TCP/IP Library

A TCP/UDP/IP protocol stack for XMOS devices. This library connects to the XMOS Ethernet library to provide layer-3 traffic over Ethernet via MII or RGMII.

Features

- TCP + UDP connection handling
- DHCP, IP4LL, ICMP, IGMP support
- Low level, event based interface for efficient memory usage
- Based on the open-source uIP stack
- Currently, the library does not officially support IPv6. However, experimental code for IPv6 support is contained in the library. Contact XMOS for more details if you require IPv6.

Typical Resource Usage

This following table shows typical resource usage in some different configurations. Exact resource usage will depend on the particular use of the library by the application.

Configuration	Pins	Ports	Clocks	Ram	Logical cores
Standard	0	0	0	~21.1K	1

Software version and dependencies

This document pertains to version 4.0.2 of this library. It is known to work on version 14.1.1 of the xTIMEcomposer tools suite, it may work on other versions.

This library depends on the following other libraries:

- lib_otpinfo (>=2.0.0)
- lib_ethernet (>=3.0.3)

Related application notes

The following application notes use this library:

- AN00121 - Using the XMOS TCP/IP library

1 Usage

The TCP/IP stack runs in a task implemented in the xtcp() function. This task connects to either the MII component in the Ethernet library or one of the MAC components in the Ethernet library. See the Ethernet library user guide for details on these components.

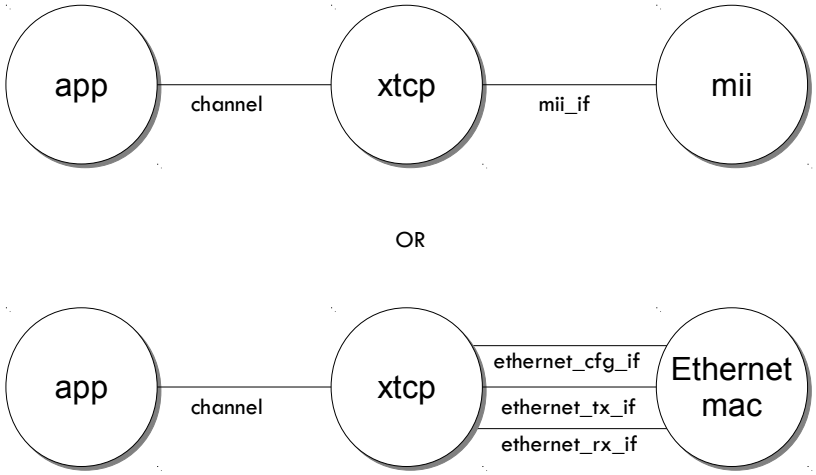

Figure 1: XTCP task diagram

Clients can interact with the TCP/IP stack via xC channels connected to the component using the client functions described in §3.3.

If your application has no need of 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.

1.1 IP Configuration

The server will determine its IP configuration based on the arguments passed into the xtcp() function. If an address is supplied then that address will be used (a static IP address configuration).

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() function can be passed a structure with an IP address that is all zeros.

1.2 Events and Connections

The TCP/IP stack client interface is a low-level event based interface. This is to allow applications to manage buffering and connection management in the most efficient way possible for the application.

Each client will receive *events* from the server. These events usually have an associated *connection*. In addition to receiving these events the client can send *commands* to the server to initiate new connections and so on.

The above Figure shows an example event/command sequence of a client making a connection, sending some data, receiving some data and then closing the connection. Note that sending and receiving may be split into several events/commands since the server itself performs no buffering.

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.

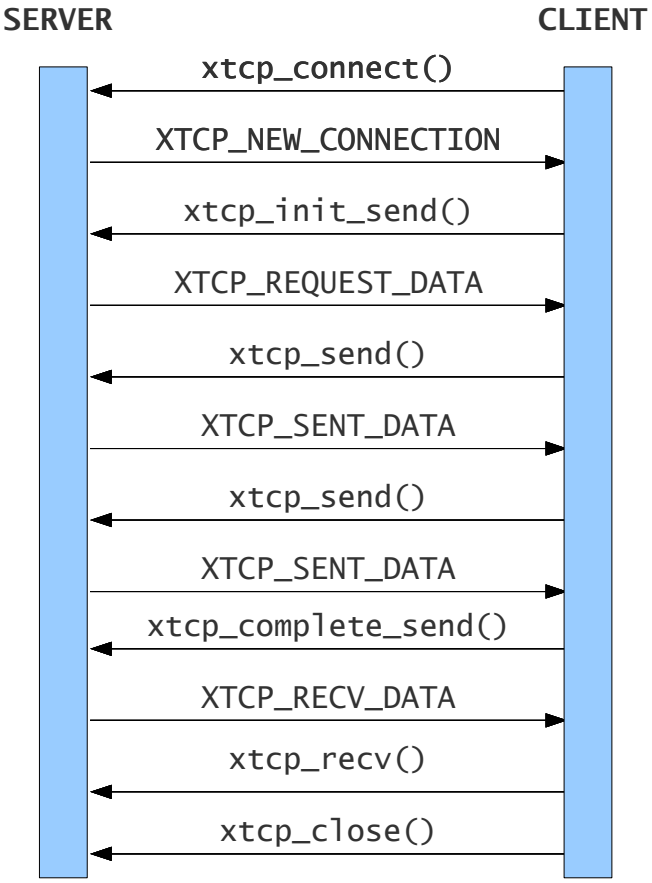

Figure 2: Example event sequence

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 §3.

1.3 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 xtcp_connect() or xtcp_listen().

1.4 New Connections

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

In the Berkley sockets API, a listening UDP connection merely reports data received on the socket, indepedent 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 xtcp_close() should be called after the connection is no longer needed.

1.5 Receiving Data

When data is received by a connection, the XTCP_RECV_DATA event is triggered and communicated to the client. At this point the client **must** call the xtcp_recv() function to receive the data.

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

1.6 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 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 can initiate a send transaction with the xtcp_init_send() function. At this point no sending has been done but the server is notified of a wish to send. The client must then wait for a XTCP_REQUEST_DATA event at which point it must respond with a call to xtcp_send().

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: 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.

The command/event exchange continues until the client calls the xtcp_complete_send() function to finish the send transaction. After this the server will not trigger any more XTCP_SENT_DATA events.

1.7 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.

1.8 Configuration

The server is configured via arguments passed to the `xtcp_server()` function and the defines described in Section §2.1.

Client connections are configured via the client API described in Section §2.1.

2 Configuration API

2.1 Configuration Defines

Configuration defines can either be set by adding the a command line option to the build flags in your application Makefile (i.e. -DDEFINE=VALUE) or by adding the file `xtcp_client_conf.h` into your 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.

XTCP_EXCLUDE_LISTEN Exclude support for the listen command from the server, reducing memory footprint.

XTCP_EXCLUDE_UNLISTEN Exclude support for the unlisten command from the server, reducing memory footprint

XTCP_EXCLUDE_CONNECT Exclude support for the connect command from the server, reducing memory footprint

XTCP_EXCLUDE_BIND_REMOTE Exclude support for the bind_remote command from the server, reducing memory footprint

XTCP_EXCLUDE_BIND_LOCAL Exclude support for the bind_local command from the server, reducing memory footprint

XTCP_EXCLUDE_INIT_SEND Exclude support for the init_send command from the server, reducing memory footprint

XTCP_EXCLUDE_SET_APPSTATE Exclude support for the set_appstate command from the server, reducing memory footprint

XTCP_EXCLUDE_ABORT Exclude support for the abort command from the server, reducing memory footprint

XTCP_EXCLUDE_CLOSE Exclude support for the close command from the server, reducing memory footprint

XTCP_EXCLUDE_SET_POLL_INTERVAL Exclude support for the set_poll_interval command from the server, reducing memory footprint

XTCP_EXCLUDE_JOIN_GROUP Exclude support for the join_group command from the server, reducing memory footprint

XTCP_EXCLUDE_LEAVE_GROUP Exclude support for the leave_group command from the server, reducing memory footprint

XTCP_EXCLUDE_GET_MAC_ADDRESS Exclude support for the get_mac_address command from the server, reducing memory footprint

XTCP_EXCLUDE_GET_IPCONFIG Exclude support for the get_ipconfig command from the server, reducing memory footprint

XTCP_EXCLUDE_ACK_RECV Exclude support for the ack_recv command from the server, reducing memory footprint

XTCP_EXCLUDE_ACK_RECV_MODE Exclude support for the ack_recv_mode command from the server, reducing memory footprint

XTCP_EXCLUDE_PAUSE Exclude support for the pause command from the server, reducing memory footprint

XTCP_EXCLUDE_UNPAUSE Exclude support for the unpause command from the server, reducing memory

footprint

UIP_USE_AUTOIP By defining this as 0, the IPv4LL application is removed from the code. Do this to save approxmiately 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.

3 Functional API

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

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

The application also needs to add lib_xtcp to its build modules:

```
USED_MODULES = ... lib_xtcp ...
```

3.1 Data Structures/Types

Type	xtcp_ipaddr_t
Description	XTCP IP address. This data type represents a single ipv4 address in the XTCP stack.

Type	xtcp_ipconfig_t
Description	IP configuration information structure. This structure describes IP configuration for an ip node.
Fields	xtcp_ipaddr_t ipaddr The IP Address of the node. xtcp_ipaddr_t netmask The netmask of the node. The mask used to determine which address are routed locally. xtcp_ipaddr_t gateway The gateway of the node.

Type	xtcp_protocol_t
Description	XTCP protocol type. This determines what type a connection is: either UDP or TCP.
Values	XTCP_PROTOCOL_TCP Transmission Control Protocol. XTCP_PROTOCOL_UDP User Datagram Protocol.

Type	xtcp_event_type_t
Description	XTCP event type. The event type represents what event is occuring on a particualr connection. It is instantiated when an event is received by the client using the xtcp_event() function.
Values	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. XTCP_RECV_DATA This event occurs when the connection has received some data. The client **must** follow receipt of this event with a call to xtcp_recv() before any other interaction with the server. XTCP_PUSH_DATA This event occurs when the connection has received a packet with the TCP push flag set indicating that the other side has temporarily finished sending data. XTCP_REQUEST_DATA This event occurs when the server is ready to send data and is requesting that the client send data. This event happens after a call to xtcp_init_send() from the client. The client **must** follow receipt of this event with a call to xtcp_send() before any other interaction with the server. 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 xtcp_send(). The server is now requesting more data so the client must** follow receipt of this event with a call to xtcp_send() before any other interaction with the server. 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 xtcp_send(). The server is now requesting for the same data to be sent again. The client must** follow receipt of this event with a call to xtcp_send() before any other interaction with the server.

Continued on next page

	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. 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. 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. XTCP_POLL This event occurs at regular intervals per connection. Polling can be initiated and the interval can be set with xtcp_set_poll_interval() XTCP_IFUP This event occurs when the link goes up (with valid new ip address). This event has no associated connection. XTCP_IFDOWN This event occurs when the link goes down. This event has no associated connection. XTCP_ALREADY_HANDLED This event type does not get set by the server but can be set by the client to show an event has been handled.

Type	**xtcp_connection_type_t**
Description	Type representing a connection type.
Values	XTCP_CLIENT_CONNECTION A client connection. XTCP_SERVER_CONNECTION A server connection.

Type	xtcp_connection_t
Description	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.

Fields	`int id` A unique identifier for the connection.
	
	`xtcp_protocol_t protocol`
	The protocol of the connection (TCP/UDP).
	
	`xtcp_connection_type_t connection_type`
	The type of connection (client/sever).
	
	`xtcp_event_type_t event`
	The last reported event on this connection.
	
	`xtcp_appstate_t appstate`
	The application state associated with the connection.
	This is set using the xtcp_set_connection_appstate() function.
	
	`xtcp_ipaddr_t remote_addr`
	The remote ip address of the connection.
	
	`unsigned int remote_port`
	The remote port of the connection.
	
	`unsigned int local_port`
	The local port of the connection.
	
	`unsigned int mss`
	The maximum size in bytes that can be send using xtcp_send() after a send event.

3.2 Server API

Function	xtcp
Description	Function implement the TCP/IP stack task. This functions implements a TCP/IP stack that clients can access via xC channels.
Type	void xtcp(chanend c_xtcp[n], size_t n, client mii_if ?i_mii, client ethernet_cfg_if ?i_eth_cfg, client ethernet_rx_if ?i_eth_rx, client ethernet_tx_if ?i_eth_tx, client smi_if ?i_smi, uint8_t phy_address, const char(& ?mac_address)[6], otp_ports_t & ?otp_ports, xtcp_ipconfig_t &ipconfig)

Continued on next page

Parameters	`c_xtcp`	The channel array to connect to the clients.
	`n`	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.
	`i_smi`	If this connection to an Ethernet SMI component is then the XTCP component will poll the Ethernet PHY for link up/link down events. Otherwise, it will expect link up/link down events from the connected Ethernet MAC.
	`phy_address`	The SMI address of the Ethernet PHY
	`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 :c:type:`xtcp_ipconfig_t` structure is used to determine the IP address configuration of the component.

3.3 Client API

3.3.1 Event Receipt

Function	xtcp_event
Description	Receive the next connect event. Upon receiving the event, the xtcp_connection_t structure conn is instatiated with information of the event and the connection it is on. This can be used in a select statement.
Type	`transaction xtcp_event(chanend c_xtcp, xtcp_connection_t &conn)`
Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection relating to the current event

3.3.2 Setting Up Connections

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

Type	`void xtcp_listen(chanend c_xtcp,`
	`int port_number,`
	`xtcp_protocol_t proto)`

Parameters	`c_xtcp` chanend connected to the xtcp server `port_number` the local port number to listen to `proto` the protocol to listen to (TCP or UDP)

Function	xtcp_unlisten
Description	Stop listening to a particular incoming port. Applies to TCP connections only.
Type	`void xtcp_unlisten(chanend c_xtcp, int port_number)`

Continued on next page

Parameters	`c_xtcp` chanend connected to the xtcp server `port_number` local port number to stop listening on

Function	**xtcp_connect**
Description	Try to connect to a remote port.

Type	`void xtcp_connect(chanend c_xtcp,`
	`int port_number,`
	`xtcp_ipaddr_t ipaddr,`
	`xtcp_protocol_t proto)`

Parameters	`c_xtcp` chanend connected to the xtcp server `port_number` the remote port to try to connect to `ipaddr` the ip addr of the remote host `proto` the protocol to connect with (TCP or UDP)

Function	**xtcp_bind_local**
Description	Bind the local end of a connection to a particular port (UDP).

Type	`void xtcp_bind_local(chanend c_xtcp,`
	`xtcp_connection_t &conn,`
	`int port_number)`

Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection `port_number` the local port to set the connection to

Function	**xtcp_bind_remote**
Description	Bind the remote end of a connection to a particular port and ip address. This is only valid for XTCP_PROTOCOL_UDP connections. After this call, packets sent to this connection will go to the specified address and port

Continued on next page

Type	`void xtcp_bind_remote(chanend c_xtcp,`
	`xtcp_connection_t &conn,`
	`xtcp_ipaddr_t addr,`
	`int port_number)`

Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection `addr` the intended remote address of the connection `port_number` the intended remote port of the connection

Function	**xtcp_set_connection_appstate**
Description	Set the connections application state data item. After this call, subsequent events on this connection will have the appstate field of the connection set

Type	`void`
	`xtcp_set_connection_appstate(chanend c_xtcp,`
	`xtcp_connection_t &conn,`
	`xtcp_appstate_t appstate)`

Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection `appstate` An unsigned integer representing the state. In C this is usually a pointer to some connection dependent information.

3.3.3 Receiving Data

Function	**xtcp_recv**
Description	Receive data from the server. This can be called after an XTCP_RECV_DATA event.
Type	`int xtcp_recv(chanend c_xtcp, char data[])`
Parameters	`c_xtcp` chanend connected to the xtcp server `data` A array to place the received data into

Continued on next page

Returns	The length of the received data in bytes

Function	**xtcp_recvi**
Description	Receive data from the xtcp server. This can be called after an XTCP_RECV_DATA event. The data is put into the array data starting at index i i.e. the first byte of data is written to data[i].
Type	`int xtcp_recvi(chanend c_xtcp, char data[], int i)`
Parameters	`c_xtcp` chanend connected to the xtcp server `data` A array to place the received data into `i` The index where to start filling the data array
Returns	The length of the received data in bytes

Function	**xtcp_recv_count**
Description	Receive a number of bytes of data from the xtcp server. This can be called after an XTCP_RECV_DATA event. Data is pulled from the xtcp server and put into the array, until either there is no more data to pull, or until count bytes have been received. If there are more bytes to be received from the server then the remainder are discarded. The return value reflects the number of bytes pulled from the server, not the number stored in the buffer. From this the user can determine if they have lost some data. see the buffer client protocol for a mechanism for receiving bytes without discarding the extra ones.
Type	`int xtcp_recv_count(chanend c_xtcp, char data[], int count)`
Parameters	`c_xtcp` chanend connected to the xtcp server `data` A array to place the received data into `count` The number of bytes to receive
Returns	The length of the received data in bytes, whether this was more or less than the requested amount.

3.3.4 Sending Data

Function	xtcp_init_send
Description	Initiate sending data on a connection. After making this call, the server will respond with a XTCP_REQUEST_DATA event when it is ready to accept data.
Type	`void xtcp_init_send(chanend c_xtcp, xtcp_connection_t &conn)`
Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection

Function	xtcp_send
Description	Send data to the xtcp server. Send data to the server. This should be called after a XTCP_REQUEST_DATA, XTCP_SENT_DATA or XTCP_RESEND_DATA event (alternatively xtcp_write_buf can be called). To finish sending this must be called with a length of zero or call the xtcp_complete_send() function.
Type	`void xtcp_send(chanend c_xtcp, char ?data[], int len)`
Parameters	`c_xtcp` chanend connected to the xtcp server `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.

Function	xtcp_sendi
Description	Send data to the xtcp server. Send data to the server. This should be called after a XTCP_REQUEST_DATA, XTCP_SENT_DATA or XTCP_RESEND_DATA event (alternatively xtcp_write_buf can be called). The data is sent starting from index i i.e. data[i] is the first byte to be sent. To finish sending this must be called with a length of zero.
Type	`void xtcp_sendi(chanend c_xtcp, char ?data[], int i, int len)`

Continued on next page

Parameters	c_xtcp	chanend connected to the xtcp serve
	data	An array of data to send
	i	The index at which to start reading from the data array
	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.

Function	**xtcp_complete_send**
Description	Complete a send transaction with the server. This function can be called after a XTCP_REQUEST_DATA, XTCP_SENT_DATA or XTCP_RESEND_DATA event to finish any sending on the connection that the event related to.
Type	`void xtcp_complete_send(chanend c_xtcp)`
Parameters	c_xtcp chanend connected to the tcp server

3.3.5 Other Connection Management

Function	**xtcp_set_poll_interval**
Description	Set UDP poll interval. When this is called then the udp connection will cause a poll event every poll_interval milliseconds.
Type	`void xtcp_set_poll_interval(chanend c_xtcp, xtcp_connection_t &conn, int poll_interval)`
Parameters	c_xtcp chanend connected to the xtcp server conn the connection poll_interval the required poll interval in milliseconds

Function	**xtcp_close**
Description	Close a connection.

Continued on next page

Type	`void xtcp_close(chanend c_xtcp, xtcp_connection_t &conn)`
Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection

Function	**xtcp_abort**
Description	Abort a connection.
Type	`void xtcp_abort(chanend c_xtcp, xtcp_connection_t &conn)`
Parameters	`c_xtcp` chanend connected to the xtcp server `conn` the connection

Function	**xtcp_pause**
Description	pause a connection. No further reads and writes will occur on the network. This functionality is considered experimental for when using IPv6.
Type	`void xtcp_pause(chanend c_xtcp, xtcp_connection_t &conn)`
Parameters	`c_xtcp` chanend connected to the xtcp server `conn` tcp connection structure

Function	**xtcp_unpause**
Description	unpause a connection Activity is resumed on a connection. This functionality is considered experimental for when using IPv6.
Type	`void xtcp_unpause(chanend c_xtcp, xtcp_connection_t &conn)`
Parameters	`c_xtcp` chanend connected to the xtcp server `conn` tcp connection structure

3.3.6 Other General Client Functions

Function	xtcp_join_multicast_group
Description	Subscribe to a particular ip multicast group address. Not available for IPv6
Type	`void` `xtcp_join_multicast_group(chanend c_xtcp, xtcp_ipaddr_t addr)`
Parameters	`c_xtcp` chanend connected to the xtcp server `addr` The address of the multicast group to join. It is assumed that this is a multicast IP address.

Function	xtcp_leave_multicast_group
Description	Unsubscribe to a particular ip multicast group address. Not available for IPv6

Type	`void`
	`xtcp_leave_multicast_group(chanend c_xtcp,`
	`xtcp_ipaddr_t addr)`

Parameters	`c_xtcp` chanend connected to the xtcp server `addr` The address of the multicast group to leave. It is assumed that this is a multicast IP address which has previously been joined.

Function	xtcp_get_mac_address
Description	Get the current host MAC address of the server.
Type	`void xtcp_get_mac_address(chanend c_xtcp,` `unsigned char mac_addr[])`
Parameters	`c_xtcp` chanend connected to the xtcp server `mac_addr` the array to be filled with the mac address

Function	xtcp_get_ipconfig
Description	Get the IP config information into a local structure. Get the current host IP configuration of the server.
Type	`void xtcp_get_ipconfig(chanend c_xtcp, xtcp_ipconfig_t &ipconfig)`

Continued on next page

Parameters	`c_xtcp`	chanend connected to the xtcp server
	`ipconfig`	the structure to be filled with the IP configuration information

APPENDIX A - Known Issues

There are no known issues with this library.

APPENDIX B - TCP/IP Library Change Log

B.1 4.0.2

- Change uIP timer.h to uip_timer.h to avoid conflict with xcore timer.h
- Update to source code license and copyright

B.2 4.0.1

- Fixed issue with link up/down events being ignored when SMI is not polled within XTCP
- MAC address parameter to xtcp() is now qualified as const to allow parallel usage

B.3 4.0.0

- Moved over to new file structure
- Updated to use new lib_ethernet
- Changes to dependencies:
  - lib_gpio: Added dependency 1.0.0
  - lib_otpinfo: Added dependency 2.0.0
  - lib_locks: Added dependency 2.0.0
  - lib_ethernet: Added dependency 3.0.0
  - lib_xassert: Added dependency 2.0.0
  - lib_logging: Added dependency 2.0.0

B.4 Legacy release history

B.5 3.2.1

- Changes to dependencies:
  - sc_ethernet: 2.2.7rc1 -> 2.3.1rc0
    - Fix invalid inter-frame gaps.
    - Adds AVB-DC support to sc_ethernet

B.6 3.2.0

- Added IPv6 support

B.7 3.1.5

- Fixed channel protocol bug that caused crash when xCONNECT is heavily loaded
- Various documentation updates
- Fixes to avoid warning in xTIMEcomposer studio version 13.0.0 or later
- Changes to dependencies:
  - sc_ethernet: 2.2.5rc2 -> 2.2.7rc1
    - Fix buffering bug on full implementation that caused crash under
    - Various documentation updates

B.8 3.1.4

- Updated ethernet dependency to version 2.2.5

B.9 3.1.3

- Updated ethernet dependency to version 2.2.4
- Fixed corner case errors/improved robustness in DHCP protocol handling

B.10 3.1.2

- Fixed auto-ip bug for 2-core xtcp server

B.11 3.1.1

- Minor code demo app fixes (port structures should be declared on specific tiles)

B.12 3.1.0

- Compatible with 2.2 module_ethernet
- Updated to new intializer api and integrated ethernet server

B.13 3.0.1

- Updated to use latest sc_ethernet package

B.14 3.0.0

- Fixed bugs in DHCP and multicast UDP
- Updated packaging, makefiles and documentation
- Updated to use latest sc_ethernet package

B.15 2.0.1

- Further memory improvements
- Additional conditional compilation
- Fix to zeroconf with netbios option enabled

B.16 2.0.0

- Memory improvements
- Fix error whereby UDP packets with broadcast destination were not received
- An initial implementation of a TFTP server

B.17 1.3.1

- Initial implementation