Project proposal

E-Sniff : A Standalone Ethernet Packet Sniffer

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, ICMP, 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 device will operate as follows. At boot time, the device will initialize the display and PS/2 keyboard, print a splash screen, then check for network connectivity. Once the network is up, the device will set up the control registers in the Ethernet controller to operate in promiscuous mode, i.e. to capture all packets it receives regardless of the intended recipient. At this point the device will wait for a user command to begin packet capture. Once the user has specified the desired packet filter, typing 'start' will begin execution of the capture program. Each packet captured will be identified and filtered based on protocol, source and destination IP address, and source and destination MAC address by analyzing certain fields in the headers of each Ethernet frame. For each packet received, the device will print the protocol, source and destination addresses, length, and any other relevant information. Depending on the speed at which the device can operate, it may also log packets to external non-volatile memory for later review. The user can type 'stop' at any time to stop packet capture. Supported commands will include:

'start' – Start capture.
'stop' – Stop capture.
'proto arp' – toggle reception of ARP/RARP packets on or off.
'proto icmp' – toggle reception of ICMP packets on or off.
'proto dhcp' – toggle reception of DHCP packets on or off.
'proto ip' – toggle reception of IP packets on or off.
'proto tcp' – toggle reception of TCP packets on or off.
'proto udp' – toggle reception of UDP packets on or off.
'proto other' – toggle reception of all other packets on or off.
'src ip <ip> <netmask>' – only display traffic from the selected source IP address(es).
'dest ip <ip> <netmask>' – only display traffic to the specified destination address(es).
'src mac <mac>' – only display traffic from the specified MAC address.
'dest mac <mac> - only display trafffic to the specified destination MAC address.
'clear filter' – clear all packet filters to default state.
'status' – display status information.
'help' – display help information.
'about' – display a short message about the project.

Other commands may be added later as needed. In addition to the information on the display, a status message will be displayed on a small LCD screen. This will inform the user of the number of packets captured/dropped and the current link state, and LEDs will inform the user of the current system load. The sniffer will also operate automatically 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 system is shown below in fig. 2-1.

System Architecture

Due to the high data rates encountered in Ethernet applications, and the comparatively low clock speed of most embedded Ethernet processors, accomplishing the above in a traditional embedded microcontroller may not be feasible. The microcontroller would have to simultaneously perform VGA monitor syncing, process user input, and capture a torrent of Ethernet frames in soft real-time.

However, a recent rise in interest in reconfigurable hardware processing systems has led to the widespread availability of soft processor cores, which would be ideal for use in this project. To simplify the job of the processor and free up CPU time for packet capture and analysis, the E-Sniff system will use custom hardware systems to perform user I/O such as VGA monitor syncing, while a custom soft processor core performs the task of capturing and analyzing packets.

Accordingly, the E-Sniff project will be implemented as an application specific system-on-a-chip, with all special-purpose hardware implemented in an FPGA. Hardware descriptions for the various custom I/O interfaces will be written in VHDL. The central processing unit will be an Altera Nios II soft processor core, which will be customized and instantiated into the design using Altera's SOPC builder. This way a robust custom processor core can be developed in a matter of weeks that will operate more efficiently than a traditional microprocessor system.

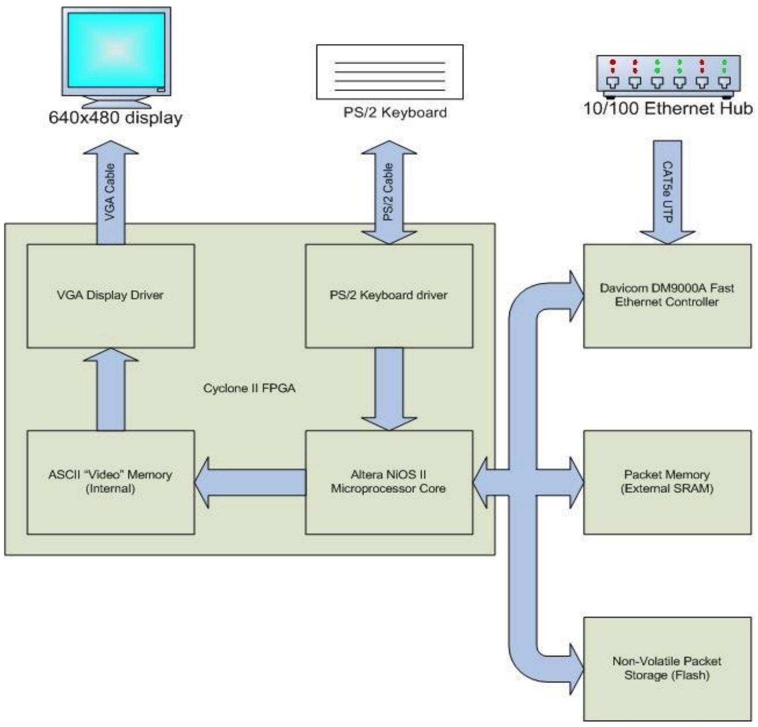
640x480 display
PS/2 Keyboard
10/100 Ethernet Hub
VGA Cable
PS/2 Cable
CAT5e UTP
VGA Display Driver
PS/2 Keyboard driver
Davicom DM9000A Fast Ethernet Controller
Cyclone II FPGA
ASCII "Video" Memory (Internal)
Altera NiOS II Microprocessor Core
Packet Memory (External SRAM)
Non-Volatile Packet Storage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

Fig. 2-1 – Block Diagram of proposed system.

Development Platform

The E-Sniff system will be implemented on an Altera DE2 Education board. This board contains a Cyclone II 32,000 LE FPGA, a Davicom DM9000A Ethernet MAC/PHY, a high-speed VGA Video DAC, PS/2 Keyboard port, VGA Monitor interface, 4 megabyte Flash, 8 megabyte SDRAM and an SD card interface – everything necessary for this project's completion. The DE2 development platform is shown in figure 2-2 below, with the peripherals utilized for this project highlighted in red.

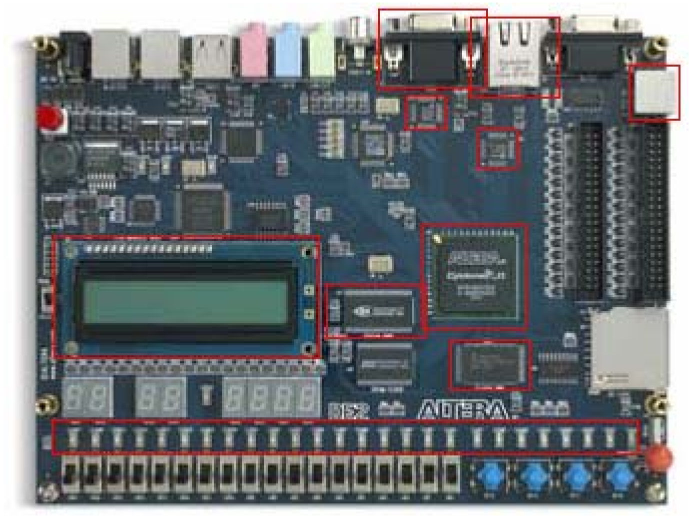

Fig. 2-2 – DE2 Development Board from Altera.

The various subsystem components in the block diagram above are detailed below, in order of importance to overall system function.

Hardware Description and Performance Specifications

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 designed in SOPC builder and will use Altera's preprogrammed interfaces to interact with the rest of the system

Functional requirements for this part of the system are as follows. The system must interface with external SDRAM, flash and the Davicom DM9000A with no read/write errors. Ideally, a read or write to SDRAM or the Ethernet controller would take less than four clock cycles. The processor must be able to write to an SD card without error, with the highest achievable data rate. 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. It will also connect to the LED's on the system board, and the 16x2 LCD.

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.

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.

Software Description and Performance Specifications

Software for the Nios II will be written using the Altera Nios II IDE and stored in the DE2's onboard flash memory. A boot loader, provided by Altera, will load the program from flash into SDRAM at boot time, permitting the use of non-volatile program memory without sacrificing efficiency.

The NiOS II core's software will operate as shown in figure 5-1 below. When the program entry point is reached, 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 to the system's onboard memory. Basic source and destination filtering operations will also be performed at this stage to minimize the number of unnecessary packets in the processing queue.

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 protocol 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.

An alternative to the above is simply to process packets as they arrive. This may prove to be a better approach since it will lead to simpler system software and reduces the potential of errors due to overloading the device with packets. In this approach, the system would process each packet as it arrived and would drop frames in times of high network congestion. Depending on operating speed of the completed system and the memory latency of the SDRAM, this method could be more or less efficient than the one previously mentioned. This problem will be investigated in more detail once the hardware development cycle is completed.

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. Keyboard data will be stored in a one-line hardware buffer and will only be processed when an entire line has been input and the user has pressed enter.

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. If 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 (~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 empty 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. Ideally, the software must be able to read a byte from the Ethernet controller and write it into SDRAM 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. A preliminary system software flowchart is shown below in figure 5-1.

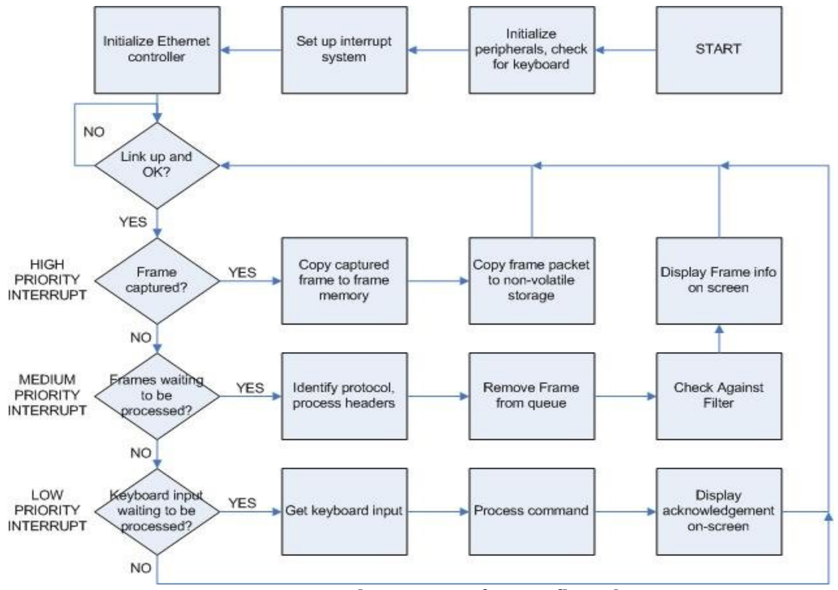
START
Initialize peripherals, check for keyboard
Set up interrupt system
Initialize Ethernet controller
NO
Link up and OK?
YES
HIGH PRIORITY INTERRUPT
Frame captured?
YES
Copy captured frame to frame memory
Copy frame packet to non-volatile storage
Display Frame info on screen
NO
MEDIUM PRIORITY INTERRUPT
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

Fig.5-1 – Preliminary software flowchart.

<u>Work Schedule</u>

Work on the E-Sniff project will be divided into four main tasks: hardware development, hardware testing, software development, and product performance testing. The amount of time allotted for each task will be based on the relative difficulty of that task and the amount of work already completed.

Hardware Development (3 weeks) : The hardware development cycle entails (1) Adapting the display and keyboard drivers, currently designed using Xilinx tools, for use with the Altera board. (2) Designing a Nios II processor with all the necessary interfaces and functions to perform the required tasks. (3) Interfacing the Nios II processor with user-defined hardware in an efficient and reliable manner. This process will be greatly simplified due to the large number of example designs and excellent documentation included with the DE2 board, as well as the availability of simple tools for Nios II processor design.

Hardware Testing (2 weeks) : The hardware testing cycle will include (1) functional testing of the hardware system to ensure proper interoperation of the processor and peripheral devices and (2) performance testing to determine timing constraints that will affect the design of the system software. This may entail writing basic test code and I/O routines and thus may overlap with the software design process.

Software Design (5 weeks) : The software design cycle will involve (1) designing software libraries to interface with the I/O devices such as the keyboard, VGA display, LCD display, LEDs, and the Ethernet controller. (2) Designing a basic packet capture program that captures and displays all data received. (3) Adding user-definable filtering operations to the packet capture program. This will be the most important and difficult aspect of the design process and will require an excellent understanding of the system hardware. This process will be simplified by the availability of example Nios II code from Altera.

Product Performance Testing (2 weeks) : The final task in this project will entail (1) Testing of the finished system to eliminate runtime errors and reduce or eliminate dropped packets. (2) Testing to determine operational effectiveness under periods of high network load and extended operation.

<u>Conclusion</u>

Due to the many facets and broad scope of this project, it should prove to be a difficult but useful exercise in digital system design. The E-Sniff project integrates elements of digital hardware design, embedded C programming, digital communications and real-time operating systems. For successful completion of the project, it will be critical to stick to deadlines and test every part of the design thoroughly before moving on to the next one. By adhering strictly to the schedule outlined above, it should be possible to complete the project in the allotted time and have a finished product ready by mid-spring.