Ethernet MAC and IP Protocol for a Spartan 6 Chip
Boston University Electronics Design Facility – Conor DuBois
Project derived from work of Xilinx and Jeremy Mans.

The purpose of this design is to parse incoming Ethernet packets and create appropriate responses. Specifically, there are three types of packets which this project can handle: ARP requests, ICMP Ping packets, and UDP packets adhering to Jeremy Mans' uTCA Control System protocol. In each case, the received and sent packets should resemble the following examples:

received ARP:
55555555555555D5 FFFFFFFFFFFF001B 216D722408060001
080006040001001B 216D7224C0A80201 000000000000C0A8
0202000000000000 0000000000000000 000000006BB63406

sent ARP:
55555555555555D5 001B216D7224000A 3501FE0708060001
080006040002000A 3501FE07C0A80202 001B216D7224C0A8
0201000000000000 0000000000000000 000000009055E43A

received Ping:
55555555555555D5 000A3501FE07001B 216D722408004500
0054000040004001 B555C0A80201C0A8 020208003A35CE15
0002030C574CA657 040008090A0B0C0D 0E0F101112131415
161718191A1B1C1D 1E1F202122232425 262728292A2B2C2D
2E2F303132333435 36377DCE21C9

sent Ping:
55555555555555D5 001B216D7224000A 3501FE0708004500
0054000040004001 B555C0A80202C0A8 020100004235CE15
0002030C574CA657 040008090A0B0C0D 0E0F101112131415
161718191A1B1C1D 1E1F202122232425 262728292A2B2C2D
2E2F303132333435 3637F12A185E

received UDP:
55555555555555D5 000A3501FE07001B 216D722408004500
0030000040004011 B569C0A80201C0A8 02028F0B0317001C
04EA000203200000 0010111111111AAAA AAAA12345678CACB
2886

sent UDP:
55555555555555D5 001B216D7224000A 3501FE0708004500
0020000040004011 B579C0A80202C0A8 020103178F0B000C
E539000203240000 0000000000000000 00000000B28CBCBC

Overview of Design Modules:

top.vhd:

This module connects sub_top.vhd to the FPGA pins described in the ucf file – simultaneously ensuring that all clock signals are buffered and set at the correct frequency. Three very important constants are specified within this module: myMAC (the MAC address of the PHY chip used), myIP (the IP address of the project), and udpPort (the port address used in UDP packets). All of these constants must be set correctly with respect to all the hardware and software involved before packets will travel freely. Additionally, top.vhd stores the high-level information associated with the UDP packets – the memory that is targeted by the write, read, and modify operations.

sub_top.vhd:

This module simply links seven different modules together. The general flow of data through these modules as the project shifts from receiving to transmitting is as follows:

Ge_mac_stream.vhd notices that a packet is coming and waits until the preamble has gone by to tell the rest of the project.

Gbe_rxpacketbuffer.vhd responds to the MAC and stores the packet in an internal block RAM.

Packet_handler.vhd notices the newly stored packet and tries to determine what type of packet it is; it then relinquishes the authority to examine the packet.

If the packet was an ARP request, arp.vhd takes over and formats a response.

If the packet was a Ping reqest, icmp.vhd takes over and formats a response.

If the packet was a UDP packet, the payload of the packet (which appears after the formatting information in the packet header) is sent into sub_transactor.vhd, one word at a time. Sub_transactor.vhd then determines what UDP operation should be executed, prompts top.vhd to modify or share information as necessary, and informs sub_packetbuffer.vhd of the payload that should be included in the response packet. Sub_packetbuffer subsequently makes all the necessary adjustments to finish formatting its response. If the incoming packet requests more data than can fit into a single response, then sub_packetbuffer.vhd and sub_transactor.vhd will create fragments with a minimal amount of delay in between.

Regardless of which type of packet occurred, all of the response packets are sent out one byte at a time to ge_mac_stream.vhd corresponding to the proper connection speed (either 1Gbps or 100Mbps).

Ge_mac_stream.vhd then makes several modifications to the packet: attaching the preamble, padding the packet out so that it exceeds the minimum data length, and affixing the Ethernet CRC checksum. These adjustments are made within a pipeline so that the

packets can emerge from the MAC at the speed intended.

ge_mac_stream.vhd:
     This module encapsulates the send and receive MAC interfaces to the PHY. Since it is a MAC, its main purpose is to deal with the connection's speed autonegotiation, preamble bits, and trailer checksums. These capabilities are all delegated to sub-modules within ge_mac_stream. Some traditional features (such as half-duplex mode) are not implemented.

eth_mdio.vhd:
     This sub-module of ge_mac_stream.vhd deals with speed autonegotiation. Specifically, it reads the PHY status using the MDIO interface and discovers the current LINK speed of the Ethernet connection (either 1Gbps or 100Mbps) which is used by the external Ethernet routing logic.

gmii_eth_rcv_stream.vhd:
     This sub-module of ge_mac_stream.vhd waits for incoming packets and does not inform the rest of the project until after the preamble. Moreover, it uses a small 16-byte asynchronous FIFO (bfifo.vhd) to move between two different 125 MHz clock domains that are likely not in complete synchrony with one another.

eth_rx_crc.vhd:
     This sub-module of ge_mac_stream.vhd waits for the CRC at the end of incoming packets and, if it's correct, tells gbe_rxpacketbuffer.vhd that the packet has been completely received and should be kept. A small arithmetic module (crc32_8.vhd) is used to calculate CRC's.

eth_tx_pad.vhd:
      This sub-module of ge_mac_stream.vhd takes packets that are about to transmit and, if necessary, pads them to the minimum Ethernet frame size (60 bytes – a 14 byte header and 46 bytes of data). It also adds 4 bytes to the end that will be overwritten by eth_tx_crc.vhd.

eth_tx_crc.vhd:
      This sub-module of ge_mac_stream.vhd is designed to add the CRC to an ethernet frame. The input frame to this module is a complete ethernet frame including the frame check sequence, but the frame check sequence is just a filler, which is overwritten by this module.

gmii_eth_tx_stream.vhd:
     This sub-module of ge_mac_stream.vhd adds the preamble to the beginning of a transmitted packet – 7 bytes of 55 followed by D5.

gbe_rxpacketbuffer.vhd:

This module serves as a wrapper for RAM that stores incoming packets. Multiple packets can be stored simultaneously – the starting addresses of the different packets are kept within an array so that the module can readily distinguish their boundaries within the RAM. When the signal from eth_rx_crc.vhd signals that the packet has correctly ended with a CRC, this module updates the array accordingly and indicates to the rest of the project that a packet is present. The pointers to the RAM are updated (dropping the current packet) once packet_handler decides that no module has any continued need for the packet. The module dpbr.vhd is used as the RAM for this module.

packet_handler.vhd:

Whenever the gbe_rxpacketbuffer.vhd indicates that a packet has come in, this module springs into action – determining whether the packet is an ARP, UDP, or Ping packet. It then grants the associated module the ability to read the incoming packet and create a new packet in response.

arp.vhd:

This module takes incoming ARP requests and creates a ARP reply packet. An ARP packet is very small and has a inflexible format so this task is not really difficult. The ARP packet consists mostly of addresses – destination MAC, source MAC, sender MAC, sender IP, target MAC, and target IP. Half of these addresses are copied from the incoming packet – the other half must be produced. This module must also coordinate its efforts with gbe_rxpacketbuffer.vhd and packet_handler.vhd.

icmp.vhd:

This module takes incoming Ping requests and creates a Ping reply packet. The method by which this feat is accomplished is not that sophisticated – data is simply echoed back with the exception of the source and destination addresses (which are swapped), the IP protocol (changed from Ping request to Ping reply), and the data checksum (trivially updated to reflect the altered IP protocol. This module must also coordinate its efforts with gbe_rxpacketbuffer.vhd and packet_handler.vhd. Note that the project's ability to respond to ARP and Ping packets can be easily tested by typing a command like “ping 192.168.2.2” into a terminal on a computer connected to the board.

sub_packetbuffer.vhd:

This module helps respond to UDP packets by dealing with all of the header information – leaving sub_transactor to focus on the Control System protocol embedded within the packet. The sub-module sub_packet_resp.vhd deals with storing and coordinating the transmission of the packets, in particular.

sub_transactor.vhd:
     This module looks at the payload of incoming UDP packets and determines whether it requests that one of the four possible operations (read, write, bitmask, and sum) be performed on the information stored in top.vhd. If so, sub_transactor.vhd performs the operation and informs sub_packetresp.vhd of the payload that should be included in the UDP response packet. This attention to the Control System protocol fulfills the main, high-level purpose of this project.

sub_packet_resp:
     This module formats and stores a UDP response packet – including the payload information provided by sub_transactor.vhd, whatever addresses need to be swapped in the header, a couple of IP checksums (calculated by ip_checksum_8bit.vhd), and other miscellaneous data fields. The formatting of UDP packets is notably more involved than the creation of ARP and Ping responses. Some parts of this task are handled by sub_packetbuffer.vhd instead of this module. This module must also coordinate its efforts with packet_handler.vhd.