Deep Packet Inspection (lite)

1.0 Abstract

A good description of Deep Packet Inspection comes from Wikipedia.

> ***Deep Packet Inspection (DPI) (also called complete packet inspection and Information eXtraction - IX -) is a form of computer network packet filtering that examines the data part (and possibly also the header) of a packet as it passes an inspection point, searching for protocol non-compliance, viruses, spam, intrusions, or defined criteria to decide whether the packet may pass or if it needs to be routed to a different destination, or, for the purpose of collecting statistical information. There are multiple headers for IP packets; network equipment only needs to use the first of these (the IP header) for normal operation, but use of the second header (TCP, UDP etc.) is normally considered to be shallow packet inspection (usually called Stateful Packet Inspection) despite this definition. (Deep packet inspection)***

For this project you will be implemented a deep packet inspector used to collect statistical information about traffic between two devices.

2.0 System Overview

The overall system may be seen in the diagram below. Two devices, called Sender and Receiver, are communicating on a shared link. The interface is synchronous, sender initiated, non-responsive, and flow controlled. The bus is a serial link with an assumed shared clock. Your device will "watch" the bus between the Sender and Receiver to determine when a transaction is occurring, capture the data that are transferred (as much as is needed to perform the inspection), and process the data to accumulate statistics.

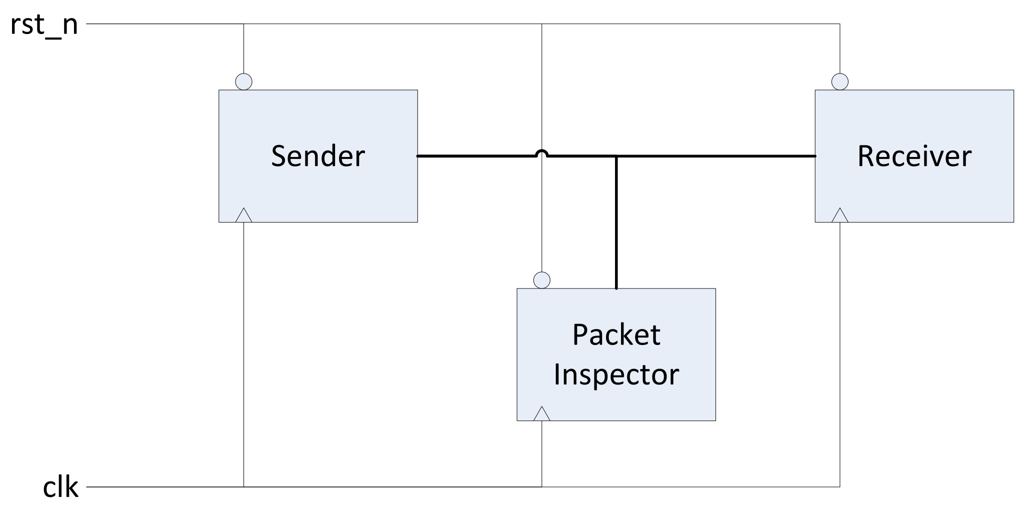


Figure 1: System Level Diagram

2.1 Protocol

The interface protocol between the Sender and Receiver consists of identifying a start pattern in a serial stream of data and capturing a specific number of bytes following the start pattern. The start pattern for the message is 32'hA5A5A5A5 (bits) and the number of bytes following the start pattern is 32 (bytes). Data are always transferred most significant bit first. The waveform diagram below shows how a message is transferred from the Sender to the Receiver.

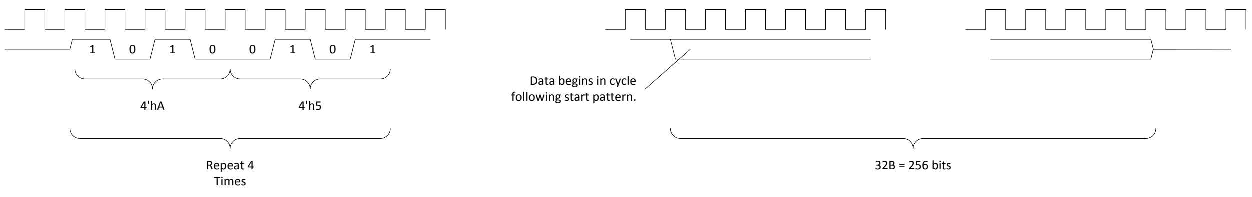


Figure 2: Message Transfer Waveform

2.2 Message Format

When the complete 32 byte message has been received it will adhere to the following structure.

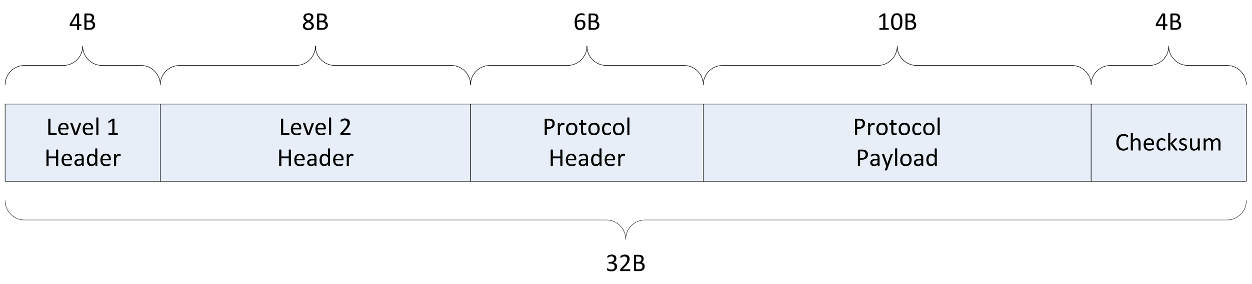


Figure 3: Message Format

2.2.1 Level 1 Header

The Level 1 Header contains a pair of 2 byte fields. The most significant (MS) quantity is the address of the Sender and is an unsigned integer. The least significant (LS) quantity is the address of the Receiver and is an unsigned integer.

2.2.2 Level 2 Header

The Level 2 Header contains four 2 byte fields. The MS quantity contains the physical address of the Sender. The next quantity contains the physical address of the Receiver. The next quantity is the port number of the transaction and the LS quantity contains a sequence number. All quantities are unsigned integers.

The port number identifies the particular protocol that the message adheres to. While this quantity may take on 2^{16}-1 values only the following need to be processed by your design. All other port numbers are to be ignored.

Port Number	Protocol
23399	Skype
20	FTP
443	HTTPS
22	SSH

23	Telnet
25	SMTP
161	SNMP
563	NNTP

Table 1: Port Numbers and their Protocols

Your module should maintain a count of the number of times a message is seen for each of these protocols, independently. I.e. you need eight (8) counters, one for each protocol.

2.2.3 Protocol Header

The Protocol Header contains six bytes of information related the particular protocol identified in the Level 2 Header. Only the Protocol Header for Skype, SSH, and Telnet need inspecting. For each of these the least significant byte in the Protocol Header contains a monotonically increasing session identifier. It is possible to have a number of Telnet sessions in process simultaneously. This byte indicates which session the payload belongs to. Your module should track the number of unique sessions that have been seen.

A simplifying feature is that the session IDs are monotonically increasing. Therefore you needn't keep track of each that occurs. Instead it is sufficient to see the first one (which may not be one (1)) and update your count when you see the next higher number and each time the session identifier increments after that. For example, say the first session ID seen is 5. Your counter should increment to 1. Later, upon seeing the number 6 for the session ID your counter should increment to 2, etc.

This capability of your module will require three (3) additional counters to track the number of unique session IDs for each of the session based protocols.

2.2.4 Protocol Payload

The Payload for each of the protocols is not important. The Payload is 10 bytes long.

2.2.5 Checksum

The Checksum for each of the messages is not important. The Checksum is 4 bytes long.

3.0 Top Level Block Diagram

The top level block diagram for your inspector can be seen below. The rst_n, data, and clk inputs are for the system interface and the counter values are for debug and verification.

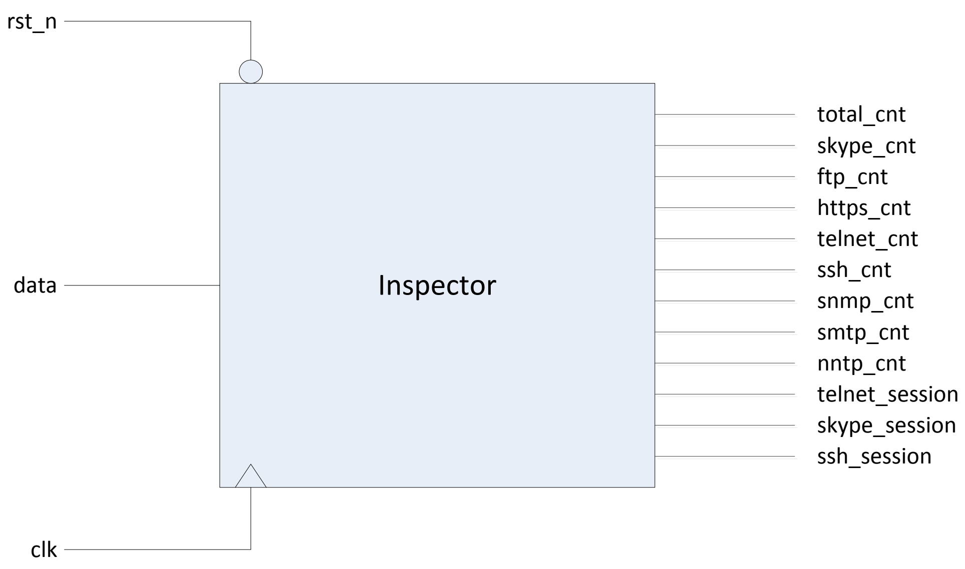


Figure 4: Top Level Block Diagram

4.0 Requirements

4.1 Syntax

[REQ0010] The top level design module **shall** be named inspector.

[REQ0020] The top level design module **shall** have the following port list.

- input rst_n
- input data
- input clk
- output [31:0] total_cnt
- output [7:0] skype_cnt
- output [7:0] ftp_cnt
- output [7:0] https_cnt
- output [7:0] telnet_cnt
- output [7:0] ssh_cnt
- output [7:0] snmp_cnt
- output [7:0] smtp_cnt
- output [7:0] nntp_cnt
- output [7:0] telnet_session
- output [7:0] skype_session
- output [7:0] ssh_session

[REQ0030] The top level testbench module **shall** be named inspector_tb.

4.2 Interface

[REQ0040] The inspector **shall** respond to the rst_n input as an active-low synchronous reset by setting all output counters to zero and resetting any internal logic as necessary.

[REQ0050] The inspector **shall** respond to the input clk as a synchronous free-running clock against which all operations occur and data are received.

[REQ0060] The inspector **shall** accept the serial stream of data as input a single bit at a time.

[REQ0070] The inspector **shall** provide an output called total_cnt as a 32-bit unsigned value indicating the total number of complete messages seen since the de-assertion of rst_n.

[REQ0080] The inspector **shall** provide an output called skype_cnt as an 8-bit unsigned value indicating the total number of Skype protocol messages seen since the de-assertion of rst_n.

[REQ0090] The inspector **shall** provide output called ftp_cnt as an 8-bit unsigned value indicating the total number of FTP protocol messages seen since the de-assertion of rst_n.

[REQ0100] The inspector **shall** provide an output called https_cnt as an 8-bit unsigned value indicating the total number of HTTPS protocol messages seen since the de-assertion of rst_n.

[REQ0110] The inspector **shall** provide an output called telnet_cnt as an 8-bit unsigned value indicating the total number of telnet protocol messages seen since the de-assertion of rst_n.

[REQ0120] The inspector **shall** provide an outpu called ssh_cnt as an 8-bit unsigned value indicating the total number of SSH protocol messages seen since the de-assertion of rst_n.

[REQ0130] The inspector **shall** provide an output called snmp_cnt as an 8-bit unsigned value indicating the total number of SNMP protocol messages seen since the de-assertion of rst_n.

[REQ0140] The inspector **shall** provide an output called smtp_cnt as an 8-bit unsigned value indicating the total number of SMTP protocol messages seen since the de-assertion of rst_n.

[REQ0150] The inspector **shall** provide an output called nntp_cnt as an 8-bit unsigned value indicating the total number of NNTP protocol messages seen since the de-assertion of rst_n.

[REQ0160] The inspector **shall** provide an output called skype_session as an 8-bit unsigned value indicating the total number of unique Skype session IDs seen since the de-assertion of rst_n.

[REQ0170] The inspector **shall** provide an output called ssh_session as an 8-bit unsigned value indicating the total number of unique SSH session IDs seen since the de-assertion of rst_n.

[REQ0180] The inspector **shall** provide an output called telnet_session as an 8-bit unsigned value indicating the total number of unique Telnet session IDs seen since the de-assertion of rst_n.

4.3 Input Data

[REQ0190] The inspector **shall** identify the start of a message by seeing the serial stream 32h'A5A5A5A5 on the input data.

[REQ0200] The inspector **shall** identify the protocol of the message from the port number in the Level 2 Header.

[REQ0210] The inspector **shall** identify the session ID, if necessary, from the Protocol Header.

5.0 Testing and Verification

There is no sample testbench provided for this project. You should create a testbench that stimulates your design printing out the resulting values of each of the counters at the end of the simulation.

5.1 Extra Credit

There is an extra credit component to this project. Up to 10 points may be awarded for demonstrating your project in the lab to the instructor or TA. Demonstration is of a fully synthesized project executing on development board hardware.

6.0 Synthesis

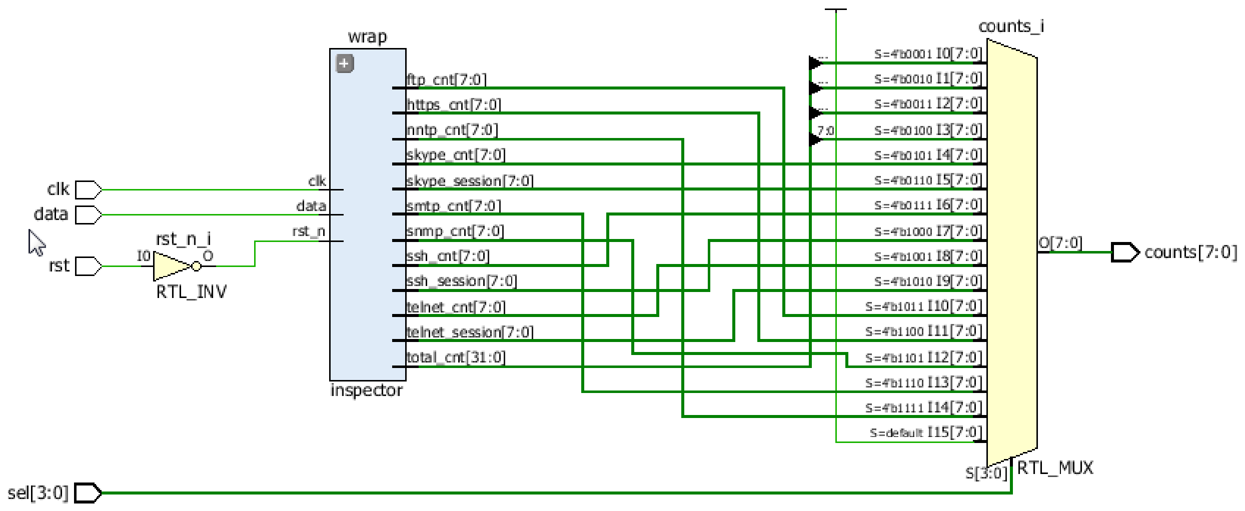
NOTE: You are to only synthesize your design, inspector. You should not include your testbench in the synthesis step.

There is a synthesis component to this project. Using the Vivado tools you should synthesize your design as you did in homework 7 for your lab 3 Verilog. Eliminate any errors and warnings over which you have control and that come from your design. The resulting top level RTL netlist; the "Slice LUTs," "Slice Registers," "IO," and "Clocking" from the Synthesis Utilization report; and any warnings that are encountered **shall** be provided in your writeup.

A master constraints file will be provided allowing the design to operate on the Basys3 development board. The wrapper only instantiates your design allowing it to synthesize for the Basys3.

6.1 RTL Netlist

Vivado is able to produce a schematic of your design allowing you to see the hardware that was built. This is a very helpful tool when you are evaluating whether your Verilog matches what you intended. Under RTL Analysis -> Elaborated Design, click on Schematic. You should see something like



This is the wrapper that has been provided along with the mux that controls which of the outputs is presently routed to the LEDs on the development board. Notice that there are select inputs allowing you to cycle through the particular count values. Since the wrapper doesn't provide any data, these will all be off. The extra credit wrapper will stimulate your design and you'll be able to see the count values.

6.1.1 Your Schematic

Click on the [+] sign at the top left of the wrapper to expand what is inside. You should see a relative mess that is your design. You should now be able to right click and save the schematic as a PDF. You may also be able to print the schematic. (Or, you could capture the screen and crop out the IDE bits, leaving only the schematic.) Any way you choose, your report **shall** include a copy of this schematic.

7.0 What to Turn In

7.1 Implementation

Submit your inspector implementation electronically to Wolfware. Remember that, irrespective of the filename, your top level module should be called inspector. The files should all compile with the command "vlog *.v" without any errors or warnings.

7.2 Testbench

Submit your testbench implementation electronically to Wolfware. Remember that, irrespective of the filename, your top level testbench module should be called inspector_tb. The files should all compile with the command "vlog *.v" without any errors or warnings. Your testbench should run to completion printing the values in the counters upon finishing the simulation with the command 'vsim –c –novopt –do "log –r *; run –all; quit" inspector_tb'.

7.3 Documentation

While there is no formal report for this project you shall turn in printed out copies of the RTL schematic that Vivado produces for your design and a printout of synthesis and utilization results. Please format the RTL schematic to be on a single page (the page may be 8.5” x 11” or 11” x 17”).

Recall that you will need to press the [+] symbol at the top left of your block in the schematic to expand it so that we are able to see what is inside of the design. The RTL schematic may be captured from using the basys3_wrapper.* files on the course web page; these only provide a wrapper to implement the design, they do not stimulate your design.

For extra credit you’ll need to present your design to the TA running with the DPI harness files provided on the course web page.

8.0 References

Wikipedia contributors. "Deep packet inspection." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 21 Mar. 2013. Web. 3 Apr. 2013.