lib_xtcp: TCP/IP library

Publication Date: 2025/7/17
Document Number: XM-007217-UG v6.2.0

XMOS

IN THIS DOCUMENT

1 Introduction

This document details the XMOS TCP library **lib_xtcp** which allows use of TCP and UDP traffic over Ethernet.

The following sections of the document describe the general usage and behaviour of the library, followed by a detailed usage with an example application and then detailed descriptions of the APIs.

This document assumes familiarity with the XMOS xcore architecture, Ethernet, and TCP/IP along with the XMOS XTC toolchain and the XC language.

lib_xtcp is intended to be used with the XCommon CMake , the *XMOS* application build and dependency management system.

This library is for use with *xcore-200* series (XS2 architecture) or *xcore.ai* series (XS3 architecture) devices, previous generations of xcore devices (i.e. XS1 architecture) are supported, but all examples and app-notes target newer devices.

1.1 Terms

The terms used in this document can appear confusing as *client* and *server* can both be used in two different ways. Firstly, for the an XC interface there are clients and servers, see *Client API* as an example. The client being application and the server being the **lib_xtcp** stack. This usage is commonly used throughout this document. Secondly, there are TCP/IP clients and servers, these are referred to as a local host or remote host, or in context such as *DHCP server* or *HTTP server*.

2 Overview

The TCP/IP library provides OSI layer 3 and 4 features. Client applications built on `lib_xtcp` can provide layers 5-7 features as needed, such as the HTTP example provided in the library's example `app_simple_webserver`, see *Getting Started* section.

Table 1: lib_xtcp and the OSI Layer Model

OSI Layers	Addressing	xcore libraries
Application	e.g. HTTP/URL	(lib_xtcp client) app_simple_webserver
Presentation	application specific	lib_xtcp client application
Session	application specific	lib_xtcp client application
Transport	Port	lib_xtcp
Network	IP	lib_xtcp
Data link	MAC	lib_ethernet (MAC)
Physical	PHY	lib_ethernet (PHY)

The TCP/IP stack runs in a task implemented in either the *`xtcp_uip()`* or *`xtcp_lwip()`* functions depending on which stack implementation is preferred. The interface to the stack are the same, regardless of which implementation is being used.

This task connects to either the RMII/RGMII MAC components or the MII component in the Ethernet library `lib_etherent`. See the figures *XTCP task diagram* and *XTCP task diagram (MII)* and the Ethernet library user guide for details on these components.

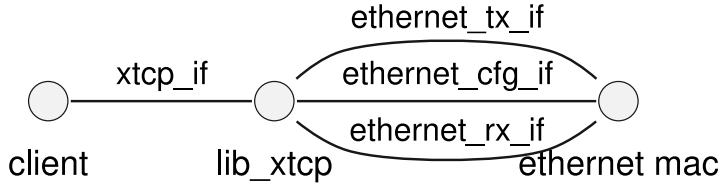

Fig. 1: XTCP task diagram

Or direct to the MII component,

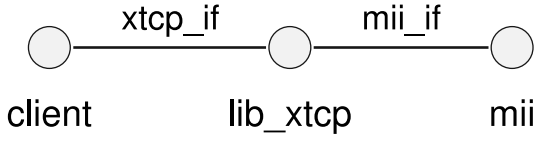

Fig. 2: XTCP task diagram (MII)

Clients can interact with the TCP/IP stack via interfaces connected to the component using the interface functions described in *Client API*.

If the application has no need to direct layer 2 traffic to the Ethernet MAC then the most resource efficient approach is to connect the `xtcp` component directly to the MII layer component.

2.1 IP Configuration

The library server will determine its IP configuration based on the **xtcp_ipconfig_t** configuration passed into the *xtcp_uip()* / *xtcp_lwip()* task (see section *Server API*). If an address is supplied then that address will be used (a static IP address configuration):

```
xtcp_ipconfig_t ipconfig = {
  { 192, 168,   0, 2 }, // ip address
  { 255, 255, 255, 0 }, // netmask
  { 192, 168,   0, 1 }  // gateway
};
```

If no address is supplied then the server will first try to find a DHCP server on the network to obtain an address automatically. If it cannot obtain an address from DHCP, it will determine a link local address (in the range 169.254/16) automatically using the Zeroconf IPV4LL protocol.

To use dynamic address, the *xtcp_uip()* and *xtcp_lwip()* functions can be passed a structure with an IP address that is all zeros:

```
xtcp_ipconfig_t ipconfig = {
  { 0, 0, 0, 0 }, // ip address
  { 0, 0, 0, 0 }, // netmask
  { 0, 0, 0, 0 }  // gateway
};
```

2.2 Events and Connections

The TCP/IP application stack client interface (see *Client API*) is a low-level event based interface. This is to allow applications to manage buffering and connections in the most efficient way possible for the application.

Each client will receive packet ready *events* from the server to indicate that the server has new data for that client. The client then collects the packet using the *get_packet()* call.

The packets sent from the server can be either data or control packets. The type of packet is indicated in the connection state **event** member. The possible packet types are defined in *Event types*.

A client will typically handle its connection to the XTCP server in the following manner:

```
xtcp_connection_t conn;
char buffer[ETHERNET_MAX_PACKET_SIZE];
unsigned data_len;
select {
  case i.xtcp.packet_ready():
    i_xtcp.get_packet(conn, buffer, ETHERNET_MAX_PACKET_SIZE, data_len);
    // Handle event
    switch (conn.event) {
      ...
    }
    break;
  }
```

The client can also call interface functions to initiate new connections, manage the connection and send or receive data.

If the client is handling multiple connections then the server may interleave events for each connection so the client has to hold a persistent state for each connection.

The connection and event model is the same from both TCP connections and UDP connections. Full details of both the possible events and possible commands can be found in *Functional API*.

2.3 New Connections

New connections are made in two different ways. Either the *connect()* function is used to initiate a connection with a remote host or the *listen()* function is used to listen on a port for remote hosts to connect to the application. In either case once a connection is established then the *XTCP_NEW_CONNECTION* event is received by the client.

By convention with POSIX sockets, a listening UDP connection merely reports data received on the socket, independent of the source IP address. In XTCP, a *XTCP_NEW_CONNECTION* event is sent each time data arrives from a new source. The API function *close()* should be called after the connection is no longer needed.

2.4 TCP and UDP

The XTCP API treats UDP and TCP connections in the same way. The only difference is when the protocol is specified on initializing connections with the interface *connect()* or *listen()* functions.

For example, a client that wishes to listen for HTTP requests over TCP connections on port 80:

```
i_xtcp.listen(80, XTCP_PROTOCOL_TCP);
```

A client could create a new UDP connection to port 15333 on a machine at 192.168.0.2 using:

```
xtcp_ipaddr_t addr = { 192, 168, 0, 2 };
i_xtcp.connect(15333, addr, XTCP_PROTOCOL_UDP);
```

2.5 Receiving Data

When data is received for a client the server will indicate that there is a packet ready and the *get_packet()* call will indicate that the event type is *XTCP_RECV_DATA* and the packet data will have been returned to the *get_packet()* call.

Data is sent from the XTCP server to client as the UDP or TCP packets arrive from the ethernet MAC. There is no buffering in the server so it will wait for the client to handle the event before processing new incoming packets.

2.6 Sending Data

When sending data, the client is responsible for dividing the data into chunks for the server and re-transmitting the previous chunk if a transmission error occurs.

> **Note**
>
> Note that re-transmission may be needed on both TCP and UDP connections. On UDP connections, the transmission may fail if the server has not yet established a connection between the destination IP address and layer 2 MAC address.

The client sends a packet by calling the *send()* interface function. A *resend* is done by calling *send()* function with the same data buffer as the previous send.

Note

The maximum buffer size that can be sent in one call to *xtcp_send* is contained in the *mss* field of the connection structure relating to the event.

After this data is sent to the server, two things can happen, shown in figure *Example TCP/IP send sequence*: Either the server will respond with an `XTCP_SENT_DATA` event, in which case the next chunk of data can be sent. Or with an `XTCP_RESEND_DATA` event in which case the client must re-transmit the previous chunk of data.

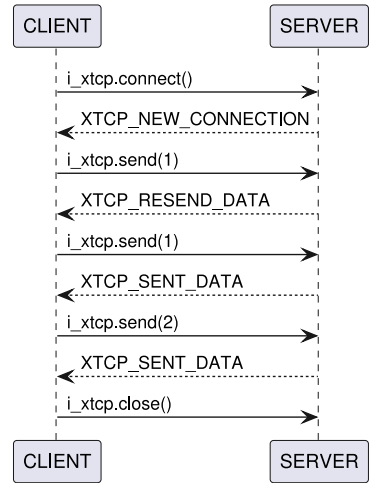

Fig. 3: Example TCP/IP send sequence

2.7 Link Status Events

As well as events related to connections. The server may also send link status events to the client. The events `XTCP_IFUP` and `XTCP_IFDOWN` indicate to a client when the link goes up or down.

2.8 Server Configuration

The server is configured via arguments passed to server task, see section *Server API* (`xtcp_uip()` / `xtcp_lwip()`) and the defines described in Section *Configuration Defines*.

2.9 Stack Configuration

The underlying stack configuration can by modified by including optional header files in the application. One or both of the following, these will override the uIP or LwIP build settings. See *Configuration Defines*.

- xtcp_client_conf.h
- xtcp_conf.h

3 Usage

3.1 Using lib_xtcp

To use this library, include **lib_xtcp** in the application's **APP_DEPENDENT_MODULES** list in *CMakeLists.txt*, for example:

```
set(APP_DEPENDENT_MODULES "lib_xtcp")
```

All functions and types can be found in the **xtcp.h** header file:

```
#include <xtcp.h>
```

3.2 Stack Selection

To choose which stack to use, simply call either *xtcp_uip()* or *xtcp_lwip()* in main.

3.3 Getting Started

The app_simple_webserver example is provided to show how the library can use TCP traffic for a very simple HTTP server.

The example targets the XCORE-200-EXPLORER dev-kit and 1000BASE-T ethernet with an RGMII PHY.

The **lib_xtcp** supports two TCP/IP stacks, either **uIP** or **LwIP** stacks. The example is configured to support both stacks, selecting the correct entry point depending on the application compiler defines. To change the selected stack please see the CMakelists.txt for the example and swap the define for either **XTCP_STACK_LWIP** or **XTCP_STACK_UIP** in **APP_COMPILER_FLAGS**.

```
set(APP_COMPILER_FLAGS ${COMPILER_FLAGS_COMMON} -DXTCP_STACK_LWIP)
```

By default the The IP address for the XCORE will be automatically assigned via DHCP if *xtcp_ipconfig_t* **ipconfig = { ... };** in **main.xc** is filled with zeros. Otherwise, to set a static IP address, insert the IPv4 address into the first row of **ipconfig** and the subnet mask to the second row, the subnet mask is typically **{ 255, 255, 255, 0 }**. For details please see section *IP Configuration*

The excerpt from the example web server shown below shows how to configure the **lib_xtcp** server with the application client here as *xhttpd*

```
int main(void) {
  xtcp_if i_xtcp[NUM_XTCP_CLIENTS];
  smi_if i_smi;
  ethernet_cfg_if i_cfg[NUM_CFG_CLIENTS];
  ethernet_rx_if i_rx[NUM_ETH_CLIENTS];
  ethernet_tx_if i_tx[NUM_ETH_CLIENTS];

  par {
    // ethernet driver setup here...

    // SMI/ethernet phy driver
    on tile[1]: smi(i_smi, p_smi_mdio, p_smi_mdc);

    on tile[0]: xtcp_lwip(i_xtcp, NUM_XTCP_CLIENTS, null,
                          i_cfg[CFG_TO_XTCP], i_rx[ETH_TO_XTCP], i_tx[ETH_TO_XTCP],
                          mac_address_phy, null, ipconfig);

    // HTTP server application
    on tile[0]: xhttpd(i_xtcp[XTCP_TO_HTTP]);
  }
  return 0;
}
```

The function **xhttpd()**, called from main will listen for a TCP connection on port 80 and shows an example of handling the events and data flowing to and from the TCP stack.

For details please see section *Events and Connections* and the notifications are defined in *Event types*.

```
void xhttpd(client xtcp_if i_xtcp)
{
  printstr("**WELCOME TO THE SIMPLE WEBSERVER DEMO**\n");

  // Initiate the HTTP state
  httpd_init(i_xtcp);

  // Loop forever processing TCP events
  while(1) {
    xtcp_connection_t conn;
    char rx_buffer[RX_BUFFER_SIZE];
    unsigned data_len;

    select {
      case i_xtcp.packet_ready(): {
        i_xtcp.get_packet(conn, rx_buffer, RX_BUFFER_SIZE, data_len);

        if (conn.local_port == 80) {
          // HTTP connections
          switch (conn.event) {
            ...
```

The project supports CMake by default, to build the project first configure then build with,

```
cd lib_xtcp

cmake -B build -G "Unix Makefiles"

xmake -j -C build
```

Once built run with,

```
xrun --xscope bin/app_simple_webserver
```

When running and with the dev-kit connected to the same network has the computer, open a browser window and enter the address printed on the xrun terminal. The browser will display a short message, “Hello World!”.

4 Configuration API

4.1 Configuration Defines

Configuration defines can either be set by adding the a command line option to the build flags in the application CMakelists file (i.e. `-DDEFINE=VALUE`) or by adding the file `xtcp_client_conf.h` into the application and then putting `#define` directives into that header file (which will then be read by the library on build).

XTCP_CLIENT_BUF_SIZE
: The buffer size used for incoming packets. This has a maximum value of 1472 which can handle any incoming packet. If it is set to a smaller value, larger incoming packets will be truncated. Default is 1472.

UIP_CONF_MAX_CONNECTIONS
: The maximum number of UDP or TCP connections the server can handle simultaneously. Default is 20.

UIP_CONF_MAX_LISTENPORTS
: The maximum number of UDP or TCP ports the server can listen to simultaneously. Default is 20.

UIP_USE_AUTOIP
: By defining this as 0, the IPv4LL application is removed from the code. Do this to save approximately 1kB. Auto IP is a stateless protocol that assigns an IP address to a device. Typically, if a unit is trying to use DHCP to obtain an address, and a server cannot be found, then auto IP is used to assign an address of the form 169.254.x.y. Auto IP is enabled by default

UIP_USE_DHCP
: By defining this as 0, the DHCP client is removed from the code. This will save approximately 2kB. DHCP is a protocol for dynamically acquiring an IP address from a centralised DHCP server. This option is enabled by default.

5 Functional API

See *Usage* section and *Getting Started* for details on usage of the following.

5.1 Data Structures/Types

typedef unsigned char **xtcp_ipaddr_t**[4]

XTCP IP address.

This data type represents a single ipv4 address in the XTCP stack.

struct **xtcp_ipconfig_t**

IP configuration information structure.

This structure describes IP configuration for an ip node.

enum **xtcp_protocol_t**

XTCP protocol type.

This determines what type a connection is: either UDP or TCP.

Values:

enumerator **XTCP_PROTOCOL_TCP**

Transmission Control Protocol

enumerator **XTCP_PROTOCOL_UDP**

User Datagram Protocol

5.2 Event types

enum **xtcp_event_type_t**

XTCP event type.

The event type represents what event is occuring on a particular connection. It is instantiated as part of the *xtcp_connection_t* structure in the function *get_packet()*.

Values:

enumerator **XTCP_NEW_CONNECTION**

This event represents a new connection has been made. In the case of a TCP server connections it occurs when a remote host firsts makes contact with the local host. For TCP client connections it occurs when a stream is setup with the remote host. For UDP connections it occurs as soon as the connection is created.

enumerator **XTCP_RECV_DATA**

This event occurs when the connection has received some data. The return_len in *get_packet()* will indicate the length of the data. The data will be present in the buffer passed to *get_packet()*.

enumerator **XTCP_SENT_DATA**

This event occurs when the server has successfully sent the previous piece of data that was given to it via a call to *send()*.

enumerator **XTCP_RESEND_DATA**

This event occurs when the server has failed to send the previous piece of data that was given to it via a call to *send()*. The server is now requesting for the same data to be sent again.

enumerator **XTCP_TIMED_OUT**

This event occurs when the connection has timed out with the remote host (TCP only). This event represents the closing of a connection and is the last event that will occur on an active connection.

enumerator **XTCP_ABORTED**

This event occurs when the connection has been aborted by the local or remote host (TCP only). This event represents the closing of a connection and is the last event that will occur on an active connection.

enumerator **XTCP_CLOSED**

This event occurs when the connection has been closed by the local or remote host. This event represents the closing of a connection and is the last event that will occur on an active connection.

enumerator **XTCP_IFUP**

This event occurs when the link goes up (with valid new ip address). This event has no associated connection.

enumerator **XTCP_IFDOWN**

This event occurs when the link goes down. This event has no associated connection.

enumerator **XTCP_DNS_RESULT**

This event occurs when the XTCP connection has a DNS result for a request.

struct **xtcp_connection_t**

This type represents a TCP or UDP connection.

This is the main type containing connection information for the client to handle. Elements of this type are instantiated by the xtcp_event() function which informs the client about an event and the connection the event is on.

5.3 Server API

void **xtcp_uip**(
    SERVER_INTERFACE_ARRAY(xtcp_if, i_xtcp, *n_xtcp*), static_const_unsigned n_xtcp, NULLABLE_CLIENT_INTERFACE(mii_if, i_mii), NULLABLE_CLIENT_INTERFACE(ethernet_cfg_if, i_eth_cfg), NULLABLE_CLIENT_INTERFACE(ethernet_rx_if, i_eth_rx), NULLABLE_CLIENT_INTERFACE(ethernet_tx_if, i_eth_tx), CONST_NULLABLE_ARRAY_OF_SIZE(char, mac_address0, MACADDR_NUM_BYTES), NULLABLE_REFERENCE_PARAM(otp_ports_t, otp_ports), REFERENCE_PARAM(*xtcp_ipconfig_t*, ipconfig),
)

This functions implements a TCP/IP stack that clients can access via interfaces. This stack will be the uIP stack.

Parameters

- **i_xtcp** – The interface array to connect to the clients.
- **n_xtcp** – The number of clients to the task.
- **i_mii** – If this component is connected to the mii() component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null
- **i_eth_cfg** – If this component is connected to an MAC component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null.
- **i_eth_rx** – If this component is connected to an MAC component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null.
- **i_eth_tx** – If this component is connected to an MAC component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null.
- **mac_address** – If this array is non-null then it will be used to set the MAC address of the component.
- **otp_ports** – If this port structure is non-null then the component will obtain the MAC address from OTP ROM. See the OTP reading library user guide for details.
- **ipconfig** – This *xtcp_ipconfig_t* structure is used to determine the IP address configuration of the component.

void **xtcp_lwip**(
    SERVER_INTERFACE_ARRAY(xtcp_if, i_xtcp, *n_xtcp*), static_const_unsigned n_xtcp, NULLABLE_CLIENT_INTERFACE(mii_if, i_mii), NULLABLE_CLIENT_INTERFACE(ethernet_cfg_if, i_eth_cfg), NULLABLE_CLIENT_INTERFACE(ethernet_rx_if, i_eth_rx), NULLABLE_CLIENT_INTERFACE(ethernet_tx_if, i_eth_tx), CONST_NULLABLE_ARRAY_OF_SIZE(char, mac_address0, MACADDR_NUM_BYTES), NULLABLE_REFERENCE_PARAM(otp_ports_t, otp_ports), REFERENCE_PARAM(*xtcp_ipconfig_t*, ipconfig),
)

This functions implements a TCP/IP stack that clients can access via interfaces. This stack will be the lwIP stack.

Parameters

- **i_xtcp** – The interface array to connect to the clients.
- **n_xtcp** – The number of clients to the task.
- **i_mii** – If this component is connected to the mii() component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null
- **i_eth_cfg** – If this component is connected to an MAC component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null.
- **i_eth_rx** – If this component is connected to an MAC component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null.
- **i_eth_tx** – If this component is connected to an MAC component in the Ethernet library then this interface should be used to connect to it. Otherwise it should be set to null.
- **mac_address** – If this array is non-null then it will be used to set the MAC address of the component.
- **otp_ports** – If this port structure is non-null then the component will obtain the MAC address from OTP ROM. See the OTP reading library user guide for details.
- **ipconfig** – This *xtcp_ipconfig_t* structure is used to determine the IP address configuration of the component.

5.4 Client API

group **Xtcp_if**

This interface API is for use by application clients that wish to leverage lib_xtcp for host-to-host data exchange.

Functions

void **get_packet**(
REFERENCE_PARAM(*xtcp_connection_t*, conn), char data[*n*], unsigned n, REFERENCE_PARAM(unsigned, length),
)

Recieve information/data from the XTCP server.
After the client is notified by *packet_ready()* it must call this function to receive the packet from the server.
If the data buffer is not large enough then an exception will be raised.

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.
- **data** – An array where XTCP server can write data to. This data array must be large enough to receive the packets being sent to the client. In most cases it should be assumed that packets of ETHERNET_MAX_PACKET_SIZE can be received.
- **n** – Size of the data array.
- **length** – An integer where the server can indicate the length of the sent packet.

void **packet_ready**()

Notifies the client that there is data/information ready for them.
After this notification is raised a call to *get_packet()* is needed.

void **listen**(int port_number, *xtcp_protocol_t* protocol)

Listen to a particular incoming port.
After this call, when a connection is established an XTCP_NEW_CONNECTION event is signalled.

Parameters

- **port_number** – The local port number to listen to
- **protocol** – The protocol to connect with (XTCP_PROTOCOL_TCP or XTCP_PROTOCOL_UDP)

void **unlisten**(unsigned port_number)

Stop listening to a particular incoming port.

Parameters

- **port_number** – local port number to stop listening on

void **close**(const REFERENCE_PARAM(*xtcp_connection_t*, conn))

Close a connection.
May still recieve data on a TCP connection. Use *abort()* if you wish to completely stop all data. Will continue to listen on the open port the connection came from.

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.

void **abort**(const REFERENCE_PARAM(*xtcp_connection_t*, conn))

Abort a connection.

For UDP this is the same as closing the connection. For TCP the server will send a RST signal and stop all incoming data.

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.

void **connect**(
unsigned port_number, *xtcp_ipaddr_t* ipaddr, *xtcp_protocol_t* protocol,
)

Try to connect to a remote port.

For TCP this will initiate the three way handshake. For UDP this will assign a random local port and bind the remote end of the connection to the host specified.

Parameters

- **port_number** – The remote port to try to connect to
- **ipaddr** – The ip addr of the remote host
- **protocol** – The protocol to connect with (XTCP_PROTOCOL_TCP or XTCP_PROTOCOL_UDP)

void **send**(
const REFERENCE_PARAM(*xtcp_connection_t*, conn), char data[], unsigned len,
)

Send data to the connection.

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.
- **data** – An array of data to send
- **len** – The length of data to send. If this is 0, no data will be sent and a XTCP_SENT_DATA event will not occur.

void **join_multicast_group**(*xtcp_ipaddr_t* addr)

Subscribe to a particular IP multicast group address.

Parameters

- **addr** – The address of the multicast group to join. It is assumed that this is a multicast IP address.

void **leave_multicast_group**(*xtcp_ipaddr_t* addr)

Unsubscribe from a particular IP multicast group address.

Parameters

- **addr** – The address of the multicast group to leave. It is assumed that this is a multicast IP address.

void **set_appstate**(
const REFERENCE_PARAM(*xtcp_connection_t*, conn), xtcp_appstate_t appstate,
)

Set the connections application state data item.

After this call, subsequent events on this connection will have the appstate field of the connection set.

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.
- **appstate** – An unsigned integer representing the state. In C this is usually a pointer to some connection dependent information.

void **bind_local_udp**(
const REFERENCE_PARAM(*xtcp_connection_t*, conn), unsigned port_number,
)

Bind the local end of a connection to a particular port (UDP).

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.
- **port_number** – The local port to set the connection to.

void **bind_remote_udp**(
const REFERENCE_PARAM(*xtcp_connection_t*, conn), *xtcp_ipaddr_t* ipaddr, unsigned port_number,
)

Bind the remote end of a connection to a particular port and ip address (UDP). After this call, packets sent to this connection will go to the specified address and port

Parameters

- **conn** – The connection structure to be passed in that will contain all the connection information.
- **ipaddr** – The intended remote address of the connection
- **port_number** – The intended remote port of the connection

void **request_host_by_name**(const char hostname[], unsigned name_len)

Request a hosts IP address from a URL.

> **Note**
>
> LWIP ONLY.

Parameters

- **hostname** – The human readable host name, e.g. "www.xmos.com"
- **name_len** – The length of the hostname in characters

void **get_ipconfig**(REFERENCE_PARAM(*xtcp_ipconfig_t*, ipconfig))

Fill the provided ipconfig address with the current state of the server.

Parameters

- **ipconfig** – IPconfig to be filled.