XMOS

AN00199: Gigabit ethernet example

Publication Date: 2025/9/19
Document Number: XM-008188-AN v1.1.1

IN THIS DOCUMENT

1 Introduction

The application note shows the use of the XMOS Ethernet library. The library allows multiple clients to access the Ethernet hardware. This application note uses the gigabit RGMII Ethernet MAC which uses 8 logical cores on a single tile (although one of those core can be shared with the application). The Ethernet library also provides 100Mbit MAC components which consume less resource on the xCORE device.

The gigabit MAC can handle line rate data packets through to the application layer and provides high performance streaming data, accurate packet timestamping, priority queuing and 802.1Qav traffic shaping.

RGMII provides the data transfer signals between the Ethernet PHY (Physical Layer Device or transceiver) and the xCORE device. The RGMII layer receives packets of data which are then routed by an Ethernet MAC layer to multiple processes running on the xCORE. SMI provides the management interface between the PHY and the xCORE device.

1.1 Block Diagram

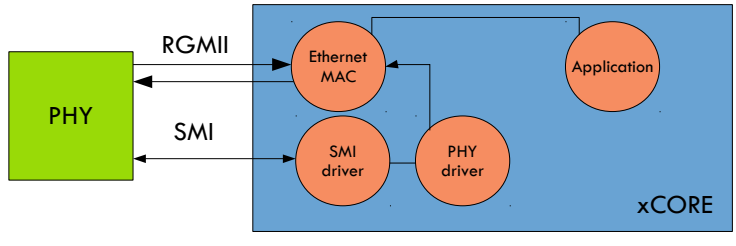

Fig. 1: Application block diagram

Fig. 1 shows the block diagram for the RGMII application. The application communicates with the Ethernet MAC that drives the RGMII data interface to the PHY. A separate PHY driver configures the PHY via the SMI serial interface.

2 Gigabit ethernet example

2.1 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_an00199* directory contains the application build configuration.

To configure the build run the following from an XTC command prompt:

```
cd app_an00199
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_an00199.xe` in the `app_an00199/bin` directory.

The example uses the MAC layer implementation in the `lib_ethernet` library. It depends on `lib_board_support` for the PHY configuration on the *XK-EVK-XE216* board. It depends on `lib_otpinfo` for reading the MAC address from the OTP memory of the xCORE device. These dependencies are specified in `APP_DEPENDENT_MODULES` in the application's *CMakeLists.txt*:

```
set(APP_DEPENDENT_MODULES  "lib_ethernet(4.1.0)"
                           "lib_board_support(1.4.0)"
                           "lib_otpinfo(2.2.1)"
```

2.2 Allocating hardware resources

The Ethernet library requires the user to declare the ports that the RGMII MAC uses to communicate with the Ethernet PHY. These ports are fixed on the xCORE-200 series so in the main program they are declared using the `RGMII_PORTS_INITIALIZER` macro provided by the library. This means the application just needs to declare a structure in `main.xc` to pass to the MAC component.

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

The application needs to control the reset line of the PHY and configure the phy via the MDIO Serial Management Interface (SMI). These are declared within `lib_board_support`.

The final ports used in the application are the ones to access the internal OTP memory on the xCORE. These ports are fixed and can be intialized with the `OTP_PORTS_INITIALIZER` macro supplied by the `lib_otpinfo` OTP reading library.

```
// These ports are for accessing the OTP memory
otp_ports_t otp_ports = on tile[0]: OTP_PORTS_INITIALIZER;
```

Note: The MAC address that the ICMP code uses is read from the OTP memory using `otp_board_info_get_mac()`. The IP address that the ICMP code uses is declared as an array in `main.xc`:

```
static unsigned char ip_address[4] = {192, 168, 1, 178};
```

These values can be altered to something that works on a given network.

2.3 The application main() function

The main function in the program sets up the tasks in the application.

```
int main()
{
  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]: xk_eth_xe216_phy_driver(i_smi, i_cfg[CFG_TO_PHY_DRIVER]);

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

    on tile[0]: icmp_server(i_cfg[CFG_TO_ICMP],
                            i_rx[ETH_TO_ICMP], i_tx[ETH_TO_ICMP],
                            ip_address, otp_ports);
  }
  return 0;
}
```

The `rgmii_ethernet_mac` and `rgmii_ethernet_mac_config` tasks communicate with the PHY and connect to the application via the three interfaces, the previously declared RGMII ports as arguments as well as an argument that determines whether the 802.1Qav traffic shaper is enabled.

The `smi` task is part of the Ethernet library and controls the SMI protocol to configure the PHY. It connects to the `ar8035_phy_driver` task which connects configuration of the PHY

The RGMII MAC is split into two tasks so that other tasks can be placed on the same logical core as the config task. In this example, the PHY driver task is placed on that core.

In this example, the `rgmii_ethernet_mac` task has two `null` arguments. These are two optional streaming channels parameters that are not used in this example but can be used for high priority, high speed traffic (see the Ethernet library user guide for details).

2.4 The PHY driver

The PHY driver task connects to both the Ethernet MAC (via the `ethernet_cfg_if` interface for configuration) and the SMI driver (via the `smi_if` interface). It is contained within `lib_board_support`.

The first action the drive does is wait for the PHY to power up and then configure the PHY. This is done via library functions provided by the Ethernet library.

The main body of the drive is an infinite loop that periodically reacts to a timer event in an xC `select` statement. A a set period it checks the state of the PHY over SMI and then informs the MAC of this state via the `eth.set_link_state` call. This way the MAC can know about link up/down events or change of link speed.

2.5 ICMP Packet Processing

The packet processing in the application is handled by the `icmp_server` task which is defined in the file `icmp.xc`. This function connects to the ethernet MAC via a transmit, receive and configuration interface:

```
[[combinable]]
void icmp_server(client ethernet_cfg_if cfg,
                 client ethernet_rx_if rx,
```

(continues on next page)

(continued from previous page)

```
                client ethernet_tx_if tx,
                const unsigned char ip_address[4],
                otp_ports_t &otp_ports)
{
```

The first thing the task performs is configuring its connection to the MAC. The MAC address is configured by reading a MAC address out of OTP (using the `otp_board_info_get_mac` function from the OTP reading library) and then calling the `set_macaddr` interface function:

```
  unsigned char mac_address[MACADDR_NUM_BYTES];
  ethernet_macaddr_filter_t macaddr_filter;

  // Get the mac address from OTP and set it in the ethernet component
  otp_board_info_get_mac(otp_ports, 0, mac_address);

  size_t index = rx.get_index();
  cfg.set_macaddr(0, mac_address);
```

After this, the task configures filters to determine which type of packets is will receive from the MAC:

```
  memcpy(macaddr_filter.addr, mac_address, sizeof mac_address);
  cfg.add_macaddr_filter(index, 0, macaddr_filter);

  // Add broadcast filter
  memset(macaddr_filter.addr, 0xff, sizeof mac_address);
  cfg.add_macaddr_filter(index, 0, macaddr_filter);

  // Only allow ARP and IP packets to the app
  cfg.add_ethertype_filter(index, ETH_FRAME_TYPE_ARP);
  cfg.add_ethertype_filter(index, ETH_FRAME_TYPE_IP);
```

The task then proceeds into an infinite loop that waits for a packet from the MAC and then processes it:

```
  while (1)
  {
    select {
    case rx.packet_ready():
      unsigned char rxbuf[ETHERNET_MAX_PACKET_SIZE];
      unsigned char txbuf[ETHERNET_MAX_PACKET_SIZE];
      ethernet_packet_info_t packet_info;
      rx.get_packet(packet_info, rxbuf, ETHERNET_MAX_PACKET_SIZE);

      if (packet_info.type != ETH_DATA)
      {
        debug_printf("Link: %s, speed %d\n", rxbuf[IF_STATUS_INDEX] ? "up" : "down", rxbuf[IF_SPEED_INDEX]);
      }
      else if (is_valid_arp_packet(rxbuf, packet_info.len, ip_address))
      {
        int len = build_arp_response(rxbuf, txbuf, mac_address, ip_address);
        tx.send_packet(txbuf, len, ETHERNET_ALL_INTERFACES);
        debug_printf("ARP response sent\n");
      }
      else if (is_valid_icmp_packet(rxbuf, packet_info.len, ip_address))
      {
        int len = build_icmp_response(rxbuf, txbuf, mac_address, ip_address);
        tx.send_packet(txbuf, len, ETHERNET_ALL_INTERFACES);
        debug_printf("ICMP response sent\n");
      }
      break;
    }
  }
}
```

The xC `select` statement will wait for the event `rx.packet_ready()` which is a receive notification from the MAC (see the Ethernet library user guide for details of the ethernet receive interface). When a packet arrives the `rx.get_packet` call will retreive the packet from the MAC.

After the packet is processed the `tx.send_packet` call will send the created reponse packet to the MAC.

Details of the packet processing functions `is_valid_arp_packet`, `build_arp_response`, `is_valid_icmp_packet` and `build_icmp_response` can be found in the `icmp.xc` file. The functions implement the ICMP protocol.

2.6 Demo Hardware Setup

- **To run the demo, connect the PC to the XTAG USB debug adapter to** xCORE-200 explorer XSYS connector
- Connect the XTAG to the host PC using a USB cable
- **Connect the ethernet jack to the host PC or to the network switch** using an ethernet cable.

Fig. 2 shows the required hardware setup.

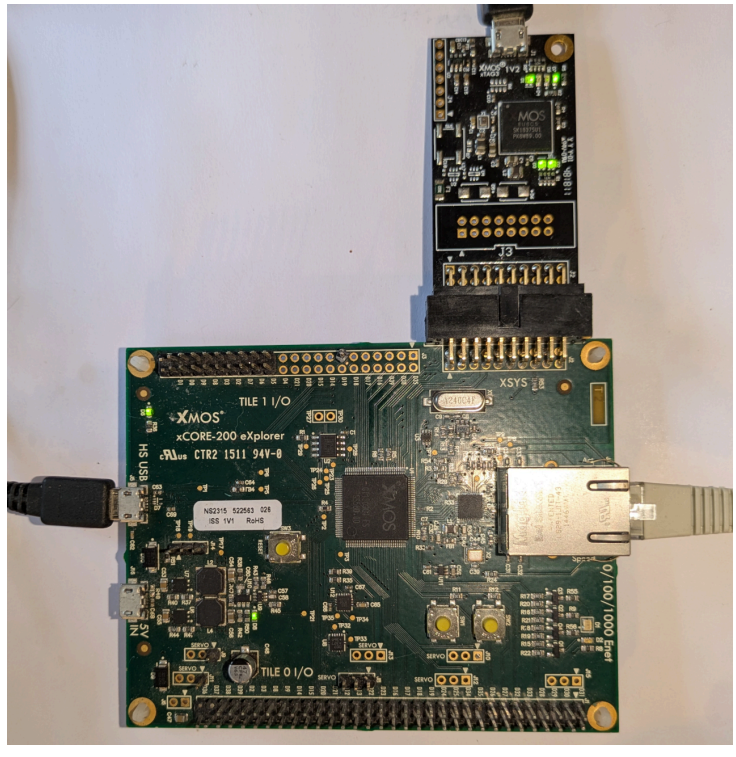

Fig. 2: Hardware Setup

2.7 Running the application

Once the `app_an00199.xe` application binary is compiled, it can be run on the *XK-EVK-XE216* board. The `xrun` tool is used from the command line to download and run the code on the xCORE device. In a terminal with XTC tools sourced, from the `app_an00199` directory, run:

```
xrun --xscope bin/app_an00199.xe
```

From a new terminal window, now ping the target:

```
$ ping 192.168.1.178
PING 192.168.1.178 (192.168.1.178) 56(84) bytes of data.
64 bytes from 192.168.1.178: icmp_seq=1 ttl=64 time=0.589 ms
64 bytes from 192.168.1.178: icmp_seq=2 ttl=64 time=0.294 ms
64 bytes from 192.168.1.178: icmp_seq=3 ttl=64 time=0.298 ms
...
```

The `xrun` console will print the status of the ICMP server and show ARP and ICMP responses:

```
ICMP server started at MAC 0:22:97:1:2:3, IP 192.168.1.178
ARP packet received
ARP response sent
ICMP packet received
ICMP response sent
ICMP packet received
ICMP response sent
ICMP packet received
ICMP response sent
...
```

2.8 Troubleshooting

If the host cannot ping the xCORE device, ensure that its wired Ethernet port is assigned a static IPv4 address. The configuration process varies by operating system but is typically done through the network settings. Fig. 3 shows how this looks on a MacBook Pro.

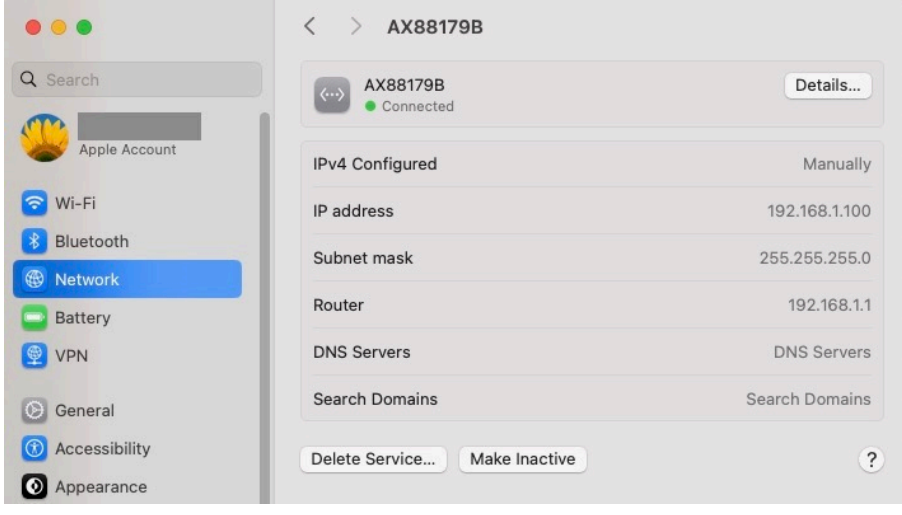

Fig. 3: Setting static IP address on a MacBook

3 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 Layer 2 Ethernet MAC Component
  https://github.com/xmos/lib_ethernet
- Ethernet Frame
  http://en.wikipedia.org/wiki/Ethernet_frame
- MAC address
  http://en.wikipedia.org/wiki/MAC_address
- Ethernet Type
  http://en.wikipedia.org/wiki/EtherType
- Internet Control Message Protocol
  http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol