AN02044: Using XMOS TCP/IP Library for UDP-based Networking

Publication Date: 2025/11/24
Document Number: XM-015354-AN v2.0.0

IN THIS DOCUMENT

1 Introduction

Transmission Control Protocol/Internet Protocol (TCP/IP) is an internetworking protocol that allows cross-platform or heterogeneous networking. It manages the flow of data in packets with headers giving the source and destination information. It provides a reliable stream delivery using sequenced acknowledgment and error detection technique. TCP/IP offers the *User Datagram Protocol* (UDP), a minimal transport protocol, wherein the packet headers contain just enough information to forward the datagrams and their error checking. UDP does not support flow control and acknowledgment.

1.1 XMOS TCP/IP

The XMOS TCP/IP component `lib_xtcp` provides a IP/UDP/TCP stack that connects to the XMOS ethernet component `lib_ethernet`. It enables several clients to connect to it and send and receive on multiple TCP or UDP connections. The stack has been designed for a low memory embedded programming environment and despite its low memory footprint provides a complete stack including ARP, IPv4, UDP, TCP, DHCP, ICMP and IGMP protocols. The library provides a TCP/IP stack (LwIP) with modifications to compile for the XCORE architecture and communicate between tasks using interfaces.

1.2 Block diagram

The application firmware organization is shown in Fig. 1.

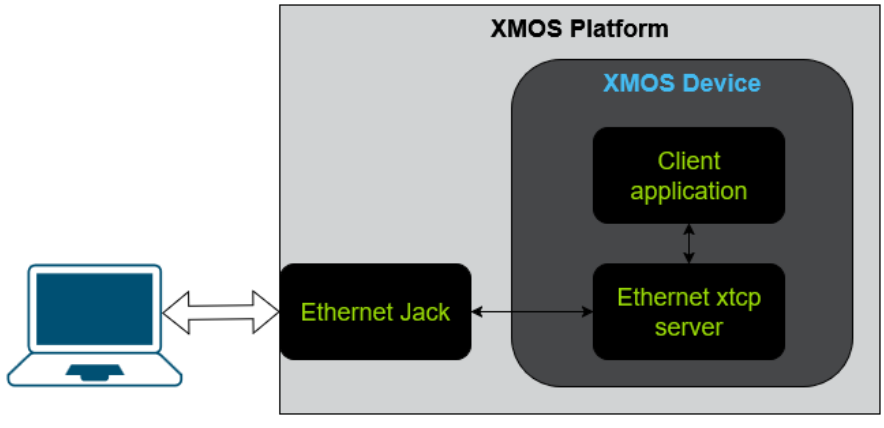

Fig. 1: Networking with an XMOS device

2 TCP/IP Application Configuration

This section describes the configuration of the application and the TCP/IP stack used in this example.

2.1 Project Configuration

The example uses the TCP/IP implementation in the **lib_xtcp** library. It depends on **lib_board_support** for the PHY configuration on the *XK-ETH-316-DUAL* board. It depends on **lib_logging** for lightweight debugging using *debug_printf()*.

These dependencies are specified in **APP_DEPENDENT_MODULES** in the application's *CMakeLists.txt*:

```
set(APP_DEPENDENT_MODULES   "lib_xtcp(7.0.0)"
                            "lib_board_support(1.4.0)"
                            "lib_logging(3.4.0)")
```

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

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

2.2 Allocating hardware resources

The TCP/IP stack connects to the RMII task from the Ethernet library which requires several ports to communicate with the Ethernet PHY. These ports are declared in the main program file (**main.xc**). In this demo the ports are set up for the Ethernet PHY connected to the *XK-ETH-316-DUAL* which uses a collection of ports to connect to the PHY defined in *xk-eth-xu316-dual.xn*:

```
port p_phy_rxd = RMII_PHY_0_RXD_4BIT;
port p_phy_txd = RMII_PHY_0_TXD_4BIT;
port p_phy_rxdv = RMII_PHY_0_RXDV;
port p_phy_txen = RMII_PHY_0_TX_EN;

port p_phy_clk = RMII_PHY_CLK_50M;
```

The MDIO Serial Management Interface (SMI) is used to transfer management information between MAC and PHY. This interface consists of two signals which are connected to two ports:

```
port p_smi_mdio = MDIO;
port p_smi_mdc = MDC;
```

2.3 IP configuration

The IP is configured via a structure passed to the **xtcp** task to suit the network where the XMOS device is used.

To assign a static IP address initialize with an address valid for the network the device will be connected to:

```
static xtcp_ipconfig_t ipconfig = {
    {192, 168, 200, 178}, /* IP address, 0 for DHCP */
    {255, 255, 255, 0},   /* submask, 0 for DHCP */
    {0, 0, 0, 0},         /* Gateway */
};
```

Alternatively it can be initialized to all zeros to enable DHCP, this will attempt to dynamically assign an IP address:

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


2.4 UDP reflector

The final part of the application is setup in `udp_reflect.xc` is a set of defines that assigns the port number for the incoming port. The size of data buffer is also set. These defines are used by the application.

```
// Defines
#define RX_BUFFER_SIZE 300
#define INCOMING_PORT 15533

#define ANY_ADDR {0, 0, 0, 0}
#define INIT_VAL -1
```


3 Application Detailed Description

3.1 The application main() function

For the UDP-based application example, the system comprises four tasks running on the multicore microcontroller.

The tasks perform the following operations.

- The RMII task `rmii_ethernet_rt_mac()` which handles RMII/Ethernet traffic.
- The SMI task `smi()` which handles the Serial Management Interface (SMI) for the Ethernet PHY. This is used to configure the Ethernet PHY via the `dual_ethernet_phy_driver()` task which manages the configuration of the PHY and communicates with the `lib_xtcp` stack.
- The XTCP server which handles the TCP/IP stack and provides the interface to the application protocol `xtcp_lwip`.
- The UDP reflector application `udp_reflect()` running in a single thread.

These tasks communicate via the use of xC channels and interface connections which allow data to be passed between application code running on separate logical threads.

Below is the source code for the main function of this application, which is taken from the source file `main.xc`

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

  par {
    on tile[0] : rmii_ethernet_rt_mac(
                   i_cfg, NUM_CFG_CLIENTS, i_rx, NUM_ETH_CLIENTS, i_tx, NUM_ETH_CLIENTS,
                   null, null,
                   p_phy_clk, p_phy_rxd, null, USE_UPPER_2B, p_phy_rxdv, p_phy_txen, p_phy_txd, null, USE_UPPER_
↪2B,
                   phy_rxclk, phy_txclk,
                   get_port_timings(PHY0_PORT_TIMINGS),
                   ETH_RX_BUFFER_SIZE_WORDS, ETH_RX_BUFFER_SIZE_WORDS, ETHERNET_DISABLE_SHAPER);

    on tile[1] : dual_ethernet_phy_driver(i_smi, i_cfg[CFG_TO_PHY_DRIVER], null);
    on tile[1] : smi(i_smi, p_smi_mdio, p_smi_mdc);

    // TCP component
    on tile[1] : xtcp_lwip(
                   i_xtcp, NUM_TCP_CLIENTS, null,
                   i_cfg[CFG_TO_TCP], i_rx[ETH_TO_TCP], i_tx[ETH_TO_TCP],
                   ipconfig);

    // The reflector thread
    on tile[0] : udp_reflect(i_xtcp[TCP_TO_APP_UDP]);
  }
  return 0;
}
```

3.2 The reflector function

The application code for receiving UDP packets of data from a network machine and reflecting it back to the same machine is implemented in the file `udp_reflect.xc`. The code performing these tasks is contained within the function `udp_reflect()` which is shown in the following code segments:

```
void udp_reflect(client xtcp_if i_xtcp) {
  debug_printf("Starting UDP reflect client: port %d\n", INCOMING_PORT);
  debug_printf("Waiting for link up...\n");

  int32_t responding_connection = INIT_VAL;  // The connection to the incoming port
  xtcp_ipaddr_t any_addr = ANY_ADDR;

  // The buffers for incoming data, outgoing responses
  char rx_buffer[RX_BUFFER_SIZE];
  char tx_buffer[RX_BUFFER_SIZE];
  int32_t dns_lookup = 0;
```

(continues on next page)

(continued from previous page)

```
int32_t dns_retries = 0;
const uint8_t hostname[] = "www.xmos.com";
xtcp_ipaddr_t dns_server = {8, 8, 8, 8};
```

In this segment of the code you can see the following.

- The buffers for holding the transmitted and received data are declared.
- The DNS server address and hostname to lookup are defined. The DNS lookup is optional and can be disabled if not required, by making `dns_lookup` non-zero.

This is followed by the client application event handling.

```
while (1) {
  int32_t client_conn;
  select {
    // Respond to an event from the tcp server
    case i_xtcp.event_ready():
      const xtcp_event_type_t event = i_xtcp.get_event(client_conn);
      switch (event) {
        case XTCP_EVENT_NONE:
          // No event to process
          break;

        case XTCP_IFUP:
          debug_printf("Link up event\n");
          // Show the IP address of the interface
          xtcp_ipconfig_t ipconfig = i_xtcp.get_netif_ipconfig(0);
          printip(ipconfig.ipaddr);

          responding_connection = i_xtcp.socket(XTCP_PROTOCOL_UDP);
          // Instruct server to listen on the incoming port
          xtcp_error_code_t listen_result = i_xtcp.listen(responding_connection, INCOMING_PORT, any_addr);
          if (listen_result != XTCP_SUCCESS) {
            debug_printf("Failed to listen on port %d, %i\n", INCOMING_PORT, listen_result);
          } else {
            debug_printf("Listening on port %d, %d\n", INCOMING_PORT, responding_connection);
          }
          break;

        case XTCP_IFDOWN:
```

Other event handling follows the above code. You can see the following in the code shown above.

- The `while` loop waits for an XTCP event and performs the appropriate function.
- When the link goes “up”, the XTCP server is instructed to listen for data on the incoming port.

4 The UDP example

4.1 Building the Application

This section assumes you have downloaded and installed the XMOS XTC tools (see *README* for required version). Installation instructions can be found here. Be sure to pay attention to the section Installation of required third-party tools.

The application uses the xcommon-cmake build system as bundled with the XTC tools.

The file *CMakeLists.txt* in the *app_an02044* directory contains the application build configuration.

The `an02044` software ZIP package should be downloaded and extracted to a chosen working directory.

To configure the build run the following from an XTC command prompt, this should only need to be run once:

```
cd an02044
cd app_an02044
cmake -G "Unix Makefiles" -B build
```

All required dependencies are included in the software package. If any dependencies are missing, they will be retrieved automatically during this step.

Finally, the application binaries can be built using `xmake`:

```
xmake -C build
```

Binary artifacts (.xe files) will be generated under the appropriate subdirectories of the `app_an02044/bin` directory — one for each supported build configuration.

For subsequent builds, the `cmake` step may be omitted. If `CMakeLists.txt` or other build files are modified, `cmake` will be re-run automatically by `xmake` as needed.

4.2 Demo Hardware Setup

1. Connect an XTAG-4 to the *XK-ETH-316-DUAL* with a ribbon cable.
2. Connect the *XK-ETH-316-DUAL* to the PC or to a network switch using an ethernet cable.
3. Connect the XTAG to the host PC using a USB cable. This allows programming of the application.
4. Connect the *XK-ETH-316-DUAL* to the host PC using a USB-C cable. This supplies power to the board.

4.3 Running the example

Once the application has been built you need to download the application binary code onto the *xcore* development kit. Here you use the tools to load the application over JTAG onto the *xcore* multicore microcontroller.

From a XTC command prompt run the following command from the *app_an02044* directory:

```
xrun --xscope ./bin/app_an02044.xe
```

Alternatively the binary can be programmed into the non-volatile flash memory with the command:

```
xflash ./bin/app_an02044.xe
```

When running the program with *xrun* you will see the output shown below:

```
Starting UDP reflect client: port 15533
Waiting for link up...
Link up event
IP: 192.168.200.178
Listening on port 15533, 0
```

4.4 Testing data transfer

The UDP example can be tested by running the python script `test_udp.py`.

```
python test_udp.py 192.168.200.178
Connecting..
Connected
Sending message: hello, world
Recv'd message: HELLO, WORLD
Closed
```

This will connect to the XMOS device at the IP address specified and send a message to the UDP port 15533. The XMOS device will respond with the same message, with the characters translated to upper case. The *xrun* console will display output similar to that shown below:

```
Got data: 13 bytes, from 192.168.200.99:52703
Sent response: 13 bytes
```

test_udp.py test script

```
#!/usr/bin/python
# Copyright 2025 XMOS LIMITED.
# This Software is subject to the terms of the XMOS Public Licence: Version 1.

import argparse
import socket


parser = argparse.ArgumentParser(description='TCP tester')
parser.add_argument('ip', type=str, help="IP address")
args = parser.parse_args()

# This simple script sends a UDP packet to port 15533 at the
# IP address given as the first argument to the script
# This is to test the simple UDP example XC program
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:

    sock.settimeout(5)

    print("Connecting..")
    try:
        sock.connect((args.ip, 15533))
        print("Connected")

        msg = "hello, world!"
        print("Sending message: " + msg)
        sock.send(bytes(msg, "ascii"))

        chunk = sock.recv(1500)
        print("Recv'd message: " + chunk.decode("ascii"))

    except socket.timeout:
        print("No response received within the timeout period.")

print("Closed")
```

5 Further reading

- XMOS XTC Tools Installation Guide
  https://www.xmos.com/xtc-install-guide
- XMOS XTC Tools User Guide
  https://www.xmos.com/view/Tools-15-Documentation
- XMOS application build and dependency management system; xcommon-cmake
  https://www.xmos.com/file/xcommon-cmake-documentation/?version=latest
- XMOS TCP/IP Component
  https://www.xmos.com/libraries/lib_xtcp
- XMOS Layer 2 Ethernet MAC Component
  https://www.xmos.com/libraries/lib_ethernet
- XMOS logging Component
  https://www.xmos.com/libraries/lib_logging
- TCP/IP - Internet protocol suite
  https://en.wikipedia.org/wiki/Internet_protocol_suite
- IP addressing
  https://en.wikipedia.org/wiki/Internet_Protocol
- Ethernet Type
  https://en.wikipedia.org/wiki/EtherType
- UDP - User Datagram Protocol
  https://en.wikipedia.org/wiki/User_Datagram_Protocol