TCP/IP Library

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

Features

- TCP and UDP connection handling
- DHCP, IP4LL, ICMP, IGMP support
- Low level, event based interface for efficient memory usage
- Supports IPv4 only, not IPv6

Stacks

This library provides two different TCP/IP stack implementations ported to the xCORE architecture.

uIP stack

The first stack ported is the uIP (micro IP) stack. The uIP stack has been designed to have a minimal resource footprint. As a result, it has limited performance and does not provide support for TCP windowing.

lwIP stack

The second stack ported is the lwIP (lightweight IP) stack. The lwIP stack requires more resources than uIP, but is designed to provide better throughput and also has support for TCP windowing.

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
UIP	0	0	0	~25.7K	1
LWIP	0	0	0	~63.6K	1

Software version and dependencies

This document pertains to version 6.0.0 of this library. It is known to work on version 14.2.4 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 (>=3.2.0)

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 either the xtcp_uip() or xtcp_lwip() functions depending on which stack implementation you wish to use. The interfaces to the stack are the same, regardless of which implementation is being used.

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.

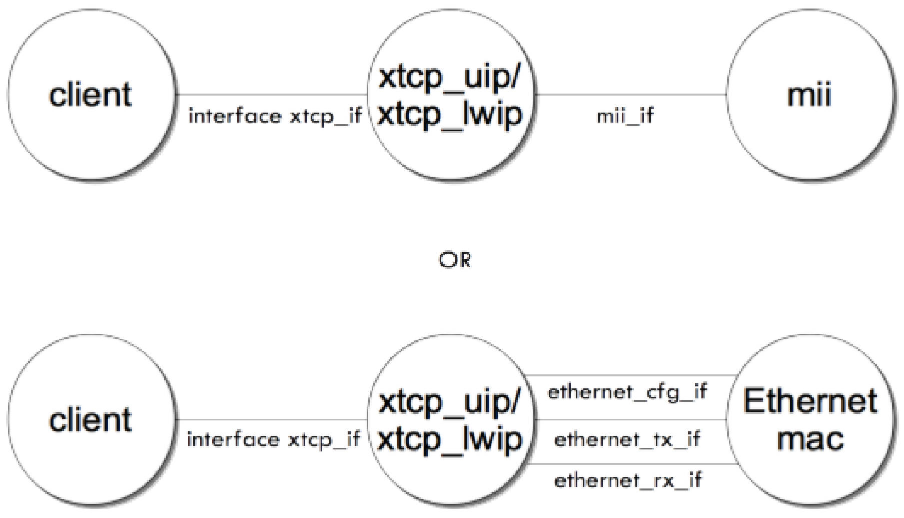

Figure 1: XTCP task diagram

Clients can interact with the TCP/IP stack via interfaces connected to the component using the interface 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 xtcp_ipconfig_t configuration passed into the xtcp_uip() / xtcp_lwip() task. 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
};
```

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 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 §3.1.

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 §3.

1.3 New Connections

New connections are made in two different ways. Either the `connect()` function is used to initiate a connection with a remote host as a client or the `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 received by the client.

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 `close()` should be called after the connection is no longer needed.

1.4 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, an HTTP client would listen for 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);
```

1.5 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.

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 sends a packet by calling the `send()` interface function.

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.

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 server task (xtcp_uip()/ xtcp_lwip()) and the defines described in Section §2.1.

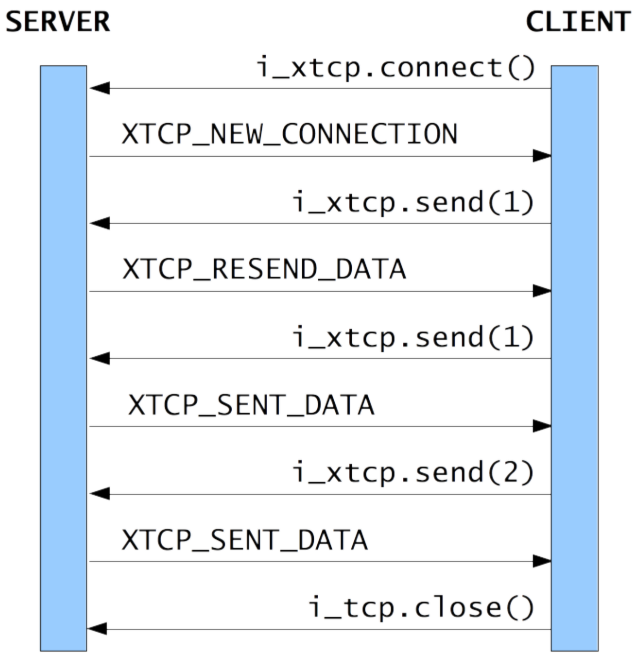

Figure 2: Example send sequence

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.

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 particular connection. It is instantiated as part of the xtcp_connection_t structure in the function get_packet().
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 return_len in get_packet() will indicate the length of the data. The data will be present in the buffer passed to get_packet(). 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(). 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. 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.

Continued on next page

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

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 client_num` The number of the client connected. `int id` A unique identifier for the connection. `xtcp_protocol_t protocol` The protocol of the connection (TCP/UDP). `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 `set_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.

Continued on next page

	`unsigned int mss`
	The maximum size in bytes that can be send using `xtcp_send()` after a send event.
	
	`unsigned packet_length`
	Length of packet recieved.
	
	`int stack_conn`
	Pointer to the associated uIP/LWIP connection.
	Only to be used by XTCP.

3.2 Server API

Function	**xtcp_uip**
Description	Function implementing the TCP/IP stack task using the uIP stack. This functions implements a TCP/IP stack that clients can access via interfaces.

Type	
	```
	void
	xtcp_uip(server xtcp_if i_xtcp[n_xtcp],
	         static const unsigned n_xtcp,
	         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_address0)[6],
	         otp_ports_t & ?otp_ports,
	         xtcp_ipconfig_t &ipconfig)
	```

Continued on next page

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.
	`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.

Function	**xtcp_lwip**
Description	Function implementing the TCP/IP stack using the lwIP stack. This functions implements a TCP/IP stack that clients can access via interfaces.

Continued on next page

Type	
	```
	void
	xtcp_lwip(server xtcp_if i_xtcp[n_xtcp],
	          static const unsigned n_xtcp,
	          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_address0)[6],
	          otp_ports_t & ?otp_ports,
	          xtcp_ipconfig_t &ipconfig)
	```

Continued on next page

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.
	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

Type	**xtcp_if**
Description	
Functions	

Function	**get_packet**
Description	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.

Type	`[[clears_notification]]`
	`void get_packet(xtcp_connection_t &conn,`
	`                char data[n],`
	`                unsigned n,`
	`                unsigned &length)`

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.

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

Type	`[[notification]]`
	`slave void packet_ready()`

Continued on next page

Type	xtcp_if (continued)
	Function: **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 listen(int port_number, xtcp_protocol_t protocol)` **Parameters**: `port_number` The local port number to listen to `protocol` The protocol to connect with (XTCP_PROTOCOL_TCP or XTCP_PROTOCOL_UDP)
	Function: **unlisten** **Description**: Stop listening to a particular incoming port. **Type**: `void unlisten(unsigned port_number)` **Parameters**: `port_number` local port number to stop listening on
	Function: **close** **Description**: 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. **Type**: `void close(const xtcp_connection_t &conn)` **Parameters**: conn The connection structure to be passed in that will contain all the connection information.

Continued on next page

Type	xtcp_if (continued)

Function	abort
Description	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.
Type	`void abort(const xtcp_connection_t &conn)`
Parameters	`conn` The connection structure to be passed in that will contain all the connection information.

Function	connect
Description	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.
Type	`void connect(unsigned port_number, xtcp_ipaddr_t ipaddr, xtcp_protocol_t protocol)`
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)

Continued on next page

Type	xtcp_if (continued)

Function	send
Description	Send data to the connection.
Type	`void send(const xtcp_connection_t &conn, char data[], unsigned len)`
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.

Function	join_multicast_group
Description	Subscribe to a particular IP multicast group address.
Type	`void join_multicast_group(xtcp_ipaddr_t addr)`
Parameters	addr: The address of the multicast group to join. It is assumed that this is a multicast IP address.

Function	leave_multicast_group
Description	Unsubscribe from a particular IP multicast group address.
Type	`void leave_multicast_group(xtcp_ipaddr_t addr)`
Parameters	addr: The address of the multicast group to leave. It is assumed that this is a multicast IP address.

Continued on next page

Type	xtcp_if (continued)
	Function: set_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 set_appstate(const xtcp_connection_t &conn, xtcp_appstate_t appstate)` **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.
	Function: bind_local_udp **Description**: Bind the local end of a connection to a particular port (UDP). **Type**: `void bind_local_udp(const xtcp_connection_t &conn, unsigned port_number)` **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.

Continued on next page

Type	xtcp_if (continued)

Function	bind_remote_udp
Description	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

Type	`void`
	`bind_remote_udp(const xtcp_connection_t &conn,`
	`                xtcp_ipaddr_t ipaddr,`
	`                unsigned port_number)`

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

Function	request_host_by_name
Description	Request a hosts IP address from a URL. LWIP ONLY.

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

Parameters	`hostname` The human readable host name, e.g. “www.xmos.com” `name_len` The length of the hostname in characters

Function	get_ipconfig
Description	Fill the provided ipconfig address with the current state of the server.

Type	`void`
	`get_ipconfig(xtcp_ipconfig_t &ipconfig)`

Parameters	`ipconfig` IPconfig to be filled.

APPENDIX A - Known Issues

The library does not support IPv6.

APPENDIX B - TCP/IP library change log

B.1 6.0.0

- CHANGE: Unified the branches of uIP and lwIP as the backend of the XTCP stack. The default is uIP. To change the stack, define XTCP_STACK in your makefile to be either UIP or LWIP. Then, instead of calling xtcp(...), call either xtcp_uip(...) or xtcp_lwip(...) respectively.
- CHANGE: The interface between the client and server is now event-driven rather than polling.
- CHANGE: Channels have been replaced by interfaces as communication medium between client and server.
- REMOVED: The following xtcp_event_types: XTCP_PUSH_DATA, XTCP_REQUEST_DATA, XTCP_POLL, XTCP_ALREADY_HANDLED
- CHANGE: The fields of packet_length and client_num have been added to the xtcp_connection_t structure.
- REMOVED: xtcp_connection_t no longer has a xtcp_connection_type_t field.
- REMOVED: The ability to pause a connection
- REMOVED: The ability to partially acknowledge a packet
- REMOVED: Support for IPv6
- REMOVED: the ability to send with an index. This functionality is easily replicated with a call to send() with the pointer of the array index location, i.e. &(data[index]).
- REMOVED: Support for XTCP_EXCLUDE_* macros which reduced functionality in order to save code size
- FIXED: Problem where ethernet packets smaller than 64 bytes would be incorrectly padded with uIP.
- Changes to dependencies:
  - lib_crypto: Removed dependency

B.2 5.1.0

- ADDED: Support for using lib_wifi to provide the physical transport

B.3 5.0.0

- ADDED: Port of LwIP TCP/IP stack
- Changes to dependencies:
  - lib_crypto: Added dependency 1.0.0

B.4 4.0.3

- ADDED: Support to enable link status notifications

B.5 4.0.2

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

B.6 4.0.1

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

B.7 4.0.0

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

B.8 Legacy release history

B.9 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.10 3.2.0

- Added IPv6 support

B.11 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.12 3.1.4

- Updated ethernet dependency to version 2.2.5

B.13 3.1.3

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

B.14 3.1.2

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

B.15 3.1.1

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

B.16 3.1.0

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

B.17 3.0.1

- Updated to use latest sc_ethernet package

B.18 3.0.0

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

B.19 2.0.1

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

B.20 2.0.0

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

B.21 1.3.1

- Initial implementation