E-Sniff : A Standalone Ethernet Packet Sniffer

Functional Requirements List and Performance Specifications

By Alex Hoyland

Advisors:

Dr. Aleksander Malinowski

And

Mr. Steven Gutschlag

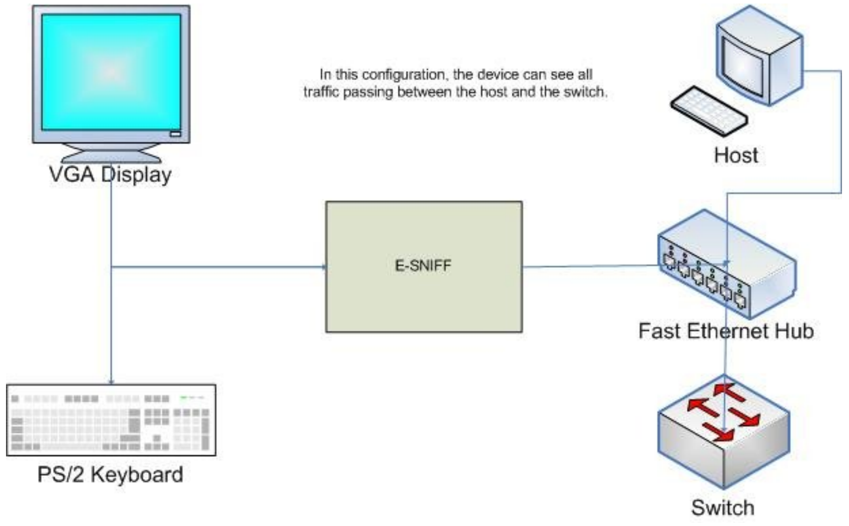
Abstract

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 and routing problems. It can 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).

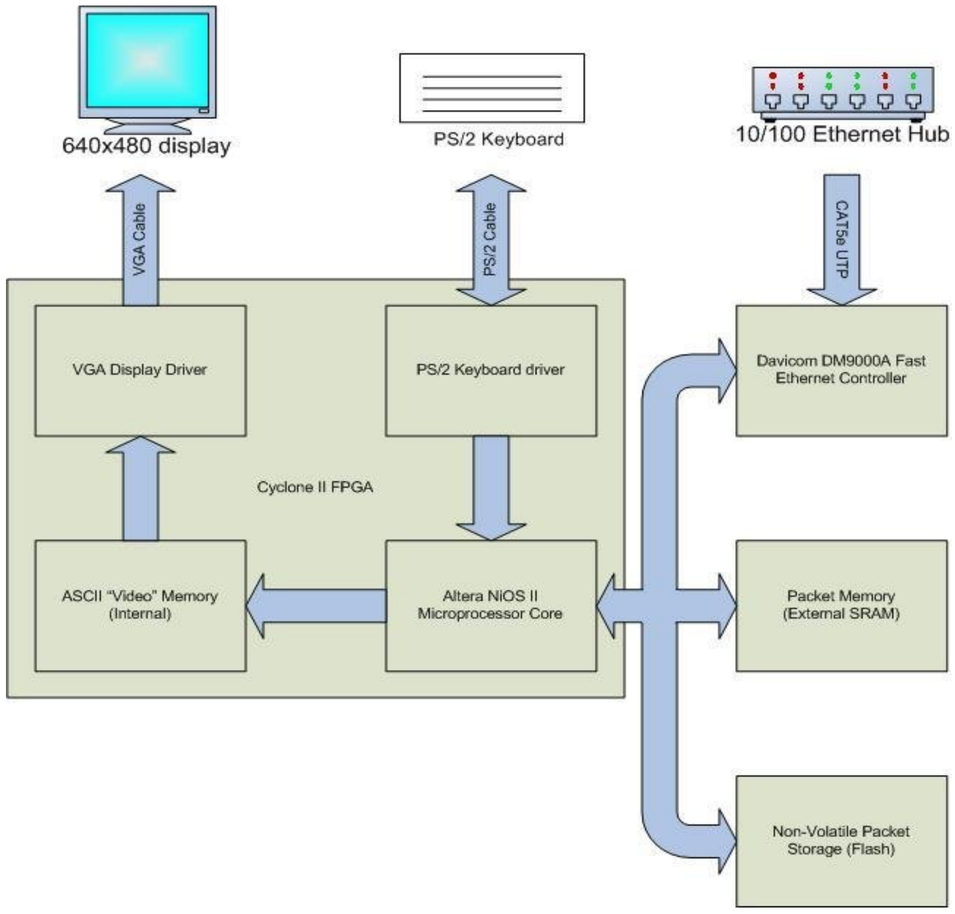
The objective of this project is to build a standalone Ethernet packet sniffer using an Altera DE2 education board with a Cyclone II FPGA. The user interface will consist of a PS/2 Keyboard and VGA monitor interface, and a 10/100 MBPS Ethernet connection for network connectivity. Packets traveling over 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.


VGA Display
In this configuration, the device can see all traffic passing between the host and the switch.
Host
E-SNIFF
Fast Ethernet Hub
PS/2 Keyboard
Switch

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.


640x480 display
PS/2 Keyboard
10/100 Ethernet Hub
VGA Cable
PS/2 Cable
CAT5e UTP
VGA Display Driver
PS/2 Keyboard driver
Davicom DM9000A Fast Ethernet Controller
Cyclone II FPGA
ASCII "Video" Memory (Internal)
Altera NiOS II Microprocessor Core
Packet Memory (External SRAM)
Non-Volatile Packet Storage (Flash)

Subsystem Component Functional Description and Performance Specifications

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.

Functional requirements for this subsystem include the ability to display all characters in memory without error regardless of the content of memory and the ability to refresh the screen at 60Hz with 640x480 resolution regardless of the current system load.

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.

The functional requirements for the video memory are as follows. This memory will be organized in a dual-port RAM. The memory will be addressed with a line offset, so that all the characters on the screen can be shifted upward in one clock cycle, simply by adding one to the offset. This will enable the microprocessor to output only one line of text when updating the display, instead of redrawing the entire display. This part of the video memory will be readable by the VGA driver and writable by the microprocessor core.

The video memory must also have a separate partition that will accept a single line of character input from the PS/2 keyboard. This small area of memory must have an asynchronous reset input that resets all character values to 20. It must be readable by the microprocessor and VGA driver and writable by the PS/2 keyboard driver.

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.

This subsystem must conform to the PS/2 standard for transmitting and receiving data to and from a keyboard. It must be able to send an arbitrary code to the PS/2 keyboard and also detect transmission errors in the start bit, stop bit and parity bit of each PS/2 frame. If an error is detected, the device will ignore the keystroke. It must also be able to output data to the VGA video memory at a rate of one character per clock cycle when each frame of data is received. The driver should able to avoid bus contention issues since both the processor and the keyboard driver will be able to drive the address lines of the VGA monitor. In addition, it must send a two clock-cycle interrupt pulse to the processor whenever the return key is pressed. It wilt translate all PS/2 keyboard scan codes to ASCII for easy processing.

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 which 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.

The functional parameters of this subsystem are already set, as it is implemented in an ASIC and cannot be changed. Its internal registers will be set to capture any packet coming over the line, to report transmission errors and to keep a running total of the number of captured packets.

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.

Functional requirements for this part of the system are as follows. The system must interface with external SRAM and the Davicom DM9000A with no read/write errors. Ideally, a read or write to SRAM or the Ethernet controller would take only one clock cycle. The processor must be able to write to a CompactFlash chip without error. It must also be able to support an input clock rate of 100 MHz. The processor will be able to interface with the external video memory and accept interrupts from the Ethernet controller, the PS/2 keyboard and the memory controller.

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 megabyte 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.

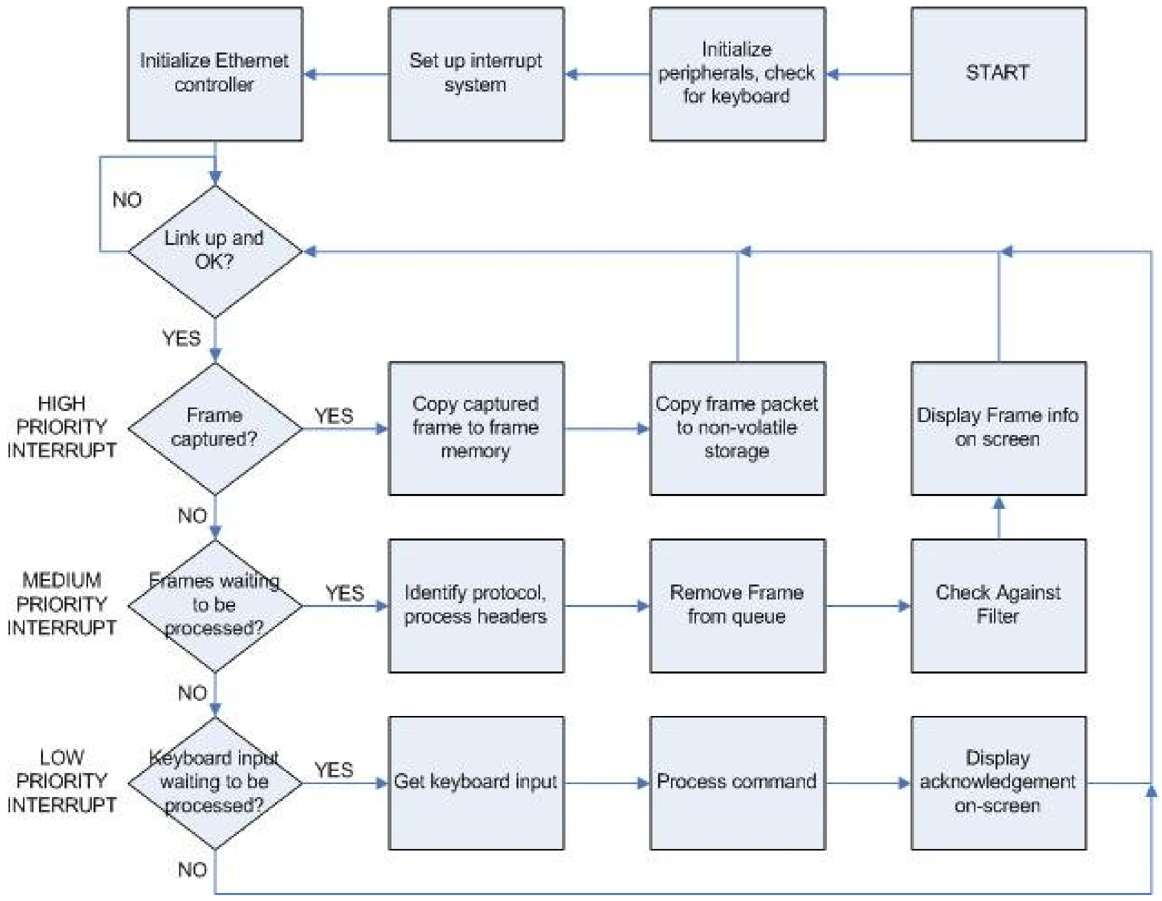
The lowest priority interrupt will process user input from the keyboard. When the enter key is pressed, the keyboard driver will request this interrupt. When the interrupt is serviced, the processor will signal that it is ready to receive the keyboard data, then process it and perform the appropriate 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.

Functional requirements for the software are as follows, and have mostly to do with timing.

A bad scenario for our packet sniffer is if we are plugged into a 100 Mbps line and are continuously receiving maximum-size packets. Since the Maximum Transmission Unit supported by the Ethernet standard is 1536 bytes, and the data rate is 100 Mbps, we have a minimum time between frames of about 100 microseconds assuming the link is at full capacity. Since this rate will never actually be achieved, we can relax this value slightly, but the software will need to process frames at a maximum rate of 1 per 150 microseconds. Since the system clock runs at 100 MHz, this will give us approximately 7,500 to 15,000 instruction cycles between frames, assuming one or two clocks per read, increment and write instruction. During this time, the processor must read in the packet and store it in SRAM (~1536 reads + 1536 writes + 3100 increments = ~6200 instructions). This leaves a 1300-instruction margin for other unexpected tasks the processor may need to handle.

The canonical worst-case scenario is when we are continuously receiving packets of the minimum length on a 100 Mbps link. This is the worst-case scenario for almost all network devices and is a common method of executing a Denial of Service attack. In this case, we receive 26-byte frames every 2 microseconds. This gives approximately 100 to 200 instructions between frames. During this time, the processor must read in the packet and store it in SRAM (26 reads + 26 writes + 52 increments = ~104 instructions. This leaves almost no margin for other unexpected tasks the processor may need to handle, and still assumes an SRAM write can be completed in two clock cycles. It may be useful to filter out packets too small to contain any relevant data, since they may significantly hinder system performance.

These scenarios lead to the following functional requirements. The software must be able to read a byte from the Ethernet controller and write it into SRAM in a maximum of 6 clock cycles. It must be able to filter packets by length, copy a maximum-length packet into memory in less than 150 microseconds and copy a minimum-length packet into memory in less than 3 microseconds. It should be able to write one line of text to the display in about 100 clock cycles, and should process packets as fast as is conceivably possible when not receiving frames. It should also be able to shut down the Ethernet controller in periods of extreme congestion so that it can process all the frames stored in SRAM before accepting more. The functional block diagram is shown below.


START
Initialize peripherals, check for keyboard
Set up interrupt system
Initialize Ethernet controller
Link up and OK?
NO
YES
HIGH PRIORITY INTERRUPT
Frame captured?
YES
Copy captured frame to frame memory
Copy frame packet to non-volatile storage
Display Frame info on screen
NO
MEDIUM PRIORITY INTERRUPT
Frames waiting to be processed?
YES
Identify protocol, process headers
Remove Frame from queue
Check Against Filter
NO
LOW PRIORITY INTERRUPT
Keyboard input waiting to be processed?
YES
Get keyboard input
Process command
Display acknowledgement on-screen
NO