AN00121: Using XMOS TCP/IP Library for UDP-based Networking on xcore-200

Publication Date: 2025/7/24
Document Number: XM-008194-AN v3.0.0

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 (XTCP)

The XMOS TCP/IP component provides a IP/UDP/TCP stack that connects to the XMOS ethernet component. 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, IPv4LL, ICMP and IGMP protocols. The library provides two stacks (one based on the open-source uIP and one based on LWIP) with modifications to work efficiently on the XMOS architecture and communicate between tasks using xC channels. This application note is based on the uIP stack provided in the library.

1.2 Block diagram

The application firmware organization is shown in Fig. 1.

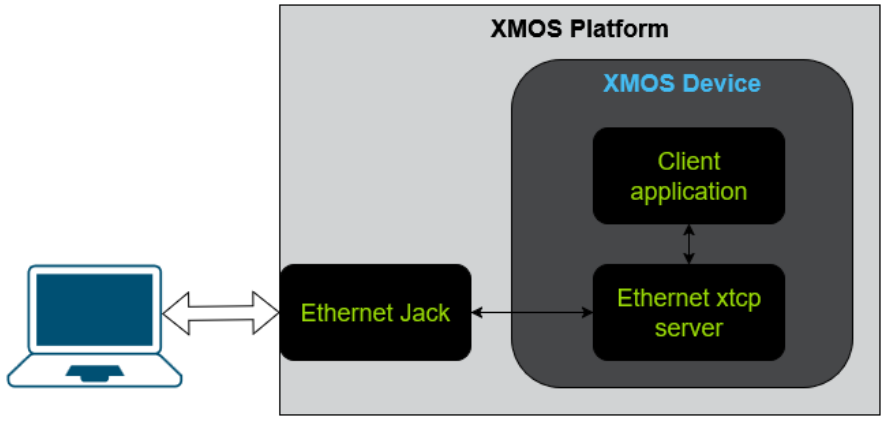

Fig. 1: Networking with an XMOS device

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_EVK_XE216* board. It depends in **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(6.2.0)"
                            "lib_board_support(1.3.0)"
                            "lib_logging(3.3.1)")
```

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 RGMII 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_EVK_XE216* which uses the Gigabit Ethernet hardware *shim* present in the *xcore* device and is accessed using a predefined struct of ports defined in the type *rgmii_ports_t*:

```
rgmii_ports_t rgmii_ports = on tile[1] : RGMII_PORTS_INITIALIZER;
```

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 = on tile[1] : XS1_PORT_1C;
port p_smi_mdc = on tile[1] : XS1_PORT_1D;
```

2.3 Stack Selection

For this example the uIP stack has be selected by default, to choose which stack to use the options are either **xtcp_uip()** or **xtcp_lwip()** in **main.xc**, this is best modified by changing the option in *CMakeLists.txt*:

```
# In the compiler flags define either XTCP_STACK_UIP or XTCP_STACK_LWIP
# according to which TCP stack is preferred to run the app note
set(APP_COMPILER_FLAGS      ${COMPILER_FLAGS_COMMON} -DXTCP_STACK_UIP)
```

This will then switch the stack and configuration in **main.xc** as shown below:

```
#ifdef XTCP_STACK_LWIP
    // TCP component
    on tile[0] : xtcp_lwip(
                    i_xtcp, 1, null,
                    i_cfg[CFG_TO_TCP], i_rx[ETH_TO_TCP], i_tx[ETH_TO_TCP],
                    mac_address_phy, null, ipconfig);
#elif defined XTCP_STACK_UIP
    // TCP component
    on tile[0] : xtcp_uip(
                    i_xtcp, NUM_TCP_CLIENTS, null,
                    i_cfg[CFG_TO_TCP], i_rx[ETH_TO_TCP], i_tx[ETH_TO_TCP],
                    mac_address_phy, null, ipconfig);
#else
#error "Please define either XTCP_STACK_LWIP or XTCP_STACK_UIP in APP_COMPILER_FLAGS"
#endif
```

2.4 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/AutoIP, 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 */
};
```

2.5 UDP reflector

The final part of the application is setup in `udp_reflect.xc` is a set of defines that assigns port numbers to the broadcast and incoming ports and also assigns the broadcast address. 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 BROADCAST_INTERVAL (6 * XS1_TIMER_HZ)
#define BROADCAST_PORT 15534
#define BROADCAST_ADDR {255, 255, 255, 255}
#define BROADCAST_MSG "XMOS Broadcast\n"
```

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 RGMII task `rgmii_ethernet_mac` which handles RGMII/Ethernet traffic.
- The RGMII config task `rgmii_ethernet_mac_config` which configures the RGMII task and handles the Ethernet PHY. This task runs on a logical core and communicates with the `lib_xtcp` stack.
- 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 `ar8035_phy_driver` task which manages the configuration of the PHY.
- The XTCP server which handles the TCP/IP stack and provides the interface to the application protocol. Either `xtcp_uip` or `xtcp_lwip`.
- The UDP reflector application `udp_reflect` running in a single core.

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];
  streaming chan c_rgmii_cfg;
  smi_if i_smi;

  par {
    on tile[1] : rgmii_ethernet_mac(
                   i_rx, NUM_ETH_CLIENTS, i_tx, NUM_ETH_CLIENTS,
                   null, null, c_rgmii_cfg, rgmii_ports,
                   ETHERNET_DISABLE_SHAPER);
    on tile[1].core[0] : rgmii_ethernet_mac_config(i_cfg, NUM_CFG_CLIENTS, c_rgmii_cfg);
    on tile[1].core[0] : ar8035_phy_driver(i_smi, i_cfg[CFG_TO_PHY_DRIVER]);

    on tile[1] : smi(i_smi, p_smi_mdio, p_smi_mdc);

#ifdef XTCP_STACK_LWIP
    // TCP component
    on tile[0] : xtcp_lwip(
                   i_xtcp, 1, null,
                   i_cfg[CFG_TO_TCP], i_rx[ETH_TO_TCP], i_tx[ETH_TO_TCP],
                   mac_address_phy, null, ipconfig);
#elif defined XTCP_STACK_UIP
    // TCP component
    on tile[0] : xtcp_uip(
                   i_xtcp, NUM_TCP_CLIENTS, null,
                   i_cfg[CFG_TO_TCP], i_rx[ETH_TO_TCP], i_tx[ETH_TO_TCP],
                   mac_address_phy, null, ipconfig);
#else
#error "Please define either XTCP_STACK_LWIP or XTCP_STACK_UIP in APP_COMPILER_FLAGS"
#endif

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

3.2 The UDP 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`. Further, a fixed packet is sent periodically to a broadcast IP address. The code performing these tasks is contained within the function `udp_reflect()` which is shown in the following pages:

```
void udp_reflect(client xtcp_if i_xtcp) {
  // The connection to the remote end we are responding to
  xtcp_connection_t responding_connection;
  // The connection out to the broadcast address
  xtcp_connection_t broadcast_connection;
  xtcp_ipaddr_t broadcast_addr = BROADCAST_ADDR;

  timer tmr;
  unsigned int time;

  // The buffers for incoming data, outgoing responses and outgoing broadcast messages
  char rx_buffer[RX_BUFFER_SIZE];
  char tx_buffer[RX_BUFFER_SIZE];
  char broadcast_buffer[RX_BUFFER_SIZE] = BROADCAST_MSG;

  // The length of the response the thread is sending
  int response_len;
  // The length of the broadcast message the thread is sending
  int broadcast_len;

  // Maintain track of two connections. Initially they are not initialized
  // which can be represented by setting their ID to -1
  responding_connection.id = INIT_VAL;
  broadcast_connection.id = INIT_VAL;

  debug_printf("Configuration: xcore-200\n");

  // Instruct server to listen and create new connections on the incoming port
  i_xtcp.listen(INCOMING_PORT, XTCP_PROTOCOL_UDP);

  tmr :> time;
```

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

- The network connections, broadcast address and the buffers for holding the transmitted and received data are declared.
- The XTCP server is instructed to listen and create new connections on the incoming port.

```
  while (1) {
    select {
      // Respond to an event from the tcp server
      case i_xtcp.packet_ready():
        // A temporary variable to hold connections associated with an event
        xtcp_connection_t conn;
        // A temporary variable to hold the length of the packet received from get_packet()
        unsigned data_len = 0;

        i_xtcp.get_packet(conn, rx_buffer, RX_BUFFER_SIZE, data_len);
        switch (conn.event) {
          case XTCP_IFUP:
            // Show the IP address of the interface
            union x_ip_addr ipconfig;
            i_xtcp.get_ipconfig(ipconfig.xtcp);
            debug_printf("dhcp: %s\n", ipaddr_ntoa(&ipconfig.ip4));

            // When the interface goes up, set up the broadcast connection.
            // This connection will persist while the interface is up
            // and is only used for outgoing broadcast messages
            i_xtcp.connect(BROADCAST_PORT, broadcast_addr, XTCP_PROTOCOL_UDP);
            break;

          case XTCP_IFDOWN:
```

Other event handling follows the above code, then the repeated broadcast.

```
      // This is the periodic case, it occurs every BROADCAST_INTERVAL timer ticks
      case tmr when timerafter(time + BROADCAST_INTERVAL) :> void:
        // A broadcast message can be sent if the connection is established
        // and one is not already being sent on that connection
        if (broadcast_connection.id != INIT_VAL) {
          debug_printf("Sending broadcast message\n");
          broadcast_len = strlen(broadcast_buffer);
          i_xtcp.send(broadcast_connection, broadcast_buffer, broadcast_len);
        }
        time += BROADCAST_INTERVAL;
        break;
```

You can see the following in the rest of the code.

- The **`while`** loop waits for an XTCP event and performs the appropriate function.
- Periodical broadcast of a fixed data is done through timer events.

4 The UDP example

4.1 Building the Application

The following 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_an00121* directory contains the application build configuration.

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

```
cd an00121
cd app_an00121
cmake -G "Unix Makefiles" -B build
```

Any missing dependencies will be downloaded by the build system as part of this configure step.

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

```
xmake -C build
```

This will build the application binary `app_an00121.xe` in the `app_an00121/bin` directory.

If *CMakeLists.txt* or other CMake configuration files have been modified, *xmake* will automatically trigger cmake to regenerate the build system as required.

4.2 Demo Hardware Setup

1. Connect an xTAG-3 to the *XK_EVK_XE216* XSYS port.
2. Connect the *XK_EVK_XE216* to the PC or to a network switch using an ethernet cable.
3. Connect 5V power to the *XK_EVK_XE216* using a USB cable.
4. Connect the XTAG to the host PC using a USB cable.

See *XMOS hardware setup for UDP demo*.

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_an00121* directory:

```
xrun ./bin/app_an00121.xe
```

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

```
xflash ./bin/app_an00121.xe
```

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

```
dhcp: 192.168.200.178
Configuration: xcore-200
New broadcast connection established:1
```

(continues on next page)

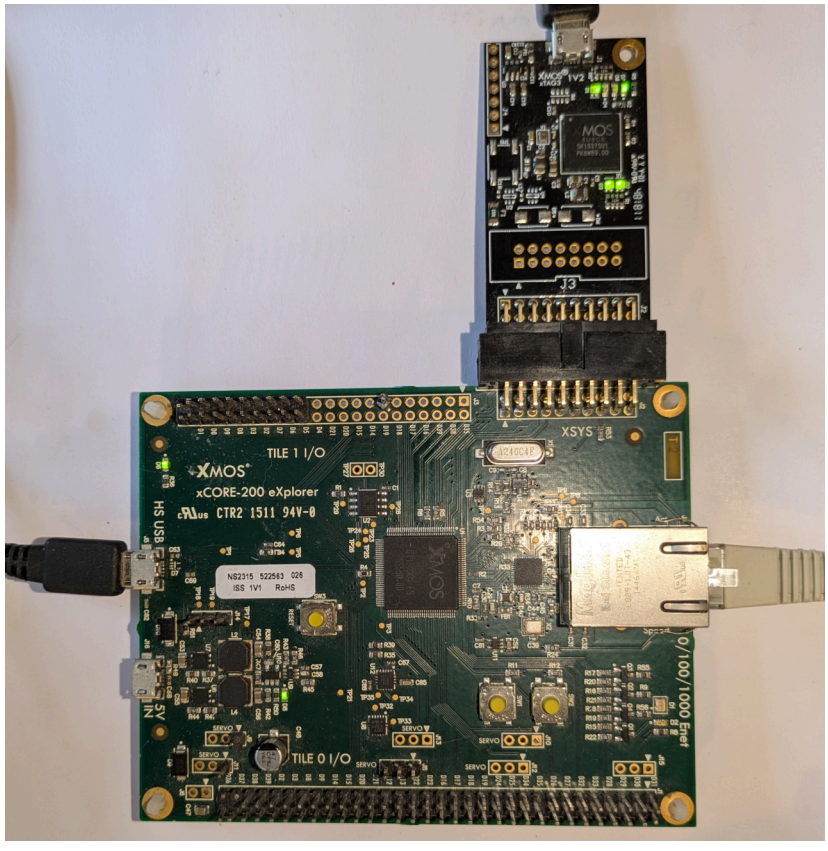

Fig. 2: XMOS hardware setup for UDP demo

(continued from previous page)

```
Sending broadcast message
Sent Broadcast
Sending broadcast message
Sent Broadcast
```

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
Revc'd message: b'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:

```
New connection to listening port: 15533
Got data: 12 bytes
Responding
```

(continues on next page)

(continued from previous page)

```
Sent Response
Closed connection: 2
```

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(20)
        print("Revc'd message: " + str(chunk))

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

print("Closed")
```

5 Further reading

- XMOS XTC Tools Installation Guide
  https://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
  http://en.wikipedia.org/wiki/EtherType
- UDP - User Datagram Protocol
  https://en.wikipedia.org/wiki/User_Datagram_Protocol