cube HAL во встроенное приложение на основе микроконтроллера STM32.

Промежуточным стеком TCP / IP является LwIP (Lightweight IP), который является стеком с открытым исходным кодом, предназначенным для встроенных устройств.

Для каждой серии предусмотрен специальный пакет прошивки STM32Cube. Он включает в себя драйвер Ethernet HAL, промежуточное ПО LwIP и примеры приложений с ОС RTOS, работающей на платах оценки ST, и без нее.

*Примечание. Этот документ применим ко всем сериям STM32 с периферийным устройством Ethernet. Однако или по причине простоты STM32F4xx и STM32CubeF4 используются в качестве эталонной платформы. То же описание, имена файлов и снимок экрана применимы также к другим сериям, предлагающим возможность подключения Ethernet, таким как STM32F107xx, STM32F2x7xx и STM32F7xx. Чтобы узнать больше о реализации примеров Ethernet на вашей серии STM32, обратитесь к документации, предоставленной в соответствующем пакете прошивки STM32Cube.*

# 1 Описание стека LwIP TCP / IP

## 1.1 Особенности стека

LwIP – это бесплатный стек TCP/IP, разработанный Адамом Данкелсом из Шведского института компьютерных наук (SICS) и лицензированный по модифицированной лицензии BSD.

Целью реализации LwIP TCP/IP является сокращение использования оперативной памяти при сохранении полноразмерного стека TCP/IP. Это делает LwIP подходящим для использования во встроенных системах.

LwIP поставляется со следующими протоколами:

- IPv4 и IPv6 (интернет-протокол v4 и v6)
- ICMP (Internet Control Message Protocol) для обслуживания и отладки сети
- IGMP (протокол управления группами Интернета) для управления многоадресным трафиком
- UDP (протокол дейтаграмм пользователя)
- TCP (протокол управления передачей)
- DNS (сервер доменных имен)
- SNMP (простой протокол управления сетью)
- DHCP (протокол динамической конфигурации хоста)
- PPP (протокол точка-точка)
- ARP (протокол разрешения адресов)

LwIP имеет три интерфейса прикладного программирования (API):

- Raw API – это родной LwIP API. Это позволяет разрабатывать приложения, используя обратные вызовы событий. Этот API обеспечивает лучшую производительность и оптимизированный размер кода, но добавляет сложности при разработке приложений.
- Netconn API – это высокоуровневый последовательный API, для которого требуется операционная система реального времени (ОСРВ). Netconn API позволяет выполнять многопоточные операции.
- BSD Socket API: Berkeley-подобный Socket API (разработанный поверх Netconn API)

Исходный код для стека LwIP можно скачать с http://savannah.nongnu.org.

## 1.2 Лицензия

LwIP лицензируется по лицензии BSD. Ниже приведена копия лицензионного документа LwIP, включенного в исходные коды...

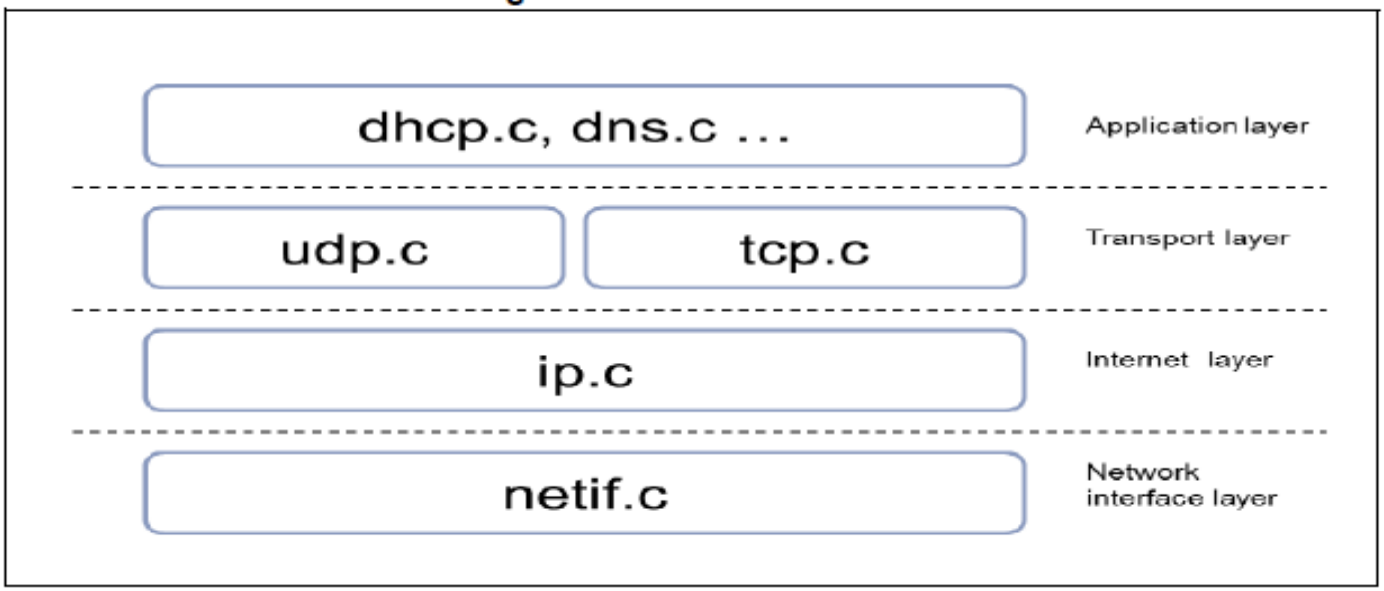
### 1.3 LwIP архитектура

LwIP соответствует архитектуре модели TCP / IP, которая определяет, как данные должны форматироваться, передаваться, маршрутизироваться и приниматься для обеспечения сквозной связи.

Эта модель включает в себя четыре уровня абстракции, которые используются для сортировки всех связанных протоколов в соответствии с объемом используемой сети (см. Рисунок 1). От низшего к высшему слои:

- Канальный уровень содержит коммуникационные технологии для одного сегмента сети (канала) локальной сети.
- Интернет-уровень (IP) соединяет независимые сети, тем самым устанавливая межсетевое взаимодействие.
- Транспортный уровень обрабатывает обмен данными между хостами.
- Прикладной уровень содержит все протоколы для конкретных служб передачи данных на межпроцессном уровне.

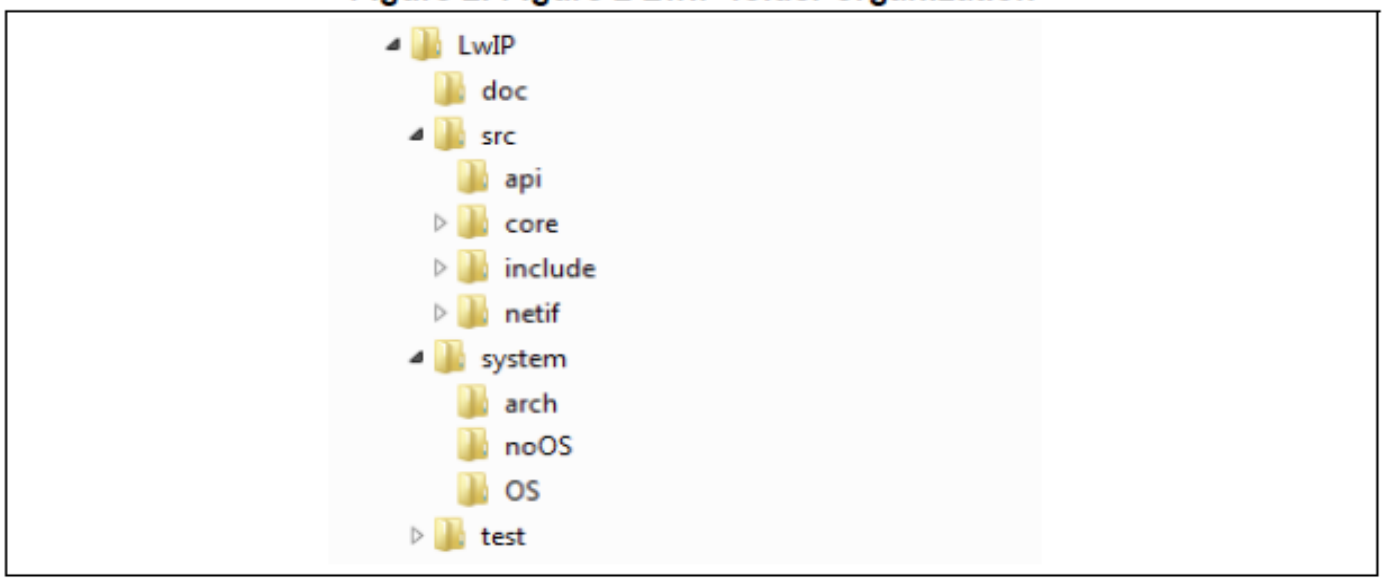
**Figure 1. LwIP architecture**




### 1.4 LwIP организация папок стека

Разархивированные файлы стека LwIP можно найти в папке \ Middlewares \ Third_Party \ LwIP.

**Figure 2. Figure 2 LwIP folder organization**



где

**doc** содержит текстовые файлы документации

**src** содержит исходные файлы стека LwIP

**api** содержит файлы API-интерфейсов Netconn и Socket

**core** содержит файлы ядра LwIP

**include** содержит файлы включения LwIP

**netif** содержит файлы сетевого интерфейса

**system** содержит файлы реализации аппаратного обеспечения порта LwIP

**arch** содержит файлы портов архитектуры STM32 (используемые типы данных, ...)

**OS** содержит файлы реализации аппаратного обеспечения порта LwIP с использованием операционной системы

**noOS** содержит файлы аппаратной реализации порта LwIP в автономном режиме

### 1.5 Обзор LwIP API

Как упомянуто выше, стек LwIP предлагает три типа API:

- Raw API
- Netconn API
- Socket API

#### 1.5.1 Raw API

Raw API основан на собственном LwIP API. Он используется для разработки приложений на основе обратного вызова.

При инициализации приложения пользователь должен зарегистрировать функции обратного вызова для различных основных событий (таких как TCP_ Sent, TCP_error, ...). Функции обратного вызова вызываются из основного уровня LwIP, когда происходит соответствующее событие.

В таблице 1 представлена сводка функций Raw API для приложений TCP.

*Таблица 1. Функции TCP Raw API*

| API функции | | Описание |
|---|---|---|
| TCP connection setup<br><br>Настройка TCP-соединения | tcp_new | Создает новый TCP PCB (блок управления протоколом). |
| | tcp_bind | Связывает TCP PCB с локальным IP-адресом и портом. |
| | tcp_listen | Запускает процесс прослушивания TCP PCB |
| | tcp_accept | Назначает функцию обратного вызова, которая будет вызываться при получении нового соединения TCP. |
| | tcp_ accepted | Сообщает стеку LwIP, что входящее TCP-соединение принято. |
| | tcp_connect | Подключается к удаленному TCP-хосту. |

| API функции | | Описание |
|---|---|---|
| Sending TCP data<br><br>Отправка данных TCP | tcp_write | Очередь данных для отправки. |
| | tcp_sent | Назначает функцию обратного вызова, которая будет вызываться при подтверждении данных удаленным хостом. |
| | tcp_output | Принудительно отправляет данные в очереди. |
| Receiving TCP<br><br>Прием TCP | tcp_recv | Устанавливает функцию обратного вызова, которая будет вызываться при поступлении новых данных. |
| | tcp_recved | Должна вызываться, когда приложение обработало входящий пакет данных (для управления окном TCP). |
| Application polling<br><br>Опрос приложений | tcp_poll | Назначает функции обратного вызова, которые будут вызываться периодически. Они могут использоваться приложением для проверки того, имеются ли оставшиеся данные приложения, которые необходимо отправить, или существуют ли соединения, которые необходимо закрыть. |
| Closing and aborting connections<br><br>Закрытие и обрыв соединения | tcp_close | Закрывает TCP-соединение с удаленным хостом. |
| | tcp_err | Назначает функцию обратного вызова для обработки соединений, прерванных LwIP из-за ошибок (таких как ошибки нехватки памяти). |
| | tcp_abort | Прерывает TCP-соединение. |

В таблице 2 приведена сводная информация о функциях Raw API для приложений UDP.

*Таблица 2. Функции UDP Raw API*

| API функции | Описание |
|---|---|
| udp_new | Создает новый UDP PCB (блок управления протоколом). |
| udp_remove | Удаляет и освобождает UDP PCB. |
| udp_bind | Связывает UDP PCB с локальным IP-адресом и портом. |
| udp_connect | Устанавливает удаленный IP-адрес UDP PCB и порт. |
| udp_disconnect | Удаляет удаленный IP-адрес UDP PCB и порт. |
| udp_send | Отправляет данные UDP. |
| udp_recv | Определяет функцию обратного вызова, которая вызывается при получении дейтаграммы*. |

** блок информации, передаваемый протоколом через сеть связи без предварительного установления соединения и создания виртуального канала.*

### 1.5.2 Netconn API

Netconn API – это высокоуровневый последовательный API, модель исполнения которого основана на блокирующей парадигме open-read-write-close.

Для правильной работы этот API должен работать в многопоточном режиме, реализуя выделенный поток для стека LwIP TCP / IP и / или несколько потоков для приложения.

В таблице 3 приведены сводные данные по функциям Netconn API.

*Таблица 3. Функции Netconn API*

| API функции | Описание |
|---|---|
| netconn_new | Создает новое соединение. |
| netconn_delete | Удаляет существующее соединение. |
| netconn_bind | Связывает соединение с локальным IP-адресом и портом. |
| netconn_connect | Подключается к удаленному IP-адресу и порту. |
| netconn_send | Отправляет данные на подключенный в данный момент удаленный IP-порт / порт (не применяется для соединений TCP). |
| netconn_recv | Получает данные из netconn. |
| netconn_listen | Устанавливает соединение TCP в режим прослушивания. |
| netconn_accept | Принимает входящее соединение через прослушивающее TCP-соединение. |
| netconn_write | Посылает данные по подключенной TCP-сети. |
| netconn_close | Закрывает TCP-соединение, не удаляя его. |

### 1.5.3 Socket API

LwIP предлагает стандартный API сокетов BSD. Это последовательный API, который внутренне построен поверх API Netconn.

В таблице 4 приведена сводка основных функций API сокетов.

*Таблица 4. Функции Socket API*

| API функции | Описание |
|---|---|
| socket | Создает новый сокет. |
| bind | Связывает сокет с IP-адресом и портом. |
| listen | Слушает сокетное соединение. |
| connect | Подключает сокет к IP-адресу и порту удаленного хоста. |
| accept | Принимает новое соединение на сокете. |
| read | Читает данные из сокета. |
| write | Записывает данные в сокет. |
| close | Закрывает сокет (сокет удаляется). |

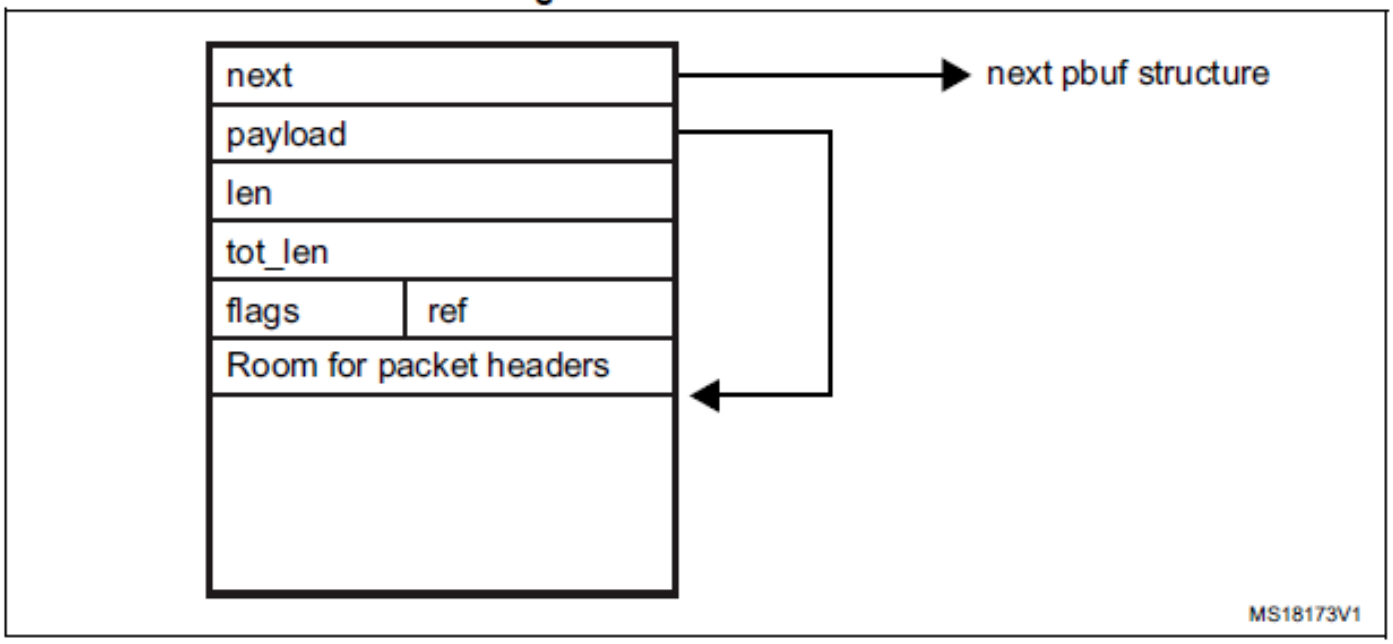
## 1.6 Управление буфером LwIP

### 1.6.1 Структура буфера пакетов

LwIP управляет буферами пакетов, используя структуру данных, называемую pbuf. Структура pbuf позволяет выделить динамическую память для хранения содержимого пакета и позволяет пакетам находиться в статической памяти.

Pbufs могут быть связаны вместе в цепочку, что позволяет пакетам распределяться по нескольким pbufs.

**Figure 3. Pbuf structure**




где

**next** содержит указатель на следующий pbuf в цепочке pbuf

**payload** полезная нагрузка содержит указатель на полезную нагрузку пакетных данных

**len** длина содержимого данных pbuf

**tot_len** сумма pbuf len плюс все поля len следующих pbufs в цепочке

**ref** 4-битный счетчик ссылок, который указывает количество указателей, указывающих на pbuf. Pbuf может быть освобожден из памяти, только когда его счетчик ссылок равен нулю.

**flags** флаги (на 4 бита) указывают тип pbuf.

LwIP определяет три типа pbuf в зависимости от типа выделения:

• PBUF_POOL

Выделение pbuf выполняется из пула статически предварительно распределенных pbuf предопределенного размера. В зависимости от размера данных, который должен быть выделен, требуется один или несколько связанных pbuf.

• PBUF_RAM

pbuf динамически выделяется в памяти (один непрерывный кусок памяти для полного pbuf)

• PBUF_ROM

Для полезной нагрузки пользователя не требуется выделять пространство памяти: указатель полезной нагрузки pbuf указывает на данные в ПЗУ, которые можно использовать только для отправки постоянных данных.

Для приема пакетов подходящим типом pbuf является PBUF_POOL. Это позволяет быстро выделить память для пакета, полученного из пула pbufs. В зависимости от размера принятого пакета выделяется одна или несколько цепочек pbufs. PBUF_RAM не подходит для приема пакетов, потому что динамическое распределение требует некоторой задержки. Это также может привести к фрагментации памяти.

Для передачи пакета пользователь может выбрать наиболее подходящий тип pbuf в соответствии с данными, которые должны быть переданы.

### 1.6.2 API управления pbuf

LwIP имеет специальный API для работы с pbufs. Этот API реализован в основном файле pbuf.c.

*Таблица 5. Функции API Pbuf*

| API функции | Описание |
|---|---|
| pbuf_alloc | Выделяет новый pbuf. |
| pbuf_realloc | Изменяет размер pbuf (только размер сжатия). |
| pbuf_ref | Увеличивает поле счетчика ссылок pbuf. |
| pbuf_free | Уменьшает количество ссылок на pbuf. Если он достигает нуля, pbuf освобождается. |
| pbuf_clen | Возвращает количество pbuf в цепочке pbuf. |
| pbuf_cat | Объединяет в цепочку два pbuf (но не изменяет счетчик ссылок цепочки pbuf в конце). |
| pbuf_chain | Цепочки двух pbufs вместе (счетчик ссылок хвостовой цепи увеличивается). |
| pbuf_dechain | Освобождает первый pbuf от его последующих pbuf в цепочке. |
| pbuf_header | Настраивает указатель полезной нагрузки, чтобы скрыть или показать заголовки в полезной нагрузке. |
| pbuf_copy_partial | Копирует (частично) содержимое буфера пакетов в предоставленный приложением буфер. |
| pbuf_take | Копирует предоставленные приложением данные в pbuf. |
| pbuf_coalesce | Создает один pbuf из очереди pbufs. |
| pbuf_memcmp | Сравните содержимое pbuf с заданным смещением с другой памятью |
| pbuf_memfind | Найти вхождение памяти в pbuf, начиная со смещения |
| pbuf_strstr | Найти вхождение строки в pbuf, начиная со смещения |

***Примечание:*** *pbuf может быть одним pbuf или цепочкой pbuf.*

*При работе с Netconn API для отправки / получения данных используются netbuf (сетевые буферы).*

*Netbuf – это просто оболочка для структуры pbuf. Он может вместить как распределенные, так и ссылочные данные.*

*Выделенный API (реализован в файле netbuf.c) предназначен для управления сетевыми буферами (выделения, освобождения, объединения в цепочку, извлечения данных,...).*

## 2 Сопряжение LwIP с драйвером STM32Cube Ethernet HAL

Этот пакет включает в себя две реализации:

- Внедрение без операционной системы (автономно)
- Внедрение с операционной системой с использованием CMSIS-RTOS API

Для обеих реализаций файл ethernetif.c используется для связи стека LwIP с сетевым интерфейсом Ethernet STM32.

Порт стека LwIP, который должен быть подключен к STM32F4xx, находится в папке «lwip/system».

Дескриптор Ethernet HAL (ETH_HandleTypeDef) должен быть объявлен в файле ethernetif.c, а также дескрипторах DMA Ethernet (ETH_DMADescTypeDef) и буферах Rx/Tx драйвера Ethernet.

В таблице 6 приведено описание API интерфейса LwIP.

*Таблица 6. Описание функций интерфейса Ethernet*

| API функции | Описание |
|---|---|
| low_level_init | Вызывает функции драйвера Ethernet для инициализации периферийного устройства STM32F4xx Ethernet |
| low_level_output | Вызывает функции драйвера Ethernet для отправки пакета Ethernet |
| low_level_input | Вызывает функции драйвера Ethernet для получения пакета Ethernet. |
| ethernetif_init | Инициализирует структуру сетевого интерфейса (netif) и вызывает для инициализации периферийное устройство Ethernet |
| ethernet_input | Вызывает low_level_input для получения пакета, а затем предоставляет его в стек LwIP |

В следующем примере показано, как инициализировать периферийное устройство Ethernet с помощью HAL API в API интерфейса:

```
static void low_level_init(struct netif *netif)
{
uint8_t macaddress[6]= {MAC_ADDR0, MAC_ADDR1, MAC_ADDR2, MAC_ADDR3,
MAC_ADDR4, MAC_ADDR5};
EthHandle.Instance = ETH;
EthHandle.Init.MACAddr = macaddress;
EthHandle.Init.AutoNegotiation = ETH_AUTONEGOTIATION_ENABLE;
EthHandle.Init.Speed = ETH_SPEED_100M;
EthHandle.Init.DuplexMode = ETH_MODE_FULLDUPLEX;
EthHandle.Init.MediaInterface = ETH_MEDIA_INTERFACE_MII;
EthHandle.Init.RxMode = ETH_RXINTERRUPT_MODE;
EthHandle.Init.ChecksumMode = ETH_CHECKSUM_BY_HARDWARE;
EthHandle.Init.PhyAddress = DP83848_PHY_ADDRESS;
```

```
/* configure ethernet peripheral (GPIOs, clocks, MAC, DMA) */
/* настроить периферийные устройства Ethernet (GPIO, часы, MAC, DMA) */
HAL_ETH_Init(&EthHandle) ;

/* Initialize Tx Descriptors list: Chain Mode */
/* Инициализировать список дескрипторов Tx: режим цепочки*/
HAL_ETH_DMATxDescListInit(&EthHandle, DMATxDscrTab, &Tx_Buff[0][0],
ETH_TXBUFNB);

/* Initialize Rx Descriptors list: Chain Mode */
/* Инициализировать список дескрипторов Rx: режим цепочки */
HAL_ETH_DMARxDescListInit(&EthHandle, DMARxDscrTab, &Rx_Buff[0][0],
ETH_RXBUFNB);
...
/* Enable MAC and DMA transmission and reception */
/* Включить передачу и прием MAC и DMA */
HAL_ETH_Start(&EthHandle);
}
```

Реализация функции ethernet_input () отличается в автономном режиме и режиме RTOS:

- В автономных приложениях эта функция должна быть вставлена в основной цикл приложения для опроса любого принятого пакета.
- В приложениях RTOS эта функция реализована в виде потока, ожидающего семафор для обработки полученного пакета. Семафор дается, когда периферийное устройство Ethernet генерирует прерывание для принятого пакета.

Файл ethernetif.c также реализует процедуры MSP для периферийных устройств Ethernet для инициализации низкого уровня (GPIO, CLK...) и прерывает обратные вызовы.

В случае реализации RTOS используется дополнительный файл (sys_arch.c). Этот файл реализует уровень эмуляции для служб RTOS (передача сообщений через почтовый ящик RTOS, семафоры и т. д.). Этот файл должен быть настроен в соответствии с текущей RTOS, то есть FreeRTOS для этого пакета.

## 3 Конфигурация LwIP

LwIP предоставляет файл с именем lwipopts.h, который позволяет пользователю полностью настроить стек и все его модули. Пользователю не нужно определять все параметры LwIP: если параметр не определен, используется значение по умолчанию, определенное в файле opt.h. Следовательно, lwipopts.h предоставляет способ переопределить большую часть поведения lwIP.

### 3.1 Поддержка модулей

Пользователь может выбрать модули, которые ему нужны для его приложения, так что размер кода будет оптимизирован путем компиляции только выбранных функций.

Например, чтобы отключить UDP и включить DHCP, в файле lwipopts.h должен быть реализован следующий код:

```
/* Disable UDP */
#define LWIP_UDP 0
/* Enable DHCP */
#define LWIP_DHCP 1
```

### 3.2 Конфигурация памяти

LwIP предоставляет гибкий способ управления размерами пула памяти и организацией.

Он резервирует область статической памяти фиксированного размера в сегменте данных. Он подразделяется на различные пулы, которые lwIP использует для различных структур данных. Например, есть пул для структуры tcp_pcb и другой пул для структуры udp_pcb. Каждый пул может быть сконфигурирован для хранения фиксированного количества структур данных. Этот номер можно изменить в файле lwipopts.h. Например, MEMP_NUM_TCP_PCB и MEMP_NUM_UDP_PCB определяют максимальное количество структур tcp_pcb и udb_pcb, которые могут быть активными в системе в данный момент времени.

Пользовательские параметры могут быть изменены в lwipopts.h. В таблице 7 приведены основные параметры оперативной памяти.

*Таблица 7. Конфигурация памяти LwIP*

| Опция памяти LwIP | Описание |
|---|---|
| MEM_SIZE | Размер памяти кучи LwIP: используется для всех распределений динамической памяти LwIP. |
| MEMP_NUM_PBUF | Общее количество pbufs MEM_REF и MEM_ROM. |
| MEMP_NUM_UDP_PCB | Общее количество структур UDP PCB. |
| MEMP_NUM_TCP_PCB | Общее количество структур TCP PCB. |
| MEMP_NUM_TCP_PCB_LISTEN | Общее количество опрашиваемых TCP PCB. |
| MEMP_NUM_TCP_SEG | Максимальное количество одновременно поставленных в очередь сегментов TCP. |
| PBUF_POOL_SIZE | Общее количество pbufs типа PBUF_POOL. |
| PBUF_POOL_BUFSIZE | Размер типа PBUF_POOL pbufs. |
| TCP_MSS | Максимальный размер сегмента TCP. |
| TCP_SND_BUF | Отправить буфер для подключения TCP. |
| TCP_SND_QUEUELEN | Максимальное количество pbufs в очереди передачи TCP. |
| TCP_WND | Объявленный размер окна получения TCP. |

# 4 Разработка приложений со стеком LwIP

## 4.1 Разработка в автономном режиме с использованием Raw API

### 4.1.1 Модель работы

В автономном режиме модель работы основана на непрерывном программном опросе, чтобы проверить, был ли принят пакет.

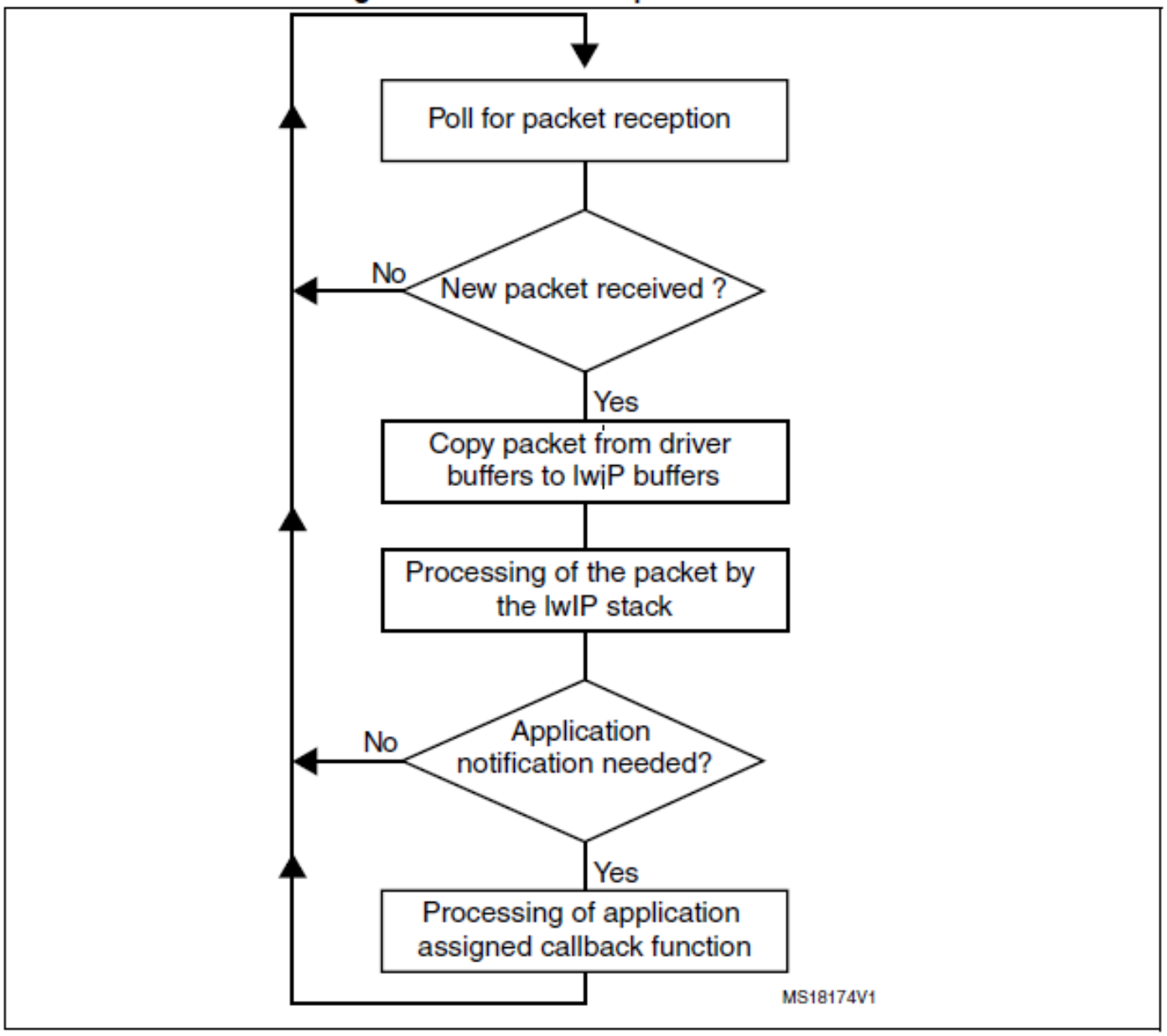
Когда пакет получен, он сначала копируется из буферов драйвера Ethernet в буферы LwIP. Чтобы скопировать пакет как можно быстрее, буферы LwIP (pbufs) должны быть выделены из пула буферов (PBUF_POOL).

Когда пакет был скопирован, он передается в стек LwIP для обработки. В зависимости от принятого пакета, стек может или не может уведомлять прикладной уровень.

LwIP связывается с прикладным уровнем, используя функции обратного вызова события. Эти функции должны быть назначены до начала процесса связи.

Обратитесь к рисунку 4 для описания блок-схемы автономной модели работы.

**Figure 4. Standalone operation model**



Для приложений TCP должны быть назначены следующие общие функции обратного вызова:

• Обратный вызов для входящего события соединения TCP, назначенный вызовом API TCP_accept.

• Обратный вызов для входящего события пакета данных TCP, назначенный вызовом API TCP_recev

• Обратный вызов для сигнализации успешной передачи данных, назначенный вызовом API TCP_sent

• Обратный вызов для сигнализации ошибки TCP (после события прерывания TCP), назначенный вызовом API TCP_err

• Периодический обратный вызов (каждые 1 или 2 с) для опроса приложения, назначенный вызовом API TCP_poll

### 4.1.2 Пример демонстрации эхо-сервера TCP

Пример эхо-сервера TCP, представленный в папке \ LwIP \ LwIP_TCP_Echo_Server, представляет собой простое приложение, которое реализует сервер TCP, который отображает любой полученный пакет данных TCP, поступающий от удаленного клиента.

В следующем примере приведено описание структуры прошивки. Это фрагмент файла main.c.

```
int main(void)
{
/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
/* Сброс всех периферийных устройств, Инициализирует интерфейс Flash и Systick. */

HAL_Init();
...
/* Initilaize the LwIP stack      Инициализировать стек LwIP*/
lwIP_init();
/* Network interface configuration    Сконфигурировать сетевой интерфейс*/
Netif_Config();
...
/* tcp echo server Init  Инициализировать сервер эхо tcp*/
tcp_echoserver_init();
/* Infinite loop   Бесконечная петля*/
while (1)
{
/* Read a received packet from the Ethernet buffers and send it to the lwIP for handling */
/* Считать полученный пакет из буферов Ethernet и отправить его в lwIP для обработки */
ethernetif_input(&gnetif);
/* Handle LwIP timeouts        Обработать тайм-аут LwIP*/
sys_check_timeouts();
}
}
```

Вызваны следующие функции:

1. Функция HAL_Init вызывается для сброса всех периферийных устройств и инициализации интерфейса Flash и таймера Systick.

2. Функция lwIP_init вызывается для инициализации внутренних структур стека LwIP и запуска операций стека.

3. Функция Netif_config вызывается для настройки сетевого интерфейса (netif).

4. Функция tcp_echoserver_init вызывается для инициализации приложения эхо-сервера TCP.

5. Функция ethernetif_input в бесконечном цикле пока опрашивает прием пакетов. Когда пакет получен, он передается для обработки стеком.

6. sys_check_timeouts Функция LwIP вызывается для обработки определенных внутренних периодических задач LwIP (таймеры протокола, повторная передача пакетов TCP ...).

### Описание функции tcp_echoserver_init

Код функции **tcp_echoserver_init** выглядит следующим образом:

```
void tcp_echoserver_init(void)
{
/* create new tcp pcb   создать новый TCP PCB*/
tcp_echoserver_pcb = tcp_new();
if (tcp_echoserver_pcb != NULL)
{
err_t err;
/* bind echo_pcb to port 7 (ECHO protocol)*/
/* привязать echo_pcb к порту 7 (протокол ECHO)*/
err = tcp_bind(tcp_echoserver_pcb, IP_ADDR_ANY, 7);
if (err == ERR_OK)
{
    /* start tcp listening for echo_pcb          начать прослушивание tcp для echo_pcb*/
    tcp_echoserver_pcb = tcp_listen(tcp_echoserver_pcb);
    /* initialize LwIP tcp_accept callback function  */
    /* инициализировать функцию обратного вызова LwIP tcp_accept  */
    tcp_accept(tcp_echoserver_pcb, tcp_echoserver_accept);
}
else
    {
    /* deallocate the pcb         освободить psb*/
    memp_free(MEMP_TCP_PCB, tcp_echoserver_pcb);
    }
}
}
```

LwIP API вызывает tcp_new для выделения нового блока управления протокола TCP (PCB) (tcp_echoserver_pcb).

Выделенный TCP PCB привязан к локальному IP-адресу и порту с помощью функции tcp_bind.

После связывания TCP PCB вызывается функция tcp_listen для запуска процесса прослушивания TCP PCB.

Наконец, функция обратного вызова tcp_echoserver_accept должна быть назначена для обработки входящих соединений TCP PCB. Это делается с помощью функции API twp_accept LwIP.

Начиная с этого момента, TCP-сервер готов принимать любое входящее соединение от удаленных клиентов.

### Описание функции tcp_echoserver_accept

В следующем примере показано, как входящие TCP-соединения обрабатываются функцией обратного вызова пользователя tcp_echoserver_accept. Это выдержка из этой функции.

```
static err_t tcp_echoserver_accept(void *arg, struct tcp_pcb *newpcb, err_t err)
{
...
/* allocate structure es to maintain tcp connection informations */
/* выделить структуры es для поддержки информации о соединениях TCP */
es = (struct tcp_echoserver_struct *)mem_malloc(sizeof(struct tcp_echoserver_struct));
if (es != NULL)
{
es->state = ES_ACCEPTED;
es->pcb = newpcb;
es->p = NULL;
/* pass newly allocated es structure as argument to newpcb */
/* передать вновь представленную структуру es в качестве аргумента newpcb*/
tcp_arg(newpcb, es);
/* initialize lwIP tcp_recv callback function for newpcb */
/* инициализировать функцию обратного вызова lwIP tcp_recv для newpcb*/
tcp_recv(newpcb, tcp_echoserver_recv);
/* initialize lwIP tcp_err callback function for newpcb */
/* инициализировать функцию обратного вызова lwIP tcp_err для newpcb */
tcp_err(newpcb, tcp_echoserver_error);
/* initialize lwIP tcp_poll callback function for newpcb */
/* инициализировать функцию обратного вызова lwIP tcp_poll для newpcb */
tcp_poll(newpcb, tcp_echoserver_poll, 1);
ret_err = ERR_OK;
...
}
```

Вызваны следующие функции:

1. Новое TCP-соединение передается функции обратного вызова tcp_echoserver_accept через параметр newpcb.

2. Структура es используется для хранения статуса приложения. Он передается в качестве аргумента соединению «newpcb» TCP PCB через вызов tcp_arg LwIP API.

3. Функция обратного вызова для приема TCP, tcp_echoserver_recv, назначается путем вызова LwIP API tcp_recv. Этот обратный вызов обрабатывает весь трафик данных с удаленного клиента.

4. Функция обратного вызова ошибки TCP, tcp_echoserver_error, назначается путем вызова LwIP API tcp_err. Этот обратный вызов обрабатывает ошибки TCP.

5. Функция обратного вызова опроса TCP, tcp_echoserver_poll, назначается путем вызова API-интерфейса LwIP tcp_poll для обработки периодических задач приложения (таких как проверка, остаются ли данные приложения для передачи).

## 4.2 Разработка с RTOS с использованием Netconn или Socket API

### 4.2.1 Операционная модель

Модель работы при работе с RTOS имеет следующие характеристики:

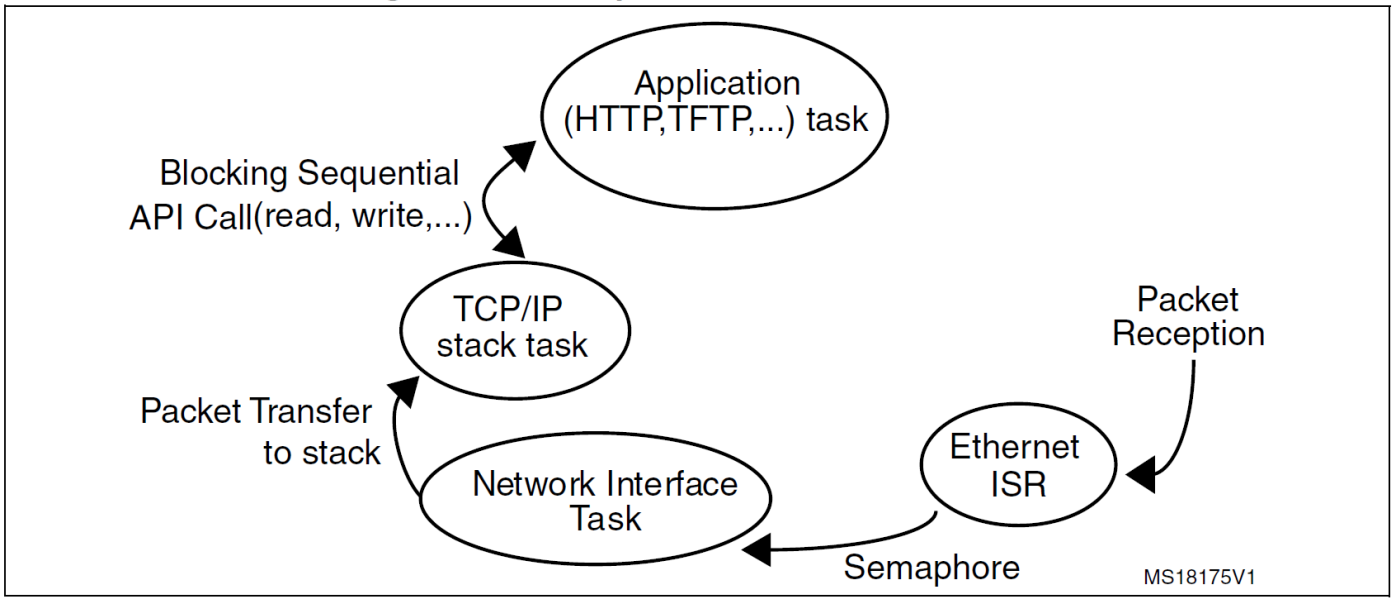
Стек TCP / IP и приложение работают в отдельных потоках.

Приложение связывается со стеком посредством последовательных вызовов API, которые используют механизм почтовых ящиков RTOS для межпроцессного взаимодействия. Вызовы API блокируют вызовы. Это означает, что поток приложения блокируется до получения ответа из стека.

Дополнительный поток, поток сетевого интерфейса, используется для получения любых полученных пакетов из буферов драйверов и их передачи в стек TCP / IP с использованием почтового ящика RTOS. Этот поток информируется о приеме пакета с использованием процедуры обслуживания прерываний приема Ethernet.

Обратитесь к рисунку 5 для описания блок-схемы модели работы lwIP с RTOS.

**Figure 5. LwIP operation model with RTOS**




### 4.2.2 Пример демонстрации эхосервера TCP с использованием API Netconn

С точки зрения приложений, API-интерфейс Netconn предлагает более простой способ, чем простой API-интерфейс для разработки приложений TCP / IP. Это потому, что он имеет более интуитивно понятный последовательный API.

В следующем примере показано приложение TCP echoserver, разработанное с помощью API Netconn. Это извлечение файла main.c.

```
int main(void)
{
...
/* Create the Start thread       Создать поток Start */
osThreadDef(Start, StartThread, osPriorityNormal, 0, configMINIMAL_STACK_SIZE * 2);
osThreadCreate (osThread(Start), NULL);
/* Start the scheduler   Запустить планировщик */
osKernelStart (NULL, NULL);
/* We should never get here as control is now taken by the scheduler */
/* Мы никогда не должны попасть сюда для управления.  Теперь это делает планировщик */
for( ;; );
}
```

Стартовый поток имеет следующий код:

```
static void StartThread(void const * argument)
{
...
/* Create tcp_ip stack thread  Создать поток стека tcp_ip */
tcpip_init( NULL, NULL );
/* Network interface configuration    Конфигурация сетевого интерфейса */
Netif_Config();
/* Initialize tcp echo server     Инициализировать эхо-сервер tcp */
tcpecho_init();
for( ;; )
{
}
}
```

Вызваны следующие функции:

1. Функция tcpip_init вызывается для инициализации модулей стека LwIP и запуска потока стека TCP / IP.
2. Функция Netif_config вызывается для настройки сетевого интерфейса (netif).
3. Поток эхо-сервера TCP создается в функции tcpecho_init.

```
void tcpecho_init(void)
{
sys_thread_new(«tcpecho_thread», tcpecho_thread, NULL,
DEFAULT_THREAD_STACKSIZE, TCPECHO_THREAD_PRIO);
}
```

### Описание функции tcpecho_thread

Поток эхо-сервера TCP имеет следующий код:

```
static void tcpecho_thread(void *arg)
{
/* Create a new connection identifier. Создайте новый идентификатор соединения. */
conn = netconn_new(NETCONN_TCP);
if (conn!=NULL)
{
/* Bind connection to well known port number 7.  Свяжите соединение с известным портом № 7.*/
err = netconn_bind(conn, NULL, 7);
if (err == ERR_OK)
{
/* Tell connection to go into listening mode. Переведите соединение в режим прослушивания.*/
netconn_listen(conn);
while (1)
{
/* Grab new connection.         Возьмите новое соединение.*/
accept_err = netconn_accept(conn, &newconn);
/* Process the new connection.        Обработайте новое соединение.*/
if (accept_err == ERR_OK)
{
while (( recv_err = netconn_recv(newconn, &buf)) == ERR_OK)
{
do
{
netbuf_data(buf, &data, &len);
netconn_write(newconn, data, len, NETCONN_COPY);
}
while (netbuf_next(buf) >= 0);
netbuf_delete(buf);
}
/* Close connection and discard connection identifier. */
/* Закройте соединение и отмените идентификатор соединения. */
netconn_close(newconn);
netconn_delete(newconn);
}
}
}
else
{
netconn_delete(newconn);
}
}
}
```

Следующая последовательность выполняется:

1. Функция API Netconn_new вызывается с параметром NETCONN_TCP, создающим новое TCP-соединение.

2. Вновь созданное соединение затем связывается с портом 7 (протокол эха), вызывая функцию API Netconn_bind.

3. После привязки соединения приложение начинает отслеживать соединение, вызывая функцию API Netconn_listen.

4. В бесконечном цикле while (1) приложение ожидает новое соединение, вызывая функцию API Netconn_accept. Этот вызов API блокирует задачу приложения, когда нет входящего соединения.

5. При наличии входящего соединения приложение может начать получать данные, вызвав функцию API netconn_recv. Входящие данные поступают в нетбуф.

6. Приложение может получить полученные данные, вызвав функцию netbuf_ data API netbuf.

7. Полученные данные отправляются обратно (отражаются) на удаленный TCP-клиент с помощью функции API Netconn_write.

8. Netconn_close и Netconn_delete используются для закрытия и удаления соединения Netconn соответственно.

# 5 Описание пакета LwIP

## 5.1 Каталоги пакетов LwIP

Пакет содержит набор приложений, работающих поверх стека LwIP и драйверов STM32Cube HAL и BSP. Прошивка состоит из следующих модулей:

• Драйверы: содержит драйверы низкого уровня микроконтроллера STM32F4xx
– CMSIS
– BSP драйверы
– драйверы HAL
• Middlewares: содержит библиотеки и компоненты протокола
– LwIP TCP / IP стек
– FatFS
– FreeRTOS
• Проекты: содержит исходный файл и конфигурации следующих приложений:
– Приложения, работающие в автономном режиме (без RTOS) на основе Raw API:
Веб-сервер
TFTP-сервер
Клиентское приложение TCP echo
Приложение эхо-сервера TCP
Клиентское приложение UDP-эхо
Приложение эхо-сервера UDP
– Приложения, работающие с операционной системой FreeRTOS:
Веб-сервер на основе Netconn API
Веб-сервер на основе API сокетов
Приложение эхо-сервера TCP / UDP на основе API-интерфейса netconn.

Приложения расположены в репозитории Projects по следующему пути: Projects \ STM324xx_EVAL \ Applications \ LwIP \, где STM324xx_EVAL относится к плате оценки STM32F4xx, такой как STM324xG_EVAL для устройств STM32F407xx / 417xx.

## 5.2 Настройки приложений

### 5.2.1 Настройка интерфейса PHY

Периферийное устройство Ethernet сопряжено с внешним физическим уровнем для обеспечения связи на физическом уровне. Определение регистров PHY и операторы определения находятся в файле конфигурации HAL stm32f4xx_hal_conf.h.

PHY может работать в режиме MII или RMII. Чтобы выбрать нужный режим, заполните параметр MediaInterface в структуре Init при инициализации периферийного устройства Ethernet.

*Примечание:*

Обратитесь к файлу readme, предоставленному в примерах Ethernet вашего устройства, чтобы узнать больше о доступных режимах интерфейса PHY на поддерживаемых платах.

### 5.2.2 Настройки MAC и IP-адреса

MAC-адрес по умолчанию установлен на 00: 00: 00: 00: 00: 02. Чтобы изменить этот адрес, измените шесть байтов, определенных в файле stm32f4xx_hal_conf.h.

IP-адрес по умолчанию установлен на 192.168.0.10. Чтобы изменить этот адрес, измените четыре байта, определенные в файле main.h.

### 5.2.3 Особенности прошивки

Этот пакет включает модули для расширения и расширения использования некоторых приложений.

Протокол DHCP поддерживается, так что MCU STM32 может выступать в качестве клиента DHCP для получения динамического IP-адреса, когда он подключен к серверу DHCP. Чтобы включить протокол DHCP, раскомментируйте следующий макрос:

#define USE_DHCP’’ из файла main.h.

Примечание. Если IP-адрес настроен с помощью DHCP, и приложение не находит DHCP-сервер в сети, к которой он уже подключен, тогда для IP-адреса автоматически устанавливается статический адрес (192.168.0.10).

Пользователь может включить контроллер LCD, определив макрос #define USE_LCD в main.h. Если он включен, будут отображаться текстовые сообщения, информирующие пользователя о состоянии приложения (назначенный IP-адрес, состояние сетевой ссылки...)

*Примечание.*

Приложения для запуска не поддерживают модули DHCP и LCD. Обратитесь к Разделу 6: Использование приложений LwIP для получения дополнительной информации.

## 5.3 Оценка настроек плат

Перед запуском примера Ethernet прочитайте соответствующий файл readme, чтобы узнать, как настроить перемычку на плате для обеспечения правильной работы.

# 6 Использование приложений LwIP

Пакет STM32Cube LwIP поставляется с несколькими приложениями, которые используют разные наборы API стека LwIP.

Gриложения делятся на три категории, как показано в таблице 8.

Таблица 8. Категории приложений LwIP

| Категория | Приложение |
|---|---|
| Getting started (basic) | TCP Echo client |
| | TCP Echo server |
| | UDP Echo client |
| | UDP Echo server |
| | TCP and UDP Echo server (Netconn API) |
| Features | HTTP Server (Raw API) |
| | HTTP Server (Netconn API) |
| | HTTP Server (Socket API) |
| Integrated | TFTP Server |

Приступая к работе приложения используют минимальную конфигурацию для запуска приложений поверх стека LwIP. Светодиоды используются для информирования пользователя о состоянии приложения.

Функции приложений обеспечивают большую гибкость и возможности. Они поддерживают сетевые протоколы, такие как HTTP, DHCP и используют ЖК-сообщения для индикации состояния приложения.

Интегрированное приложение поддерживает компонент промежуточного программного обеспечения FatFS и протокол TFTP для передачи файлов на карту microSD ™, расположенную на плате оценки, и с нее.

## 6.1 Начало работы с приложениями

### 6.1.1 TCP эхо-клиент

Это приложение используется для проверки основного соединения TCP. MCM STM32 действует как TCP-клиент, который подключается к TCP-серверу. Клиент отправляет строку, и сервер возвращает клиенту ту же строку.

Чтобы протестировать клиентское приложение TCP echo, выполните следующие действия:

1. Убедитесь, что настройки перемычки STM324xx-EVAL правильные.

2. Создайте и запрограммируйте демонстрационный код во флэш-память STM32F4xx.

Светодиоды указывают на успех или неудачу инициализации LwIP (динамическое распределение адресов «DHCP» не поддерживается для этого приложения).

3. На удаленном ПК откройте окно командной строки. Под Windows выберите Пуск> Все программы> Стандартные> Командная строка.

4. В командной строке введите:

C: \> echotool / p tcp / s

где:

– / p tcp – протокол TCP (протокол TCP)

– / s фактический режим соединения (режим сервера)

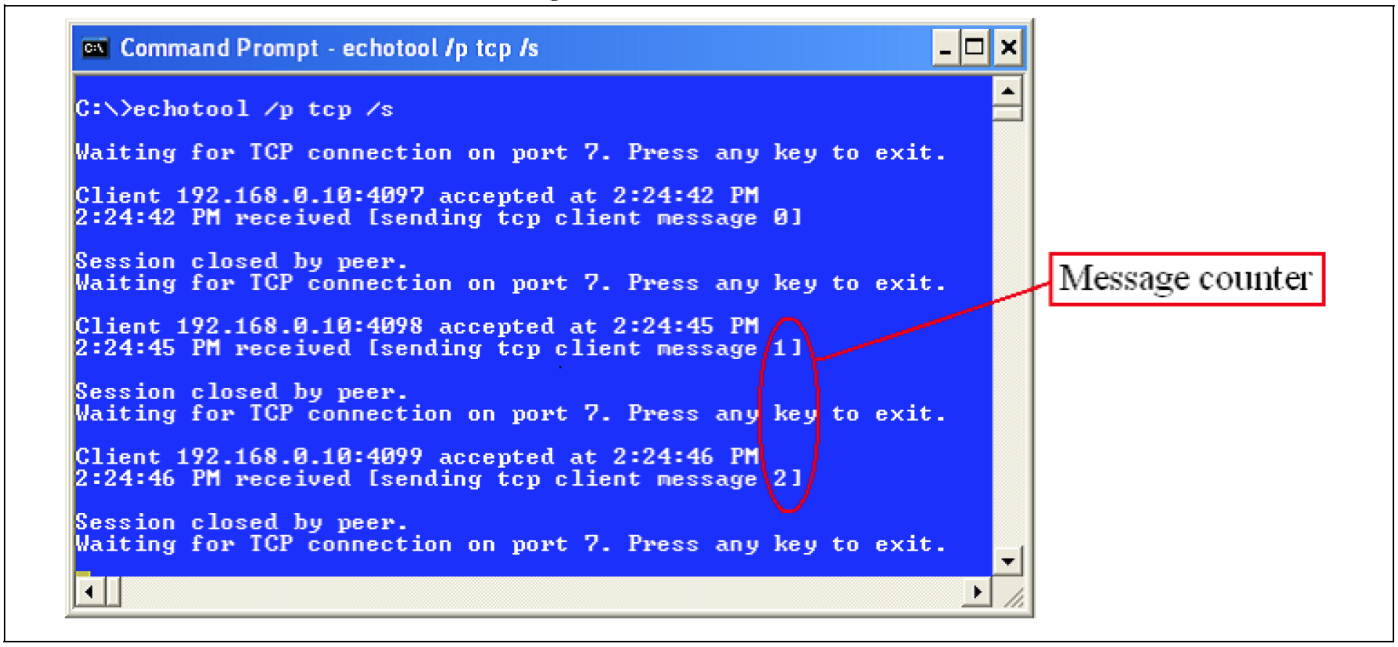
5. При нажатии кнопки Key на плате STM324xx-EVAL клиент отправляет строку, и сервер возвращает клиенту ту же строку.

*Примечание.*

Убедитесь, что IP-адрес удаленного ПК идентичен адресу, указанному в файле main.h (по умолчанию 192.168.0.11).

На рисунке 6 показан пример этой командной строки и ответа модуля.

**Figure 6. TCP echo client**




### 6.1.2 эхо-сервер TCP

Это приложение используется для проверки основного соединения TCP. MCU STM32 действует как TCP-сервер, ожидающий клиентских запросов. Это просто повторяет все, что отправлено.

Чтобы проверить демонстрацию эхо-сервера TCP, выполните следующие действия:

1. Убедитесь в правильности настроек перемычки STM324xx-EVAL.

2. Создайте и запрограммируйте демонстрационный код во флэш-память STM32F4xx.

Светодиоды показывают успех или неудачу инициализации LwIP (динамический адрес

выделение «DHCP» не поддерживается для этого приложения).

3. На удаленном ПК откройте окно командной строки. Под Windows выберите Пуск> Все программы> Стандартные> Командная строка.

4. В командной строке введите:

C: \> echotool IP_address / p tcp / r 7 / n 15 / t 2 / d Testing LwIP TCP echo server

где:

– IP_address – это фактический IP-адрес платы. По умолчанию используется следующий статический IP-адрес: 192.168.0.10

– / p tcp – это протокол (протокол TCP)

– / r – фактический удаленный порт на эхо-сервере (эхо-порт)

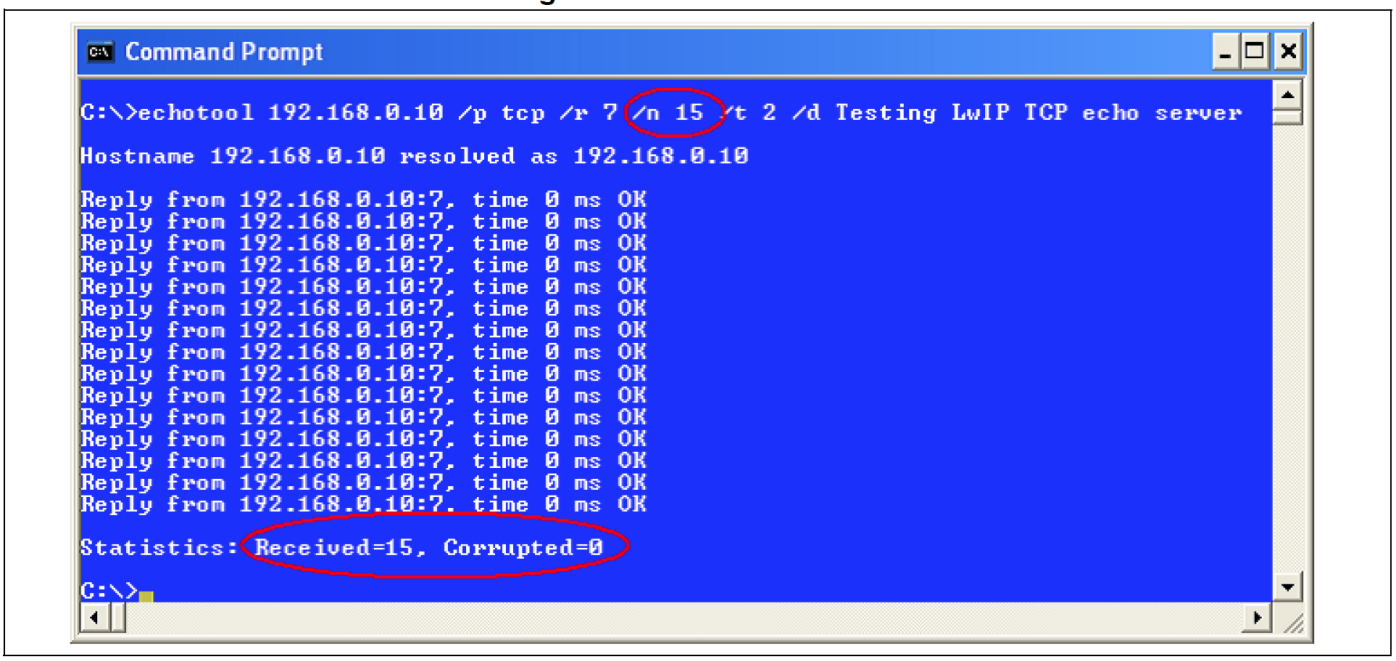
– / n – количество эхо-запросов (например, 15)

– / t – время ожидания соединения в секундах (например, 2)

– / d – это сообщение, которое будет отправлено для эха (например, «Тестирование эхо-сервера LwIP TCP»)

На рисунке 7 показан пример этой командной строки и ответа модуля.

**Figure 7. TCP echo server**




*Примечание:*

Статистика показывает количество полученных и поврежденных пакетов в конце теста.

### 6.1.3 UDP-эхо-клиент

Это приложение используется для проверки базовых эхо-соединений UDP. MCU STM32 действует как клиент UDP, который подключается к серверу UDP.

Чтобы проверить демонстрацию эхо-клиента UDP, выполните следующие действия:

1. Убедитесь в правильности настроек перемычки STM324xx-EVAL.

2. Создайте и запрограммируйте демонстрационный код во флэш-память STM32F4xx.

Светодиоды указывают на успех или неудачу инициализации LwIP (динамическое распределение адресов «DHCP» не поддерживается для этого приложения).

3. На удаленном ПК откройте окно командной строки. Под Windows выберите Пуск> Все программы> Стандартные> Командная строка.

4. В командной строке введите:

C: \> echotool / p udp / s

где:

– / p udp – протокол (протокол UDP)

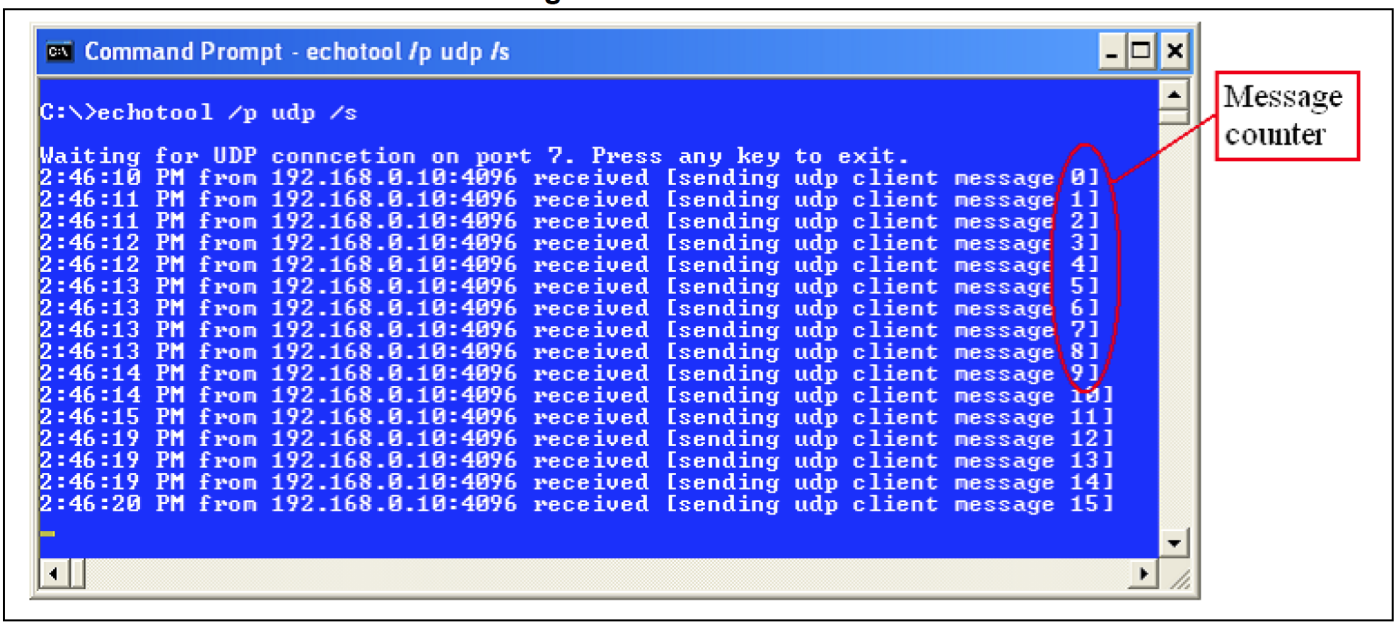
– / s фактический режим соединения (режим сервера)

5. При нажатии кнопки Key на плате STM324xx-EVAL клиент отправляет строку, и сервер возвращает клиенту ту же строку.

Примечание. Убедитесь, что IP-адрес удаленного ПК идентичен адресу, указанному в файле main.h (по умолчанию 192.168.0.11).

На рисунке 8 показан пример этой командной строки и ответа модуля.

**Figure 8. UDP echo client**




### 6.1.4 UDP эхо-сервер

Это приложение используется для тестирования основных соединений UDP. MCU STM32 действует как UDP-сервер, ожидающий клиентских запросов.

Чтобы протестировать приложение эхо-сервера UDP, выполните следующие действия:

1. Убедитесь в правильности настроек перемычки STM324xx-EVAL.
2. Создайте и запрограммируйте демонстрационный код во флэш-память STM32F4xx.

Светодиоды указывают на успех или неудачу инициализации LwIP (динамическое распределение адресов «DHCP» не поддерживается для этого приложения).

3. На удаленном ПК откройте окно командной строки. Под Windows выберите Пуск> Все программы> Стандартные> Командная строка.
4. В командной строке введите:

C:\>echotool IP_address /p udp /r 7 l/ 7 /n 15 /t 2 /d Testing LwIP UDP echo server

где:

– IP_address – это фактический IP-адрес платы. По умолчанию используется следующий статический IP-адрес: 192.168.0.10

– / p – протокол (протокол UDP)

– / r – фактический удаленный порт на эхо-сервере (эхо-порт)

– / l – фактический локальный порт для клиента (эхо-порт)

– / n – количество эхо-запросов (например, 15)

– / t – время ожидания соединения в секундах (например, 2)

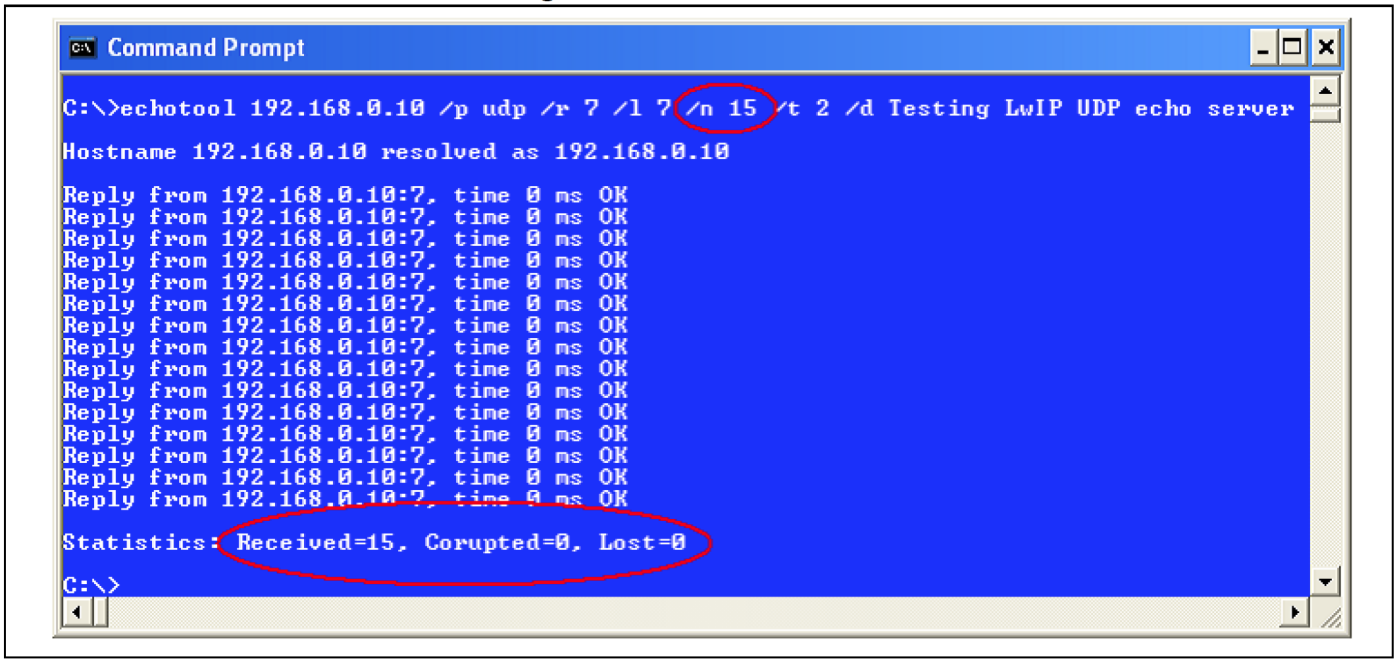
– / d – это сообщение, которое нужно отправить для эха (например, «Тестирование эхо-сервера LwIP UDP»)

На рисунке 9 показан пример этой командной строки и ответа модуля.

*Примечание.*

Статистика, показывающая количество полученных и поврежденных пакетов, дается в конце теста.

Figure 9. UDP echo server




### 6.1.5 UDP эхо-сервер TCP на основе точки доступа netconn

Эта демонстрация предоставляет приложение службы echo для протоколов TCP и UDP:

• Чтобы проверить демонстрацию netconn UDP TCP эхо-сервера в режиме TCP-сервера, см. Раздел 6.1.2: эхо-сервер TCP.

• Чтобы проверить демонстрацию netconn UDP TCP эхо-сервера в режиме UDP-сервера, см. Раздел 6.1.4: эхо-сервер UDP.

## 6.2 Особенности применения

### 6.2.1 Веб-сервер на основе сырого API

Это приложение реализует веб-сервер на основе необработанного API LwIP. Он используется для подключения к MCU STM32 из веб-клиента и для загрузки HTML-страниц.

Приложение веб-сервера реализует следующие функции:

• Разбор URL
• CGI (общий интерфейс шлюза)
• SSI (Включение на стороне сервера)
• Динамическая генерация заголовка
• HTTP Post запрос

Чтобы протестировать приложение веб-сервера, выполните следующие действия:

1. Убедитесь в правильности настроек перемычки STM324xx-EVAL.

2. В файле main.h раскомментируйте параметры «USE_DHCP» или «USE_LCD», чтобы включить функции клиента DHCP или ЖК-экрана.

3. Соберите и запрограммируйте код приложения во флэш-память STM32F4xx.

4. Если определены «USE_ DHCP» и «USE_LCD», на ЖК-экране отображается сообщение, указывающее на успешное или неудачное распределение IP-адреса DHCP, в противном случае светодиоды указывают на результат этой операции.

5. После назначения IP-адреса (статического или динамического) пользователь может запустить приложение.

6. На удаленном ПК откройте веб-клиент (Mozilla Firefox или Internet Explorer) и введите IP-адрес платы в веб-браузере. По умолчанию используется следующий статический IP-адрес: 192.168.0.10.

**Figure 10. Web server home page**


### Включения на стороне сервера (SSI)

SSI – это метод, используемый для динамического включения динамических данных в код HTML.

Это делается путем размещения определенного тега внутри HTML-кода веб-страницы. Тег должен иметь следующий формат:

<!--#tag-->

Для страницы преобразования АЦП в коде HTML используется следующий тег:

<!--#t-->

Когда есть запрос на веб-страницу ADC (которая имеет расширение «.shtml»), сервер анализирует веб-страницу и, когда тег найден, он заменяется значением преобразования ADC. (См. рисунок 8)

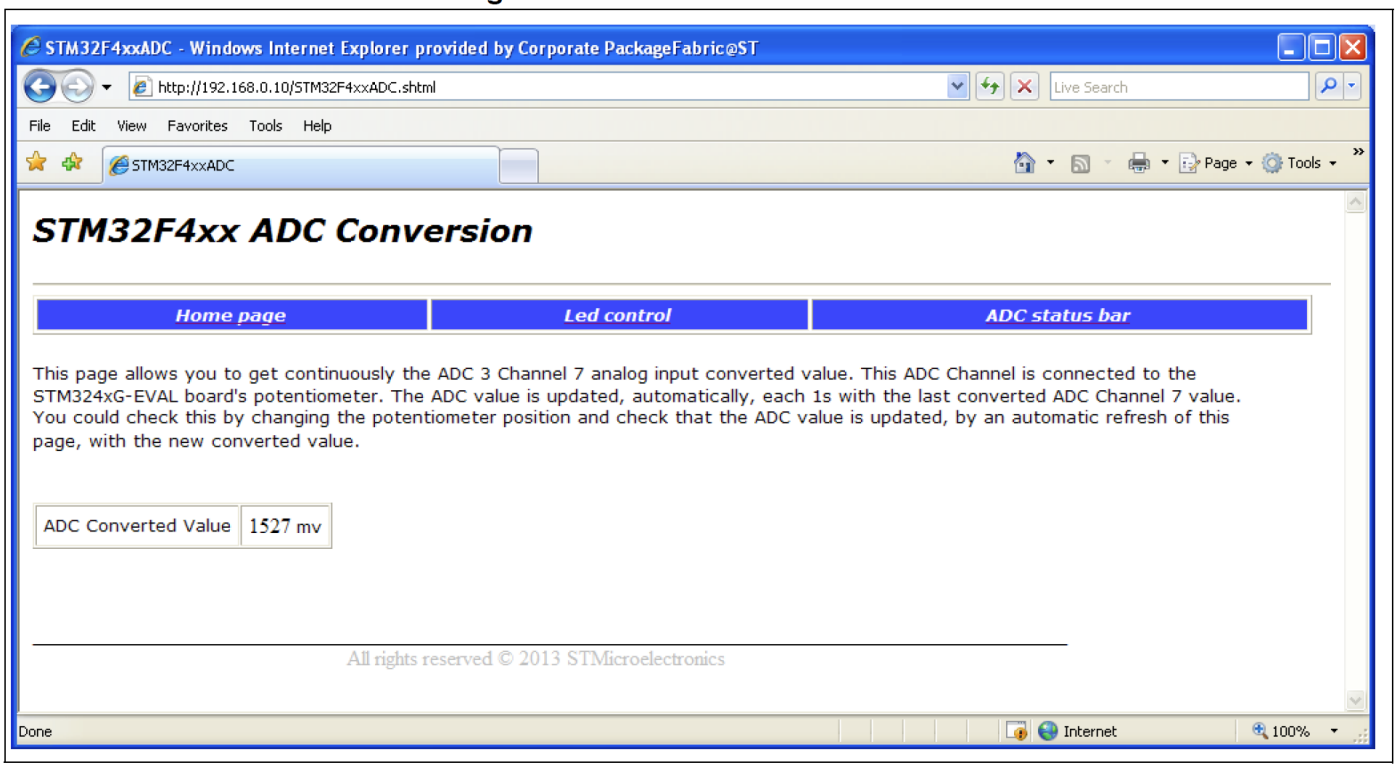
### Общий интерфейс шлюза (CGI)

CGI – это стандартная веб-методика, используемая для выполнения запроса, поступающего от клиента на стороне сервера, а затем для возврата ответа клиенту.

В LwIP предлагаемый CGI работает только с запросами метода GET и может обрабатывать до 16 параметров, закодированных в URI. Функция обработчика CGI, выполняемая на стороне сервера, возвращает файл HTML, который HTTP-сервер отправляет клиенту.

В демонстрации HTTP-сервера этот метод используется для управления четырьмя светодиодами (LED1, LED2, LED3 и LED4) оценочной платы.

Figure 11. SSI use in HTTP server

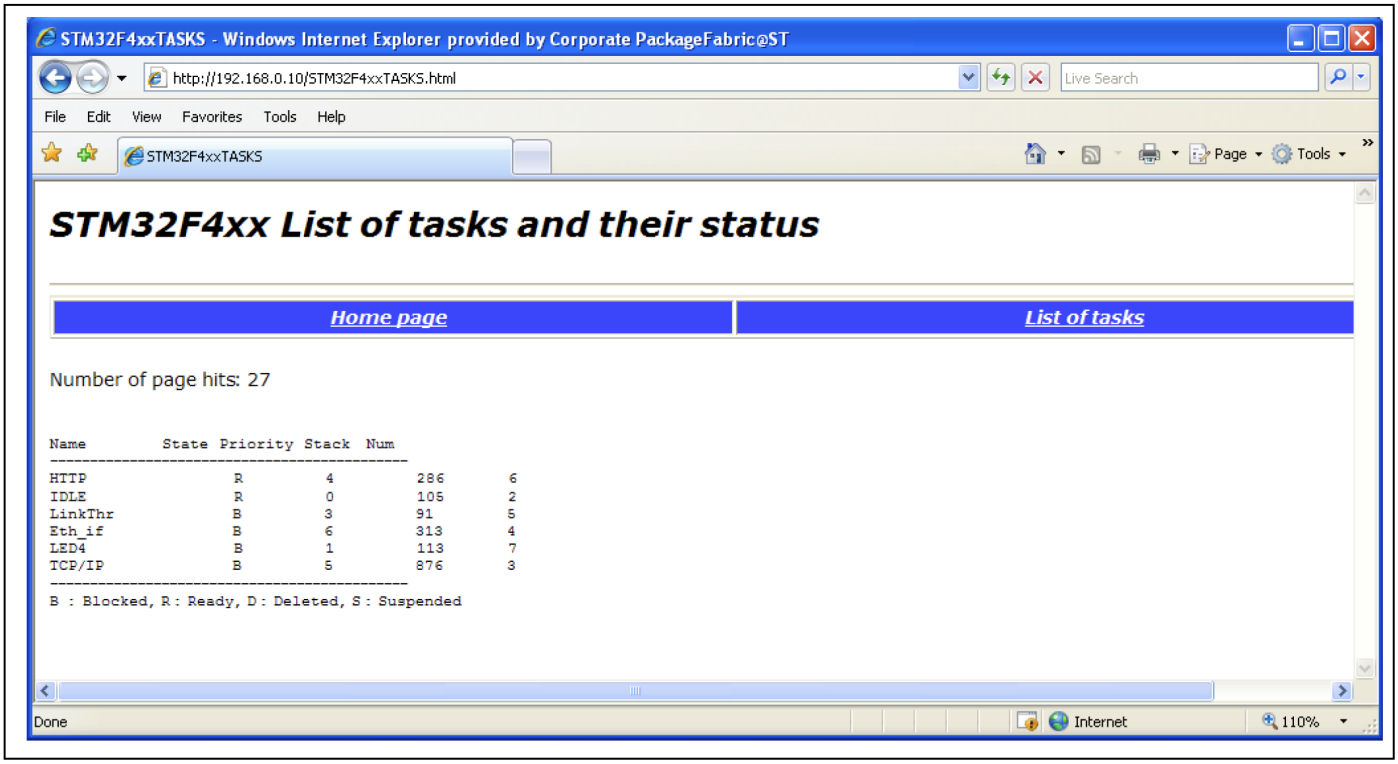



### 6.2.2 Веб-сервер на основе netconn API

Это приложение реализует веб-сервер на основе API-интерфейса netconn. Он используется для подключения к MCU STM32 из веб-клиента и для загрузки HTML-страниц.

Этот веб-сервер содержит две HTML-страницы. Первый дает общую информацию о микроконтроллерах STM32F4xx и стеке LwIP. Второй перечисляет запущенные задачи и их статус. Эта страница автоматически обновляется каждую секунду (см. Рисунок 12).

Figure 12. Web server list of task page



Чтобы протестировать демоверсию netconn HTTP-сервера, выполните следующие действия:

1. Убедитесь в правильности настроек перемычки STM324xx-EVAL.
2. В файле main.h раскомментируйте опции «USE_DHCP» или «USE_LCD», чтобы включить функции клиента DHCP или ЖК-экрана.
3. Соберите и запрограммируйте код приложения во флэш-память STM32F4xx.
4. Если определены «USE_ DHCP» и «USE_LCD», на ЖК-экране отображается сообщение, указывающее на успешное или неудачное распределение IP-адреса DHCP, в противном случае светодиоды указывают результат этой операции.
5. После назначения IP-адреса (статического или динамического) пользователь может запустить приложение.
6. На удаленном ПК откройте веб-клиент (Mozilla Firefox или Internet Explorer) и введите IP-адрес платы в веб-браузере. По умолчанию используется следующий статический IP-адрес: 192.168.0.10.

### 6.2.3 Веб-сервер на основе API сокетов

Это приложение, реализующее веб-сервер на основе сокета API. Чтобы проверить эту демонстрацию, обратитесь к Разделу 6.2.2: Веб-сервер на основе netconn API.

## 6.3 Интегрированные приложения

### 6.3.1 TFTP сервер

TFTP-сервер – это приложение для передачи файлов, для которого требуется удаленный TFTP-клиент. Файлы передаются на карту microSD, расположенную на плате STM324xx-EVAL, и обратно.

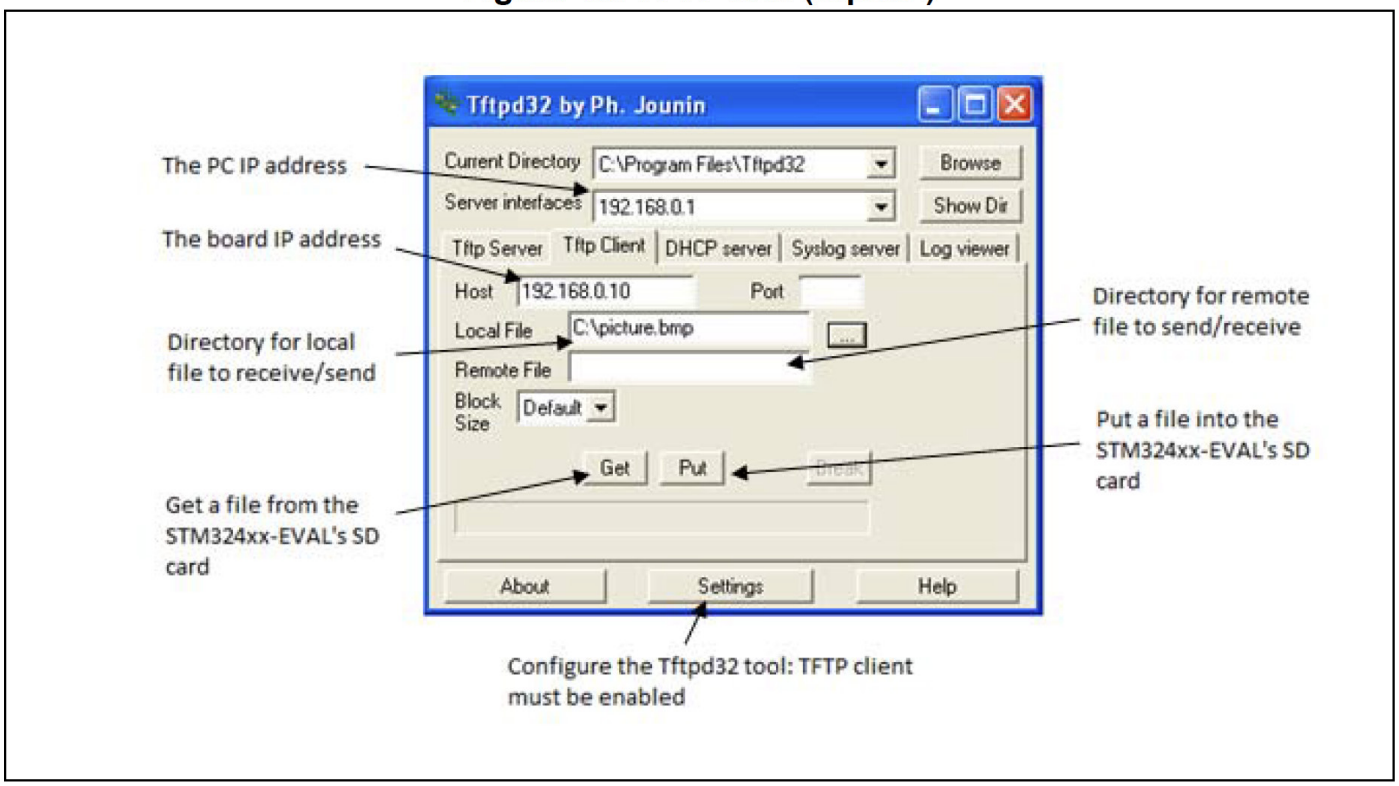
Сервер TFTP ожидает запроса от удаленного клиента TFTP. Оценочная плата STM324xx-EVAL должна быть подключена через удаленный компьютер для загрузки или загрузки файла. Для этого на удаленном ПК должен быть установлен клиент TFTP. Это можно сделать с помощью инструмента tftpd32, который можно найти по адресу http://tftpd32.jounin.net.

Чтобы протестировать приложение TFTP-сервера, выполните следующие действия:

1. Убедитесь в правильности настроек перемычки STM324xx-EVAL.
2. В файле main.h раскомментируйте параметры «USE_DHCP» или «USE_ LCD», чтобы включить функции клиента DHCP или ЖК-экрана.
3. Соберите и запрограммируйте код приложения во флэш-память STM32F4xx.
4. Если определены «USE_ DHCP» и «USE_LCD», на ЖК-экране отображается сообщение, указывающее на успешное или неудачное распределение IP-адреса DHCP, в противном случае светодиоды указывают на результат этой операции.
5. После назначения IP-адреса (статического или динамического) пользователь может запустить приложение.
6. На удаленном ПК откройте клиент TFTP (например, TFTPD32) и настройте адрес сервера TFTP (адрес хоста в TFTPD32).
7. Начните передачу файлов на карту micro SD, расположенную на плате STM324xx-EVAL, и обратно.

На рисунке 13 представлен обзор инструмента tftpd32.

**Figure 13. TFTP tool (tftpd32)**




*Примечание.*

Перед загрузкой / загрузкой файла с / на плату STM324xx-EVAL убедитесь, что карта microSD вставлена в специальный разъем.

## 7 Заключение

Пакет LwIP позволяет использовать стек TCP / IP lwIP с API-интерфейсом STM32Cube HAL. Этот стек с открытым исходным кодом предлагает услуги полномасштабного стека TCP / IP, сохраняя при этом относительно низкое использование ОЗУ / ПЗУ.

Описаны два подхода для разработки приложений TCP / IP, либо в автономном режиме, либо с использованием операционной системы реального времени (RTOS) для многопоточных операций.

## Приложение A FAQ

### A.1 Как выбрать статический или динамический (DHCP)

Распределение IP-адресов?

При комментировании макроса #define USE_DHCP, расположенного в main.h, статическому IP-адресу назначается микроконтроллер STM32 (по умолчанию 192.168.0.10, это значение можно изменить из файла «main.h»).

Если макрос #define USE_DHCP не закомментирован, протокол DHCP включен, и STM32 будет действовать как клиент DHCP

### A.2 Как ведет себя приложение, когда кабель Ethernet отключен?

Когда кабель отключен, периферийное устройство Ethernet прекращает передачу и прием трафика, а сетевой интерфейс отключается. Если используется ЖК-

контроллер, отображается сообщение, информирующее пользователя о том, что кабель не подключен, в противном случае включается красный светодиод оценочной платы.

Когда пользователь повторно подключает кабель, возобновляется трафик Ethernet и настраивается сетевой интерфейс. Если используется ЖК-контроллер, отображается сообщение, информирующее пользователя о новом IP-адресе со статическим или динамическим назначением, в противном случае включается желтый светодиод оценочной платы.

### A.3 Как приложение может быть перенесено на другое оборудование?

Когда используется другая аппаратная платформа, проверьте конфигурацию GPIO в функции HAL_ETH_MspInit () для периферийного устройства Ethernet и в HAL_PPP_MspInit () или HAL_MspInit (), если приложению требуется больше периферийного устройства PPP.