VHDL IP Stack

School of Computer Science and Electrical Engineering
University of Queensland, Brisbane, Australia.
http://www.csee.uq.edu.au/

THE UNIVERSITY OF QUEENSLAND

Last Modified: 23 February 2001

Contents

1.0 About this design 1
2.0 Files needed for this design 2
List of Files 2
File Descriptions 2
3.0 Description of the design 4
Design Overview 4
Design Structure 4
4.0 Module Descriptions 5
Ethernet Receiver 5
Ethernet Sender 6
ARP 6
ARP Sender 7
Internet 8
Internet Sender 8
ICMP 9
UDP 9
5.0 Design Considerations 9
Design Limitations 9
Interfacing and Enhancing the Design 10
Design Notes 10
6.0 Memory Map 11

1.0 About this design

This IP stack for an FPGA is a complex design because of the number of layers and the complexity of each that is required. It is limited to 10Mb/s operation and is designed for a full duplex switched network. It implements the lower layers of a standard TCP/IP stack. Further implementation is needed to make it work specifically for a certain purpose (eg a web server). There is support to read and write to RAM from the PC via the parallel port as well, for debugging and tests purposes (this maybe easily removed). Note the design only supports IP and ARP frames, other protocols such as RARP and 802.2 frames are not supported.

2.0 Files needed for this design

List of Files

- arp3.vhd
- arpsnd.vhd
- crcgenerator.vhd
- ethernet.vhd
- ethernetsnd.vhd
- globalconstants.vhd
- icmp.vhd
- internet.vhd
- internetsnd.vhd
- memorymultiplexor-sv01.vhd
- pctosraminterface-sv06.vhd
- sram512kleft16bit50mhzreadreq-sv05.vhd
- sraminterfacewithpport-sv01.vhd
- stack3.vhd (top level)
- udp.vhd
- stackpins.ucf
- cpldnet.vhd
- cpldnetpins.ucf
- XSVSRAMUility.exe

File Descriptions

arp3.vhd

Handles ARP requests and replies that are received, and manages the ARP table.

arpsnd.vhd

Sends ARP requests, replies and handles lookups.

crcgenerator.vhd

Generates CRC checksums.

ethernet.vhd

Receives ethernet frames off the network.

ethernetsnd.vhd

Puts Ethernet frames onto the network.

globalconstants.vhd

Contains the IP and MAC addresses of the device.

icmp.vhd

Responds to ICMP echo requests with an echo reply.

internet.vhd

Handles IP datagrams and reassembly.

internetsnd.vhd

Sends IP datagrams and handles fragmentation.

memorymultiplexor-sv01.vhd

Allows either the PC or the IP stack to access RAM. (See the PC to SRAM Interface Design project for more details)

pctosraminterface-sv06.vhd

Handles the protocol for the communication between the PC and the board.

sram512kleft16bit50mhzreadreq-sv05.vhd

Main interface to the left bank RAM, controls reading and writing.

sraminterfacewithpport-sv01.vhd

Top level for the RAM files.

stack3.vhd

Top level for the design, also contains a RAM arbitrator to share RAM usage between the different layers.

udp.vhd

Simple implementation of UDP.

stackpins.ucf

Constraints file for the design.

cpldnet.vhd

CPLD reconfiguration to set the outputs of the CPLD that are connected to the PHY. (Compile to an SVF file)

cpldnetpins.ucf

Constraints file for the CPLD design.

XSVRAMUtility.exe

Executable file for reading and writing to the onboard RAM from the PC.

3.0 Description of the design

Design Overview

This VHDL design is a simple implementation of a IP stack, but it is lacking a few transport level protocols (such as TCP). It does, however, provide a rich set of lower layers that should allow higher level protocols (such as TCP) to be plugged easily on top of what already exists. Where possible, the design has been kept RFC compliant.

The protocol stack was designed and built on an XSV-300 board (version 1.0) from the XESS Corp. (www.xess.com) using Foundation 3.1 and associated tools to make the design. The board makes use of an LXT970A Ethernet Transceiver (PHY) from Level One (an Intel company) to encode and decode the signals onto a 10Mb/s full duplex twisted pair switched network. (The chip has been forced to 10Mb/s operation and can be changed by reprogramming the CPLD, however this means much of the code will need to be rewritten – see Design Limitations in Design Considerations for more information).

Design Structure

The design is based upon the layered model of a TCP/IP stack – there is an Ethernet (Host to Network) layer that interacts with the PHY device, and also communicates with the ARP and Internet layers. ARP (which holds the ARP lookup table), on the receive side, is considered to act as another protocol on the Internet layer parallel to IP, and on the transmit side it lies transparently between the Internet layer and the Ethernet layer, with a connection to ARP on the send side. The main Internet layer protocol (IP) communicates with the transport layer and its associated protocols (ICMP, UDP) (see Figure 1). Each individual layer is split up into separate VHDL files and processes to simplify the design.

NB For simplicity from here on, the ARP, Ethernet and IP protocols are called the ARP layer, Ethernet layer and Internet layer respectively (this further distinguishes ARP from any layer).

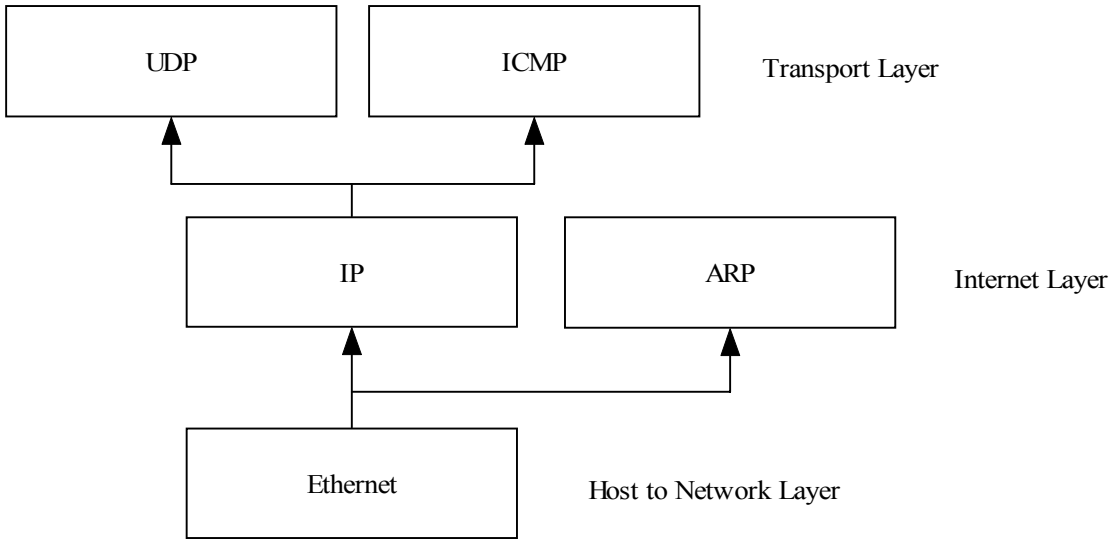

Figure 1. – The layout of the receive side of the IP stack.

It is important to make a distinction between the send and receive sides of the stack, as they differ slightly. Coming back down the stack, the only transport layer protocol that currently sends data (ICMP) lies at the top, and is connected to the Internet Send layer, which lies directly below it. The Internet Send

layer only has the Internet Protocol (IP) available at this layer, as ARP is handled differently on the send side. Directly below the Internet Send layer lies the ARP Send "pseudo" layer, which is transparent to the layers above and below it. The ARP Send layer is also connected to the ARP layer (see Figure 2). Below the ARP Send layer lies the Host to Network Send (Ethernet) layer, which is connected to the PHY and thus the physical network.

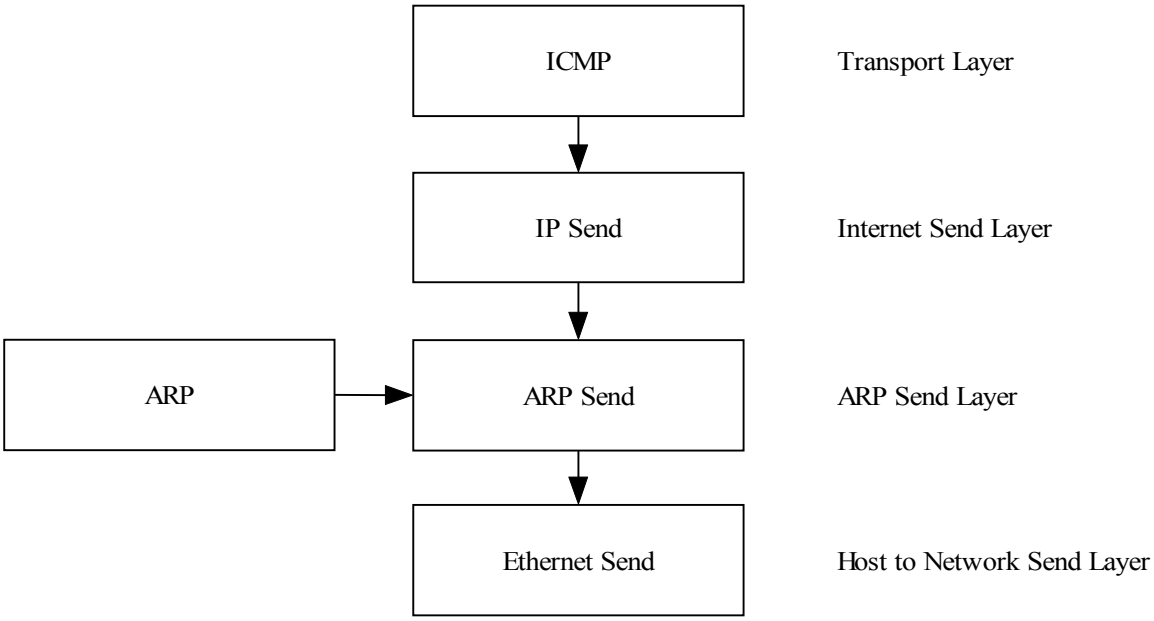

Figure 2. – The layout of the send side of the IP stack.

Any more transport layer protocols should be able to be plugged in parallel to the ICMP layer, with little modification to the stack required. Each layer looks similar to a normal TCP/IP stack in concept and in behaviour, with the slight exception of ARP and ARP Send. See Module Descriptions for a more detailed operation of each layer.

4.0 Module Descriptions

Ethernet Receiver

The Ethernet receiver is connected to the PHY via an MII, with a 4 bit (nibble) wide data bus (rxdata), a receive data valid signal (rx_dv), a receive error signal (rx_er) and a receive clock (rx_clk). When the PHY receives a frame, rx_dv is asserted for the entirety of the frame. When in 10Mb operation, the PHY strips the first 60 bits off the preamble, and leaves the SFD as the first nibble received. On each rising edge of rx_clk, a valid nibble of data will be valid to be latched.

The Ethernet receiver ignores the SFD, then processes the frame header. If the target MAC field in the header of the frame being received does not equal the one set in the global constants package or is not a broadcast, then the frame will be ignored by jumping to a state and doing nothing until rx_dv goes back low. The source MAC field of the header is ignored. If the Ethernet type field is not either 0800h or 0806h then the frame is also ignored, as the only frames accepted are IP and ARP messages.

After the Ethernet type is received, two signals are asserted to tell the above layers that a new frame is being processed, and the type of message that is contained in the frame (either ARP or IP). After each byte (every second nibble) is received, another signal is asserted to tell the above layers that a new byte

has been received and the byte is being passed up. The four CRC bytes are also passed to the next layer, and are ignored on the next layer. As each byte is received (including the frame header and data but not including the SFD) it is also passed to the CRC generator so the CRC can be checked later.

Once rx_dv goes low and the frame has finished being received, the CRC is checked, and the end frame signal is asserted. If the CRC is not zero (i.e. a correct CRC) then the valid signal gets zero, and the next layer should ignore the frame due to the invalid CRC. Otherwise, both the end frame and frame valid signals are asserted at the same time to tell the next layer that a valid frame has been fully received.

rx_er is ignored.

The Ethernet receiver passes the received frame as a byte stream, which saves slow RAM accesses and saves potential frames being lost as when a very small frame follows a large one the above layer could miss it, as the time to read and then store a large frame to somewhere else in RAM is longer than it takes to receive a very small frame. However, it does not check for frames of an invalid length (below 46 and above 1500 bytes for Ethernet), and will only operate at 10Mb/s.

Ethernet Sender

The Ethernet sender is connected to the PHY via a similar interface as the Ethernet receiver. From the PHY to the Ethernet sender there is the transmit clock (tx_clk) which is the clock used to synchronise the data between the two devices. The other connections in the MII are a four bit transmit data bus (txdata), a transmit enable line (tx_en) and a transmit error line (tx_err – not used) going to the PHY.

The Ethernet sender can transmit either ARP or IP frames, as specified by the frame type input. Once it has been instructed to send a frame, it will generate the preamble for the frame (7 bytes of 55h followed by 5Dh) and construct the header (based on the destination MAC input; the device MAC from the global constants package; and the frame type input to set the Ethernet type field). Once the header has been constructed, the Ethernet sender will read the data from RAM (the location depends on the type of frame) and send each byte read. Any frames smaller than 46 bytes are automatically padded to 46 bytes with whatever (pseudo random) data is currently contained in RAM.

All the data is sent to the PHY a nibble at a time using the txdata bus, least significant nibble first. As each byte is sent, it is also given to the CRC generator. Once all data has been sent, 4 consecutive bytes of zeros are sent to the CRC generator to correctly calculate the CRC for the frame. The 4 bytes of CRC are then sent at the end of the frame, LSB first.

Once the frame has been sent a signal is asserted to the layer who requested it to inform them that the frame was sent. The Ethernet sender is then not available for the time it would take to send 12 bytes over the network. tx_er is not used. The collision and carrier sense inputs are ignored from the PHY for both the Ethernet and Ethernet Sender (the PHY does not make proper use of these in full duplex mode).

ARP

The ARP layer handles ARP replies and ARP requests that are received, and also manages the ARP table and receives ARP table lookups. It also tells the ARP Sender to generate ARP replies when needed.

When an Ethernet frame is received with an Ethernet type field of 0806h, then the ARP receives an ARP frame from the Ethernet Receiver via a byte stream. As ARP messages are always a standard length, the

padding on the Ethernet frame and the CRC included in the byte stream can easily be ignored. If the ARP message if not meant for the IP specified in the global constants package, if it is not a valid ARP message for Ethernet and IPv4, or if it is anything other than an ARP request or reply, than the message is ignored. If it is an ARP request, then the sender is added to the ARP table and the ARP sender is informed to send out a reply, and the ARP table waits until notification is received that the reply has been sent. If it is an ARP reply, then the sender is added to the ARP table.

Because of the two-way handshake between the ARP and the ARP Sender, a possible lockup could have occurred. Consider the following situation:

- The ARP Sender sends out an ARP request (request A)
- An ARP request arrives from someone else (request B)
- ARP needs the ARP Sender to say that it has sent the reply to request B before it can receive more requests and replies
- The ARP Sender waits for ARP to receive the reply to request A and add it to the ARP table before it can send out more frames to the Ethernet Sender

Both ARP and the ARP Sender will now be waiting on each other in a lockup situation, as each requires a signal from the other to continue. This results in all incoming ARP messages being missed and no frames will be sent until the ARP Sender times out and resets. To avoid this, the signals to tell the ARP Sender to send an ARP reply are set and reset in a separate process, leaving the main ARP process free to receive further replies and set the ARP table until the ARP Sender then becomes free to send the reply. Should a second request arrive before the required ARP reply comes in, then the first request (request B) will now be ignored and the signals will be set to tell the ARP Sender to send a reply to the second request when it becomes free.

The ARP table is a two entry first in first out table. Lookups are performed on the table by setting the lookup IP input. If the IP is in the ARP table, than the ARP table entry valid output is asserted and the corresponding MAC address is outputted. When an entry is added to the table, it first checks to see if that entry already exists. If it does exist, then the entry is moved to the newest entry position in the table and updated. If the entry isn't previously in the table, then it is added into the newest entry position, the previous newest entry is moved into the older entry position and the older entry is discarded.

ARP Sender

The ARP sender lies transparently between the Ethernet Send and the Internet Send layers and constructs ARP requests and replies; handles ARP table lookups; and passes frames from the Internet Send layer to the Ethernet Send layer. The ARP Sender will always be unavailable while the Ethernet Sender is sending a frame.

The ARP Sender will construct ARP replies when requested to do so by the ARP layer. The target IP is obtained from the input from the ARP layer to the ARP Sender, and the target MAC address is obtained by performing a lookup in the ARP table by using the inputted IP. Once the ARP reply has been constructed and stored to location 0 in RAM then the ARP Sender will inform the Ethernet Send layer to send a frame. Once it has been sent, the ARP Sender will inform the ARP layer that the reply has been sent.

If the ARP Sender receives a frame from the Internet Sender, then it will perform a lookup on the destination IP (received from the Internet Send layer) in the ARP table. If it does not have a valid entry for the destination IP, then it will generate an ARP request for that IP into location 0 in RAM, and inform the Ethernet Sender to send the request. A timeout counter is then started, and it has 21.5 seconds in which to receive the ARP reply and put a valid entry for that IP into the ARP table or the counter will

overflow, and it will inform the above Internet send layer that the frame has been sent and return to an idle state, with the frame not sent. If the entry was originally valid or became valid after the ARP request was sent, then it will inform the Ethernet Sender to send the frame to the looked up MAC address. Once the frame has been sent, it informs the Internet Send layer that the frame has been sent.

Internet

The Internet layer receives datagrams from the Ethernet Receiver, strips the IP header, works out what is to be done with it, and if no more fragments are coming, then passes the datagram to the required protocol on the above transport layer.

Each datagram is received from the Ethernet Receiver via a byte stream. The amount of bytes expected is pulled from the IP datagram length field in the IP header, so the CRC that is sent at the end of the byte stream can easily be ignored. Different signals are asserted to the Internet layer when a new frame arrives, when each consecutive byte arrives, when the frame is finished and if the frame is valid (i.e. it has the correct CRC). This is also the same way ARP receives the byte stream from the Ethernet Receiver.

The length of the header of the datagram is taken from the first byte received of the IP header, and any options that may be included in the IP header are then ignored. The protocol, length, identification and source IP fields of the header are latched. If the destination IP field isn't the same as the IP in the global constants package or isn't a broadcast IP; the checksum on the header isn't correct; or the datagram isn't for IPv4 then the datagram is dropped.

When a new datagram is received, it is stored into the first available buffer (the two IP buffers at locations 10000h and 20000h in memory and are numbered 0 and 1 respectively). Reassembly of incoming packets is restricted to packets that arrive in network order. Out of order packets will be dropped as will duplicate packets for data that arrived previously. The IP layer always tries to write to buffer 0 first and will only write to buffer 1 if buffer 0 contains a partially received datagram. When a buffer is utilised the source IP, identification number and protocol from the IP header are stored and all three must match the incoming frame to continue using the buffer. Every time a new valid fragment is received, a timeout counter is restarted, and when this reaches its maximum value, the buffer is freed again for a new incoming packet. Any fragment size that is valid will be received and reassembled properly.

Internet Sender

The Internet Sender handles the creation of IP datagrams from TPDUs and the fragmentation of the TDPUs if they are too large. Once it forms the correct header and creates the datagram, it passes the nearly formed IP datagram down to the ARP Sender if it is available.

When the Internet Sender receives a new TPDU to send, it first creates the header, getting the destination IP, protocol and length fields from its inputs. The identification field is obtained from a counter that increments approximately every 20us, and the source IP is obtained from the global constants package. The options in the IP header are never used. When the header has been created, the header checksum is created and added to it. The data is then pulled from RAM (the location depends on the address offset input - this facilitates the use of different buffers for different protocols) and stores the data to location 800h. The ARP Sender is then informed that a new frame is ready to be sent, and the Internet Sender will then do nothing until it is informed from the ARP Sender that the frame has been sent.

Fragmentation occurs when the datagram that is required to be sent is above 1480 bytes. This is because 1500 bytes is the maximum frame size allowable over Ethernet, and that has to include a 20 byte IP header. When a packet needs to be fragmented, 1044 byte fragments (1024 bytes of data, 20 bytes of IP header) are sent until a final fragment with less than 1480 bytes of data can be sent. Fragments are sent in order as soon as they can be passed out of the Ethernet layer. The Internet Sender will not be able to send more data until it has finished transmitting the entire TPDU. As the Internet Sender only sends 1024 bytes of data for every fragment except the last, it will send more packets than usually required, but this greatly simplifies the hardware required to implement fragmentation.

ICMP

The ICMP protocol implemented only responds to an echo request. Any other ICMP message is ignored. ICMP can respond a valid ping containing between 0 and 2^{16} - 8 bytes of data.

If an IP datagram is received with the protocol field set to ICMP and the ICMP code and type fields in the ICMP header are for an echo request, then ICMP will strip the ICMP headers and recreate them into an echo reply format. It will get the ICMP data out of one of the Internet buffers depending on the value of the buffer select input. It will then store the created ICMP echo reply and optional data to location 40000h in RAM and inform the Internet Sender to create an IP datagram to send the echo reply.

UDP

This is a simple implementation of UDP that catches UDP messages on a specified port and stores them to location 30000h in RAM. Once an IP datagram arrives with the protocol field set to UDP, then the UDP process will remove the UDP headers and check the port, and then move the data to the new location in RAM. The UDP checksum is ignored (the pseudo header is not recreated).

5.0 Design Considerations

Design Limitations

There are several key limitations to the design of the IP stack, most of which are due to the limited amount of hardware, RAM and buffer space available on an FPGA (assuming the IP stack shouldn't take up over half the FPGA in size). Several of the main limitations are listed below.

- Only a small number of connections can be handled at once – if more than two other addresses try talking to the board simultaneously with all of them sending data to the board, a lot of the time will be spent sending off and processing ARP requests and replies as the ARP table only has two entries. The ARP table can easily grow in size if needed, but it needs an extra 80 flip-flops and more logic for table lookups for every entry added.
- The lack of buffer space also creates problems if multiple datagrams are being received and reassembled at once, or if the transport layer protocols are busy then datagrams will have to be dropped as no IP buffers will be free unless more IP buffers or transport layer buffers are allocated. Increasing the number buffers results in a large increase of memory usage and logic needed for controlling them.

- Due to the large timeouts on the IP buffers and the ARP Sender, any packet loss can result in several modules of the system being tied up for a long period of time. However, if these timeouts are shortened, then late arriving packets can cause other lockups or cause packets not to be sent at all.
- The last important limitation is only running the design at 10Mb/s – due to the complexity of logic required, and other problems such as generating the CRC in time and increased RAM accesses, running the systems at 100MHz would not be feasible unless a lot of optimising was done, or handshaking between the modules was increased allowing different modules to run at different clock rates. Also, running the connection at 10/100Mb/s would mean that two different versions of both the Ethernet and Ethernet Receiver would have to be used, as the timings to and from the PHY are different at the different transfer speeds.

Interfacing and Enhancing the Design

The IP stack is designed with the intention to not be just a stand-alone design, and as such would be useful for algorithms that require a fast transfer of data which isn't provided by the parallel port. However, the IP stack is a lot more demanding of hardware and RAM and is more complex than the parallel port interface. Preferably, it could simply receive and store data in RAM, and inform the application when new data has arrived. If the board could be configured to support partial reconfiguration of the Virtex FPGA, then perhaps reconfiguration over the network is a (non-trivial) application.

To enhance the design, many other protocols such as TCP, RARP and support for 802.2 frames can be added. However, the IP stack was written with the intention that a multiplexer (much like the RAM arbitrator) would be used between the transport and the Internet Send layers. This would keep the send signals of the transport layer protocol asserted until it had been informed from the Internet Send layer that the datagram had been sent. This allows transport layer protocols to easily be added without the Internet Send layer caring, facilitating the addition of higher level protocols. Unfortunately, this was not written in time.

Several other enhancements are possible – for example, when receiving and storing data, only 8 bits of the 16 bits of data of each RAM address are used, and although more logic would be required, a lot of RAM would be saved if the full 16 bits of data were used. Due to a lack of time, the designs have not been optimised (in many of the files, logic can be reduced by the addition of a few states, and the two huge case statements in arpsnd.vhd can be optimised into one by playing with the nextState signals). Another thing that can be added to aid the design is several timing constraints that might help the implementation of it.

Design Notes

Please note that this design has been fully tested and no problems have been witnessed with it running at 50MHz (even though Foundation reports a much lower clock speed should be the maximum for several reasons). However, synthesising and implementing with a full optimisation effort is recommended.

In the reprogramming of the CPLD to set the PHY to the correct modes, several of the PHY outputs to the CPLD were also mapped to LEDs, and as such these LEDs should not be driven by anything else unless the CPLD vhd file is changed, the svf file is rewritten and then the CPLD is reprogrammed with the new svf. The list of remapped LEDs are:
LEDS (Speed LED) – LED 5 of the right hex display – active low indicates 100Mb/s operation
LEDL (Link LED) – LED 6 of the right hex display – active low indicates a valid link
LEDR (Receive LED) – Bar(0) – active low indicates a frame is being received

LEDT (Transmit LED) – Bar(1) – active low indicates a frame is being transmitted
These are all feed through nets. Check cpldnet.vhd for more information on these connections.

Another two bar LEDs are used as indicators to show when the two IP buffers are busy. Bar(2) and bar(3) are mapped to IP buffer 0 and 1 respectively.

6.0 Memory Map

The IP stack only uses the left bank of memory to hold the data used by the stack. Below is a memory map of what portions of the RAM are used and what parts are free. The free parts of the memory are available to hold more buffers. Each address is only considered to be 8 bits (one byte) - the other 8 bits are ignored.

Memory Range (hex)	Memory Usage
00000 - 007FF	ARP Send buffer – holds ARP replies / requests to be sent (1500 bytes)
00800 – 01000	IP Sender buffer – holds IP frames to be sent (1500 bytes)
01001 - 0FFFF	Free
10000 - 1FFFF	IP Receive buffer 0 – holds reconstructed IP packets (64k bytes)
20000 - 2FFFF	IP Receive buffer 1 – holds reconstructed IP packets (64k bytes)
30000 - 3FFFF	UDP Receive buffer – holds received UDP TPDUs (64k bytes)
40000 - 4FFFF	ICMP Reply buffer – holds created echo replies (64k bytes)
50000 - 5FFFF	Free
60000 - 6FFFF	Free
70000 - 7FFFF	Free