Application Note, V1.0, Jan. 2007

AP16106

XC167CI

IwIP (Light weight) TCP/IP Stack on 16-bit XC167CI Microcontroller

Microcontrollers

Never stop thinking

Edition 2007-01-15

Published by
Infineon Technologies AG
81726 München, Germany

AP16106

Revision History:	2007-01	V1.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

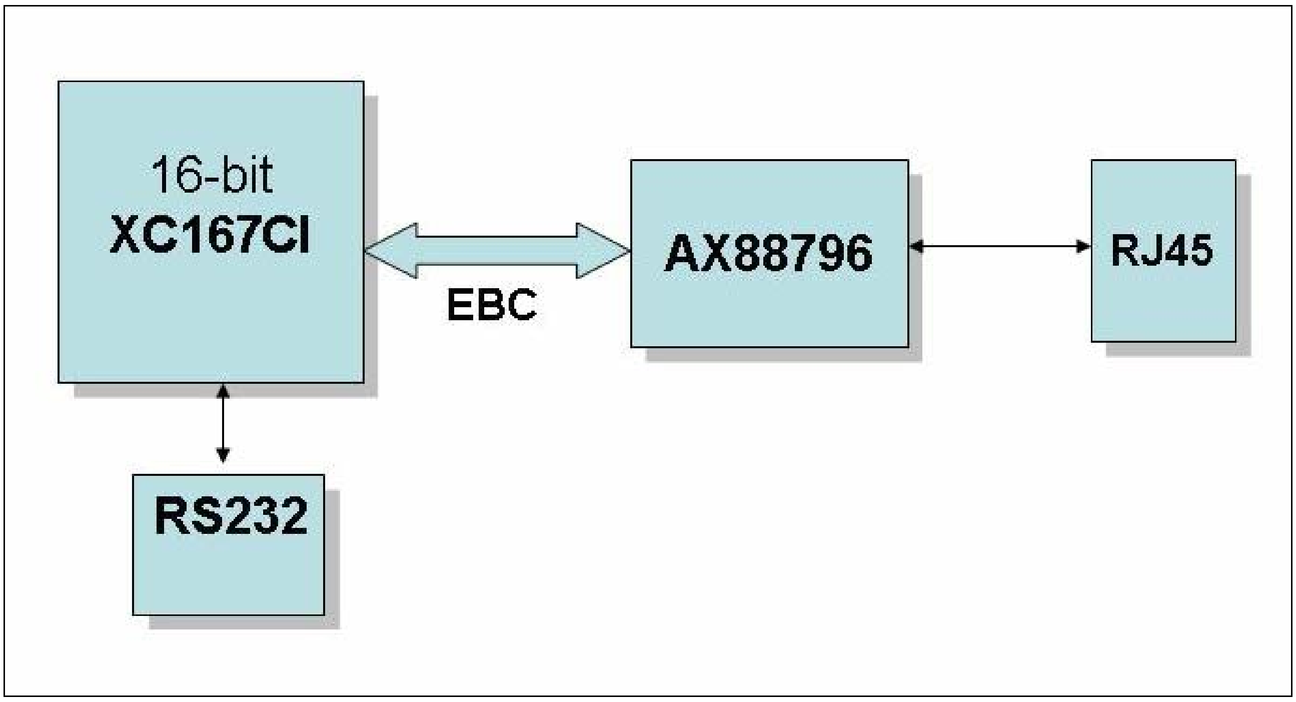
Table of Contents **Page**

1 Introduction

TCP/IP is a communication protocol stack designed to provide a reliable data stream between two hosts. It is a popular means of communicating data over a network. Most people use the protocol everyday to check email, browse the web, instant message and download files. TCP/IP is also becoming more utilized in embedded systems. A microcontroller in home based appliance or process controller is not directly connected to internet but instead is a host or industrial LAN using an Ethernet protocol.

This application note describes the method to provide lwIP (Light weight TCP/IP) stack over Ethernet connectivity on Infineon's XC167CI 16-bit microcontroller. The focus of the lwIP TCP/IP implementation is to reduce the RAM usage while still having a full scale TCP. The Ethernet network is implemented using Asix AX88796 Fast Ethernet controller with embedded 10/100Mbps PHY/Transceiver and 8k * 16 bit SRAM. This Ethernet controller is interfaced to Infineon's XC167CI microcontroller through External Bus Controller (EBC).

The Solution block diagram is as shown below:



2 Hardware

The hardware includes Infineon's 16-bit XC167CI as a target microcontroller to which AX88796 Ethernet Controller is interfaced using lwIP stack. The board used for this purpose is Keil's XC167CI Evaluation board.

2.1 The XC167CI 16-bit Microcontroller

The XC167CI is a new derivative of the popular C166 microcontroller family that is based on the enhanced C166S V2 architecture which outperforms existing 16-bit solutions. Impressive DSP performance and advanced interrupt handling combined with an integrated powerful peripheral set and a high performance on-chip flash makes the XC167CI the instrument of choice for demanding industrial and automotive applications. In addition to the well-known C167 peripheral set the XC167CI has a flexible on-chip PWM unit for all kind of motor control applications and an I^2C module.

The Architecture of XC167CI has been optimized for high instruction throughput and minimum response time to external interrupts. Integrated intelligent peripheral systems reduce the need for CPU intervention. The high flexibility of this architecture perfectly supports the diverse and varying needs of different application areas such as industrial drives and control, or automotive body. All this combined in a small PG-TQFP-144 package enables very high levels of system integration.

The XC167CI main features are as follows:

- High Performance 16-bit CPU with 5-Stage Pipeline
  - 25 ns Instruction Cycle Time at 40 MHz CPU Clock (Single-Cycle Execution)
  - 1-Cycle Multiplication (16 ×16 bit), Background Division (32 / 16 bit) in 21 Cycles
  - 1-Cycle Multiply-and-Accumulate (MAC) Instructions
  - Enhanced Boolean Bit Manipulation Facilities
  - Zero-Cycle Jump Execution
  - Additional Instructions to Support HLL and Operating Systems
  - Register-Based Design with Multiple Variable Register Banks
  - Fast Context Switching Support with Two Additional Local Register Banks
  - 16 Mbytes Total Linear Address Space for Code and Data
  - 1024 Bytes On-Chip Special Function Register Area (C166 Family Compatible)
- 16-Priority-Level Interrupt System with 77 Sources, Sample-Rate down to 50 ns.
- 8-Channel Interrupt-Driven Single-Cycle Data Transfer Facilities via Peripheral Event Controller (PEC), 24-Bit Pointers Cover Total Address Space.
- Clock Generation via on-chip PLL (factors 1:0.15 … 1:10), or via Prescaler (factors 1:1 … 60:1).
- On-Chip Memory Modules
  - 2 Kbytes On-Chip Dual-Port RAM (DPRAM)
  - 4 Kbytes On-Chip Data SRAM (DSRAM)
  - 2 Kbytes On-Chip Program/Data SRAM (PSRAM)
  - 128 Kbytes On-Chip Program Memory (Flash Memory)
- On-Chip Peripheral Modules
  - 16-Channel A/D Converter with Programmable Resolution (10-bit or 8-bit) and Conversion Time (down to 2.55 μs or 2.15 μs).
  - Two 16-Channel General Purpose Capture/Compare Units (32 Input/Output Pins).

- Capture/Compare Unit for flexible PWM Signal Generation (CAPCOM6) (3/6 Capture/Compare Channels and 1 Compare Channel).
- Multi-Functional General Purpose Timer Unit with 5 Timers.
- Two Synchronous/Asynchronous Serial Channels (USARTs).
- Two High-Speed-Synchronous Serial Channels.
- On-Chip Twin CAN Interface (Rev. 2.0B active) with 32 Message Objects. (Full CAN/Basic CAN) on Two CAN Nodes, and Gateway Functionality.
- I^2C Bus Interface (10-bit addressing, 400 Kbit/s) with 3 Channels (multiplexed).
- On-Chip Real Time Clock, Driven by Dedicated Oscillator.

• Idle, Sleep, and Power Down Modes with Flexible Power Management.

• Programmable Watchdog Timer and Oscillator Watchdog.

• Up to 12 Mbytes External Address Space for Code and Data.

  - Programmable External Bus Characteristics for Different Address Ranges.
  - Multiplexed or Demultiplexed External Address/Data Buses.
  - Selectable Address Bus Width.
  - 16-Bit or 8-Bit Data Bus Width.
  - Five Programmable Chip-Select Signals.
  - Hold- and Hold-Acknowledge Bus Arbitration Support.

• Up to 103 General Purpose I/O Lines, partly with Selectable Input Thresholds and Hysteresis.

• On-Chip Bootstrap Loader.

• Supported by a Large Range of Development Tools like C-Compilers, Macro-Assembler, Packages, Emulators, Evaluation Boards, HLL-Debuggers, Simulators, Logic Analyzer Disassemblers, Programming Boards.

• On-Chip Debug Support via JTAG Interface.

• 144-Pin TQFP Package, 0.5 mm (19.7 mil) pitch.

2.2 The Keil's XC167CI Evaluation Board

The MCBXC167 is designed to be a very flexible evaluation board for Infineon XC167 chip. It supports up to two CAN interfaces (enabled using jumpers). The MCBXC167 evaluation board can be expanded to build hardware prototypes. You can debug your application software via the On-Chip Debugging System (OCDS) for example by connecting the Keil ULINK USB-JTAG Adapter or via the Serial Port (COM) using the Keil Monitor-166.

2.2.1 Hardware Blocks

This section describes the hardware blocks of the MCBXC167 board. It provides a circuit description that helps you to understand the MCBXC167 board components and describes how you may interface to the various I/O interface available.

The following block diagram shows the various memory, I/O, configuration, and power systems on the board.

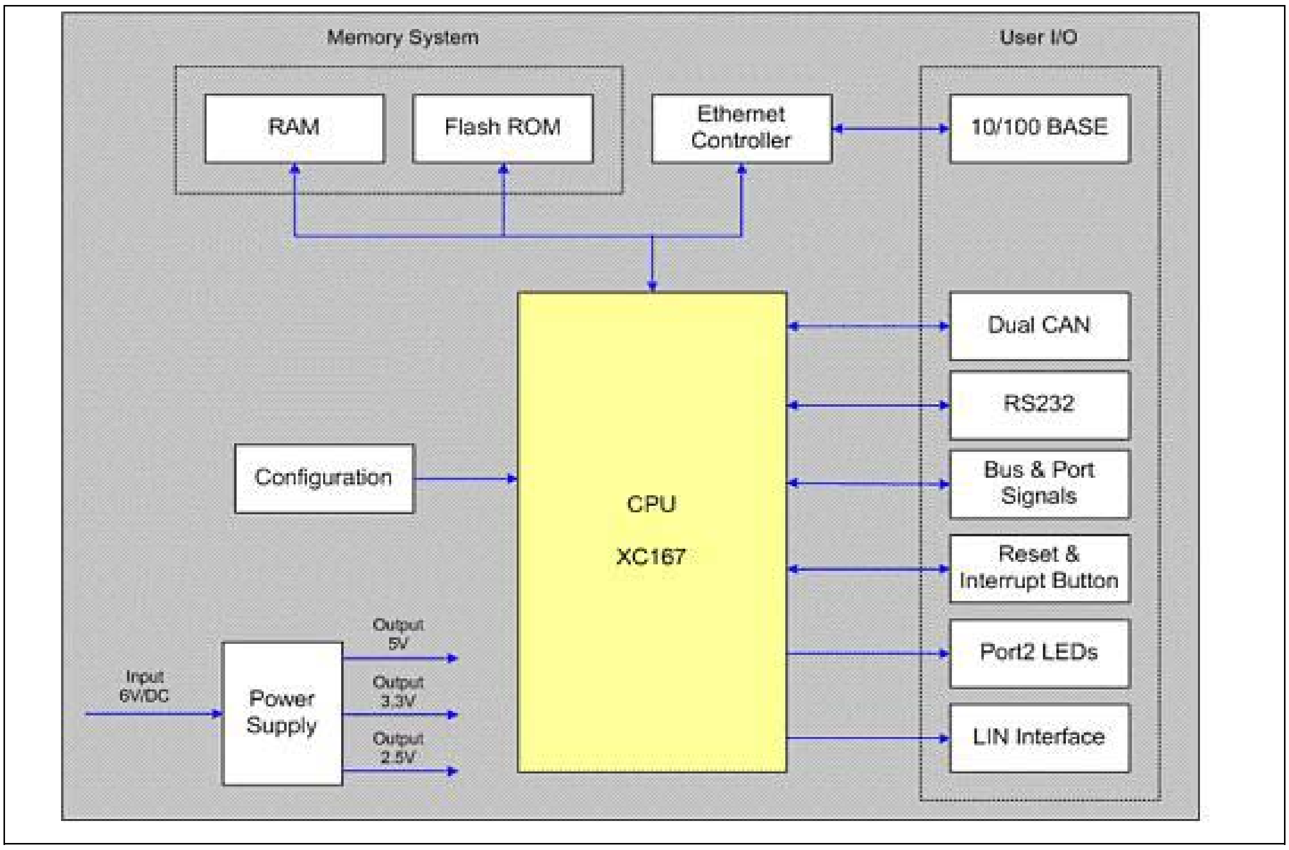
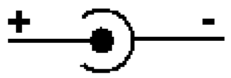


Figure 1 Keil XC167CI System Block Diagram

Power Supply

Power is supplied to the MCBXC167 board by an external 6 Volt DC power supply which should be capable of providing 400mA. Connection is made using a standard barrel plug. On the board, 5 Volts, 3.3 Volts, and 2.5 Volts are generated by voltage regulators IC8 and IC6.

!!Using a power supply with wrong polarity will damage the Board!!



The Center hole provides a positive voltage.

XC167 Micro-Controller

The Infineon XC167 microcontroller provided with the MCBXC167 (IC4) board is a high-end XC16x device with on-chip CAN. An 8.0 MHz crystal provides the clock signal.

Memory Devices

The MCBXC167 evaluation board provides a high speed RAM device (IC2) and a Flash ROM device (IC5). The chip select signals are provided by the XC167 microcontroller. CS0/ is used for the Flash ROM and CS1/ is used for the RAM.

OCDS Interface Connector

The OCDS (On-Chip Debug System) of the XC167 device is connected to the OCDS interface connector. OCDS allows application debugging and programming of on-chip and off-chip Flash devices. The standard OCDS 16-pin connector is supported by Keil ULINK and many third party tool vendors.

Ethernet Interface

The Ethernet Controller AX88796 (IC7) is used to interface the Ethernet connector. The MCBXC167 NET board provides a standard 10/100 Base (10/100Mbit) interface that allows direct connection to most Ethernet networks. The CS2/ chip select from the XC167 microcontroller is used to access the Ethernet controller as a memory mapped device.

CAN Drivers

The MCBXC167 board supports dual CAN interfaces using the TLE 6253 CAN driver (IC1) and termination resistors (R18 and R19). The driver settings may be changed by the configuration jumpers.

LIN Interface

The Single-wire transceiver TLE 6258 on the MCBXC167 supports LIN specification 1.2.

Status LED's

The MCBXC167 BASIC board has the following status LED's:

- **POWER** shows that the power supply is connected to the board.
- **RESOUT** indicates the status of the Reset Output pin.
- **P2.8 – P2.15** are connected to eight Port 2 pins and can be used to display program status while testing your applications.

The MCBXC167 NET board adds the following status LED's for the Ethernet controller:

- **ACT** shows activity (transmit or receive) on the Ethernet connection.
- **SPEED** is active in the 100Mbit mode of the Ethernet controller.
- **LINK** indicates a physical link to another Ethernet device.

Serial Port

The MCBXC167 board supports the ASC0 on-chip Serial UART and uses a MAX232 (IC10) to convert the logic signals to RS-232 voltage levels. The ASC0 UART might be used in bootstrap mode to download debugging utilities like the Monitor-166 or Flash Programming tools.

The serial port (COM) is configured as a standard 3-wire interface. The DB9 connector is wired to loop the PC's handshaking signals back to the PC.

Technical Data

Parameter	MCBXC167 NET	MCBXC167 BASIC
Supply Voltage	6V DC	6V DC
Supply Current	250mA typical, 400mA maximum	250mA typical, 400mA maximum
XTL Frequency	8 MHz (Allows up to 40MHz on-chip clock)	8 MHz (Allows up to 40MHz on-chip clock)
Memory	512 Kbyte high-speed RAM 2 Mbyte off-chip Flash ROM	512 Kbyte high-speed RAM
CPU	Infineon XC167	Infineon XC167
Peripherals	1 × RS232 Interface, 1 × OCDS Interface, 2 × CAN Interfaces, 1 × 10/100 Mbps Ethernet Controller, 1 × LIN Interface	1 × RS232 Interface, 1 × OCDS Interface

2.3 Asix AX88796 Ethernet Controller

The AX88796 Fast Ethernet Controller is a high performance and highly integrated local CPU bus Ethernet controller with embedded 10/100 Mbps PHY/Transceiver and 8K * 16 bit SRAM. The AX88796 supports both 8 bit and 16 bit local CPU interfaces which include MCS-51series, 80186 series, MC68K series CPU and ISA bus. The AX88796 implements both 10 Mbps and 100 Mbps Ethernet function based on IEEE802.3 / IEEE802.3u LAN standard. The AX88796 also provides an extra IEEE802.3u compliant Media Independent Interface (MII) to support other media applications. Using MII interface, Home LAN PHY type media can be supported. The chip also provides optional Standard Print Port (parallel port interface), can be used for printer server device or treat as general I/O port. The chip also supports up to 3/1 additional General purpose In/Out pins.

Features:

- Highly integrated with embedded 10/100Mbps MAC, PHY and Transceiver.
- Embedded 8K * 16 bit SRAM.
- Compliant with IEEE 802.3/802.3u 100BASE-TX/FX specification.
- NE2000 register level compatible instruction.
- Single chip local CPU bus 10/100Mbps Fast Ethernet MAC Controller.
- Support both 8 bit and 16 bit local CPU interfaces include MCS-51 series, 80186 series and and MC68K series CPU.
- Support both 10Mbps and 100Mbps data rate.
- Support both full-duplex and half-duplex operation.
- Provides an extra MII port for supporting other media. For example, Home LAN application.
- Support EEPROM interface to store MAC address.
- External and internal loop-back capability.
- Support Standard Print Port for printer server application.
- Support up to 3/1 General Purpose In/Out pins.
- 128-pin LQFP low profile package.
- Low Power Consumption, typical under 100mA.
- 0.25 Micron low power CMOS process. 25MHz Operation, Pure 3.3V operation with 5V I/O tolerance.
- Non lead free part number is AX88796 L.
- RoHS compliant part number is AX88796 LF.

3 Application

The application development starts with the initialization and configuration part for XC167CI and AX88796 chip.

3.1 Initialization and Configuration

The *AX88796.c* module provides all the functions for AX88796 initialization, configuration and packet transmission / reception management. The Ethernet controller can be initialized either by using external EEPROM (where all configuration words are automatically downloaded into the AX88796 internal registers at reset time) or by filling them through software. In this application note the Ethernet controller is initialized and configured by the function.

```
static err_t AX88796_init(struct netif *netif) ;
```

This function initializes packet transmission / reception control registers at start-up and sets up the AX88796, using its register set. The register sets included are as follows:

a) Page Start and Page Stop registers for the Receiver Buffer Ring:

```
        HVAR (unsigned char, PSTART) = SM_RSTART_PG;
        HVAR (unsigned char, PSTOP)  = SM_RSTOP_PG - 1;
```

b) Current Page and Boundary Page registers for initialization of Receiver Buffer Ring:

```
        HVAR (unsigned char, CPR1)   = SM_RSTART_PG + 1;
        HVAR (unsigned char, BNRY)   = SM_RSTART_PG;
```

c) Transmit Page Start Register initialization during Transmission of Packet:

```
        HVAR (unsigned char, TPSR)   = SM_TSTART_PG;
```

d) Interrupt Service Routine (ISR) Registers initialization:

```
        HVAR (unsigned char, ISR)    = 0xFF;
```

e) The Ethernet MAC Address :

```
         #define      MYMAC [6]       “00:03:19:00:00:02”
```

```
err_t  AX88796if_init(struct netif *netif) ;
```

This function initializes the lwIP network interface, and calls *AX88796_init( )* to initialize the hardware.
The header file *AX88796.h* defines all AX88796 internal registers, for more details refer to the AX88796 datasheet.

3.2 Ethernet Packet Management

Polling method is used to serve the Interrupt Request. Whenever the interrupt request arrives for transmission or reception of a packet *AX88796_service( )* routine is called which is the actual interrupt service routine and called whenever AX88796 needs servicing.

```
AX88796_service( ):
```

This function checks for the Interrupt service Request at ISR register. Here, the ISR register is checked for Receive Packet. When the packet is received correctly i.e. ISR’s ‘0’th bit is high, *AX88796if_input( )* is called to get a received packet and then forwards the packet to protocols handler i.e. higher level interface. This function is called when *AX88796* needs service. It tests *AX88796* whether it needs service.

`AX88796if_input( ):`

This function is called when a packet is received by the *AX88796* and is fully available to read. It moves the received packet to a *pbuf* which is forwarded to the IP network layer or ARP module. It transmits a resulting ARP reply or queued packet. It uses *AX88796_input( )* function to move the packet from *AX88796* to a newly allocated *pbuf.* Function is called from *AX88796_service( ).*

`AX88796if_output( ) :`

Before writing a frame to the *AX88796*, the ARP module is asked to resolve the Ethernet MAC address. The ARP module might undertake actions to resolve the address first, and queue this packet for later transmission. It uses *AX88796_output( )* to transfer the packet. This function is called from lwIP.

`AX88796_output( ) :`

This function transmits a frame size initially. If the receiver is capable of accepting the frame of the given size, it transfers a packet to *AX88796* for transmission.

`AX88796if_service( ) :`

This function is called in a polling manner, or only after the *AX88796* has raised an interrupt request.

Summarizing, the *AX88796.c* module moves the received packet from the Ethernet controller memory to the reception message and the packet to transmit from the transmission message to the Ethernet controller memory.

3.3 lwIP TCP/IP Stack

The TCP/IP stack used in the solution is the lwIP open source stack from www.sics.se. It provides the necessary protocols for Internet communication, with a very small code footprint and RAM requirements. The lwIP implementation is designed to have only the absolute minimal set of features needed for a full TCP/IP stack. It can only handle a single network interface and contains only a rudimentary UDP implementation, but focuses on the IP, ICMP and TCP protocols. The full TCP/IP suite consist of numerous protocols, ranging from low level protocol such as ARP which translates IP addresses to MAC addresses, to application level protocols. The lwIP is mostly concerned with the TCP and IP protocols and upper layer protocols will be referred to as "the application".

The lwIP main features are as follows:

- IP (Internet Protocol) including packet forwarding over multiple network interfaces.
- ICMP (Internet Control Message Protocol) for network maintenance and debugging.
- UDP (User Datagram Protocol) for Datagram data.
- TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery / fast transmit.
- Specialized non-copy API for enhanced performance.
- Optional Berkeley Socket API.
- Very small code size.
- Very low RAM usage, configurable at compile time.
- Any number of concurrently active TCP connections, maximum amount configurable at compile time.
- Any number of passively listening (server) TCP connections, maximum amount configurable at compile time.
- Free for both commercial and non-commercial use.
- RFC compliant TCP and IP protocol implementation, including flow control, fragment reassembly and retransmission time-out estimation.

A pbuf (Packet Buffer) is lwIP's internal representation of a packet, and is designed for the special needs of minimal stack. The pbuf structure has support for allocating dynamic memory to hold packet contents, and for letting packet data reside in static memory.

Pbuf's are of three types:

PBUF_POOL:
This type consists of fixed size pbufs allocated from a pool of fixed size pbufs. This are mainly used by network device drivers since the operation of allocating a single pbuf is fast and is therefore suitable for use in interrupt handler.

PBUF_ROM:
PBUF_ROM pbufs are used when an application sends data that is located in memory managed by the application.

PBUF_RAM:
Pbufs of PBUF_RAM type are used when an application sends data that is dynamically generated. In this case, the pbuf system allocates memory not only for application data, but also for headers that will be prepended to the data.

The pbuf module provides functions for manipulation of pbufs which are as follows:

`pbuf_alloc( ) :`
This function allocates pbufs of any of the three types described above.

`pbuf_ref( ):`
This function increases the reference count. If the reference count reaches zero the pbuf is deallocated.

`pbuf_free( ):`
This function deallocates the assigned pbufs which first decreases the reference count of pbuf.

`pbuf_realloc( ):`

This function shrinks the pbuf so that it occupies just enough memory to cover the size of the data.

`pbuf_header( ):`
This function adjusts the payload pointer and the length fields so that the header can be prepended to the data in the pbuf.

`pbuf_chain( ) and pbuf_dechain( ):`
These functions are used for chaining and de-chaining the pbufs.

In lwIP device drivers for physical network hardware are represented by a network interface structure similar to that in BSD. The network interface structure is shown below:

```
struct netif {
        struct netif *next ;
        char name[2];
        int num ;
        struct ip_addr ip_addr;
        struct ip_addr netmask;
        struct ip_addr gw;
        void (* input) (struct pbuf *p, struct netif *inp);
        int (* output) (struct netif *netif, struct pbuf *p, struct ip_addr
                        *ipaddr);
        void *state;
};
```

Figure 2 The netif structure

The network interfaces are kept on a global linked list, which is linked by the **next** pointer in the structure. Each network interface has a name, stored in the **name** field in fig. This two letter name identifies the kind of device driver used for the network interface and is only used when the interface is configured by a human operator during runtime. The **num** field is used to distinguish different network interfaces of the same kind.

The three IP addresses **ip_addr**, **netmask**, **gw** are used by the IP layer when sending and receiving packets. The **input** pointer points to the function the device driver should call when the packet has been received. A network interface is connected to a device driver through the **output** pointer. Finally, the **state** pointer points to device driver specific state for the network interface and are set by the device driver.

To send the data, the application passes a pointer to the data as well as the length of the data to the stack. lwIP uses an event driven interface where the application is invoked in response to certain events. lwIP calls the application when data is received, when data has been successfully delivered to the other end of the connection, when a new connection has been set up, or when data has to be retransmitted. The application is also periodically polled for new data. The application provides only one callback function; it is up to the application to deal with mapping different network services to different ports and connections.

3.4 Application Layer

The lwIP uses an event-driven interface where the application is invoked in response to certain events. An application running on top of the lwIP stack is implemented as a C function that is called in response of the following connection events:

a) Accept
b) Sent
c) Receive
d) Error
e) Polling

lwIP provides two Application Program's Interfaces (APIs) for programs to use for communication with the TCP/IP code: the sequential API (often just called "the API") and the raw TCP/IP interface. This document is intended as a description of the latter. For lwIP versions lower than 0.5, this API was not documented.

The sequential API provides a way for ordinary, sequential, programs to use the lwIP stack. It is quite similar to the BSD socket API. The model of execution is based on the open-read-write-close paradigm. Since the

TCP/IP stack is event based by nature, the TCP/IP code and the application program must reside in different execution contexts (threads).

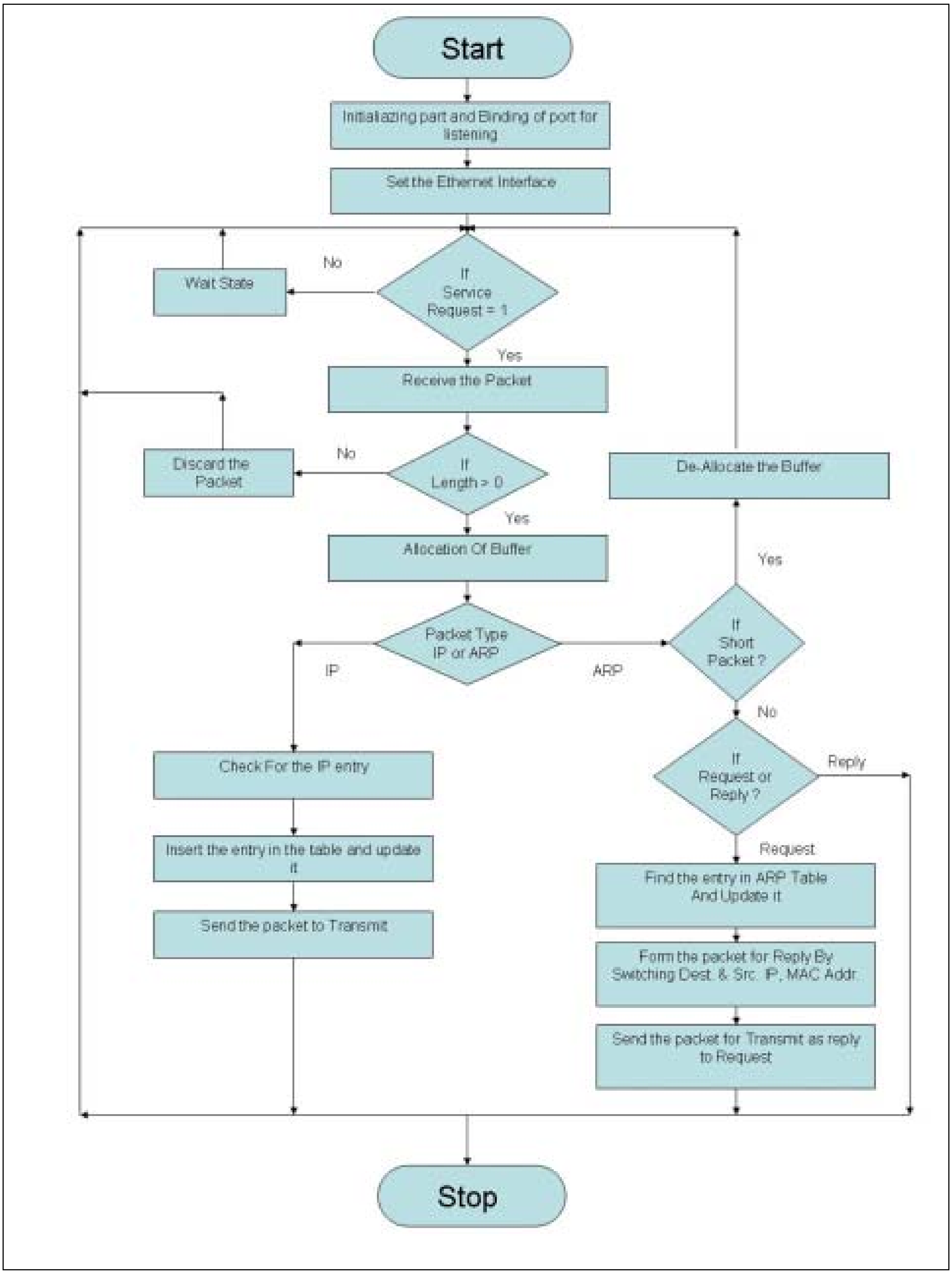
The raw TCP/IP interface allows the application program to integrate better with the TCP/IP code. Program execution is event based by having callback functions being called from within the TCP/IP code. The TCP/IP code and the application program both run in the same thread. The sequential API has a much higher overhead and is not very well suited for small systems since it forces a multithreaded paradigm on the application.

The raw TCP/IP interface is not only faster in terms of code execution time but is also less memory intensive. The drawback is that program development is somewhat harder and application programs written for the raw TCP/IP interface are more difficult to understand. Still, this is the preferred way of writing applications that should be small in code size and memory usage.

Both APIs can be used simultaneously by different application programs. In fact, the sequential API is implemented as an application program using the raw TCP/IP interface.

A simple example application has been implemented as a demonstration of stack functionalities. The application is a simple HTTP server that manages a typical Web Site: a homepage and several linked pages.

4 Flowchart


Start
Initializing part and Binding of port for listening
Set the Ethernet Interface
Wait State
No
If Service Request = 1
Yes
Receive the Packet
Discard the Packet
No
If Length > 0
Yes
De-Allocate the Buffer
Allocation Of Buffer
Yes
Packet Type IP or ARP
IP
ARP
If Short Packet ?
No
If Request or Reply ?
Reply
Request
Check For the IP entry
Insert the entry in the table and update it
Send the packet to Transmit
Find the entry in ARP Table And Update it
Form the packet for Reply By Switching Dest. & Src. IP, MAC Addr
Send the packet for Transmit as reply to Request
Stop

5 Example: TELNET Application with Ping Request

In this example TELNET application has been implemented using lwIP protocol stack. Before implementing the TELNET application, connection has been checked between AX88796 using Ping request and reply application. The implementation steps are as follows:

1) Initialize the memory, buffer, timer for XC167-C1, network interface, IP address, gateway Subnet mask.

   `mem_init( ), pbuf_init( ), GPT1_init( ), netif_init( )` etc.

2) Set up the Ethernet interface by initializing it for given network interface, ip address, gateway, subnet mask.

```
ethif= netif_add (&ipaddr, &netmask, &gw, NULL, AX88796if_init, ip_input).
```

3) Set this Ethernet interface as default interface.

```
netif_set_default(ethif).
```

4) Initialize TELNET application by assigning a New Process control block (pcb) and binding this pcb to port 23 which is TELNET port and in listening state.

```
pcb = tcp_new( )
tcp_bind(pcb, IP_ADDR_ANY, 23)
pcb = tcp_listen(pcb)
```

5) Globally enable the interrupts.

```
PSW_IEN        = 1.
```

6) AX88796 is initialized in polling mode. So it always polls AX88796 and check whether the service is required. So whenever service is required, interrupt occurs and packet is received by AX88796.

```
AX88796if_service(ethif).
```

7) AX88796 checks the received frame length and allocates a buffer for a frame.

```
p=pbuf_alloc(PBUF_RAW, len, PBUF_POOL)
```

8) According to the Type of the packet IP or ARP, reply is sent back.

```
switch (htons (ethhdr->type))
 {
 case ETHTYPE_IP:
   q= etharp_ip_input(netif, p);
   pbuf_header(p,-14);
   netif->input(p, netif);
   break;
 case ETHTYPE_ARP:
   q=etharp_arp_input(netif, (struct eth_addr *) &netif->hwaddr, p);
   break;
```

9) After the successful reception of reply, the TELNET port is bind to given IP address and depending on the event, the bank of blinking LED's which are initialized by the timer are switched ON or OFF.

```
      err_t lwip_tcp_event (void *arg, struct tcp_pcb *pcb, enum lwip_event ev,
                              struct pbuf *p, u16_t size, err_t err)
  {
         int retvalue = ERR_OK;
         char * Welcome = "Hello World!\n\rYou can switch on and off the
                            blinking LED\n\rPress 0 to stop the blinking and 1 to
                            start it\n\rPress q to quit\n\r\n\r";
         char * OFF = "Blinking LED Switched OFF\n\r";
         char * ON = "Blinking LED Switched ON\n\r";

        switch(ev) {
        case LWIP_EVENT_ACCEPT:
              tcp_write(pcb, (unsigned char *)Welcome, strlen(Welcome), 1);
        break;
        case LWIP_EVENT_SENT:
        break;
         case LWIP_EVENT_RECV:
              if (p!=NULL) {
              switch (*(unsigned char *)p->payload) {
                        case '0':
                        GPT1_vStopTmr_GPT1_TIMER_2(GPT12E_T2CON_T2R);
                        tcp_write(pcb, (unsigned char *) OFF, strlen(OFF), 1);
              break;
                        case '1':
                        GPT1_vStartTmr_GPT1_TIMER_2(GPT12E_T2CON_T2R);
                        tcp_write(pcb, (unsigned char *)ON, strlen(ON), 1);
              break;

              default:
              break;
              }
            }
           break;
           }
return retvalue;
}
```

6 Conclusion

The solution provides a better way to use lwIP to add Ethernet connectivity to Infineon's XC167CI with minimum system resources. The stack also supports TELNET application as it is explained in the example. lwIP is not thread safe i.e. it interferes if it is used in multithreading environment. But the system with minimum system resources and memory it provides a better option to add Ethernet connectivity. As lwIP is an open source code, the user can easily modify the code as per the application.

Table 1 Definitions

Acronym	Definition
TCP	Transmission Control Protocol
IP	Internet Protocol
UDP	User Datagram Protocol
DHCP	Dynamic Host Configuration Protocol
ICMP	Internet Control Message Protocol
ARP	Address Resolution Protocol
MAC	Media Access Control
LAN	Local Area Network
DNS	Domain Name System
OSI	Open System Interconnection

7 References

[1] Infineon's 16-bit XC167CI Datasheet

[2] Asix AX88796 Ethernet Controller Technical Reference Manual

[3] Keil's XC167CI Evaluation Board Product Datasheet

[4] lwIP V0.5 Reference Manual

8 LWIP Copyright