Microchip TCP/IP Lite Stack

Authors: *Janaki Kuruganti,*
Marius Cristea
Microchip Technology Inc.

INTRODUCTION

This application note describes the structure and the interface for the Microchip Transmission Control Protocol/Internet Protocol (TCP/IP) lite stack library, and includes some simple demo applications. The purpose of the TCP/IP lite stack implementation is to provide optimized (low Flash and RAM footprint) TCP/IP stacks for microcontrollers with ≥8KB Flash (UDP only) and ≥16KB Flash (TCP/IP), while still having full functional TCP/IP v4 stack. The stack will allow customers to add wired communication and interoperability with other systems to their applications over Ethernet.

The Microchip TCP/IP lite stack is implemented in a configurable and modular way allowing users to include only the intended features or functionality to their application. The stack is written in C programming language and it is intended to be compiled with the MPLAB® XC8 compiler.

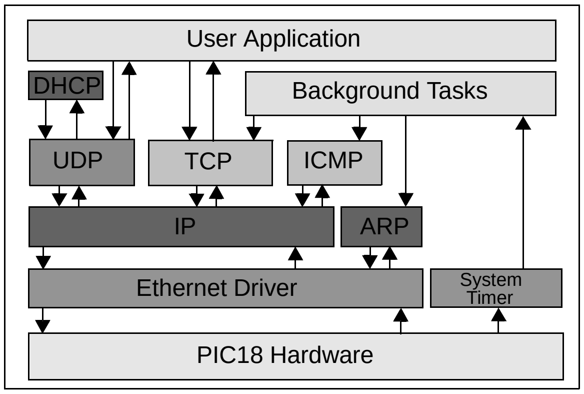
Acronyms

- TCP – Transmission Control Protocol
- ICMP – Internet Control Message Protocol
- DHCP – Dynamic Host Configuration Protocol
- MAC address – Media Access Control Address
- RAM – Random Access Memory
- IP – Internet Protocol
- ARP – Address Resolution Protocol
- UDP – User Datagram Protocol
- ACK – Acknowledgment Packet
- MSS – Maximum Segment Size
- RX – Receive
- TX – Transmit

TCP/IP STACK ARCHITECTURE

The TCP/IP lite library implementation is based on the TCP/IP communication model as shown in Figure 1:

FIGURE 1: TCP/IP COMMUNICATION MODEL



The TCP/IP stack is divided into multiple layers (Figure 1). Each layer in the Microchip TCP/IP lite stack can access directly one or more layers situated directly above or below it.

The TCP/IP stack needs a background task called periodically by the user, in order to handle asynchronous events like managing a timeout, checking the status for the Ethernet controller and parsing the received buffers.

The code implementing each protocol resides in a separate source file, while the services and Application Programming Interfaces (APIs) are defined through header/include files.

STACK CONFIGURATION

The stack uses C compiler definitions in the `tcpip_config.h` file, which makes it easily configurable by the user. The `tcpip_config.h` file contains multiple configuration parameters, though the user has to customize the following stack parameters in order to connect the board to the network:

1. MAC_ADDRESS:

The hardware MAC address (Media Access Control address) of the board should be globally unique for each Ethernet port. In case of the PICDEM2.net™ board there are two unique MAC addresses provided by Microchip on the back of the board. The default value is `{0x00, 0xDE, 0xAD, 0x00, 0xBE, 0XFF}`.

2. dhcpName:

It refers to the human readable board's name. It is used by the DHCP server to assign a human readable name to a MAC address. The default value is `PIC18F97J60 ETHERNET`.

3. DHCP_NAME_SIZE

It is the length of the dhcpName string. The size needs to be updated each time the dhcpName is changed. The default value is `20`.

4. ENABLE_TCP_DEBUG

It enables debug messages from the TCP protocol to be sent to the syslog module.

5. ENABLE_IP_DEBUG

It enables debug messages from the User Datagram Protocol (UDP) to be sent to the syslog module.

TCP/IP STACK BUFFER MANAGEMENT

Overview

The TCP/IP stack uses by default the least possible memory, so that users have maximum possible memory available to be allocated for their applications. Users are responsible for providing all the needed buffers for each TCP/IP protocol/connection used, as described further on.

Ethernet controller Random Access Memory (RAM) used by the TCP/IP stack:

- Ethernet received packets are kept into the Ethernet controller RAM memory. The Ethernet controller will receive and store multiple received Ethernet packets until the TCP/IP stack has time to process them. The buffer for each received packet is managed by the Ethernet controller automatically and a buffer descriptor will be available for the user. The Ethernet controller will start dropping the received packets if it has no more available memory.
- Ethernet packet to be transmitted is also built and kept in the Ethernet controller memory. The TCP/IP stack supports only one transmit (TX) packet at a time.

Buffers Used by the UDP Protocol

For creating UDP packets the stack allows the user to directly use the Ethernet controller RAM memory for storing the user payload. The user will call the API to start a UDP packet, transfer the payload and send the packet through the wire.

When receiving data over the UDP protocol, the Ethernet Controller will manage the buffer for the received packet. If the packet was received successfully and there is a user callback registered for the incoming port, the stack will call the registered function (callback) and give the user the opportunity to access the payload directly from the Ethernet controller. This will avoid copying the payload multiple times.

Buffers Used by the TCP Protocol

In case of TCP, the user needs to allocate some memory for each TCP connection. There are a few types of buffers needed by the TCP:

- The socket area, where the internal information about the TCP connection is kept.
- The memory area for each socket is created and passed to the stack by the user through the API.
- The receive (RX) and the transmit (TX) buffers for each TCP connection are created by the user and passed to the stack via the stack API. Each socket can have only one RX and one TX buffer at a time. The stack always needs one RX buffer available to receive data from the remote host. The stack functions only for a short period of time without the RX buffer, before asking for packet retransmission.

TCP/IP STACK LIMITATIONS

The TCP/IP stack has some limitations because of the limited memory, for both RAM and Flash, available on an 8-bit microcontroller.

Address Resolution Protocol (ARP)

The number of entries in the Address Resolution Protocol (ARP) table should be limited to eight. The limit of the maximum entries in the table is user configurable (refer to the `tcpip_config.h` file). The number of entries in the ARP table should be at least equal to the maximum number of device connections, otherwise there will be a performance degradation caused by the ARP request for each IP address that is not found in the ARP table.

IPv4

The stack supports only ICMPv4, UDPv4 and TCPv4 packets. All the other types of packets will be discarded by the stack.

The stack does not support IP fragmentation. Fragmented IP packets will be dropped by the stack. Fragmentation is required if the IP packet size is larger than the maximum transmit unit, usually 1500 bytes. If such packets have to be sent, the user will be required to break them down into multiple smaller sized packets to avoid any fragmentation.

The TCP/IP stack will not interpret the IP header options.

The TCP/IP stack does not support the loop-back address 127.0.0.1. The 127.0.0.1 address is handled as a general IPv4 address and the stack will try to find the MAC address associated with this IP address.

Currently, the stack does not support multi-casting. The user cannot send and receive packets with multi-cast addresses.

Internet Control Message Protocol (ICMP)

ICMP is an optional protocol and only the ICMP ECHO_REPLY message is implemented. The stack will reply only to an ICMP ECHO_REQUEST, all the other ICMP messages will be ignored.

User Datagram Protocol (UDP)

In order to save some memory, when UDP packets are transmitted, the stack allows building the packets directly inside the Ethernet controller in transmit buffer. Since there is only one transmit buffer, no transmission can take place once the UDP packet creation has started and until the packet has been sent. The time between starting a UDP packet and the moment when the packet is sent should be kept to a minimum.

Dynamic Host Configuration Protocol (DHCP) Client

Only DHCP DISCOVER and DHCP REQUEST message types are supported. The client will be able to inquire and retrieve an IP address. DHCP INFORM type message to request more information from the DHCP server is not supported. DHCP RELEASE type message to release DHCP information and deactivate the Client's IP address is also not supported.

Transmission Control Protocol (TCP)

Packets received out of order are dropped and the stack requests retransmission of the expected packet.

Urgent Pointer is ignored and the data will pass to the application in the same way as data received without Urgent Pointer.

The Push Flag is ignored and the data is passed by the user every time there is something in the socket's RX buffer.

In order to reduce the protocol complexity, the number of timeouts and the duration between, the messages' retransmission is defined as a fixed-time interval and it is user configurable.

The delayed Acknowledgment packet (ACK) functionality is not implemented. Each ACK packet will be sent with payload if there are any outstanding TX data or without payload right away.

Only the Maximum Segment Size (MSS) option from the TCP header options is supported. All the other options are silently ignored by the stack.

RUNNING THE TCP/IP STACK DEMOS

Required Hardware and Software to Run the Demo

1. PICDEM.net® 2 Board
2. PICDEM.net 2 Power Supply
3. Microchip TCP/IP Lite Stack and Demo Code
4. MPLAB X v2.35 or later
5. XC8 v1.33 C Compiler or later
6. Microchip Debugger/Programmer (e.g., PICkit™ 3, ICD 3, REAL ICE™)
7. PC with Windows®, Linux® or Mac OS®
8. TCP/IP Demo Application
9. DHCP Server (without it the board cannot release an IP address and the demo will not work)
10. Ethernet cables:
    - Straight-Through – if the board is connected to a router/switch
    - Crossover – if the board is connected directly to the computer

Setting Up the Hardware

1. Connect the PICDEM.net 2 board (connector J1) using an Ethernet cable to an Ethernet network (it can be connected directly to the Ethernet port of a PC). The board has to be able to connect to a running DHCP Server.
2. Connect the power supply to the PICDEM.net 2 board using the J7 connector.
3. Connect the Microchip Debugger/Programmer to the PICDEM.net 2 board using connector J4.
4. Load one of the Demo projects in MPLAB X.
5. For the next steps, refer to the setup chapter related to each demo.

TCP SERVER IMPLEMENTATION DEMO

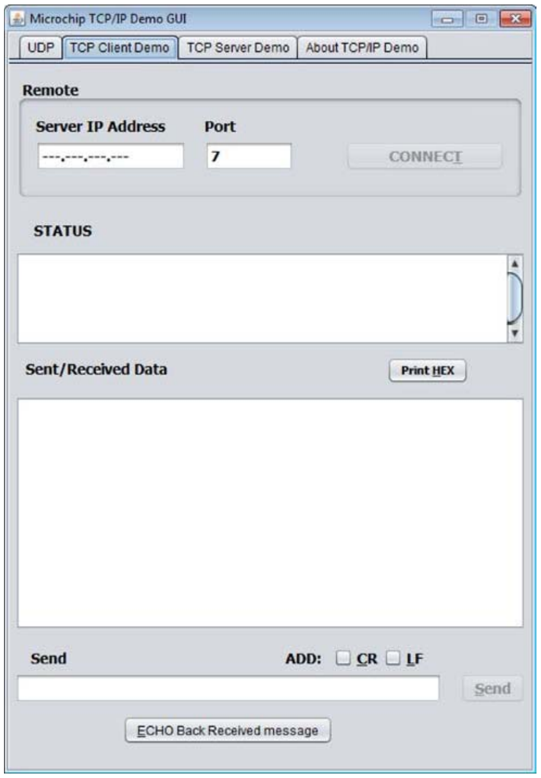
Overview

This is a TCP echo server implementation example, listening on port 7. The server is started on the PICDEM board and it will wait for any incoming connection. The server will echo back all the received data once the connection with a client is established. There is only one active connection created for this demo, but the TCP/IP stack supports multiple TCP connections on the same board. The user needs to create a new server (create buffers, initialize and start listening) for each new connection. The firmware is intended to run on PICDEM.net 2 boards.

Setting Up the Software for TCP Server Demo

1. Open MPLAB X and load the TCP Server Demo project.
2. Compile the latest TCP Server Demo code with XC8.
3. Program the PICDEM.net 2 board.
4. If the DHCP server is running in the Ethernet network used for the demo the PICDEM.net 2 board, it should have its own IP address. The IP address will come up on the LCD of the board. This address will be used by the client to connect to the server running on the PICDEM board.
5. Start TCP/IP Demo Java application on the computer (as shown in Figure 2).
6. Go to the **TCP Client Demo** tab.
7. Go to the Server IP Address and set the IP address printed on the PICDEM board LCD and set the port number to 7.
8. Push the **Connect** button.
9. When the computer is connected to the PICDEM board, a message in the Received/ Send Data windows will appear (e.g., Connected to 192.168.0.21 Port: 7).
10. Type in the Send window and push the **Send** button to send the string. Both sent data and received data will appear with different colors in the Sent/Received Data window.
11. Pressing the **Disconnect** button will close the TCP connection. A Connection Closed message will appear.
12. Repeat steps 6 to 10 to test the connection using different string lengths.
13. To generate TCP traffic to the board, the ECHO Back Received Message should be enabled. The Send file should be filled with the message to be sent to the board. Pushing the **Send** button will initiate the data exchange. To stop the TCP traffic, push the **ECHO Back Received Message** button again. In this case, in the Sent/ Received Data windows only the received messages will appear.

FIGURE 2: MICROCHIP TCP CLIENT DEMO FOR JAVA APPLICATION



TCP Server Demo Firmware

BUFFER CREATION

The user needs to create the socket, the RX and the TX buffers.

EXAMPLE 1: RX BUFFER

```
// create the socket for the TCP Server
tcpTCB_t port7TCB;

// create the TX and RX buffers
uint8_t rxdataPort7[20];
uint8_t txdataPort7[20];
```

TCP Server Implementation

These are the steps required to implement the TCP Server:

1. Initializing the TCP stack. The function should be called before any other TCP function.

   ```
   TCP_Init();
   ```

2. Inserting and initializing the socket in order to create the connection. It keeps all needed information for the TCP connection.

   ```
   TCP_SocketInit(&port7TCB);
   ```

3. Assigning the local port for the server. The function will assign a port for the socket to listen on. The server will listen on this port for any incoming connections. A default port number will be assigned in the absence of a user-supplied listen port.

   ```
   TCP_Bind(&port7TCB, 7);
   ```

4. Adding the receive buffer to a socket. The function will insert the buffer into the socket for storing the received data.

   ```
   TCP_InsertRxBuffer(&port7TCB,
   rxdataPort7, sizeof(rxdataPort7));
   ```

5. Starting the TCP server. The function will set the TCP stack to listen to a port for a connection request. If the TCP handshake completes successfully, the user can exchange data with the remote over the TCP connection. Only one connection request is accepted at one time. The TCP stack can handle multiple connections for a particular port number. But for each new connection the user needs to create a new socket, an RX buffer, and start a new instance of the server for the same port.

   ```
   TCP_Listen(&port7TCB);
   ```

6. Checking the status of the socket. This function checks if the pointer provided as parameter is already registered with to the TCP/IP stack as a socket. If the pointer is a valid socket, the function will return the state of that socket. The possible states of the socket are defined in the `tcpv4.h` file.

   ```
   socket_state = TCP_SocketPoll(&port7TCB);
   ```

7. Checking if there are any data received in the socket. The function will return the number of bytes available in the RX buffer.

   ```
   rxLen = TCP_GetRxLength(&port7TCB);
   ```

8. Reading how many bytes are available in the RX buffer and getting the buffer ready to be used by the user. The function will return the number of bytes available in the buffer. After calling this function, the user can access the buffer in a safe way. Once the function is called, the stack will not save any further data received into this RX buffer. The user should provide, as fast as possible, another RX buffer to the stack, in order to avoid packet retransmission.

   ```
   rxLen = TCP_GetReceivedData
   (&port7TCB);
   ```

9. Sending the buffer to the remote machine. The API will allow the user to send data over an active TCP connection. The data cannot be sent if the connection is not established between the local and the remote host.

   ```
   TCP_Send(&port7TCB, txdataPort7,
   txLen);
   ```

10. Checking if the TX buffer was sent correctly (this means that the remote host acknowledged all the received bytes). This function needs to be called before trying to send anything, because the socket can handle only one buffer at a time.

    ```
    TCP_SendDone(&port7TCB)
    ```

11. Closing a TCP connection. This function will close the TCP connection. Socket connection closing will happen after the TCP connection handshake is done (the connection closing is not done right away). The user needs to check the socket state periodically until the socket is in closed state. When the socket is in Closed state, the RX buffer and the TX buffer can be safely reused.

    ```
    TCP_Close(&port7TCB)
    ```

12. Removing the socket. When the socket is closed, if the user wants to remove the socket from the internal socket list, the following API will remove the pointer.

    ```
    TCP_SocketRemove(&port7TCB)
    ```

13. Background task. This function needs to be called periodically by the application, in order to handle the timeouts from the TCP stack. The TCP background task is called once per second to handle the TCP stack timeouts.

    ```
    TCP_Update();
    ```

Source Code for the TCP Server Implementation

The TCP Server Demo Code (source code and prebuilt hex file) is available on the Microchip website.

EXAMPLE 2: TCP ECHO SERVER IMPLEMENTATION

```
void DEMO_TCP_echo_server(void)
{
    // create the socket for the TCP Server
    static tcpTCB_t port7TCB;

    // create the TX and RX buffers
    static uint8_t rxdataPort7[20];
    static uint8_t txdataPort7[20];

    uint16_t rxLen, txLen, i;
    socket_state_t socket_state;
    rxLen = 0;

    // checking the status of the socket
    socket_state = TCP_SocketPoll(&port7TCB);

    switch(socket_state)
    {
        case NOT_A_SOCKET:
            //Inserting and initializing the socket
            TCP_SocketInit(&port7TCB);
        case SOCKET_CLOSED:
            //configure the local port
            TCP_Bind(&port7TCB, 7);
            // add receive buffer
            TCP_InsertRxBuffer(&port7TCB, rxdataPort7, sizeof(rxdataPort7));
            // start the server
            TCP_Listen(&port7TCB);
            break;
        case SOCKET_CONNECTED:
            // check if the buffer was sent, if yes we can reuse the buffer
            if(TCP_SendDone(&port7TCB))
            {
                // check to see if there are any received data
                rxLen = TCP_GetRxLength(&port7TCB);
                if(rxLen > 0)
                {
                    //make sure it safe to use the receive buffer;
                    rxLen = TCP_GetReceivedData(&port7TCB);

                    //simulate some buffer processing copy from the RX buffer to the TX buffer
                    for(i = 0; i < rxLen; i++)
                    {
                        txdataPort7[i] = rxdataPort7[i];
                    }
                    // reuse the rx buffer
                    TCP_InsertRxBuffer(&port7TCB, rxdataPort7, sizeof(rxdataPort7));

                    txLen = rxLen;
                    //send data back to the source
                    TCP_Send(&port7TCB, txdataPort7, txLen);
                }
            }
            break;
        default:
            // we should not end up here
            break;
    }
}
```

TCP CLIENT IMPLEMENTATION DEMO

Overview

This is a TCP client implementation that will connect to a server that runs on a computer on port 60. The user needs to modify the server IP address into the firmware. Once the connection is established, the client will send status packets to the server every two seconds. The packets sent by the client contain temperature reading, potentiometer value, buttons and LED status. From the computer/server the user can send text messages that will be printed on the second line of the PICDEM.net 2 board's LCD. Messages longer than 16 characters will be truncated and only the first 16 characters will appear on the LCD. From the server, the user can also turn the LEDs on the board ON or OFF using the GUI push buttons.

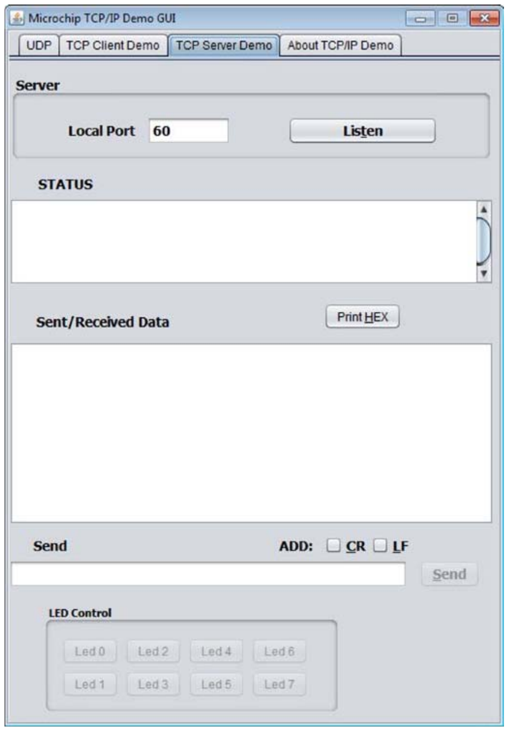
For this demo there is only one active connection implemented, but the TCP/IP stack supports multiple TCP connections on the same board. For each new connection the user needs to create a new socket, an RX buffer, and try to connect to the server. The firmware was developed to run on PICDEM.net 2 boards.

The TCP Client Demo will try to connect to the server every two seconds. This was implemented so that the user has easy access to the Wireshark protocol analyzer.

Setting Up the Software for TCP Client Demo

1. Find the IP Address of the computer where the TCP/IP Server Java Demo application runs on.
2. Open MPLAB X and load the TCP Client Demo project.
3. Modify the Server IP address from TCP/IP Stack Demo Code (e.g., `remoteSocket.addr.s_addr = MAKE_IPV4_ADDRESS(192,168,0,3);`).
4. Compile the project using the XC8 compiler.
5. The firmware to the PICDEM.net 2 board can be programmed using MPLAB X.
6. If the DHCP server is running in the Ethernet network used for the demo, the PICDEM.net 2 board should get its own IP address. The IP address will come up on the LCD of the PICDEM board.
7. Start the TCP/IP Demo Java application on the computer.
8. Go to the **TCP Server Demo** tab.
9. Go to the Server → Local Port and change the port number to 60.
10. Push the **Listen** button.
11. When the PICDEM.net 2 board connects to the computer, a message in the Sent/Received Data window will appear (e.g., 192.168.0.21: Connected).
12. Type in the Send field and the message will be sent to the board. The message will be sent when either the **Enter** or the **Send** buttons are pushed. Both sent data and received data will be shown in different colors in the Received/Send Data window.
13. Messages that came from the board will be automatically shown in the Sent/Received Data window and include information on raw temperature reading from the sensor, button 0 to 3 states, LED 0 to 7 states and the raw value of the on-board potentiometer. Values are in hexadecimal format.
14. Pushing the LED 0 to LED 7 buttons will initiate the sending of a command to the board. The corresponding LED from the boards will be turned ON or OFF. The implementation supports only one LED turning ON or OFF at a time.
15. Pushing the **Disconnect** button will close the TCP connection. A "Client disconnected" message will appear.
16. Repeat steps 8 to 14 to test the connection.

FIGURE 3: MICROCHIP TCP CLIENT DEMO IN JAVA APPLICATION



Local Port: 60

TCP Server Demo Firmware

BUFFER CREATION

The user needs to create the socket, the RX and the TX buffers. The TX buffer will be created by the user and passed to the TCP stack then it's ready to be sent.

EXAMPLE 3: RX BUFFER CREATION

```
// create the socket for the TCP Client
tcpTCB_t port60TCB;

// create the TX and RX buffers
uint8_t rxdataPort60[50];
uint8_t txdataPort60[80];
```

TCP Client Implementation

These are the steps required to start and to implement the TCP Client:

1. Initializing the TCP stack. The function should be called before any other TCP functions are called.

   ```
   TCP_Init();
   ```
2. Inserting and initializing the socket in order to create the connection. The socket keeps all the needed information for the TCP connection.

   ```
   TCP_SocketInit(&port60TCB);
   ```
3. Setting the local port for the client (this step is not mandatory). The TCP stack will use the next available port number to be used as a local port. The user needs to use the `TCP_Bind` function to make sure that a certain port number will be used when the connection is initiated. This is useful when the server accepts connections only from a certain port number.

   ```
   TCP_Bind(&port60TCB, 1024);
   ```
4. Adding the RX buffer to the socket. The function will insert the buffer into the socket; it will be used for saving the received data.

   ```
   TCP_InsertRxBuffer(&port60TCB,
   rxdataPort60, sizeof(rxdataPort60));
   ```
5. Starting the Client. The `TCP_Connect` function will initiate the TCP connecting procedure for connection to the server. If the TCP handshake was done successfully, the user can exchange data with the remote server over the TCP connection. The user needs to provide the port number to connect on (this is the port number where the server listens on).

   ```
   remoteSocket.addr.s_addr =
   MAKE_IPV4_ADDRESS(192,168,0,3);
   remoteSocket.port = 60;
   TCP_Connect(&port60TCB,
   &remoteSocket);
   ```
6. Checking the status of the socket. This function checks if the pointer provided as a parameter is registered internally to the TCP/IP stack as a socket. If the pointer is a valid socket, the function will return the state of that socket. The possible states of the socket are defined in the `tcpv4.h` file.

   ```
   socket_state = TCP_SocketPoll(&port7TCB);
   ```
7. Checking if there is any data received in the socket. The function will return the number of bytes available in the RX buffer.

   ```
   rxLen = TCP_GetRxLength(&port60TCB);
   ```
8. Reading how many bytes are available in the RX buffer and get the buffer ready to be used by the user. The function will return the number of bytes available in the buffer. After calling this function, the user can access the buffer safely. Once the function is called, the stack will not save further received data into this RX buffer. The user should provide, as quickly as possible, another RX buffer to the stack (in order to avoid packet retransmission).

   ```
   rxLen = TCP_GetReceivedData
   (&port60TCB);
   ```
9. Sending the buffer to the remote machine. The API will allow the user to send data over an active TCP connection. The data cannot be sent if the connection is not established between the local and the remote host.

   ```
   TCP_Send(&port60TCB, txdataPort60,
   txLen);
   ```
10. Checking if the TX buffer was sent correctly (this means that the remote host acknowledges all the received bytes). This function needs to be called before trying to send anything, because the socket can handle only one buffer at a time.

    ```
    TCP_SendDone(&port60TCB)
    ```
11. Closing a TCP connection. This function will close the TCP connection. Socket connection closing will happen after the TCP connection handshake is done (the connection closing is not done right away). The user needs to check the socket state periodically until the socket is in Closed state. When the socket is in Closed state, both the RX buffer and the TX buffer can be safely reused.

    ```
    TCP_Close(&port60TCB)
    ```

12. Removing the socket. When the socket is closed, if the user wants to remove the socket from the internal socket list, the following API will remove the pointer.

    ```
    TCP_SocketRemove(&port60TCB)
    ```

13. Background task. This function needs to be called periodically by the application, in order to handle the timeouts from the TCP stack. The TCP background task is called once per second to handle the TCP stack timeouts.

    ```
    TCP_Update();
    ```

Source Code for the TCP Client Implementation

The TCP Client Demo Code (source code and prebuilt hex file) is available on the Microchip website.

EXAMPLE 4: SOURCE CODE SNIPPET FOR TCP CLIENT IMPLEMENTATION

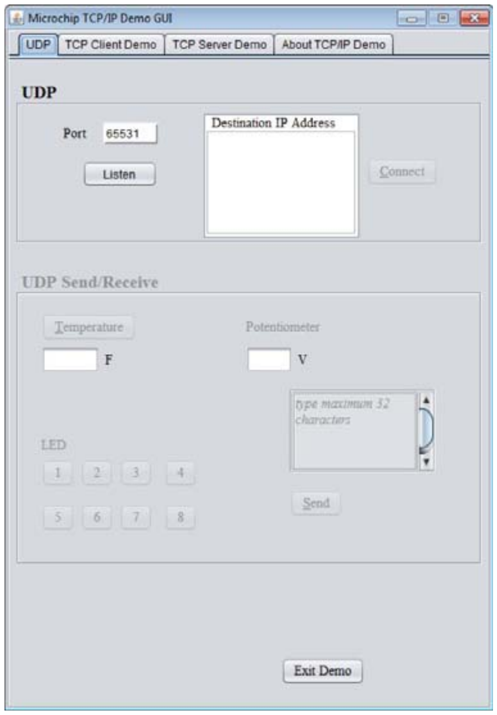
```
void DEMO_TCP_client(void)
{
    // create the socket for the TCP Client
    static tcpTCB_t port60TCB;

    // create the TX and RX Client's buffers
    static uint8_t rxdataPort60[50];
    static uint8_t txdataPort60[80];

    static time_t t_client = 0;
    static time_t t_client_old = 0;
    uint16_t rx_len, i;
    socket_state_t socket_state;
    rx_len = 0;

    socket_state = TCP_SocketPoll(&port60TCB);

    time(&t_client);

    switch(socket_state)
    {
        case NOT_A_SOCKET:
            // Inserting and initializing the socket
            TCP_SocketInit(&port60TCB);
        case SOCKET_CLOSED:
            // if the socket is closed we will try to connect again

            if(t_client >= t_client_old)
            {
                 // try to connect once at 2 seconds
                t_client_old = t_client + 2;
                TCP_InsertRxBuffer(&port60TCB, rxdataPort60, sizeof(rxdataPort60));
                TCP_Connect(&port60TCB, &remoteSocket);
            }
            break;
        case SOCKET_CONNECTED:
            // implement an echo client over TCP
            // check if the previous buffer was sent
            if (TCP_SendDone(&port60TCB))
            {
                rx_len = TCP_GetReceivedData(&port60TCB);
                // handle the incomming data
                if(rx_len > 0)
                {
              /*…………………………………………………………………………….
               LED Command parsing and LCD updates was removed from this example. The full code is available in
               the source code.
            …………………………………………………………………………….*/
                    // reuse the rx buffer
                    TCP_InsertRxBuffer(&port60TCB, rxdataPort60, sizeof(rxdataPort60));
                }
                if(t_client >= t_client_old)
                {
                  // send board status message only once at 2 seconds
              t_client_old = t_client + 2;
            /*…………………………………………………………………………….
               Composing the TX message in the TX buffer was removed from this example. The full code is
               available in the source code.
            …………………………………………………………………………….*/

                    //send data back to the source
                    TCP_Send(&port60TCB, txdataPort60, strlen(txdataPort60));
                }
            }
            break;
        default:
            // we should not end up here
            break;
    }
}
```

UDP DEMO

Overview

This is a UDP Client and Server implementation. It consists of UDP Send (UDP Client) and UDP Receive (UDP Server) implementations. As UDP Send, PICDEM board sends potentiometer and temperature readings as UDP packets. As UDP Receive, PICDEM.net 2 board starts listening to any incoming UDP packets, such as toggle LEDs and display data on LCD on port 65531. The port numbers can be anything between 49152 and 65535.

FIGURE 4: TCP/IP UDP DEMO IN JAVA APPLICATION



Setup the Software for UDP Send (Client) Demo

1. Start MPLAB X and load the UDP Demo project.
2. The firmware can be programmed to the PICDEM.net 2 board by using the XC8 compiler.
3. PICDEM.net 2 board sends LISTEN UDP packets on port 65531 every second until it is connected to the UDP Server.
4. Start TCP/IP Demo Java application on the computer.
5. Go to the **UDP** tab and enter the port number to 65531.
6. Push the **Listen** button. This will open a UDP socket at port 65531.

Note: Make sure the DEST_PORT in the `udp_demo.h` file and the port number entered in the TCP/IP Demo application are the same.

7. TCP/IP Demo application starts listening to the packets on port 65531 and displays IP addresses of the devices which are on the network.
8. User can select the IP address of the device from the list, to whom to talk to and click on **Connect** button.
9. TCP/IP Demo displays the IP address of the connected device.
10. Turn the knob on the PICDEM.net 2 board. The PICDEM.net 2 board sends UDP packets to display the potentiometer reading in volts.
11. Click on the **Temperature** button on the PC. The PICDEM.net 2 board sends UDP packets to display the ambient temperature reading in Fahrenheit.
12. Click on any of the LED buttons from 1 to 8. The PICDEM.net 2 board receives UDP packets and toggles LEDs on the board.
13. In GUI, type a text of maximum 32 characters and click on the **Send** button. The PICDEM.net 2 board receives UDP packets and displays the text on the LCD board.
14. Push the **Disconnect** button to close the connection between the computer and the PICDEM.net 2 board.
15. Repeat steps 5 to 14 to verify the UDP send and the UDP receive packets on port 65531.

UDP Send Implementation

In order to start the UDP packet, the following steps are required:

1. Start UDP Packet

The function will start the UDPv4 Packet, which starts the IPv4 packet and writes the UDP Header. The UDP Header fields' – checksum and Data length – are initially set to '`0`'.

- `UDP_Start (uint32_t destIP, uint16_t srcPort, uint16_t destPort);`

2. Write UDP Packet

There are four methods of writing a UDP packet, depending on the size and order of data written.

- `UDP_WriteBlock (uint8_t* data, uint16_t length)` – Writes a block of data.
- `UDP_Write8 (uint8_t data)` – Writes 1 byte of data.
- `UDP_Write16 (uint16_t data); -`

Writes 2 bytes of data in Host Order.
- UDP_Write32 (uint32_t data); - Writes 4 bytes of data in Host Order.

3. Send UDP Packet

The function will insert the total payload length into the UDP header, compute the checksum of the UDP packet and send the packet on the wire.

- UDP_Send();

UDP Receive Implementation

In order to receive the UDP packet, the following steps are required:

1. Port Handling

In the udpv4_port_handler_table.h file, the UDP_CallBackTable function needs to be updated with the receiving port number and its callback function.

EXAMPLE 5: PORT HANDLING

```
typedef struct
{
    uint16_t portNumber;
    ip_receive_function_ptr callBack;
} udp_handler_t;

const udp_handler_t UDP_CallBackTable [] = \
{
     {portNumber, &callback}
};
```

2. Receive UDP Packet

If the checksum is correct, the function will match the port number to the corresponding function handler (callback) and the length of the UDP payload will be passed as a parameter to the callback. Any UDP packets with invalid checksums are discarded.

- UDP_Receive (uint16_t udpcksm);

3. Read UDP Packet

There are four methods of reading a UDP packet, depending on the size and order of data.

- UDP_ReadBlock (uint8_t* data, uint16_t length) – Reads a block of data.
- UDP_Read8 (uint8_t data) – Reads 1 byte of data.
- UDP_Read16 (uint16_t data) - Reads 2 bytes of data in Host Order.
- UDP_Read32 (uint32_t data) - Reads 4 bytes of data in Host Order.

Source Code for the UDP Client/Server Implementation

The UDP Demo Code (source code and prebuilt hex file) is available on the Microchip website.

EXAMPLE 6: UDP CLIENT IMPLEMENTATION

```
void DEMO_UDP_Send()
{
     bool started = false;
     if(!claim_ip_check())
     {
         started = UDP_Start(0xFFFFFFFF,65533,65531);
         if(started==SUCCESS)
         {
             UDP_Write8(LISTEN);          // Write the Transmit Buffer data
             UDP_Send();
         }
     }
}
```

EXAMPLE 7: UDP SERVER IMPLEMENTATION

```
const udp_handler_t UDP_CallBackTable [] = \
{
    {65531, & DEMO_UDP_Recv }
};

void DEMO_UDP_Recv(int length)
{
    UDP_ReadBlock(&data,sizeof(data));
/*
    Process the Receive Buffer data
*/
}
```

RUNNING TCP/IP STACK ON PICDEM.net 2 BOARD USING DIFFERENT ETHERNET CONTROLLERS

This section describes the Hardware (PICDEM.net™ board) and the Software (MPLAB® X project) setup while using different Ethernet controllers.

PICDEM.net 2 Board with ENC97J60

HARDWARE SETUP

1. Connect the Ethernet cable to the connector J1 (i.e., the RJ45 jack on the left side of the board).
2. Open the bridge Jumper JP9.
3. Apply 9V power supply to the board at the connector J7.

SOFTWARE SETUP

1. Open MPLAB X IDE.
2. Open one of the TCP/IP Demo projects.
3. In the Project properties window, select the option PICDEM.net2_ENC97J60 device configuration.

Note:	This will enable the ENC97J60 driver, TCP/IP stack and other drivers which are used for the PICDEM.net 2 board.

PICDEM.net 2 Board with ENC28J60

HARDWARE SETUP

1. Connect the Ethernet cable to the J2 connector (i.e., the RJ45 jack on the right side of the board).
2. Bridge the Jumper JP9.
3. Close the bridge Jumper JP9.
4. Apply 9V power supply to the board at the connector J7.

SOFTWARE SETUP

1. Open MPLAB X IDE.
2. Open one of the TCP/IP Demo projects.
3. In the Project properties, select the option PICDEM.net2_ENC28J60 device configuration.

Note:	This will enable the ENC28J60 driver, the TCP/IP stack and other drivers which are used for the PICDEM.net 2 board.

PICDEM.net 2 Board with ENCx24J600 – SPI Interface

HARDWARE SETUP

1. Make sure to open the Jumper JP9 on the PICDEM.net 2 board.
2. Insert the Fast 100 Mbps Ethernet PICtail™ Plus Daughter Board J4 header into the PICDEM.net Board J5 connector with pin 1 of the PICtail board (labeled RE2) aligned with the RE2 on the connector J5 of the PICDEM.net 2 board.
3. Connect the Ethernet cable to the connector J7 on the PICtail board.
4. Apply 9V power supply to the PICDEM.net 2 board at the connector J7.

SOFTWARE SETUP

1. Open MPLAB X IDE.
2. Open one of the TCP/IP Demo projects.
3. In the Project properties, select the option PICDEM.net2_ENCx24J600_SPI device configuration.

Note:	This will enable the ENCx24J600 driver, the TCP/IP stack and other drivers which are used for the PICDEM.net 2 board.

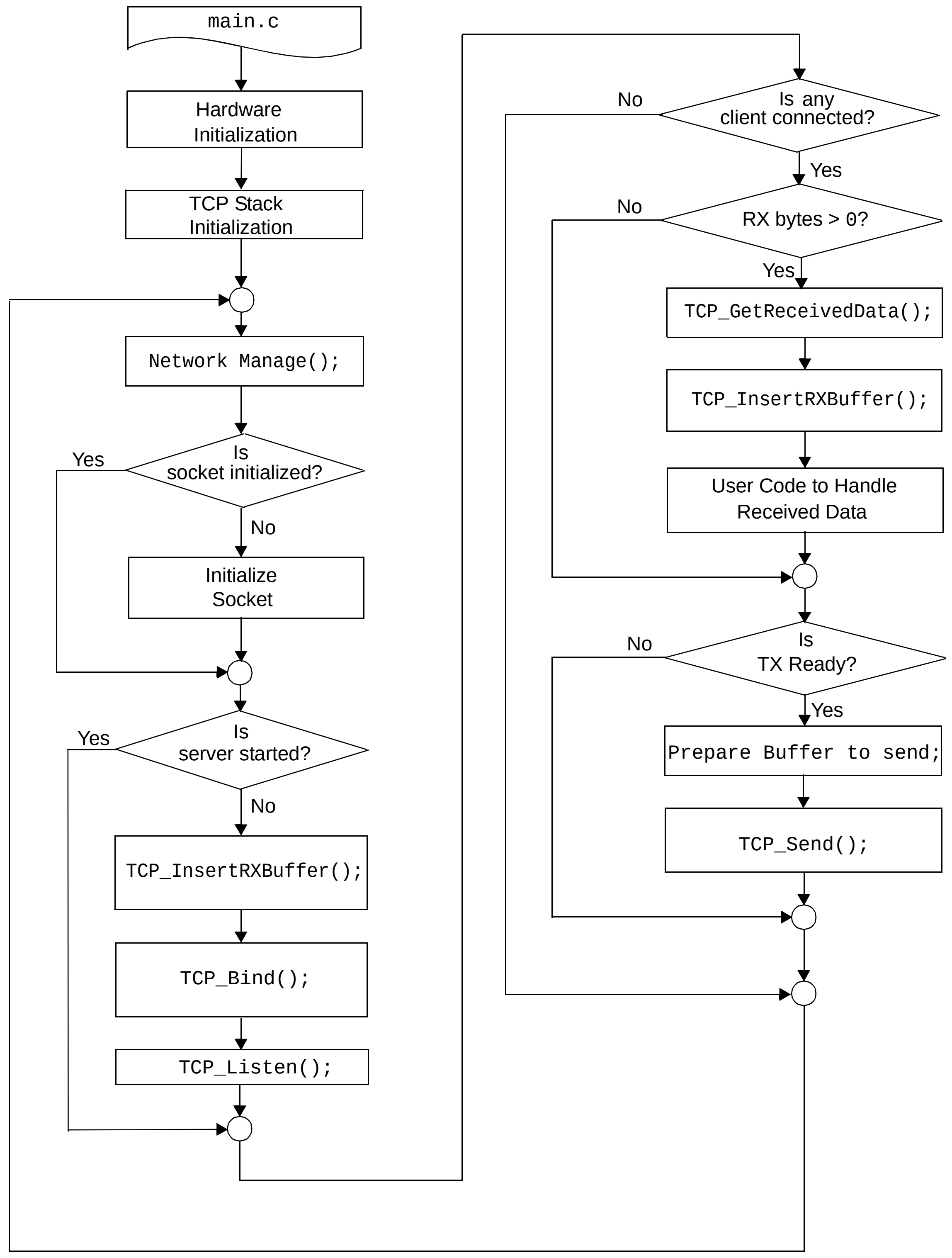
CONCLUSION

This application note presents some very simple software solutions for implementing a TCP Server, a TCP Client and exchange data over UDP, based on the Microchip TCP/IP stack. The TCP/IP lite stack provides an effective and modular implementation allowing network connectivity to the embedded system with limited resources.

APPENDIX A: REFERENCES

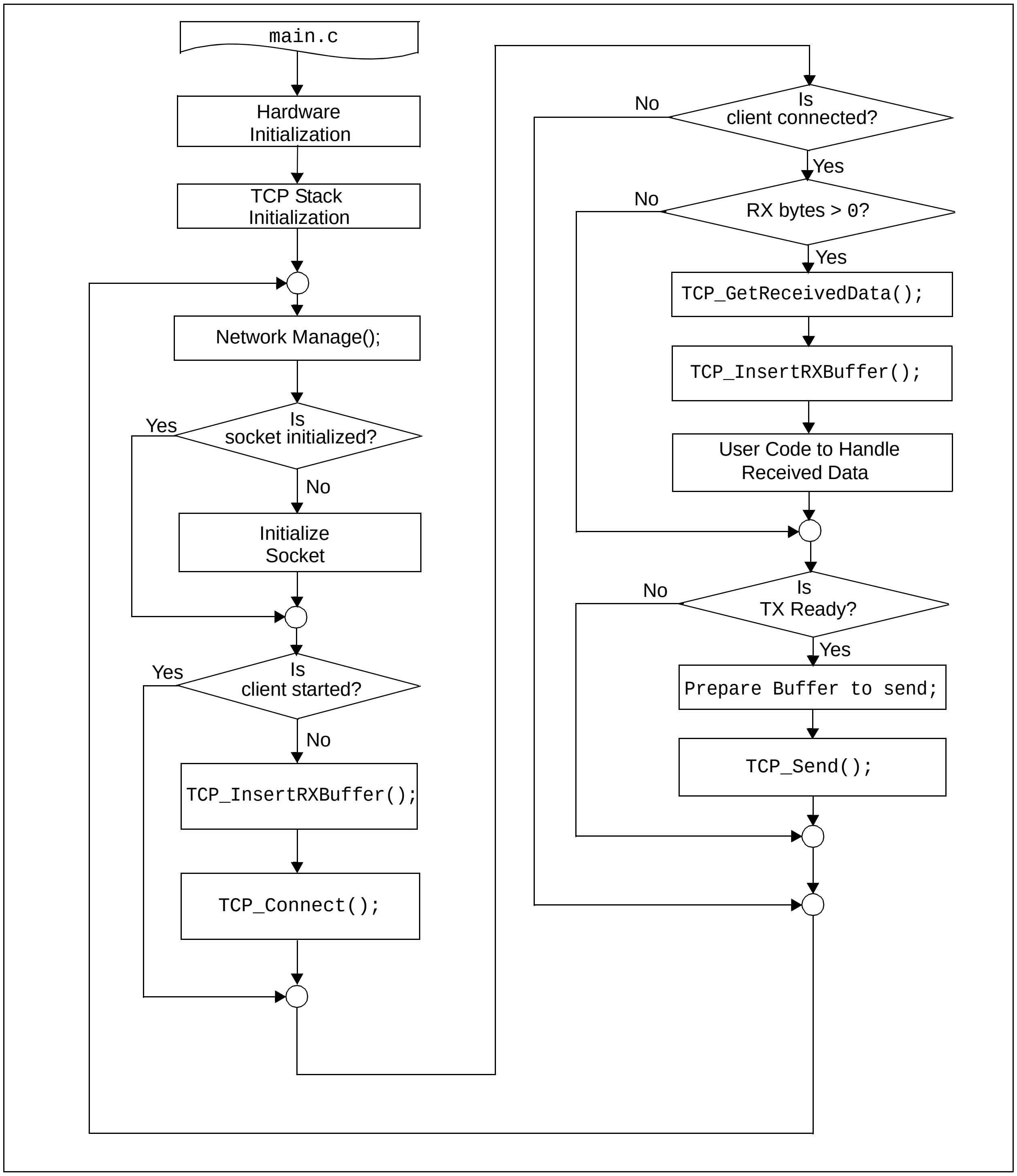
1. *User Datagram Protocol, RFC 768*
2. *Internet Protocol, DARPA Internet Program Protocol Specification, RFC 791*
3. *Internet Control Message Protocol, DARPA Internet Program Protocol Specification, RFC 792*
4. *Transmission Control Protocol, DARPA Internet Program Protocol Specification, RFC 793*
5. *Requirements for Internet Hosts, Communication Layers, RFC 1122*
6. *An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48 bit Ethernet Address for Transmission on Ethernet Hardware, RFC 826*
7. *Domain Names – Implementation and Specification, RFC 1035*
8. *Clarifications to the DNS Specification, RFC 2181*
9. *Service Name and Transport Protocol Port Number Registry* (www.iana.org)

APPENDIX B: FLOWCHART FOR TCP SERVER DEMO

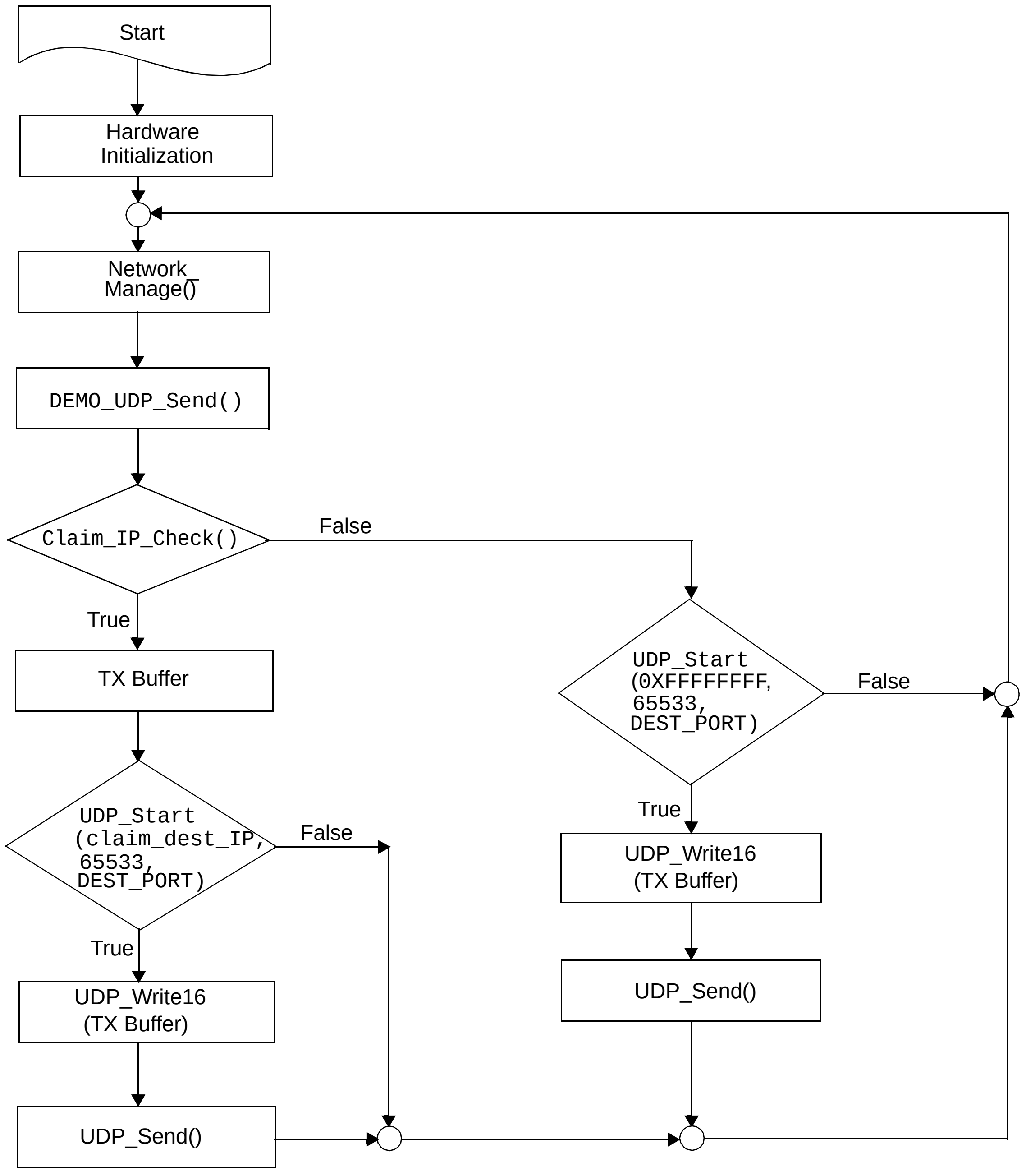


Note: Since the demo containing the TCP/IP stack is very large, the software flowchart presented is only for the main application with the focus of the TCP Server routine found in the `main.c` file.

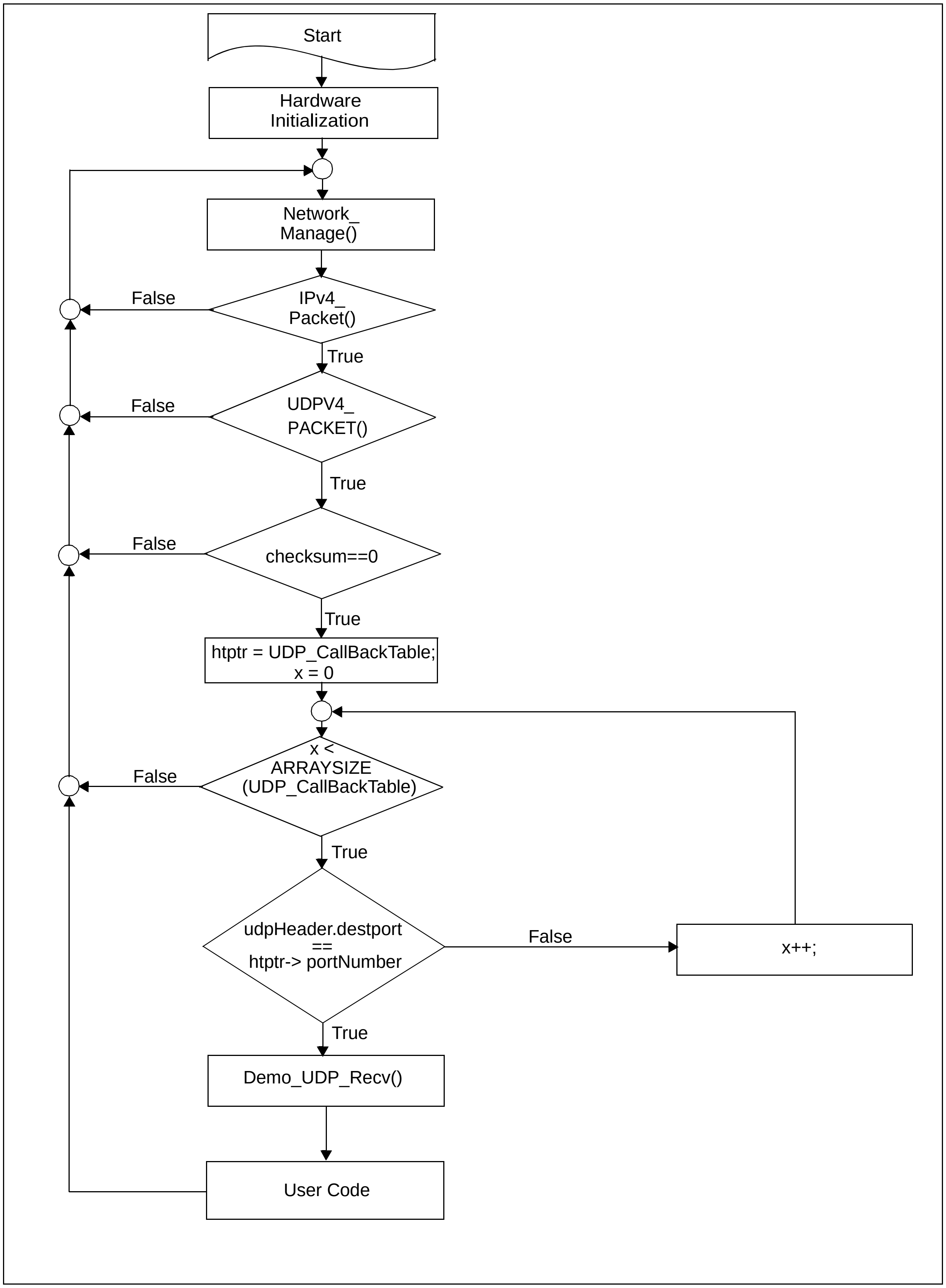
APPENDIX C: FLOWCHART FOR TCP CLIENT DEMO



APPENDIX D: FLOWCHART FOR UDP DATA EXCHANGE DEMO



APPENDIX E: FLOWCHART FOR UDP PACKET RECEIVING IN UDP DEMO



ISBN: 978-1-5224-0002-8

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

07/14/15