ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

How to Integrate an lwIP TCP/IP Stack into Embedded Applications

Anju Puthenpurayil, Central Applications Engineer

Abstract

The use of a TCP/IP stack is widespread in Ethernet communication interfaces for local and wide area networks. Lightweight TCP/IP (lwIP) is a scaled down implementation of the TCP/IP protocol focused on reducing RAM usage. This article provides guidance on integrating the lwIP TCP/IP stack into an embedded application, ultimately streamlining the development process and saving time and effort.

Introduction

The lightweight TCP/IP (lwIP) stack is a compact implementation of the TCP/IP protocol suite tailored to minimize RAM usage, making it well-suited for embedded systems. It provides three distinct application programming interfaces (APIs):

- Low level raw APIs
- Higher level netconn APIs
- BSD style socket APIs

This article will focus exclusively on examples using the raw API interface. The raw API is event-driven and is designed to operate without an underlying operating system. Applications employing the raw API implement callback functions that are triggered by core events.

Despite being more intricate than the socket API, the raw API offers significantly higher throughput due to its lower overhead.

Various demo examples built on the lwIP TCP/IP stack will be discussed utilizing Analog Devices' MAX32570 microcontroller. The initial section delves into a ping demo, demonstrating how to ping the device from a PC. The subsequent section provides insights into a TCP Echo server example, which serves as a rudimentary server demo, useful for testing TCP connections.

ADI's MaximSDK software development kit contains the necessary software and tools to develop firmware for ADI's MSX32xxx microcontrollers. It includes an lwIP stack library file, "MaximSDK\Libraries\lwIP". Figure 1 shows the lwIP library files in the MaximSDK folder structure.

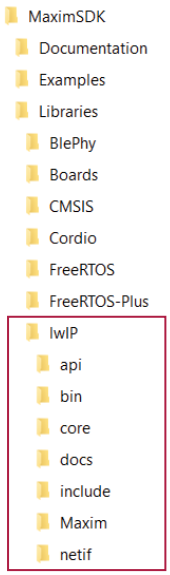

Figure 1. ADI's MaximSDK lwIP library files

The lwIP folder contains different subfolders:

- API folder (netconn and socket APIs)
- Core folder (lwIP core files including "tcp.c", "ip.c," etc.)
- Netif folder (network interface files)
- Include folder (all lwIP include files)
- Maxim folder (customized mac driver for ADI's microcontroller)

The lwIP architecture follows a TCP/IP model structure. The TCP/IP protocol is a combination of different protocols at various layers. TCP/IP is normally considered to be a 4-layer system as shown in Figure 2.

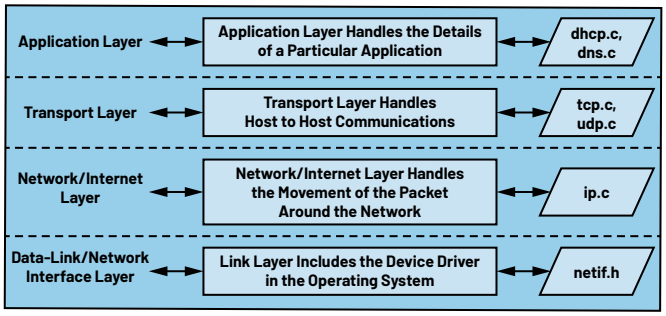

Figure 2. TCP/IP protocol layers.

An lwIP project always contains a configuration file named "lwipopts.h" and a default configuration file called "opt.h". The "opt.h" file contains all the default stack configurations and its module configurations, whereas the "lwipopts.h" allows the user to fully configure the stack and its modules. Note that this file does not include all the possible lwIP options. So, if a configuration is not defined in the "lwipopts.h" file, the default configuration defined in the "opt.h" will be considered.

Similarly, the lwIP library has one application specific header file called "lwipcfg.h". The controller's IP address, gateway address, netmask address, and MAC address should be defined in the "lwipcfg.h" file as shown in Figure 3.

```
/** Default Static IP Address Configuration: (Configured by User) **/
#define LWIP_PORT_INIT_IPADDR(addr)     IP4_ADDR((addr), 192,168,100,200)
#define LWIP_PORT_INIT_GW(addr)         IP4_ADDR((addr), 192,168,100,100)
#define LWIP_PORT_INIT_NETMASK(addr)    IP4_ADDR((addr), 255,255,255,0)

/** Default Ethernet MAC Address Values: (Configured by User) **/
#define MAC_BYTE1           0x00
#define MAC_BYTE2           0x18
#define MAC_BYTE3           0x80
#define MAC_BYTE4           0x03
#define MAC_BYTE5           0x25
#define MAC_BYTE6           0x70
#define MAC_LEN             6

#endif /* LWIP_LWIPCFG_H */
```

Figure 3. lwipcfg header file.

To establish a connection between lwIP and the underlying hardware drivers, a platform-specific adaptation layer is necessary. For instance, when implementing the lwIP stack for a microcontroller, a tailored driver is required to bridge the lwIP stack and the microcontroller's Ethernet MAC drivers. This custom driver should encompass the following functionalities:

- Initialization function: This function is responsible for initializing the MAC driver specific to the microcontroller.
- Send function: It facilitates the transfer of data received from the TCP stack to the Ethernet MAC driver for subsequent transmission.
- Receive function: This handles the forwarding of packets received from the Ethernet MAC driver to the TCP stack.

For the ADI microcontroller, a pre-existing customized driver can be found in the MaximSDK at the path "MaximSDK\Libraries\lwIP\Maxim\mxc_eth.c." This driver serves as a wrapper around the microcontroller's own Ethernet MAC (EMAC) peripheral library, which is located in the peripheral drivers at "C:\MaximSDK\Libraries\PeriphDrivers\Source\EMAC."

Ping Example

The "ping" command is a simple tool used for network troubleshooting. It performs an Internet Control Message Protocol (ICMP) echo request by sending a signal to a specific IP address and waits for a reply. When the destination receives this request, it replies with an echo reply packet. This section will explain how to perform a basic ping test from a Windows PC to a microcontroller to check their connectivity. It will also cover how to use the microcontrollers' ping module to communicate with a PC.

Here's how the Windows ping utility works:

- It sends four data packets to the microcontroller and waits for a response.
- The microcontroller sends these data packets back to the PC as a reply known as echo reply requests.

To run the ping test:

- Connect microcontroller EVKIT to a PC using an Ethernet cable.
- Open the command prompt and type "ping <IP address of microcontroller>" and press enter.

A reply in the command prompt such as that depicted in Figure 4 implies proper connectivity between the PC and the microcontroller.

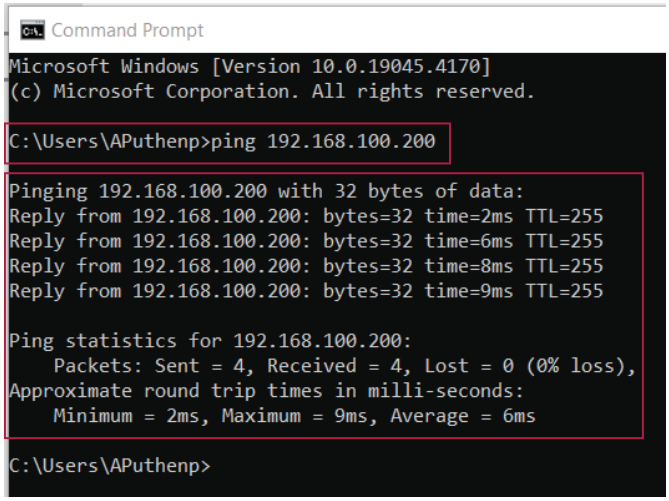

Figure 4. Ping output in command prompt.

To Test Ping from the Microcontroller

The "lwIP_Ping" file is the ping example for the ADI's MAX32570 microcontroller included within the MaximSDK. It can be found at "C:\MaximSDK\Examples\MAX32570\lwIP_Ping" and the following guidance is provided:

- The IP address of the microcontroller is set using "lwipcfg.h" file. The IP address of the microcontroller and the PC should be in the same series.
- The IP address of the PC should be provided as the gateway address in the microcontroller "lwipcfg.h" file.
- Connect the PC and MAX32570 EVKIT using an Ethernet cable.
- Run the ping example code.
- Open the serial terminal in eclipse (Window-> Show view -> Terminal). The terminal will show a ping result if the ping is successful as shown in Figure 5.

```
Link Status: Up
ping: send 192.168.100.100
ping: recv 192.168.100.100 0 ms
ping: send 192.168.100.100
ping: recv 192.168.100.100 0 ms
ping: send 192.168.100.100
ping: recv 192.168.100.100 0 ms
ping: send 192.168.100.100
ping: recv 192.168.100.100 0 ms
ping: send 192.168.100.100
```

Figure 5. Ping output in serial terminal.

The command prompt shows only ping statistics. To view the actual data sent, a tool called Wireshark is needed. Wireshark captures packets from a network connection. After opening Wireshark, the Ethernet option should be selected. Details will be shown such as source and destination MAC addresses, source and destination IP addresses, communication protocol, and additional data sent. This information is displayed in Wireshark as depicted in Figure 6.

As shown in the example, the data sent is 0x00, 0x01... to 0x1F. However, what if the user wants to change the data sent?

438	44.620241	192.168.100.100	192.168.100.200	ICMP	74	Echo (ping) request id=0x0001, seq=1396/29701, t
439	44.620502	192.168.100.200	192.168.100.100	ICMP	74	Echo (ping) reply id=0x0001, seq=1396/29701, t
440	44.650921	192.168.100.100	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.252 for an
441	44.651014	fe80::e6dd:921c:4b9…	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
442	44.746968	192.168.100.100	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
443	44.746974	192.168.100.100	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
444	45.542972	192.168.100.200	192.168.100.100	ICMP	74	Echo (ping) request id=0xafaf, seq=4/1024, ttl=2

```
> Frame 444: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{9CC74755-B309-4D77-99D5-A5D2058
> Ethernet II, Src: MaximInt_03:25:70 (00:18:80:03:25:70), Dst: CeLink_61:94:31 (a0:ce:c8:61:94:31)
> Internet Protocol Version 4, Src: 192.168.100.200, Dst: 192.168.100.100
> Internet Control Message Protocol
```

```
0000  a0 ce c8 61 94 31 00 18  80 03 25 70 08 00 45 00   ···a·1·· ··%p··E·
0010  00 3c 00 03 00 00 ff 01  71 40 c0 a8 64 c8 c0 a8   ·<······ q@··d···
0020  64 64 08 00 57 4b af af  00 04 00 01 02 03 04 05   dd··WK·· ········
0030  06 07 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ········ ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f                     ········ ··
```

Figure 6. Ping data packet in Wireshark.

Modify Data Sent from Microcontroller Ping

The data to be sent with ping is set in the "ping. c" file. The "Ping.c" file is the ping sender module. The size of the data to be sent is set using "PING_DATA_SIZE" in the "ping.c" file. The data size is set as 32 bytes as shown in Figure 7.

```
/** ping additional data size to include in the packet */
#ifndef PING_DATA_SIZE
#define PING_DATA_SIZE 32
#endif
```

Figure 7. Ping data packet size.

The data to be sent is also defined in "ping.c" file. The additional data buffer is filled with some data as shown in Figure 8 "0x00, 0x01, 0x02...to 0x1F."

```
ICMPH_TYPE_SET(iecho, ICMP_ECHO);
ICMPH_CODE_SET(iecho, 0);
iecho->chksum = 0;
iecho->id = PING_ID;
iecho->seqno = lwip_htons(++ping_seq_num);

/* fill the additional data buffer with some data */
for (i = 0; i < data_len; i++) { ((char *)iecho)[sizeof(struct icmp_echo_hdr) + i] = (char)i; }

iecho->chksum = inet_chksum(iecho, len);
```

Figure 8. Ping data packet.

Depending on the application, if the user wants to change the data, the data buffer can be modified in the "ping .c" file. For example, changing all 32 bytes of data to "0x01, 0x01...0x01" is shown in Figure 9.

```
/* fill the additional data buffer with some data */
for (i = 0; i < data_len; i++) { ((char *)iecho)[sizeof(struct icmp_echo_hdr) + i] = (char)1; }

iecho->chksum = inet_chksum(iecho, len);
```

Figure 9. Modified ping data packet.

The modified "ping .c" file provides results in Wireshark as shown in Figure 10. The data is then updated with the new parameters.

```
0000  a0 ce c8 61 94 31 00 18  80 03 25 70 08 00 45 00   ···a·1·· ··%p··E·
0010  00 3c 00 04 00 00 ff 01  71 3f c0 a8 64 c8 c0 a8   ·<······ q?··d···
0020  64 64 08 00 38 3b af af  00 05 01 01 01 01 01 01   dd··8;·· ········
0030  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01   ········ ········
0040  01 01 01 01 01 01 01 01  01 01                     ········ ··
```

Figure 10. Modified ping data packet in Wireshark

TCP Echo Server

The ping examples utilize ICMP to determine the responsiveness of a target system. It sends the intended recipient an echo request through the network using default data. When the destination address gets this request, it replies with an echo reply packet.

If a user wants to send customized data from one device to another, they use the TCP protocol for data transmission. The Echo service is a standard TCP functional mainly used to check reachability and identify routing problems. In this service, a server and client are established using TCP. When the server gets a message from a client, it sends that same message back.

In the MaximSDK, the "lwIP TCP" source file demonstrates how to use TCP functions within the lwIP library. In this scenario, the microcontroller acts as a TCP server and waits for a client request. The data sent from the client is echoed back. The "tcpecho_raw.c" application source files should be used within the TCP Echo server example. Follow the steps below to assign the TCP Echo server.

To set up the TCP Echo server:

- Create a socket
- Bind the socket to the advertised port number
- Once bound, it starts listening for incoming connections
- When a connection is requested, it accepts the request from the client machine
- The server then receives data from the client
- Lastly, it sends back the same data

Figure 11 shows a snippet of code giving an overview of the firmware structure, which is part of the main function. The config_emac function initializes the EMAC and the MXC_ETH_Init initializes the lwIP stack.

```
int main(void)
{
    ……
    ……
    /* EMAC init */
    result = config_emac();
    ……
    /* LWIP Stack init */
    result = MXC_ETH_Init(&lwIP_config);
    ……
    /* TCP Echo Server init */
    tcpecho_raw_init();
    /* Infinite loop */

    while (1) {
        result = MXC_ETH_Tick();

        if (result) {
            break;
        }
    }

    return result;
}
```

Figure 11. A snippet of code giving an overview of the firmware structure.

After the EMAC and lwIP stack initialization, the TCP Echo server is initialized using tcpecho_raw_init. The Echo server initialization structure is shown in Figure 12.

```
void tcpecho_raw_init(void)
{
    /* Create new tcp pcb */
    tcpecho_raw_pcb = tcp_new_ip_type(IPADDR_TYPE_ANY);

    if (tcpecho_raw_pcb != NULL) {
        err_t err;

      /* Bind tcp pcb to port 7 */
        err = tcp_bind(tcpecho_raw_pcb, IP_ANY_TYPE, 7);

        if (err == ERR_OK) {
            /* Start tcp listening for echo pcb  */
            tcpecho_raw_pcb = tcp_listen(tcpecho_raw_pcb);
            /* Initialize lwIP tcp accept callback function  */
            tcp_accept(tcpecho_raw_pcb, tcpecho_raw_accept);
        } else {
            /* abort? output diagnostic? */
        }
    } else {
        /* abort? output diagnostic? */
    }
}
```

Figure 12. The Echo server initialization structure.

Echo server initialization will create a new socket. It will then bind the assigned IP address and port number to the new socket. After binding, it will listen for a connection from the remote client.

To test the TCP server example, use echotool.exe PC client utility. The echotool.exe file should be saved in the C drive and the command prompt should be opened from the C drive. In client mode, it sends data to the server and checks whether it came back as shown in Figure 13. Be sure to use the echo tool in client mode to test the server demo.

How to Test the TCP Server Example:

- Ensure all connections are working properly.
- Build the example code using eclipse.
- Run the code in debug mode.
- Open the command prompt window on the remote PC.
- Enter the following in the command prompt:
  - *"C:\>echotool IP_address /p tcp /r 7 /n 15 /t 2 /d LwIP TCP echo server Example"*

    IP_address is the actual board IP address. The static IP address is 192.168.100.200
    /p tcp is the protocol (TCP protocol)
    /r is the actual remote port on the echo server (echo port)
    /n is the number of the echo requests
    /t is the connection timeout in seconds
    /d is the message to be sent for echo (for example, "LwIP TCP echo server Example")

```
C:\>echotool 192.168.100.200 /p tcp /r 7 /n 15 /t 2 /d LwIP TCP echo server Example

Hostname 192.168.100.200 resolved as 192.168.100.200

Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK
Reply from 192.168.100.200:7, time 0 ms OK

Statistics: Received=15, Corrupted=0
```

Figure 13. TCP Echo server output.

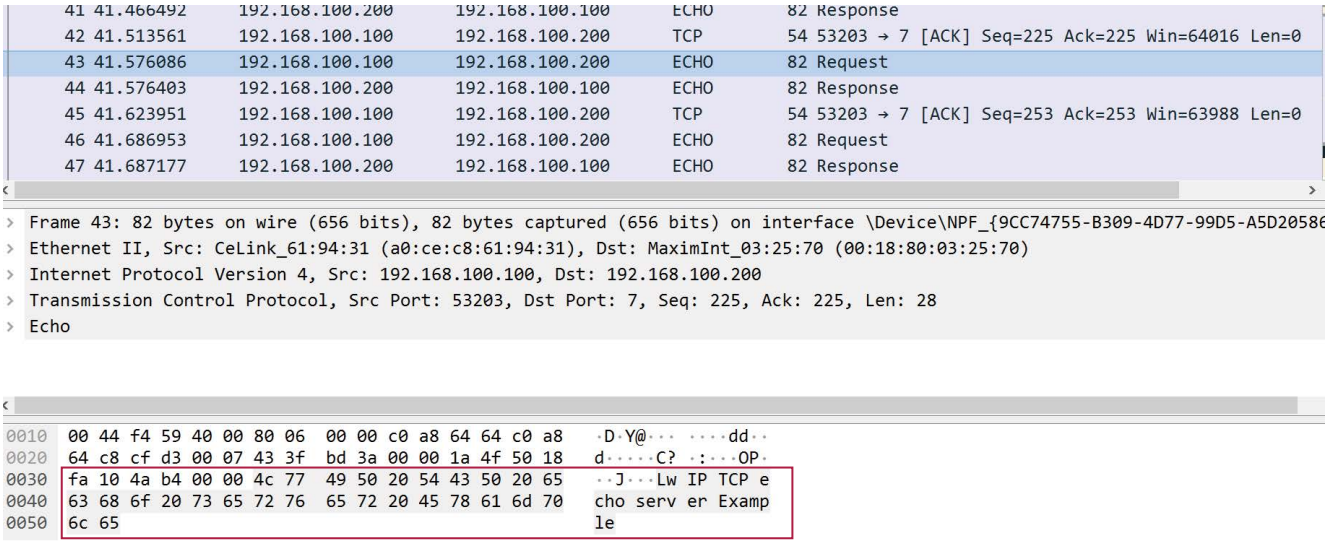

Figure 14. TCP Echo server output in Wireshark.

The TCP protocol and data sent through the network can be verified using the Wireshark software. Packets sent through the network will be available in Wireshark as shown in Figure 14. The data sent through the command prompt is "LwIP TCP echo server Example". The same data is seen in the Wireshark software.

Conclusion

Understanding and effectively utilizing the capabilities of the lwIP stack along with the ICMP-based ping tool and the TCP protocol opens up a wide array of possibilities for network diagnostics and data transmission. ADI's MAX32570 microcontroller and MaximSDK provide a robust foundation for implementing the lwIP stack and building reliable communication systems. By following the outlined examples in this article, network issues can be troubleshooted, creating seamless connections and data integrity assurance.

References

[1] nongnu.org/lwIP/2_1_x/index.html

wireshark.org/download.html

github.com/PavelBansky/EchoTool

About the Author

Anju Puthenpurayil is currently a central applications engineer in the CAC team at Analog Devices. Anju holds a master's degree in VLSI and embedded systems from Defence Institute of Advanced Technology (DIAT) Pune, India. Her commitment to stay up to date with industry trends and advancements makes the perfect contribution to the success of Analog Devices' product offerings.