XMOS

AN00120: 100Mbps RMII ethernet example

Publication Date: 2025/10/22
Document Number: XM-008190-AN v3.1.0

IN THIS DOCUMENT

1 Introduction

This application note demontrates the use of the XMOS Ethernet library. `lib_ethernet` provides the Media Access Control (MAC) function for the Ethernet stack that allows interfacing to MII, RMII or RGMII Ethernet PHYs. The application note uses `lib_ethernet` to provide a simple IP stack capable of responding to an ICMP ping message.

It demonstrates the real-time RMII MAC which uses four cores and provides high performance streaming data, accurate packet timestamping, priority queuing and 802.1Qav traffic shaping. RMII provides the data transfer signals between the Ethernet PHY (Physical Layer Device or transceiver) and the xCORE device.

The RMII 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.

The application communicates with the Ethernet MAC that drives the RMII data interface to the PHY. A separate PHY driver configures the PHY via the SMI serial interface. The application is tested on the *XK-ETH-316-DUAL* board which is an xcore.ai based board. It has two TI DP83825I PHYs connected to it and can support applications requiring up to two ethernet ports. This application instantiates one RMII MAC port that connects to DP83825I PHY_0 on the board.

2 100Mbit RMII ICMP example

2.1 Thread Diagram

Fig. 1 shows how the various functions are mapped to the threads inside the xCORE. The hardware platform maps the RMII pins to tile[0] and the SMI pins to tile[1] and hence the associated MAC and SMI tasks must be placed accordingly. Note that the PHY driver task can be combined with the SMI interface by the xC compiler so that it only occupies one hardware thread. The `icmp_server` application itself, has no need to access IO ports and communicates via interfaces and so may be placed on either tile, however is on tile[0] in this example.

The RMII ethernet MAC itself always consists of four hardware threads which use shared memory to communicate and hence are always place on the same tile as each other.

This particular example only uses 4 + 1 + 1 (6) of the total 16 hardware threads on the xCORE and less than 4% (40 kB) of the total RAM (1024 kB) and so has plenty of room for other functionality.

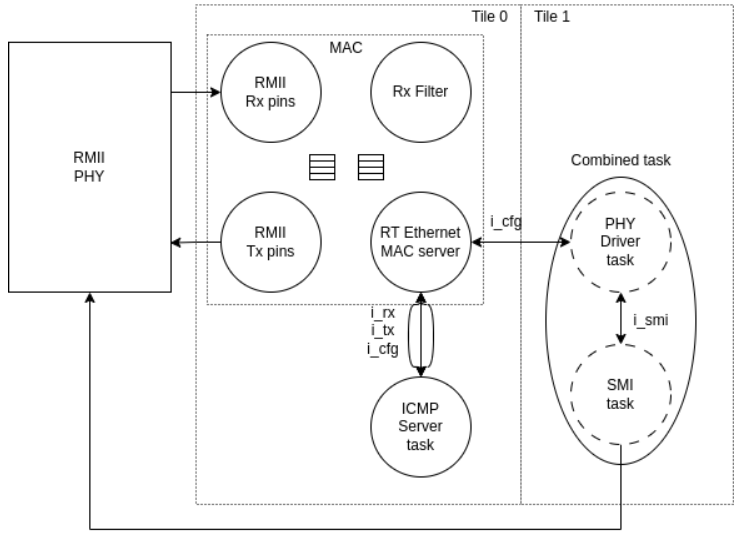

Fig. 1: Application thread diagram

2.2 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_an00120* directory contains the application build configuration.

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

```
cd app_an00120
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_an00120.xe` in the `app_an00120/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-ETH-316-DUAL* board. 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)"
```

2.3 Allocating hardware resources

The Ethernet library requires several ports to communicate with the Ethernet PHY. These ports are declared in the main program file (`main.xc`). In this examples the ports are set up for the PHY 0 on the *XK-ETH-316-DUAL* board which is the PHY in the top left corner of the board. Note that, in the default hardware configuration, it is also necessary to populate PHY_1 (top middle of the board) due to the way the 50 MHz ethernet clock is distributed.

Actual port names such as `XS1_PORT_1A` are specified in the XN file `xk-eth-316-dual.xn` and the source code refers to them with symbolic names such as `PHY_0_TX_EN`, with the actual to symbolic name mapping being specified in the XN file:

```
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;
```

In addition to the ports, two clock blocks are required, one each for the ethernet TX and RX clocks.

```
clock phy_rxclk = on tile[0]: XS1_CLKBLK_1;
clock phy_txclk = on tile[0]: XS1_CLKBLK_2;
```

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;
```

2.4 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];
  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);
    on tile[0]: icmp_server(i_cfg[CFG_TO_ICMP],
                            i_rx[ETH_TO_ICMP], i_tx[ETH_TO_ICMP],
                            ip_address, mac_address_phy);
  }
```

The `rmii_ethernet_rt_mac` creates a 100M RMII MAC instance that connects to the PHY on board. It internally starts the four tasks, `rmii_master_rx_pins_4b`, `rmii_master_tx_pins`, `mii_ethernet_filter` and `mii_ethernet_server`, that make up the MAC implementation. These tasks handle communicating with

the PHY at the pin level (`rmii_master_rx_pins_4b`, `rmii_master_tx_pins`), filtering received packets based on a MAC address lookup table based filtering, moving them into the receive queues (`mii_ethernet_filter`) and communicating with the client processes, facilitating packet transfer between the clients and the network (`mii_ethernet_server`).

The `rmii_ethernet_rt_mac` tasks takes the previously declared ports as arguments as well as the required buffer size for the packet buffer within the MAC. In addition, a structure provided by `lib_board_support` is passed in which configures the clock-block and pad delays to ensure the capture windows (setup and hold times) are met for this relatively fast IO interface.

The `smi` task is part of the Ethernet library and controls the SMI protocol to configure the PHY. It connects to the `dual_ethernet_phy_driver` task which handles configuration of the PHY and monitors the line state.

The `dual_ethernet_phy_driver` function is implemented in `lib_board_support` and it configures the PHY over the SMI interface.

Note

The MAC address that the ICMP code uses is declared as an array in `main.xc`:

```
static unsigned char mac_address_phy[MACADDR_NUM_BYTES] = {0x00, 0x22, 0x97, 0x01, 0x02, 0x03}
```

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.

2.5 The PHY driver

The PHY drive task `dual_ethernet_phy_driver` connects to both the Ethernet MAC (via the `ethernet_cfg_if` interface for configuration) and the SMI driver (via the `smi_if` interface):

```
    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);
```

The first action the driver 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 and specific register initialisation required in this instance.

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

2.6 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,
                 client ethernet_tx_if tx,
                 const unsigned char ip_address[4],
                 const unsigned char mac_address[MACADDR_NUM_BYTES])
{
```

The first thing the task performs is configuring its connection to the MAC. The MAC address is configured by calling the `set_macaddr` interface function:

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

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

```
// Add broadcast filter
memset(macaddr_filter.addr, 0xff, MACADDR_NUM_BYTES);
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 retrieve the packet from the MAC.

After the packet is processed the `tx.send_packet` call will send the created response 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.

2.7 Demo Hardware Setup

The demo uses the *XK-ETH-316-DUAL* board. Fig. 2 shows the hardware.

2.8 Running the application

Once the `app_an00120.xe` application binary is compiled, it can be run on the *XK-ETH-316-DUAL* 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_an00120` directory, run:

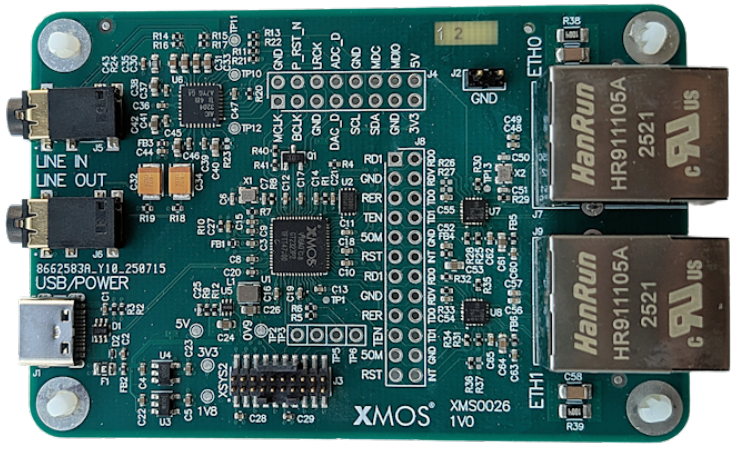

Fig. 2: Hardware Setup for XMOS RMII Ethernet library demo

```
xrun --xscope bin/app_an00120.xe
```

Once this command has executed the application will be running on the xCORE device.

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
...
```

2.9 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.

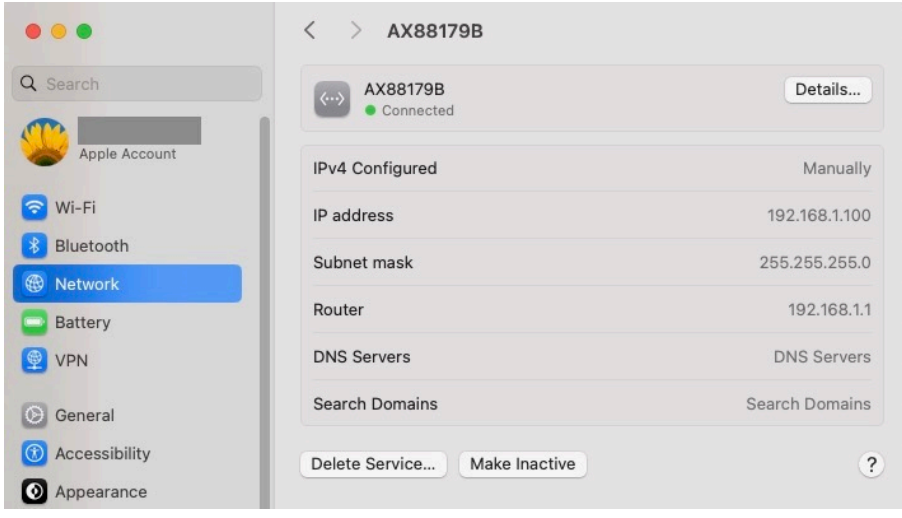

Fig. 3: Setting static IP address on a MacBook

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