COM-5502SOFT
IP/TCP SERVER/UDP/ARP/PING STACK for 10GbE
VHDL SOURCE CODE OVERVIEW

Overview

10Gigabit-speed IP protocols like TCP/IP and UDP/IP can demand a high level of computation on processors. The trend has been to move the implementation of these fast but highly repetitive tasks to a TCP offload engine (TOE) to free the application processor from frequent interrupts.

The COM-5502SOFT is a generic Internet protocol stack (including the **VHDL source code**) designed to support near 10Gbps throughputs on any low-cost FPGAs running at 156.25 MHz.

The modular architecture of VHDL components reflects the various internet protocols implemented within: TCP servers[1], UDP transmit, UDP receive, ARP, NDP, PING, IGMP (for multicast UDP), DHCP server and DHCP client. Ancillary components are also included for streaming. These components can be easily enabled or disabled as needed by the user's application.

The VHDL source code is fully portable to a variety of FPGA platforms.

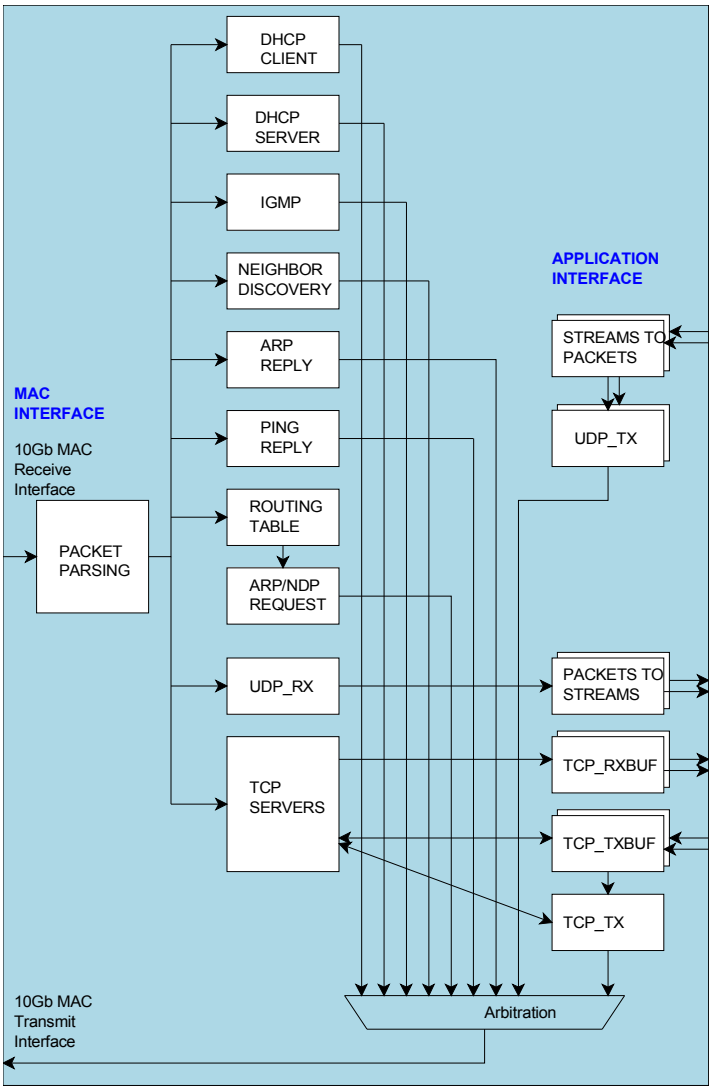
The maximum number of concurrent TCP connections can be adjusted prior to VHDL synthesis depending on the available FPGA resources.

The code is written specifically for IEEE 802.3 Ethernet packet encapsulation (RFC 894). It supports IPv4, IPv6, jumbo frames.

The code interfaces seamlessly with the COM-5501SOFT 10Gbps Ethernet MAC for the MAC / PHY layers implementation or the COM-5401SOFT 10/100/1000 Mbps Ethernet MAC. However, the MAC interface is generic and simple enough to interface with any Ethernet MAC component with minimum glue logic.

Wireshark Libpcap network capture files can be used as receiver input for simulation purposes.

Block Diagram



[1] See COM-5503SOFT for TCP clients.

MSS • 845 Quince Orchard Boulevard Ste N • Gaithersburg, Maryland 20878-1676 • U.S.A.
Telephone: (240) 631-1111 Facsimile: (240) 631-1676 www.ComBlock.com
 Issued 1/23/2021

Target Hardware

The code is written in generic VHDL so that it can be ported to a variety of FPGAs capable of running at 156.25 MHz or above.

Device Utilization Summary

(Excludes 10G Ethernet MAC and XAUI)
Device: Xilinx Artix-7

	UDP-only: 1 UDP rx, 1 UDP tx, 0 TCP server, ARP, Ping, routing table, IPv4 only, 8KB UDP tx buffer
Flip Flops	3198
LUTs	2740
36Kb block RAM	7.5
DSP48	0

	TCP IPv4 only: 0 UDP rx, 0 UDP tx, 1 TCP server, ARP, Ping, routing table, IPv4 only, MTU 1500, 32KB TCP buffers
Flip Flops	3515
LUTs	3855
36Kb block RAM	26
DSP48	0

	TCP IPv4 only: 0 UDP rx, 0 UDP tx, 2 TCP servers, ARP, Ping, routing table, IPv4 only, 32KB TCP buffers
Flip Flops	4255
LUTs	5174
36Kb block RAM	41
DSP48	0

	1 UDP rx, 1 UDP tx, 1 TCP server, ARP, Ping, NDP, routing table, IPv4. IPv6, 32KB TCP buffers
Flip Flops	7293
LUTs	9215
36Kb block RAM	32.5
DSP48	0

TCP Throughput

The TCP throughput is primarily a function of the tx/rx buffers sizes and of the two-way delay. For example, if the two way delay (NIC + FPGA) is 90us

Buffers sizes	TCP throughput
2kB	133 Mbits/s
8kB	673 Mbits/s
32kB	2.8 Gbits/s
64kB	5.56 Gbits/s
128kB	9.3 Gbits/s
256kB	9.3 Gbits/s

If the two-way delay is only 45us, the same TCP throughput can be achieved with half-sized buffers.

The buffer size is determined prior to synthesis by the generic parameters
TCP_TX/RX_WINDOW_SIZE

Throughput Performance Examples

UDP
IPv4 UDP throughput using 512-Byte data frames:
8.64 Gbits/s

IPv4 UDP throughput using 2048-Byte data frames:
9.62 Gbits/s

TCP
IPv4 TCP single server, uni-directional stream,
MTU = 1500 Bytes, equal length maximum size IP frames:
9.38 Gbits/s

TCP
IPv6 TCP single server, uni-directional stream,
MTU = 1500 Bytes, equal length maximum size IP frames:
9.23 Gbits/s

IPv4 TCP single server, bi-directional streams,
MTU = 1500 Bytes
8.82 Gbits/s in each direction

IPv4 TCP single server, uni-directional stream,
MTU = 8252 Bytes, equal length maximum size IP frames, buffer size = 32K Bytes:
9.88 Gbits/s

IPv6 TCP single server, uni-directional stream,
MTU = 8252 Bytes, equal length maximum size IP frames, buffer size = 32K Bytes:
9.86 Gbits/s

TCP Latency Performance Examples

The transmit and receive latency depend on the frame length. For a maximum frame length of 1460 bytes, FPGA 156.25 MHz processing clock:

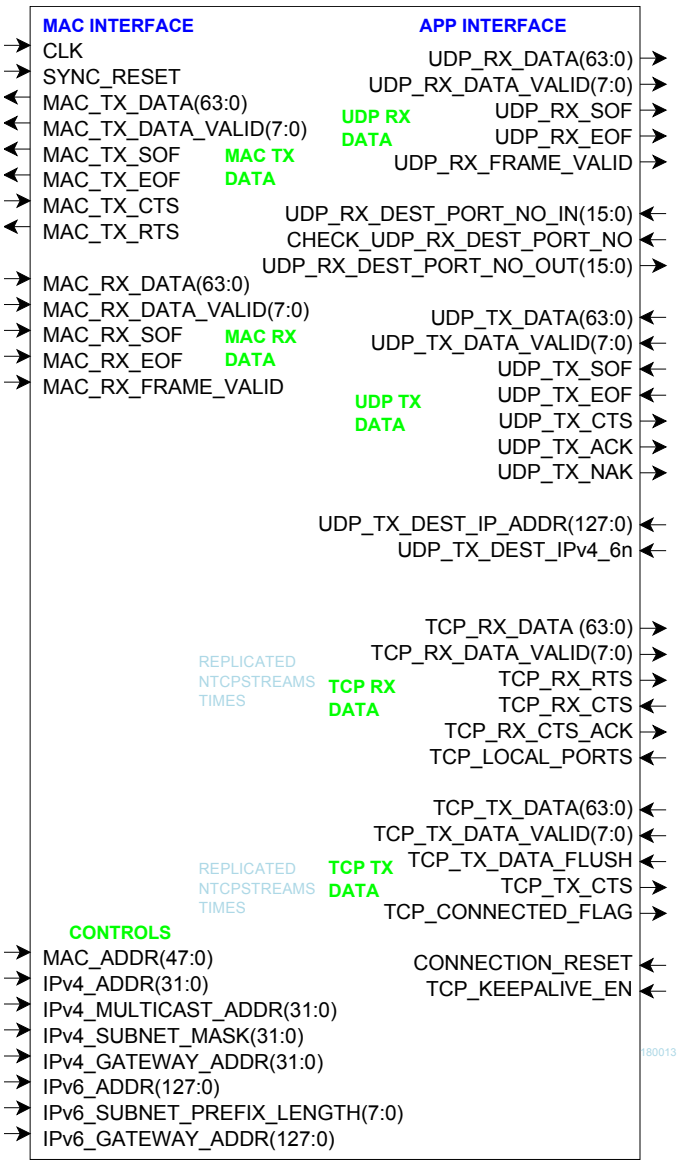
- Transmit latency (from the 1st byte of payload data input to the 1st byte of payload data output to the Ethernet MAC): 23.9µs
- Receive latency (from the 1st byte of Ethernet MAC input to the 1st byte of payload data output): 12.2µs

If latency is more important than throughput, the transmit segmentation threshold can be reduced to X payload bytes. In this more general case,

- Transmit latency (from the 1st byte of payload data input to the 1st byte of payload data output to the Ethernet MAC): 0.5 + 2X/125 µs

The receive latency (from the 1st byte of Ethernet MAC input to the 1st byte of payload data output): 0.5 + X/125 µs

Interfaces



Component Interface

This interface comprises three primary signal groups:

- MAC interface (direct connection to COM-5501SOFT Ethernet MAC or equivalent)
- TCP streams
- UDP frames or UDP streams to/from the user application.

All signals are clock synchronous. See the clock/timing section.

Configuration

The key configuration parameters are brought to the interface so that the user can change them dynamically at run-time. Other, more arcane, parameters are fixed at the time of VHDL synthesis.

Pre-synthesis configuration parameters

The following configuration parameters are set prior to synthesis in the *com5502pkg.vhd* package or at the top level component *com5502.vhd*.

Configuration parameters in ***com5502pkg.vhd***	*Description*
Maximum number of concurrent TCP streams	NTCPSTREAMS_MAX. This applies to all COM5502 components instantiated in a project. It primarily affects the data width of the TCP interface.
Configuration parameters in ***com5502.vhd***	*Description*
Number of concurrent TCP streams for a given COM5502 instance.	NTCPSTREAMS This applies to a given COM5502 instance. Each additional TCP stream requires additional resources (RAM block, logic).
UDP transmit instantiation	NUDPTX instantiated (1) / disabled (0) Note: a component handles multiple ports.
UDP receive instantiation	NUDPRX instantiated (1) / disabled (0) Note: a component handles multiple ports
Enable IPv6 protocols	IPv6_ENABLED '1' to allow IPv6 protocols in addition to the baseline IPv4. '0' to ignore IPv6 messages.
MTU size	MTU Maximum Transmission Unit: IP frame maximum byte size. -- Typically 1500 for standard frames, 9000 for jumbo frames. -- A frame will be deemed invalid if its payload size exceeds this MTU value. -- Should match the values in MAC layer) -- elastic buffers at the user interface should be sized to contain at least 4 IP frames payload. See ADDR_WIDTH generic parameter.
TCP buffers sizes	TCP_TX_WINDOW_SIZE TCP_RX_WINDOW_SIZE Window size is expressed as 2**n Bytes. Thus a value of 15 indicates a window size of 32KB. This generic parameter determines how much memory is allocated to buffer tcp streams. It applies equally to all concurrent streams (no individualization). Purpose: tradeoff memory utilization vs throughput. Memory size ranges from 2KB (multiple streams/lower individual throughput) to 1MB (single stream/maximum throughput) The window scale option is recommended on the client side when this server's buffers are larger than 64KB.
DHCP server instantiation	DHCP_SERVER_EN instantiated (1) / disabled (0) The DHCP server assigns dynamic IPv4 addresses to DHCP clients from a pool of local IPv4 addresses.
DHCP client instantiation	DHCP_CLIENT_EN '1' to instantiate a DHCP client within. DHCP is a protocol used to dynamically assign IP addresses at power up from remote DHCP servers, like a gateway. '0' when a fixed (static) IP address is defined by the user. One can instantiate both DHCP server and DHCP client at the same time, but not enable them simultaneously
IGMP instantiation	IGMP_EN instantiated (1) / disabled (0) Enable when using UDP multicast addresses
Inactive input stream timeout	TX_IDLE_TIMEOUT When segmenting a TCP transmit stream, a packet will be sent out with pending data if no new data was received within the specified timeout. Expressed as integer multiple of 4µs.
TCP keep-alive period	TCP_KEEPALIVE_ PERIOD period in seconds for sending no data keepalive frames. "Typically TCP Keepalives are sent every 45 or 60 seconds on an idle TCP connection, and the connection is dropped after 3 sequential ACKs are missed"
CLK frequency	CLK_FREQUENCY CLK frequency in MHz. Needed to compute actual delays.

Configuration parameters in ***ping_10g.vhd***	*Description*
Maximum ping size	MAX_PING_SIZE maximum IP/ICMP size (excluding Ethernet/MAC, but including the IP/ICMP header) in 64-bit words. Larger echo requests will be ignored. The ping buffer contains up to 18Kbits total (for a queued IP/ICMP response waiting for the tx path to become available)
Configuration parameters in ***arp_cache2.vhd***	*Description*
Routing table refresh period	REFRESH_PERIOD(19:0) Refresh period for this routing table. Expressed as an integer multiple of 100ms. Default value is 3000 (5 minutes).
Configuration parameters in ***tcp_txbuf_10G.vhd*** ***tcp_rxbufndemux2_10G.vhd***	*Description*
Elastic buffer size	ADDR_WIDTH Specifies the elastic buffer size for each stream. Data width is fixed at 8 bytes. Thus ADDR_WIDTH = 11 indicates a buffer size of 128 Kbits. Maximum value = 12 (256Kbits) Note that the buffer size must be large enough to store two complete IP frames payloads (see MTU above).
Configuration parameters in ***udp_tx_10g.vhd***	*Description*
UDP checksum enable (IPv4)	UDP_CKSUM_ENABLED Enable (1) / Disable (0) UDP checksum computation for IPv4. Objective is to save FPGA resources.

Configuration parameters in ***stream_2_packets_10g.vhd***	*Description*
Maximum packet size when segmenting a stream to packets	MAX_PACKET_SIZE When segmenting a transmit stream, a packet will be sent out as soon as MAX_PACKET_SIZE bytes are collected. The recommended size is 512 bytes for a low overhead.
Retransmission timer	TX_RETRY_TIMEOUT A re-transmission attempt will be made periodically until routing information is available and the transmit path to the MAC is available. The retry period is expressed as an integer multiple of 4µs.

Run-time configuration parameters

The user can set and modify the following controls at run-time. All controls are synchronous with the user-supplied global CLK.

Run-time configuration	*Description*
MAC address MAC_ADDR(47:0)	This network node 48-bit MAC address. The user is responsible for selecting a unique 'hardware' address for each instantiation. Natural bit order: enter x0123456789ab for the MAC address 01:23:45:67:89:ab It is essential that this input matches the MAC address used by the MAC/PHY.
Dynamic vs static IP DYNAMIC_IP	'1' for dynamic addressing '0' for static IP address. The device IP address can be assigned dynamically by an external DHCP server, or defined as static address by the user. Dynamic addressing requires instantiating a DHCP client: set the generic parameter DHCP_CLIENT_EN = '1'.
IPv4 address REQUESTED_IPv4_ADDR (31:0)	Static address when DYNAMIC_IP = '0' Last dynamically assigned address when DYNAMIC_IP = '1'. Address 0.0.0.0 can also be used in conjunction with dynamic addressing if the user does not 'remember' the last dynamic IP address. 4 bytes for IPv4. Byte order: (MSB)192.68.1.30(LSB)
IPv4 Subnet Mask IPv4_SUBNET_MASK(31:0)	Subnet mask to assess whether an IP address is local (LAN) or remote (WAN) Byte order: (MSB)255.255.255.0(LSB) Ignored when the DHCP client feature is enabled, as the DHCP server provides the subnet mask.
IPv4 Gateway IP address IPv4_GATEWAY _ADDR(31:0)	One gateway through which packets with a WAN destination are directed. Byte order:

	(MSB)192.68.1.1(LSB) Ignored when the DHCP client feature is enabled, as the DHCP server provides the gateway information.
IPv4 Multicast address IPv4_MULTICAST _ADDR(31:0)	to receive UDP multicast messages. One multicast address only. 0.0.0.0 to signify that IP multicasting is not supported here. IGMP must be instantiated to declare that this node belongs to a multicast group.
IPv6 address IPv6_ADDR(127:0)	Local IP address. 16 bytes for IPv6. Byte order example: (MSB)FE80:: 0102:0304:0506:0708(LSB)
IPv6 Subnet Prefix Length IPv6_SUBNET_PREFIX _LENGTH (7:0)	Valid range 64-128
IPv6 Gateway IP address IPv6_GATEWAY _ADDR(127:0)	One gateway through which packets with a WAN destination are directed. Must be on the same local network as this device.
TCP_KEEPALIVE_EN	Keep-alive is a mechanism to detect when a TCP connection is interrupted. Keep-alive messages are sent periodically. Three missed keep-alive messages cause a TCP reset. Enable (1)/ Disable (0) for each stream.

Throughout this document **CTS** and **RTS** refer to flow control signals "**C**lear **T**o **S**end" and "**R**eady **T**o **S**end" respectively. CTS is generated by the data sink to indicate it can process and/or store incoming data. RTS is generated by the data source to indicate that data bits are available, should the data sink raise its CTS flag.

UDP-Application Interface

UDP transmit interface	
UDP transmit word UDP_TX_DATA (63:0)	Input: send 0 to 8 bytes. Byte order: MSB first (easier to read contents during simulation). Unused bytes are expected to be zeroed.
UDP data valid UDP_TX_DATA_VALID (7:0)	Input. Indicates the meaningful bytes in UDP_TX_DATA. 0xFF for 8 bytes, 0x80 for one byte, 0xC0 for two bytes, etc.
UDP_TX_SOF UDP_TX_EOF	Inputs. 1 CLK wide markers to delineate the frame boundaries. SOF = Start Of Frame EOF = End Of Frame Must be aligned with UDP_TX_DATA_VALID
Flow control UDP_CTS	Output '1' = Clear To Send '0' = input buffer is nearly full. Do not send more data. The user must check the Clear-To-Send flag before sending additional data. The timing is not precise (it is safe to send data for a few clocks after CTS goes low), thanks to an input elastic buffer.
Transmission acknowledgements UDP_TX_ACK UDP_TX_NAK	Outputs UDP_TX_ACK: 1 CLK-wide pulse indicating that the previous UDP frame was successfully sent. UDP_TX_ACK 1 CLK-wide pulse indicating that the previous UDP frame could not be sent (destination not present for example). USAGE: wait until the previous UDP tx frame UDP_TX_ACK or UDP_TX_NAK to send the follow-on UDP tx frame

UDP receive interface	
UDP rx word UDP_RX_DATA(63:0)	Output. Receive 0 to 8 bytes. Byte order: MSB first (easier to read contents during simulation) All words in a frame contain 8 bytes, except the last word which may contain fewer.
UDP rx data valid UDP_RX_DATA_VALID (7:0)	Output. Indicates the meaningful bytes in UDP_RX_DATA. 0xFF for 8 bytes, 0x80 for one byte, 0xC0 for two bytes, etc.
Start Of Frame / End Of Frame UDP_RX_SOF UDP_RX_EOF	Outputs. 1 CLK wide markers to delineate the frame boundaries. SOF = Start Of Frame EOF = End Of Frame. Aligned with UDP_RX_DATA_VALID
UDP_RX_FRAME_VALID	Output. The frame validity UDP_RX_FRAME_VALID is displayed at the end of frame when UDP_RX_EOF = '1' The user is responsible for discarding bad frames. Always check UDP_RX_FRAME_VALID at the end of packet UDP_RX_EOF = '1') to confirm that the UDP packet is valid. External buffer may have to backtrack to the the last valid pointer to discard an invalid UDP packet. Reason: we only knows about bad UDP packets at the end.
CHECK_UDP_RX_DEST_PORT_NO	Input. '1' when the COM5502 component filters out UDP frames sent to a destination port other than the user-specified UDP_RX_DEST_PORT_NO_IN '0' when UDP frames with any destination port (but with the right IP address) are passed to the user.
UDP_RX_DEST_PORT_NO_IN	Input. User-specified UDP rx destination port (enabled when CHECK_UDP_RX_DEST_PORT_NO = '1'

TCP-Application Interface

Prior to synthesis, one must configure the following constants:

- The maximum number of TCP servers NTCPSTREAMS_MAX in com5502.pkg. This limit applies to all instantiated COM5502 components in a project:
- The number of TCP servers NTCPSTREAMS for a given COM5502 instance, as declared in the generic section.

TCP receive interface (for TCP connection # I)	
TCP local port TCP_LOCAL_PORTS(I)(15:0)	Input. TCP_SERVER port configuration. Each one of the NTCPSTREAMS streams handled by this component must be configured with a distinct port number. This value is used as destination port number to filter incoming packets, and as source port number in outgoing packets.
TCP rx word TCP_RX_DATA(I)(63:0)	Output. Receive 0 to 8 bytes. Byte order: MSB first (easier to read contents during simulation)
TCP rx data valid TCP_RX_DATA_VALID (I) (7:0)	Output. Indicates the meaningful Bytes in TCP_RX_DATA. 0xFF for 8 Bytes, 0x80 for one Byte, 0xC0 for two Bytes, etc. Partially filled words can remain at the interface for several clock periods until the remaining word bytes are received. However, when the received word is full (0xFF), it stays at the interface for one and only one clock.
Ready To Send TCP_RX_RTS(I)	Output. Usage: TCP_RX_RTS goes high when at least one byte is in the output queue (i.e. not yet visible at the output TCP_RX_DATA). The application should then raise TCP_RX_CTS for one clock to fetch the next word 2 CLKs later.

	Note that the next word may be partial (<8 bytes) or full.
Flow control: Clear To Send TCP_RX_CTS(I)	Input. Flow control signal. '1' to indicate that the user is ready to accept the next TCP rx word. This signal can be pulsed or continuous. The latency between TCP_RX_CTS and TCP_RX_DATA_VALID is two clocks. This Clear-To-Send signal can remain '1' if the application is capable of handling the high throughput. The code will ignore this TCP_RX_CTS signal when no new data is being received.

The TCP interface is replicated NTCPSTREAMS times, depending on the number of connections implemented.

See the TCP receive interface timing section for details.

DHCP Server Application Interface

When instantiated, the DHCP server assigns IPv4 addresses dynamically to DHCP clients requesting an IPv4 address. The addresses are taken from a pool of DHCP_SERVER_NIPs consecutive addresses starting at address with least significant byte DHCP_SERVER_IP_MIN_LSB. The address is leased for DHCP_SERVER_LEASE_TIME seconds. The DHCP client is expected to renew the lease before it expires. Together with the leased IPv4 address, the DHCP server also provides the client with IP addresses for the WAN router (DHCP_ROUTER) , its subnet mask and a DNS (DHCP_SERVER_DNS).

DHCP server configuration	
DHCP_SERVER_EN2	Input. Enable the DHCP server at run-time. It only applies if the DHCP server is instantiated. Mutually exclusive with DYNAMIC_IP (chose DHCP client OR server, but not both)
DHCP_SERVER_IP_MIN_LSB	LSB of first address in the DHCP server pool of IPv4 addresses.
DHCP_SERVER_NIPs	Number of IPv4 addresses in the DHCP server pool. Maximum of 128 entries.
	For example, if IPv4_ADDR = 172.16.1.3, IP_MIN = 10, NIPs = 10, the DHCP server will assign and keep track of IP addresses in the range 172.16.1.10 and 172.16.1.19 (inclusive).

Limitations

This software does not support the following:

- IEEE 802.3/802.2 encapsulation, RFC 1042, only the most common Ethernet encapsulation.

Only one gateway is supported at any given time.

Software Licensing

The COM-5502SOFT is supplied under the following key licensing terms:

1. A nonexclusive, nontransferable corporate/organization license to use the VHDL source code internally, and
2. An unlimited, royalty-free, nonexclusive transferable license to make and use products incorporating the licensed materials, solely in bitstream format, on a worldwide basis.

The complete VHDL/IP Software License Agreement can be downloaded from http://www.comblock.com/download/softwarelicense.pdf

Configuration Management

The current software revision is 0.

Directory	Contents
/project_1	Xilinx Vivado 2017.4 project
/doc	Specifications, user manual, implementation documents
/src	.vhd source code, .ucf constraint files, .pkg packages. One component per file.
/sim	Testbenches, Wireshark capture files as simulation stimulus
/bin	.bit configuration file for use with COM-1800/COM-5104 hardware modules.
/use_example	Source code for use with COM-1800/COM-5104 hardware modules Test components (stream to packets segmentation, etc) are in directory \use_example\src

Key project file:

Xilinx ISE project file: com-5502_ISE14.xise

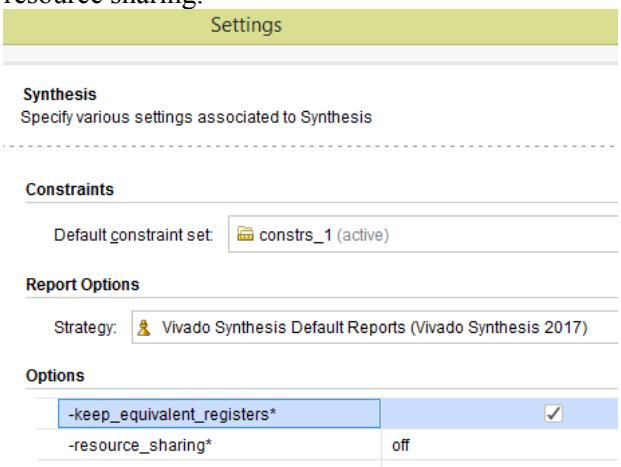
VHDL development environment

The VHDL software was developed using the following development environment:

(a) Xilinx Vivado 2017.4 as synthesis tool

(b) Xilinx Vivado 2017.4 as VHDL simulation tool

For best FPGA place and route timing, the recommended Xilinx Vivado synthesis settings are the default + keep equivalent registers + no resource sharing.



Ready-to-use Hardware

Use examples are available to run on the following Comblock hardware modules:

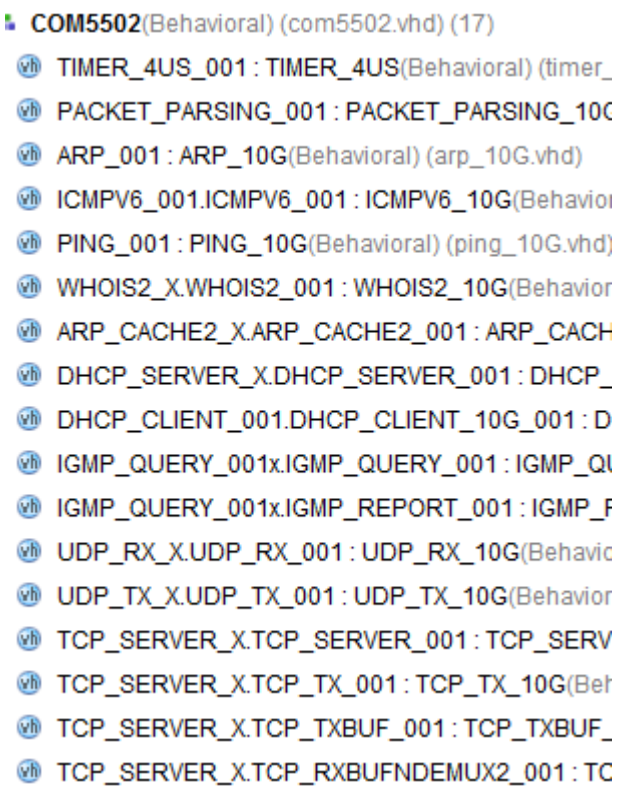
- COM-1800 FPGA (XC7A100T) + ARM + DDR3 SODIMM socket + GbE LAN development platform
  http://www.comblock.com/com1800.html
- ComBlock COM-5104 10G Ethernet network interface (SFP+ connector to 4-lane XAUI to FPGA)
  http://www.comblock.com/com5104.html

All hardware schematics are available online at
comblock.com/download/com_1800schematics.pdf
comblock.com/download/com_5104schematics.pdf

Acronyms

Directory	Contents
ARP	Address Resolution Protocol (only for IPv4)
CTS	Clear To Send (flow control signal)
DNS	Domain Name Server
EOF	End Of Frame
LAN	Local Area Network
LSB	Least Significant Byte in a word
MSB	Most Significant Byte in a word
MTU	Maximum Transmission Unit (frame length)
NDP	Neighbor Discovery Protocol
RX	Receive
RTS	Ready To Send (flow control signal)
SOF	Start Of Frame
TCP	Transmission Control Protocol
TX	Transmit
UDP	User Datagram Protocol
WAN	Wide-Area Network

Top-Level VHDL hierarchy



The code is stored with one, and only one, component per file.

The root entity (highlighted above) is *COM5502.vhd*. It contains instantiations of the IP protocols and a transmit arbitration mechanism to select the next packet to send to the MAC/PHY.

The root also includes the following components:

- The *PACKET_PARSING_10G.vhd* component parses the received packets from the MAC and efficiently extracts key information relevant for multiple protocols. Parsing is done on the fly without storing data. Instantiated once.
- The *ARP_10G.vhd* component detects ARP requests and assembles an ARP response Ethernet packet for transmission to the MAC. Instantiated once. ARP only applies to IPv4. For IPv6, use neighbour discovery protocol instead.
- The *DHCP_SERVER_10G.vhd* component manages a pool of IPv4 addresses. It assigns them dynamically to DHCP clients upon request. The server also supplies the subnet mask, the gateway address and a DNS address.
- The *DHCP_CLIENT_10G.vhd* component requests an IPv4 address from a remote DHCP server when dynamic addressing is selected. The server also supplies the subnet mask, the gateway address and a DNS address.
- The IGMP_REPORT_10G.vhd component sends an IGMP membership report to declare this network node as belonging to a multicast group. The *IGMP_QUERY.vhd* component responds to membership queries.
- The *ICMPV6_10G.vhd* component detects incoming IP/ICMPv6 neighbor solicitations on the fly and responds with the local MAC address information.
- The *PING_10G.vhd* component detects ICMP echo (ping) requests and assembles a ping echo Ethernet packet for transmission to the MAC. Instantiated once. Ping works for both IPv4 and IPv6.
- The *WHOIS2_10G.vhd* component generates an ARP request broadcast packet (IPv4) or a Neighbor solicitation message (IPv6) requesting that the target identified by its IP address responds with its MAC address.
- The *ARP_CACHE2_10G.vhd* component is a shared routing table that stores up to 128 IP addresses with their associated 48-bit MAC addresses and a ‘freshness’ timestamp. This component determines whether the destination IP address is local or not. In the latter case, the MAC address of the gateway is returned. Only records regarding local addresses are stored (i.e. not WAN addresses since these often point to the router MAC address anyway). An arbitration circuit is used to arbitrate the routing request from multiple transmit instances. Instantiated once.
- The flexible *UDP_TX_10G.vhd* component encapsulates a data packet into a UDP frame addressed from any port to any port/IP destination. Supports both IPv4 and IPv6. Generally instantiated once, irrespective of the number of source or destination UDP ports. However, multiple

instantiations can easily be implemented by modifying the COM5502.vhd top level code (search for the TX_MUX_00x and RT_MUX_00x processes). Multiple instances are useful when multiple UDP sources need transmit arbitration to prevent collisions.

- The *UDP_RX_10G.vhd* component validates received UDP frames and extracts the data packet within. As the validation is performed on the fly (no storage) while received data is passing through, the validity confirmation is made available at the end of the packet. The calling application is therefore responsible for discarding packets marked as 'invalid' at the end. See *PACKETS_2_STREAM_10G.vhd* for assistance in discarding invalid packets. Instantiated once, irrespective of the number of UDP ports being listened to.
- The *TCP_SERVER_10G.vhd* component is the heart of the TCP protocol. It is written parametrically so as to support NTCPSTREAMS concurrent TCP connections. It essentially handles the TCP state machine of a TCP server: initially listening for connection requests from remote TCP clients, establishing and tearing down the connections and managing flow control and byte ordering while the connections are established. Since this is a server, it does not know a priori whether the protocol is IPv4 or IPv6 (it depends on the client), so each server is given two IP addresses, one for each IP version.
- The *TCP_TX_10G.vhd* component formats TCP tx frames, including all layers: TCP, IP, MAC/Ethernet. It is common to all concurrent streams and is thus instantiated once.
- The *TCP_TXBUF_10G.vhd* component stores TCP tx payload data in individual elastic buffers, one for each transmit stream. The buffer size is configurable prior to synthesis through the ADDR_WIDTH generic parameter.
- The *TCP_RXBUFNDEMUX_10G*.vhd component demultiplexes several TCP rx streams. This component has two objectives: (1) tentatively hold a received TCP frame on the fly until its validity is confirmed at the end of frame. Discard if invalid or further process if valid. (2) demultiplex multiple TCP streams, based on the destination port number.

Additional components are also provided for use during system integration or tests.

- *STREAM_2_PACKETS_10G.vhd* segments a continuous data stream into packets. The transmission is triggered by either the maximum packet size or a timeout waiting for fresh stream bytes.
- *PACKETS_2_STREAM_10G.vhd* reassembles a data stream from received valid packets while discarding invalid packets. The packet's validity is assessed at the end of packet. It is designed to connect seamlessly with the *TCP_RX.vhd* component.
- *LFSR11P64.vhd* is a pseudo-random sequence generator used for test purposes. It generates a PRBS11 test sequence commonly used for bit error rate testing at the receiving end of a transmission channel. The 64-bit wide output allows for high-speed operation (10 Gbits/s).
- *BER64.vhd* is a bit error rate tester expecting to receive a PRBS11 test sequence. It synchronizes with the received bit stream and count errors over a user-defined window. The 64-bit wide output allows for high-speed operation (10 Gbits/s).

VHDL simulation

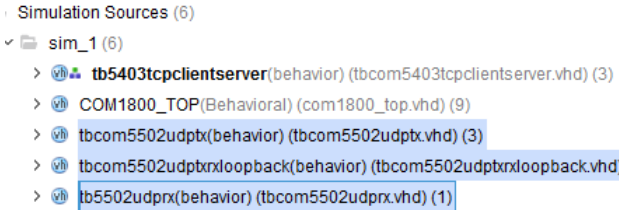
Test benches are provided for HDL simulation of UDP transmit, UDP receive.

Several test benches use Wireshark Libpcap network capture files as stimulus. See Libcap File Player

For TCP server simulation, a TCP client simulator is needed (not supplied), because of the interactive nature of the TCP protocol. The COM-5502SOFT + COM-5503SOFT bundle allows the comprehensive TCP server - TCP client simulation.

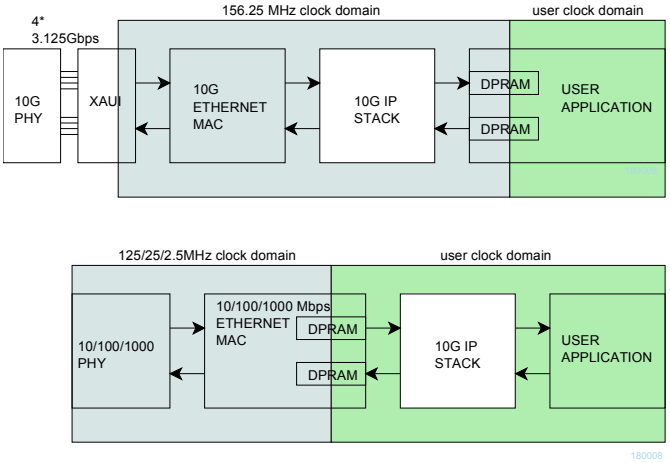
The testbenches (tb*.vhd) are located in the /sim directory

Quick start:
In the Xilinx Vivado, open a .xpr project. The available testbenches are displayed as illustrated below. Start the simulator. In the simulator, open the stored .wcfg configuration file which bears the same name as the testbench.



Clock / Timing

The COM-5502SOFT can connect to 10G Ethernet MAC as well as to lower-speed 10/100/1000Mbps Ethernet MAC without any code change. However, the clock domains are different, as illustrated by the two use-cases below.



At 10G speed, the COM-5502SOFT uses the same 156.25 MHz clock as the 10G Ethernet MAC. If the user application uses a different clock, dual-port RAMs must be used to cross the clock domain

When the COM-5502SOFT is connected with the lower-speed 10/100/1000 Mbps tri-mode Ethernet MAC, dual-port RAMs within the Ethernet MAC are used to cross the clock domains. The COM-5502SOFT can then use the same clock as the user application.

The COM-5502SOFT code is written to run at 156.25 MHz on a Xilinx Artix7 -1 speed grade with 2 concurrent TCP streams instantiated.

UDP Receive Latency

In order to minimize the latency, UDP payload bytes are forwarded directly to the user application interface with only a partial validation. This allows the application to start processing the UDP payload data without delay since frame errors are very rare. However, the complete validation information is only available at the end of the UDP frame (UDP_RX_EOF). The user application is responsible to discarding invalid frames based upon the UDP_RX_FRAME_VALID confirmation.

Validation checks performed prior to the first UDP payload word (UDP frame is not forwarded to the user application if any of these checks fail):

- IP datagram
- Destination IP address
- IPv4 header checksum
- UDP protocol
- Destination UDP port (when enabled)

Validation checks performed at the end of UDP frame (user is responsible for discarding the frame if any of these checks fail):

- UDP checksum

TCP Receive Latency

In the baseline code, the TCP receive payload data goes through the *TCP_RXBUFNDEMUX2_10G*.vhd component which conveniently discards bad frames and packs payload data into neat 64-bit words.

The 'price to pay' for this convenience is a delay which can be significant as the user application is notified of valid payload bytes at the end of an IP frame.

Alternative lower-latency method:

When low-latency is a priority, the *TCP_RXBUFNDEMUX2_10G*.vhd component may be bypassed (requires minor code editing). In this case, the user application is responsible to discarding invalid frames based upon the RX_FRAME_VALID confirmation at the end of frame RX_EOF.

```
--// RX TCP PAYLOAD -> EXTERNAL RX BUFFER
-- Latency: 1 CLK after the received IP payload frame.
RX_DATA: out std_logic_vector(63 downto 0);
    -- TCP payload data field. Each byte validity is in RX_DATA_VALID(I)
    -- IMPORTANT: always left aligned (MSB first): RX_DATA_VALID is x80,xc0,xe0,xf0,....x0
RX_DATA_VALID: out std_logic_vector(7 downto 0);
    -- delineates the TCP payload data field
RX_SOF: out std_logic;
    -- 1 CLK pulse indicating that RX_DATA is the first byte in the TCP data field.
RX_TCP_STREAM_SEL_OUT: out std_logic_vector(NTCPSTREAMS-1 downto 0);
    -- output port based on the destination TCP port
RX_EOF: out std_logic;
    -- 1 CLK pulse indicating that RX_DATA is the last byte in the TCP data field.
    -- ALWAYS CHECK RX_FRAME_VALID at the end of packet (RX_EOF = '1') to confirm
    -- that the TCP packet is valid. External buffer may have to backtrack to the the last
    -- valid pointer to discard an invalid TCP packet.
    -- Reason: we only knows about bad TCP packets at the end.
RX_FRAME_VALID: out std_logic;
    -- verify the entire frame validity at the end of frame (RX_EOF = '1')
RX_FREE_SPACE: in SLV16xNTCPSTREAMStype;
    -- External buffer available space, expressed in bytes.
    -- Information is updated upon receiving the EOF of a valid rx frame.
    -- The real-time available space is always larger
```

Validation checks performed prior to the first TCP payload word in an IP frame (TCP payload data is not forwarded out to the user application if any of these checks fail):

- IP datagram
- Destination IP address
- IPv4 header checksum
- TCP connection
- TCP protocol
- Destination TCP port
- No gap in received sequence
- Non-zero data length
- Originator is identified (no spoofing)

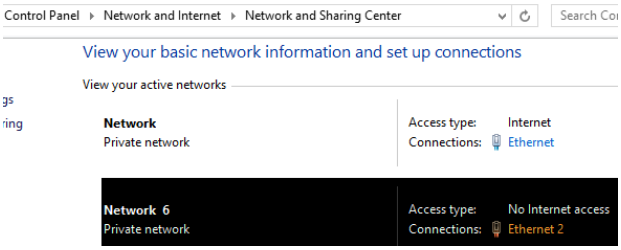
Validation checks performed at the end of TCP frame (user is responsible for discarding the frame if any of these checks fail):

- TCP checksum

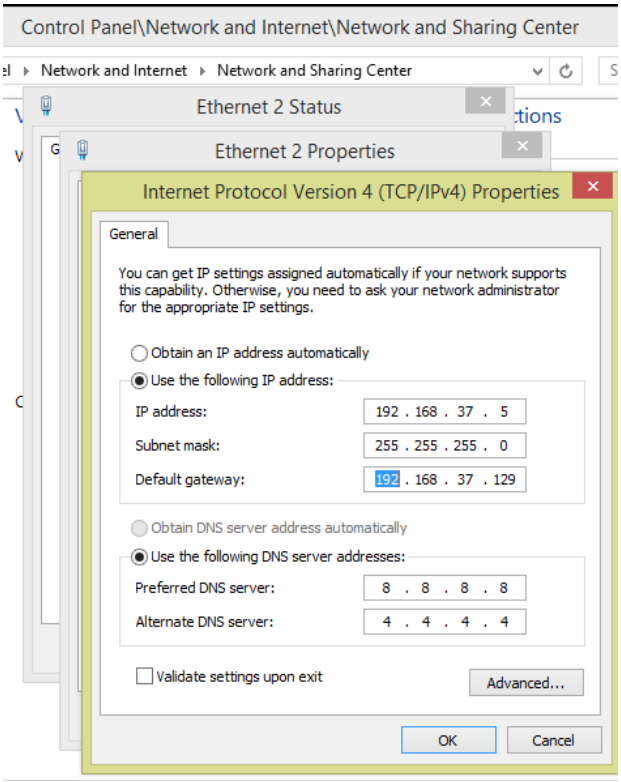
Troubleshooting

1. PC cannot ping or receive UDP frames or establish a TCP connection.

One likely cause may be Windows security. Declaring the network adapter as a "Private Network" makes it easier to access the FPGA board from the PC.



One method is to define the default gateway field in the network adapter IPv4 configuration as the FPGA board IPv4 address .



TCP receive interface timing (simple)

The receive interface for TCP and UDP are somewhat different. Each TCP stream stores the receive data in an independent output buffer.

Read each full 8-Byte word APP_RX_DATA(63:0) when the word is full, that is when APP_RX_DATA_VALID = xFF for one CLK.

Regulate the receive throughput using the "Clear-To-Send' SIGNAL APP_RX_CTS: '1' to enable, '0' to stop.

Partially-filled output words are available at the interface, as indicated by APP_RX_DATA_VALID = x80 (1 Byte), xC0 (2 Bytes), ... xFE (7 Bytes). The partial output words can stay at the interface for extented periods, that is until the 8-Byte output word is completely filled.

To summarize, the simple rules for reading TCP data are as follows:

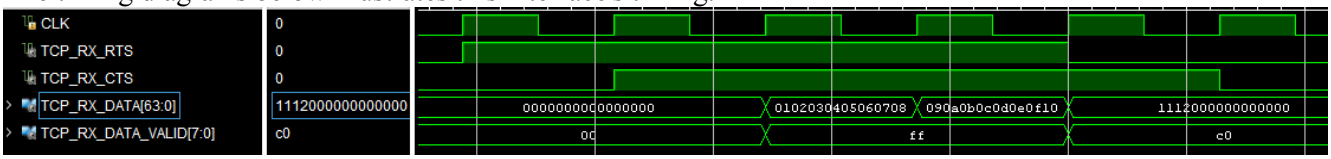
```
if(TCP_RX_DATA_VALID = x"FF") then
        -- READ complete 8-byte word
        OUTPUT <= TCP_RX_DATA(63:0);
elsif(TCP_RX_DATA_VALID /= x"00") then
        -- LOOK/PEEK at a PARTIAL word TCP_RX_DATA, filled MSB first
        -- The complete 8-byte word will be available later at the next occurrence
        -- of (TCP_RX_DATA_VALID = x"FF")
        PEEK <= TCP_RX_DATA(63:?)
end if;

-- TCP_RX_CTS can stay high all the time, unless the data flow is too high
-- TCP_RX_RTS and TCP_RX_CTS_ACK are generally for monitoring purposes
```

TCP receive interface timing (detailed)

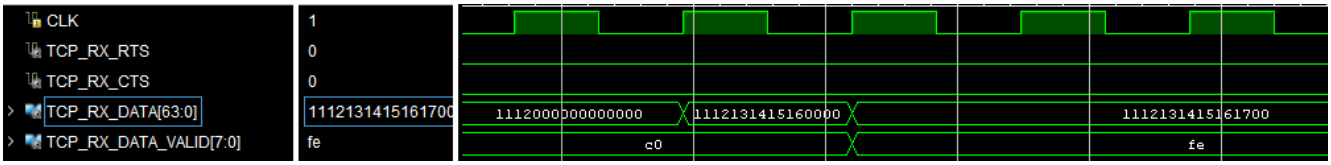
The application controls the output rate through TCP_RX_CTS (Clear-To-Send) pulses. One TCP_RX_CTS pulse will fetch the next word as long as it contains at least one Byte. The TCP_RX_RTS goes high when at least one Byte is unread in the output buffer. Thus the application should do the following:

1. Check TCP_RX_RTS until it indicates data hidden in the output buffer
2. Send a APP_RX_CTS pulse (1 clock wide)
3. Get the data Byte(s) in APP_RX_DATA(63:0). The number of Bytes available is shown in APP_RX_DATA_VALID(7:0)
4. Wait until the output word is full (TCP_RX_DATA_VALID(7:0) = xFF) to get the full word contents. Filling the output word with incoming Bytes is automatic. Note that this may take several clocks, depending on the rate at which data is sent over the LAN.
5. Repeat steps 1-5

The timing diagrams below illustrates this interface's timing.



The transmitted sequence is 010203..etc.
The first two output words contain 8 valid data Bytes. They are available one clock after requested the application generates the TCP_RX_CTS pulse.
The third output word contains only two Bytes (the last two Bytes received at this time). The next TCP_RX_CTS is ignored as there is no other data waiting in the output buffer.

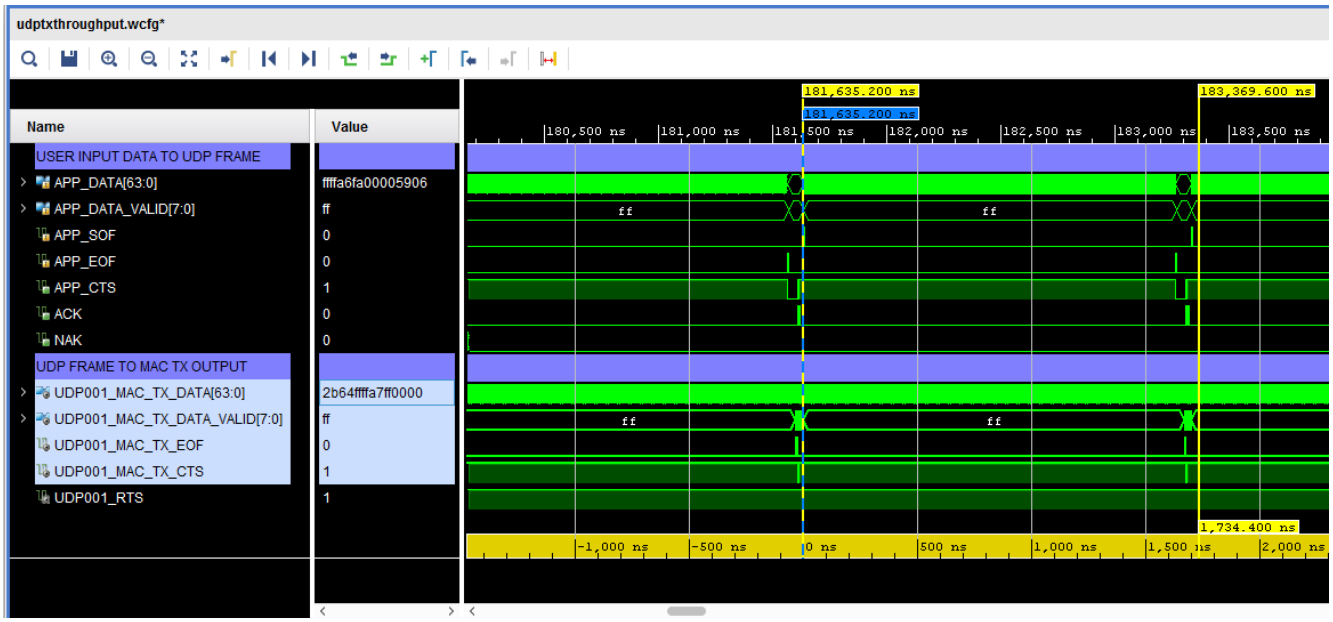


The third output word is being filled as data arrives. Note that the number of valid Bytes is updated with some delay as the component must confirm each frame's validity at the end of each Ethernet frame.

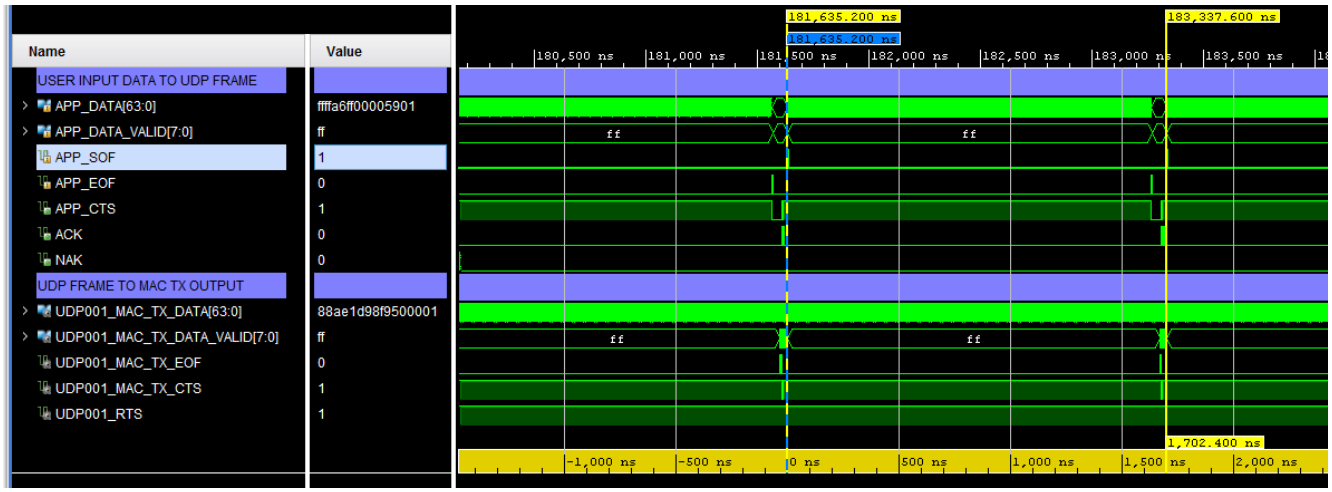
UDP Transmitter Latency

Before sending a UDP frame, the data must be stored in a buffer while the checksum is being computed. Therefore, the transmit latency depends to a large part on the size of the UDP frame, since transmission can only start after the last word is received from the user.

For example: in the case of a 2048-byte frame, the transmit latency is 1.734us



The 10Gbits/s capacity is nearly fully utilized in the case of 2048-byte UDP frames: 2048 bytes/1.702us = 9.62 Gbits/s



Libcap File Player

Real network packets captured by the popular Wireshark LAN analyzer can be used as realistic stimulus for the COM-5502 software. The *tbcom5502.vhd* test bench reads a libpcap-formatted file as captured by Wireshark and feeds it to the COM-5502 receive path. The input file must be named *input.cap* and be placed in the same directory as the Vivado project.

The libpcap file format is described in http://wiki.wireshark.org/Development/LibpcapFileFormat

Note that Wireshark is sometimes unable to capture checksum fields when the PC operating system offloads the checksum computation to the network interface hardware. In order to still be allowed to simulate, set SIMULATION := '1' in the generic map section of the COM5502.vhd component. When doing so,

(a) the IP header checksum is considered valid when read as x"0000".
(b) The TCP checksum computation is forced to a valid 0x0001, irrespective of the 16-bit field captured by Wireshark.

Components details

WHOIS2.VHD

Before sending any IP packet, one must translate the destination IP address into a 48-bit MAC address. A look-up table (within *arp_cache2.vhd*) is available for this purpose. Whenever there is no entry for the destination IP address in the look-up table, an ARP request is broadcasted to all asking for the recipient to respond with an ARP response. The main task of the *whois2.vhd* component is to assemble and send this ARP request.

ARP_CACHE2.VHD

A block RAM is used as cache memory to store 128 MAC/IP/Timestamp records. Each record comprises (a) a 48-bit MAC address, (b) the associated IP address (32-bit IPv4 or 64-bit local IPv6) and (c) a timestamp when the information was last updated. The information is updated continuously based on received ARP responses and received IP packets. The component keeps track of the oldest record, which is the next record to be overwritten.

Whenever the application requests the MAC address for a given IP address (search key), this component searches the block RAM for a matching IP address key. If found, it returns the associated MAC address. If the search key is not found or is older than a refresh period, this component asks whois2.vhd to send an ARP request packet.

The code is optimized for fast access. Response time is between 32ns and 850ns depending on the record location in memory.

This routing table is instantiated once and shared among multiple instances requiring routing services. An arbitration circuit is used to sequence routing requests from several transmit instances (for example several instantiations of the UDP_TX component).

ComBlock Compatibility List

FPGA development platform
COM-1800 FPGA (XC7A100T) + ARM + DDR3 SODIMM socket + GbE LAN development platform
Network adapter
COM-5104 10G Ethernet network interface
Software
COM-5501SOFT 10Gbps Ethernet MAC. VHDL source code.
COM-5401SOFT 10/100/1000 Mbps Ethernet MAC. VHDL source code.
COM-5502SOFT IP/UDP/TCP/ARP/PING stack for 10GbE. VHDL source code.
COM-5503SOFT IP/UDP/TCP CLIENT/ARP/PING stack for 10GbE. VHDL source code.

ComBlock Ordering Information

COM-5502SOFT IP/TCP SERVER/UDP/ARP/PING PROTOCOL STACK for 10GbE, VHDL SOURCE CODE

ECCN: EAR99

MSS • 845 Quince Orchard Boulevard Ste N •
Gaithersburg, Maryland 20878-1676 • U.S.A.
Telephone: (240) 631-1111
Facsimile: (240) 631-1676
E-mail: sales@comblock.com