E-Sniff : A Standalone Ethernet Packet Sniffer

By Alex Hoyland

Advisors:

Dr. Aleksander Malinowski

And

Mr. Steven Gutschlag

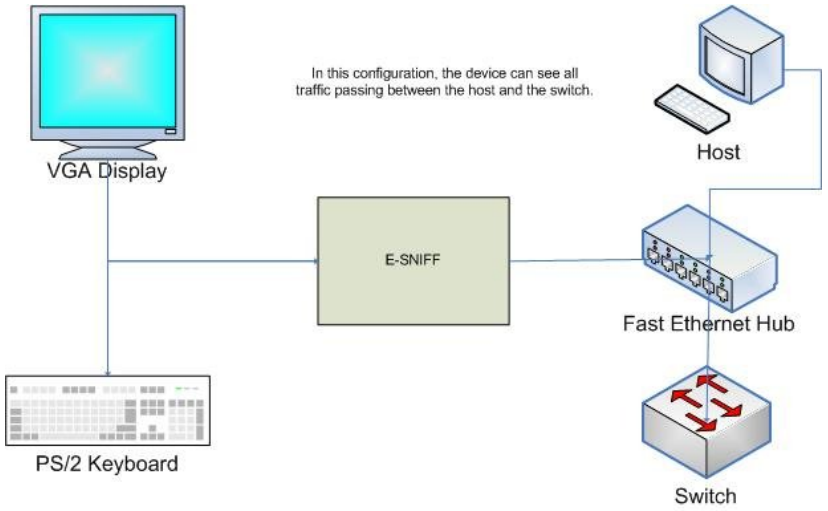
Summary

With the growing complexity of IP networks, it has become increasingly difficult to determine the source of network problems. Packet loss can occur due to any number of sources, from congestion to poorly written firewall rules to routing problems. It can at times be difficult to determine where in the network packets are lost, and system administrators will often find it helpful to view the traffic going over the line. This is the job of a packet sniffer. Many commercial and open-source sniffer applications are available for PCs; however, it would often be useful to have a special-purpose sniffing device so that one could avoid running sniffing software on high-volume server systems. A small Ethernet sniffing device could also be used for covert eavesdropping on network traffic, if one were so inclined.

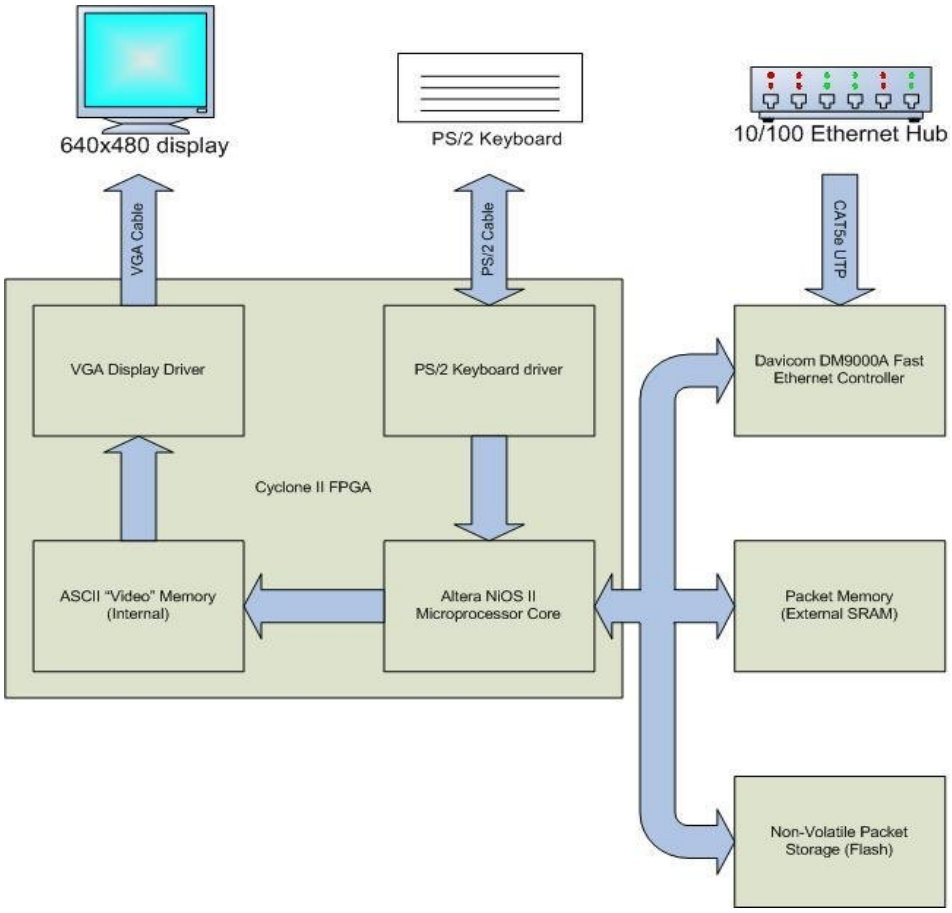
The objective of this project is to build a standalone Ethernet packet sniffer using an Altera DE2 education board with a Cyclone II FPGA. The user interface will consist of a PS/2 Keyboard and VGA monitor interface, and a 10/100 MBPS Etehrnet connection for network connectivity. Packets traveling over the line will be captured, analyzed, and displayed in a convenient format on the VGA display. The user will be able to enter simple keyboard commands to filter the captured packets based on protocol, source and destination address, and TCP/UDP port number. Network protocols supported by this device will include ARP, RARP, IP, TCP, UDP and DHCP. The device will receive all data going over the line, and will not transmit, making it very difficult to detect. In addition, it will be able to store captured packets in non-volatile memory for later review.

Functional Description

The goal of this project is to produce a device that will display basic information about network traffic in a convenient format. E-Sniff will be implemented on an Altera DE2 education board. The board will be connected to a VGA monitor and a PS/2 keyboard for user I/O, and will interface with a network hub via a 10/100 Fast Ethernet connection. The line to be monitored will be plugged into the hub, and another line will be connected from the hub to the line's original destination.



When Ethernet frames are sent over this line, the device will intercept them, identify the relevant protocols, and display information about the packet header on the VGA monitor. It will also store the packet in some form of non-volatile memory for later review. The sniffer will continue to operate without a monitor or keyboard attached, and therefore could also be used for covert eavesdropping on network traffic. The device will not transmit any information onto the network, making it very difficult to detect. A high-level block diagram of the proposed system is shown below.



Subsystem Component Functional Description

VGA display driver: The VGA display driver will be implemented in hardware. It will generate the Hsync and Vsync signals needed to display output at 640x480 resolution, and will output pixel color information. Since this monitor will only display letters, video memory will store ASCII character codes, with each memory location corresponding to a letter on the screen. The VGA hardware will fetch the character code corresponding to each pixel and output the pattern of pixels needed to display the letter. In this way, the microprocessor core will be able to update the screen simply by writing ASCII character codes into video memory. The font will be stored in a ROM within the VGA hardware. Characters will be of fixed width to simplify the task of mapping them onto the screen.

Video memory: The video memory will be organized so that each line return will shift all previous characters up one row. This will result in a screen full of text in which new lines appear at the bottom and old ones disappear off the top of the screen. This may be done in software or employ a shift-register architecture, which would simplify the software and allow for faster execution times.

Keyboard driver: The keyboard driver will accept letter and numerical inputs from an industry-standard PS/2 keyboard. A number of punctuation marks will also be supported, for use in typing IP addresses and subnet masks. Since a PS/2 interface is simply a clocked serial port, the keyboard scan codes will be read by a special-purpose shift register. Once received, the keyboard scan codes will be translated into ASCII characters and stored in memory. When the enter key is pressed, the entire string of user input will be sent to the microprocessor core, which will interpret the commands and react accordingly. At boot time, the driver will test to see whether a keyboard is connected. If one is not present, it will send a signal to the processor to start packet capture automatically with default settings.

Davicom DM9000A 10/100 Ethernet Controller: The Ethernet controller, built into the DE2 board, implements a physical layer network interface for Ethernet and IEEE 802.3 networks. It will be set up with a unique MAC address and put into promiscuous mode, which will enable it to capture packets for whwich it was not the intended recipient. The device will then begin to capture frames from the network. When a packet is received, the device stores it in internal SRAM and requests an interrupt from the NiOS II processor core, which will read the frames into its internal memory structure and extract the relevant information for display.

Altera NiOS II processor core: The NiOS II is a general-purpose microprocessor core from the Altera corporation, and will function as the heart of this system. It will be programmed to set up the Ethernet controller, set up internal memory and initialize the VGA monitor. It will then begin polling the ethernet controller for received packets and relevant network statistics. When a packet is received, the processor will check it against the current packet filtering rules. If it matches the packet filter, the processor will copy it to external SRAM and signal the Ethernet controller to continue capturing. Time permitting, packets will also be captured to a non-volatile storage medium such as a Compactflash card for review at a later time.

When not receiving captured packets, the processor will strip the headers off of packets already in memory and display the relevant information they contain on-screen. It will also periodically poll the keyboard hardware for user input and process whatever is entered. Timing will be crucial in this part of the system, as the Ethernet controller may receive packets faster than the microprocessor core can process them. In the event of a packet overflow, the NiOS II will print a message informing the user of the number of packets lost while the system is stalled. A preliminary software flowchart is shown on the next page.

Software Flowchart

The NiOS II core's software will operate as shown in the diagram below. When the system is activated, it will initialize the peripherals (Monitor, keyboard, Ethernet controller, etc.) and begin monitoring the link state. Whenever the link is good, the processor will enable three interrupts of high, low and medium priority.

The first and highest priority interrupt will execute whenever the Ethernet controller has received a frame. The microprocessor will check whether the frame is valid and then copy it from the controller's internal SRAM, storing it in the 8 MB external SRAM on the DE2 board, and in non-volatile memory for later reference.

The second interrupt will be of medium priority and will process the packets for display. It will begin by fetching a packet out of the queue, stripping off its headers and reading the protocol, source and destination information. It will check the results against simple filters to determine if the packet is to be displayed. If so, it will then write a line of text to the screen describing the packet, its source, destination and protocol. In periods of congestion when the frame memory is full, this routine may signal the high priority interrupt to stop receiving frames so that it can catch up.

The lowest priority interrupt will process user input from the keyboard. When the enter key is pressed, the keyboard driver will request this interrupt. When the interrupt is serviced, the processor will signal that it is ready to receive the keyboard data, then process it and perform the appropriate action. An acknowledgement message will display to show the user that their command has been processed. Supported commands will include start/stop capture, source and destination filters, and protocol filtering.