Adding Ethernet Connectivity to DSP-Based Systems

M. Nashaat Soliman *Novra Technologies Inc.*

ABSTRACT

An ever-increasing number of appliances incorporate Ethernet connectivity. With the popularity of the Internet, IP protocol connectivity is required in many applications. In this application note we present a functional structure for an Ethernet driver and a mini IP/UDP stack that can be incorporated in any Internet appliance. This driver structure is flexible to allow for hardware changes (changing the Ethernet chip) with minimal code changes. We suggest some implementation constructs using DSP/BIOS kernel objects. We also highlight the advantages of using the DSP/BIOS objects to solve certain problems as opposed to a structure without an operating system.

Contents

List of Figures

1 Introduction

1.1 Hardware Architecture

Ethernet is a very popular local area network (LAN) solution. As more applications require some kind of network connection, the implementation of an Ethernet connection can prove to be a winning choice. Figure 1 shows the hardware architecture to add an Ethernet connection to a C54X DSP family.

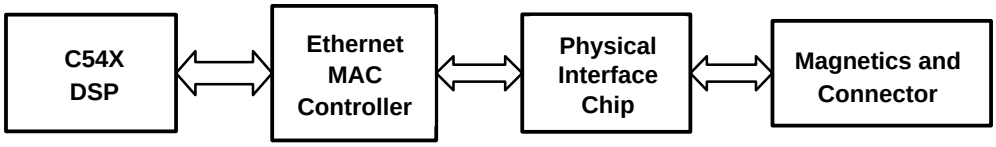

Figure 1. Hardware Architecture for Ethernet Connection

The hardware components are the Ethernet MAC controller and the physical interface chip, although some manufactures offer an integrated solution that combines the two components onto a single chip. The communication between the DSP and the physical interface chip (**PHY**) would still be the same.

There is a magnetics module between the RJ45 connector and the Ethernet PHY device that typically consists of a transformer and a common mode choke. The main function of the transformer is to provide electrical isolation between the system and the outside world; also, the common mode chokes keep system noise from transferring to the twisted pair cable.

The Ethernet MAC controller is connected to the external parallel interface. Usually an interrupt is tied to the chip to get notifications of the different data transfer events to and from the MAC controller external DMA. If the DSP supports DMA to external memory then this would be the preferred mechanism to transfer data. In that case, the Ethernet MAC can use the DSP ready line to request externally generated wait states (if the MAC controller supports this type of handshaking). This allows the MAC controller to accept data at the proper speed without any intervention from the CPU.

The PHY chip is managed through the standard media independent interface management (MII) whether it is integrated within a single chip or not. Communication with the PHY chip is required for resetting it, getting link status and setting the auto negotiation parameters. The data sheet for the PHY chip contains the description of the interface and frame format for communication as well as the register configuration for the chip.

The MAC controller will accept complete Ethernet frames with Ethernet headers from the DSP. They will be stored in an internal buffer and then scheduled for transmission whenever the media is available. The number of buffers in the chip and the queuing capability will vary from chip to chip. Received packets are queued in the Ethernet MAC chip until the DSP reads them out and frees the buffer. Again, different chips will have different mechanisms. The data sheet describes the register configuration as well as the procedures for transmit and receive.

In this document the main focus is the software structure and how to isolate it from the hardware.

1.2 Ethernet Frame Structure

There are two types of Ethernet frames: the IEEE 802.3/802.2 encapsulation and the Ethernet encapsulation(RFC 894) . Ethernet frames are more popular due to their simplicity. Figure 2 shows the Ethernet frame structure.

Destination MAC address (6 bytes)	Source MAC address (6 bytes)	Protocol ID (2 bytes)	Payload Data	CRC

Figure 2. The Structure of Ethernet Frame

Source and destination addresses identify the sender and the recipient of the frame, and the protocol ID identifies the type of the payload data (0x0800 for IP traffic). The CRC (Cyclic Redundancy Check) is computed and inserted by the Ethernet chip, and the DSP code isn't involved.

As a node on the Ethernet bus, your device has to have a unique MAC address. The IEEE administers MAC addresses, and you need to get an OUI (Organization Unique Identifier) from the IEEE to assign MAC addresses to your devices. *Refer to reference 1 for more information.*

Your MAC address is used to form the frames originating from your device. Other devices that need to communicate back to you use this MAC address. An important component in the application code is the Address Resolution Protocol (ARP), which will enable discovery of the MAC addresses based on the IP addresses.

ARP protocol is specified in reference 2.

1.3 IP (Internet Protocol)

1.3.1 Why IP Packets?

IP packets are carried inside Ethernet frames or any other link layer protocol (PPP, SLIP, HDLC…etc). So the IP packets can traverse many heterogeneous networks to arrive to their destination. Even though your destination may be on the same network (same Ethernet wire) IP is implemented in every networked solution. Every PC can talk to IP devices (otherwise it won't go on the internet). From this we sum the answer as…

If you want your device to communicate with any other device that can go on the Internet, you have to be able to communicate using IP protocol.

1.3.2 IP Packet Format

IP packets have their own addressing mechanism. IP version 4 – used by the majority of systems –has a 4-byte address. This means that for your device to be addressable it needs an IP address, which is configurable by the device user. MAC addresses are hard coded. On rare occasions your device will need to change its MAC address. Figure 3 shows the IP packet format.

The payload of an IP packet is another protocol data unit for a higher application. The application protocols that we will address here in this application note are UDP and ICMP. Other application protocols can be implemented like IGMP and TCP. They are supported by the functional structure given in this application note but their specific implementation requirements are not discussed here. *Reference 1 describes all of these protocols in details.*

Offset in Bytes	Field Description			
0	4-bit version (0x4)	4-bit header length	8-bit Type of service	16-bit total length of the packet including data
4	16-bit identification (incrementing counter)		3-bit flags	13-bit fragment offset
8	8-bit time to live (TTL)	8-bit application protocol ID		16-bit header checksum
12	32-bit source IP address			
16	32-bit destination IP address			
20	Variable length data (payload) …. …. ….			

Figure 3. Packet Format for IP Packets

1.4 User Datagram Protocol (UDP)

So, what goes in the IP payload? It is UDP datagrams. UDP is very straightforward. You have a message to send. You put the message in an envelope (which is the UDP header), and then give it to the IP service to send it. Therefore, it is the application's responsibility to form the UDP headers. Figure 4 shows the UDP datagram structure.

Source port (2bytes)	Destination port (2 bytes)	Length	Checksum	Payload

Figure 4. UDP Datagram Structure

Source and destination ports allow different messages to be exchanged for different applications between two end systems, as Figure 5 illustrates. The application on end system 2 can talk to the two different applications on end system 1. It does so by sending messages on two different destination ports.

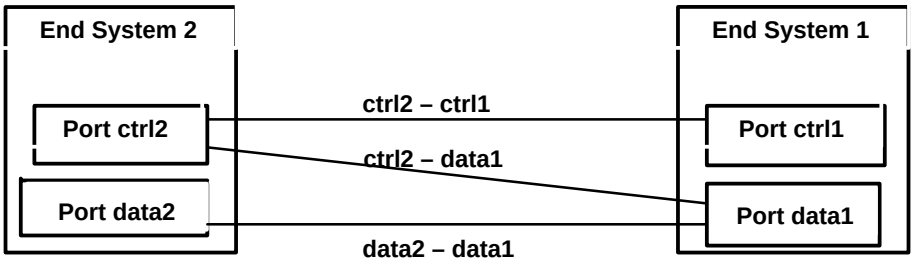

Figure 5. UDP Ports and Different Applications Messages

1.5 Internet Control Message Protocol (ICMP)

A very common way for trouble shooting network configuration is the echo request/echo reply message pair (ping). They are part of the ICMP protocol. Our implementation for the stack will include this functionality in order to provide a basic network troubleshooting functionality. The way our implementation handles ping messages requires minimal knowledge of the protocol.

ICMP is another type of datagram that is carried inside IP packet. *Reference 1 provides a comprehensive discussion of the protocol and packet format.*

1.6 Communication Architecture

The communication architecture is composed of different layers. They interact with each other to provide the communication service to the application generating or receiving the data. Our application will use the architecture shown in Figure 6.

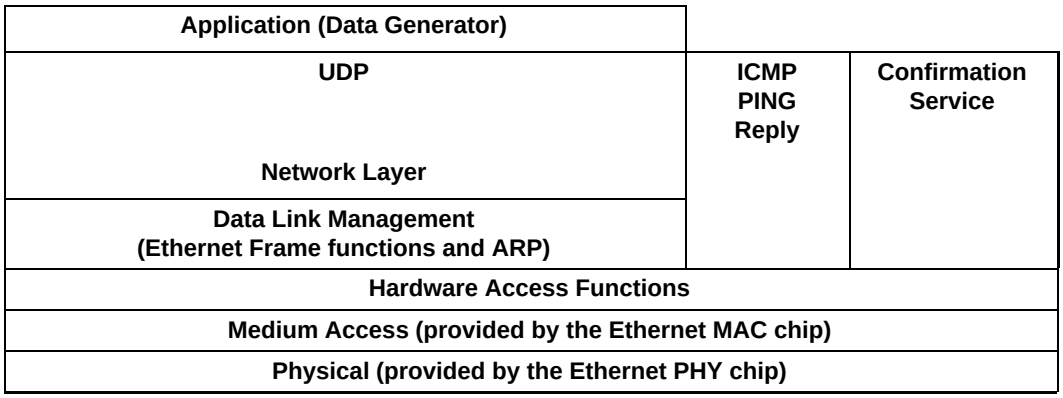

Figure 6. The Communication Architecture in our UPD/IP Application

Even though this architecture might seem complicated, and violates the layering principal, it is more efficient. The ping packets do not need to go through Link management functions. Instead, they are replied to quite quickly by swapping some fields and bouncing the packet back.

Also, the confirmation service, which isn't really part of a standard stack, presents a fast track for acknowledging the reception of critical control packets (if the application is actually using any). Later on we will discuss the implementation of this service.

2 Functional Structure

In this section we will divide and specify the functionalities of these layers and even divide them in sub-layers if necessary. We begin with the Ethernet hardware access functions and work our way up.

2.1 Hardware Access Functions

The hardware access functions will provide the interface between Ethernet chip and link layer functions. They are required to provide an interface that is totally independent of how the Ethernet chip works or how it is connected to the DSP.

There are two basic sub layers that can be identified in the hardware access functions: the I/O layer and the functional layer. The I/O functions will provide read/write services that hide the nature of the physical interface (i.e., whether DMA is used, 8-bit interface, …etc). Also these functions will provide the MII interface read/write functions.

This functional layer offers a set of commands that can be sent to the Ethernet chip. These commands are: Initialize, Send Frame, Get received Frame Information, and Get Received Frame. These functions are the service primitives of the Ethernet chip. Figure 7 shows this structure.

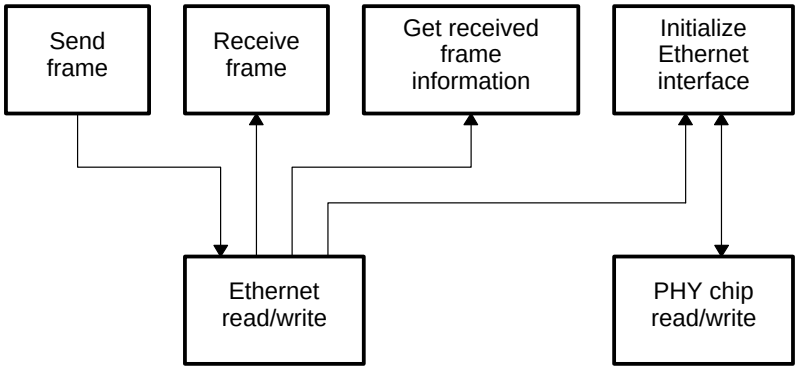

Figure 7. Functional Structure for Hardware Access Functions

The advantages of this functional structure are how it can adapt to design changes. You can change the PHY chip or change the way the Ethernet chip is connected to the bus and only worry about changing a single set of subroutines, rather than making changes all over your code.

It might not seem very clear why we included a “Get received Frame information” as a service primitive. The argument is that the Ethernet chip may, depending on its configuration, receive some frames that it is not interested in processing (broadcast for example). These frames need not be copied entirely from the Ethernet chip to the DSP; they can be deleted from the chip directly. This will save time, since we need to check the first 42 bytes only to get all the information on the Frame.

The receive frame service supports zero-length receive, which clears the frame from the Ethernet chip buffer without moving it to the DSP memory.

2.2 Data Link Management

The functions in this layer handle the IP packets queuing for transmission. The IP packets formed in the higher layers are encapsulated with an Ethernet header and sent to the chip for physical transmission. One important function is to find the proper MAC address for transmitting this packet. This is done by maintaining a table that has the known mappings. If the address is a broadcast or multicast, direct mapping rules apply. If the address is outside the scope of the LAN, the MAC address of the default gateway is used (in a sense this is a routing decision).

Also this layer dispatches incoming Ethernet packets to the proper destination function. So Ping packets are passed to the ICMP ping reply process and other IP packets go the IP packet processor. Figure 8 shows the interaction between data link management functions.

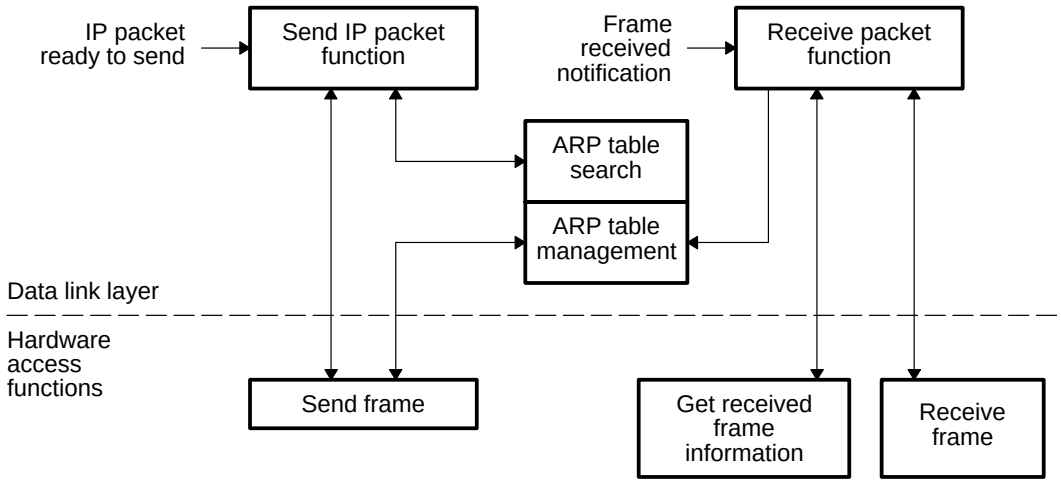

Figure 8. Interaction Between Data Link Management Functions

If the ARP table doesn't have the entry corresponding to the IP address of the packet, the management function will send an ARP query frame and the packet will be dropped. The receive function gets frame information first before deciding to receive the frame or clear it.

The ARP table administration is described in reference 2.

2.3 The Network/UDP Layer

Even though it might seem unusual, the UDP layer and the network layer are merged together. Why? The only protocol that uses the IP encapsulation services is UDP, so save the extra call, save the extra code, and merge them together.

The functions in this layer need to put IP and UDP headers in front of the data. The application provides the data, and specifies the destination and source ports, and the destination IP address. The function encapsulates the data with a header that is formed based on this information.

Also, there is a function that receives the IP packets, verifies they are addressed to the device (i.e., the destination address is the same as its own address), and then dispatches them to the packet processors based on the destination port. In some cases, some information about the source may be provided depending on the type of processing the application does with the data (is it receiving from multiple sources).

Forming the IP header is a fairly straightforward process. Most of the fields are fixed, especially with only one protocol to serve (UDP). Basic information that should be filled is easily calculated from the data length and the destination information. The check sum is a one's complement sum of the header treated as 16-bit words. The carry is added to the sum until there is no carry.

Fragmentation option in the IP service is not implemented; we are assuming the application is nice enough to keep its individual packets under the 1500-byte limit on Ethernet frames. This assumption is realistic for most embedded applications, since it would be better if the application segments its traffic.

Figure 9 shows a typical packet. The default values selected for the header will be fixed; they usually fit most applications, unless a special requirement is set on the TTL or the TOS fields. The offsets shown are the offset from the beginning of the buffer in words.

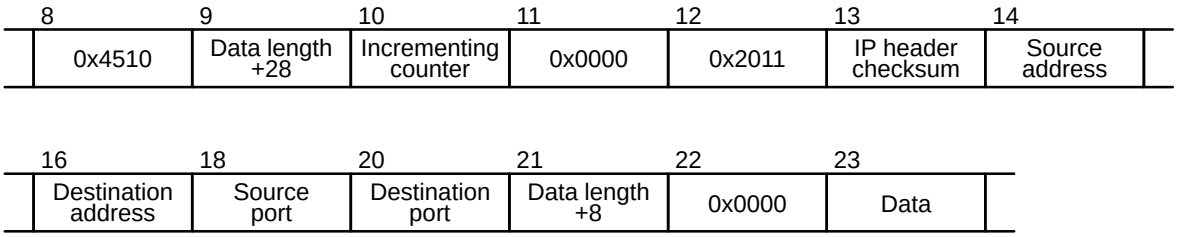

Figure 9. Typical Header, With Default Fields That Fits Most Applications

The packet is formed in this fashion so that Ethernet headers can be inserted in front. The Ethernet header is 7 words only; the first word is dummy word and is ignored when the packet is sent. However, it makes sure the data and the headers are on the double word boundary so that efficient assembly code can be used if the application needs it.

When the locations for the headers are reserved up front, the communication functions don't need to copy the data to form the packet; they just insert the headers and move it to the Ethernet chip. We will see how the kernel objects can be used to implement such functions easily.

2.4 Icmp Ping Reply, and Confirmation Service

ICMP ping reply, and confirmation services provide a quick and convenient way to acknowledge a packet by echoing it back. For acknowledgement you interchange all the source addresses and destination addresses. The simplicity of this approach follows from the fact that all checksums are the same and your code doesn't have to recalculate any of them and the packet is ready for transmission.

For Ping reply, you will need to change the packet type into reply by subtracting 8 from the first byte in the ICMP header. The checksum needs to be adjusted by adding 8 in the high byte and adding the carry. Again, no need for checksum calculations, and the packet is ready to go. This approach saves the time spent to search the ARP table for the reply MAC address. You need to be able to tunnel Complete Ethernet Packets through the Driver

3 Implementation Challenges

The implementation of this communication hierarchy without a proper operating system will not provide reusable and well-defined interfaces. The following problems need to be addressed.

3.1 Synchronization

The Ethernet chip needs a synchronization method so the write operations aren't performed until any ongoing read operation is complete. A global variable needs to be continuously polled and wouldn't provide a notification when it is changed.

3.2 Notification

Many components will need an event to start processing. For example, the Send IP packet function needs to be notified that there is a packet ready for transmission before it can start processing it. Also, the receive IP packet function needs a notification mechanism: the Ethernet chip will notify the DSP through an interrupt of a receive event. However, it is not suitable to run the receive function in the ISR context.

3.3 Queuing

Packets that are ready for transmission need to be queued until the hardware is available. Therefore, a mechanism for storing the ready packets, or pointers to these packets in a FIFO list needs to be implemented. Received packets can be processed on the fly, relying on the internal Ethernet chip buffers to store incoming frames during the processing of the current packet.

3.4 Buffering

The application has data that needs to be sent out. It needs memory to put this data in. There are many techniques that can be used to secure the memory needed by the application. However, we must guarantee that this memory includes a space for the headers. We need to consider the number of buffers that must be available and the size of these buffers as well.

3.5 Scheduling

How are these different components going to run? A complex question needs a simple answer. Especially if some functions have to wait on shared resources, because the Ethernet chip is shared between transmit and receive functions. Also, the ARP management function needs to run periodically to expire the entries in the table.

4 Implementation Model Using DSP/BIOS Objects

Now that we have presented the challenges, we will present an implementation model using some DSP/BIOS objects to address these challenges. The main three objects that we will use are Semaphores, Queues, and Tasks. Semaphores can be used to grant mutually exclusive access to shared resources. They also can be used to notify a task of some event. The queues can be used to store packets ready for transmission (or descriptors for these packets). They can also be used to allocate buffers for the data in some situations.

We will need two tasks, one Interrupt service routine, two queues, and three semaphores to implement this stack. The two tasks are Send Packet, and Receive Packet. The rest of the functions are called either by the Data generator (to add headers) or by the Send Packet Task to actually move the data to the hardware.

The semaphores are Transmit Packet Ready, Receive Packet Ready, and Ethernet Chip Available. The Transmit Packet Ready is used to notify the Sender Task that there is a fresh packet added to the transmit queue, and the Receive Packet Ready does a similar function. The Ethernet Chip Available is used by the sender and receiver to regulate access to the hardware.

The two queues hold a structure called Packet Descriptor. One acts as a repository for available descriptors, the other one acts as a queue for packets that are ready for transmission. The Interrupt service routine services the Ethernet interrupt and posts the semaphore when there is a received packet.

4.1 Sending Packets

The API to send raw data will do the following:

- Call encapsulation function to put UDP/IP header on the data.
- Get a free descriptor structure from the free descriptor queue. If the queue is empty, the packet is dropped.
- Put the descriptor in the ready for transmission queue.
- Post the semaphore to notify the task that there is a packet ready for transmission.

Now, what would the sender task do?

- Pend on the semaphore for ready packets.
- Get the descriptor for the ready packet.
- Complete the Ethernet headers for the packet (if necessary)
- Pend on the Ethernet Chip Ready semaphore.
- Call the hardware access functions to send the packet to the chip.
- Free the descriptor so it can be reused.

4.2 Receiving Packet

When the Ethernet chip receives a packet it will interrupt the DSP. The Interrupt service routine will post the semaphore to notify the receive task. So the receive task has the following logic flow…

- Pend on the receive packet semaphore.
- Pend on the Ethernet Chip available semaphore.
- Get the received packet information.
- Decide if it needs to process the packet or not.
- Move the Packet to the DSP memory if it needs to process it. Otherwise delete it.
- Call the proper processing function for the packet depending on which port it was addressed to. ICMP packets go to the ping reply function.

4.3 Packet Descriptors

This structure contains all the necessary information to form the Ethernet packet. The descriptor structure is filled by the application and read by the sender process. The descriptor contains the following information.

- Address of the Buffer containing the packet.
- Length of the packet in bytes
- Flag to indicate if the packet needs Ethernet headers or the header was filled by the application.

The use of descriptors allows the queuing of packets just by pointing at them; this gives the application the flexibility to use any memory management scheme for buffer allocation. Free descriptors are stored in a queue. Descriptors that have information are queued in the packet ready queue. The free descriptor queue is used for two reasons; In case of static memory allocation it guarantees the memory for the descriptors; also, you can control how many packets can be queued by controlling the length of the free queue.

When the packet is sent the sender task will free the descriptor. If the application can't get a free descriptor, the packet can be either delayed or dropped.

4.4 Memory Allocation

The DSP/BIOS kernel comes with a dynamic memory manager. This is ideal for allocating packets of varying size; however there is no guarantee on how long an allocation request would take. If you need to guarantee the memory allocation time, you have to use static allocation. One way of managing the buffers is to store them in a queue. The free buffers queue will contain buffers that can be used by the application to form packets.

This mechanism limits the allocation time to the time of que_get(). The tradeoff is the wasted memory. Allocating buffers of fixed size means that the buffer has to be bigger than any possible packet (1500 Bytes). This means wasted memory.

According to your application requirements, you may choose to have two queues: one for big buffers and the other one for smaller buffers. The application will then get the buffer from the appropriate queue depending on the data size.

Note that no matter how you allocate the buffer, the hardware access functions should free that buffer so it can be reused. Also note that there is usually no use to queue received packets by the application. For efficiency purposes it is better to consume the packets as they arrive.

Figure 10 and Figure 11 illustrate the different components of the implementation and their interaction.

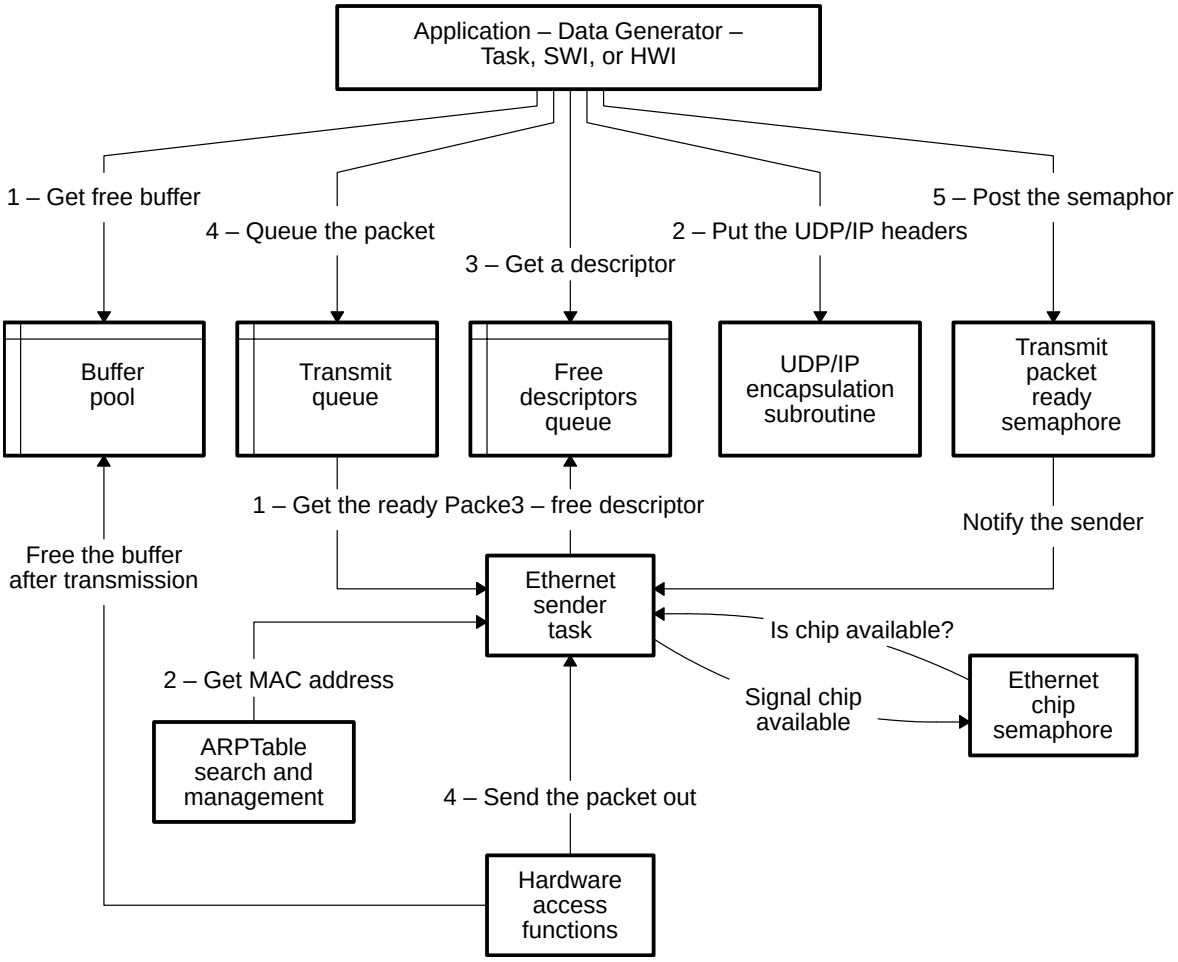

Figure 10. Interaction Sequence for Send Packet Operation

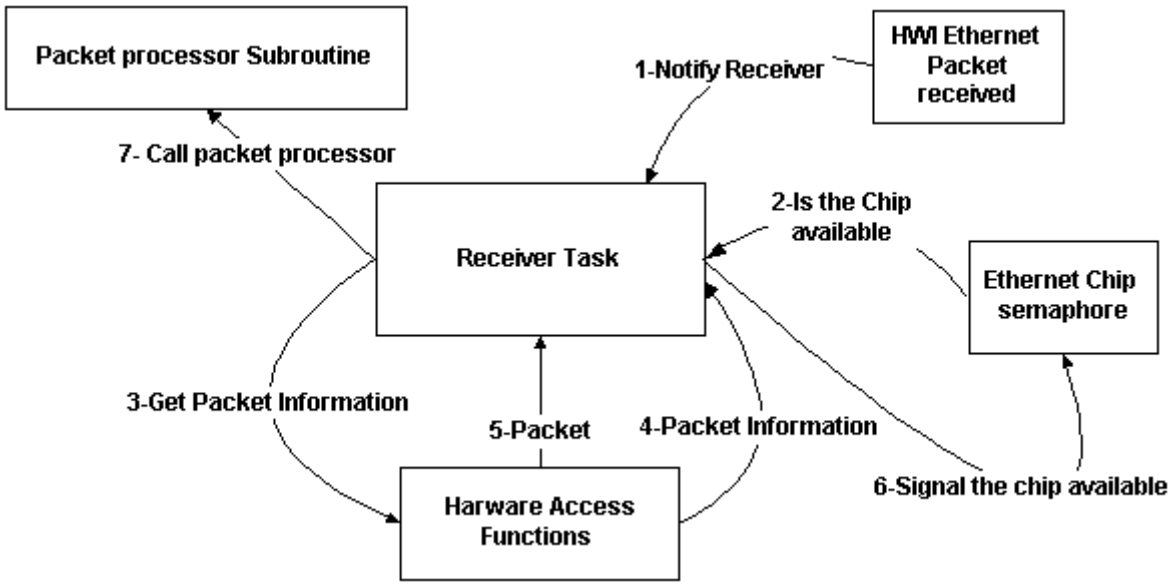

Figure 11. Interaction Sequence for Receive Packet Operation

5 Conclusions and Summary

The DSP/BIOS is a rich operating system that provides useful objects for building complex systems. The implementation that we suggest in this application note confirms this fact. This design may be modified to fit different application needs.

The main factors that an embedded application continuously struggles with are the memory requirements and real-time performance. This design of a driver/communication service can be resized and adapted to the needs of your application. For example you can store incoming packets in a queue so they can be processed by another task.

It is important that you make use of the features of the Ethernet MAC chip that you are using in your design. Some chips provide a sophisticated queue management for transmitter side so you can offload the transmit queue functionality on the chip hardware.

6 References

1. “TCP/IP Illustrated Volume1”, W. Richard Stevens.
2. “An Ethernet Address Resolution Protocol”, RFC 826.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265