FPGA-based Ethernet Sniffer for Real-Time Networks

João Faria, Arnaldo Oliveira, Paulo Pedreiras, Rui Santos

Abstract **– This paper presents an *Ethernet* sniffer based on dedicated hardware, able to carry out the timestamping of network events with a level of resolution and precision compatible with the specific needs of real-time protocols. The sniffer is based in FPGA technology, and autonomously stores in local memory the received messages data and associated information (timestamps and size). This data is then fed to the host computer, via an USB connection, and stored in a file format compatible with *Wireshark*. This allows using standard tools to subsequently analyze the traffic. Furthermore, in the scope of this work it was also developed custom tools to automate the analysis of timing properties of the traffic, including the generation of graphics and statistical data, which are common operations in the analysis of real-time protocols. The paper includes experimental results of a prototype implementation showing that this tool permits meticulous temporal measurements, with a resolution of $10ns$ and a maximum error of $100ns$, which improve the results obtained with standard software-based applications by over one order of magnitude.**

Resumo **– Este artigo apresenta um *sniffer Ethernet*, baseado em hardware dedicado, capaz de efectuar a marcação temporal dos eventos de rede com uma resolução e precisão compatíveis com as necessidades específicas dos protocolos de tempo-real. O *sniffer* é baseado em tecnologia FPGA e armazena autonomamente as mensagens que circulam na rede, bem como informação complementar relevante (marca temporal e tamanho). Estes dados são enviados para um computador externo, via uma ligação USB, e armazenados num ficheiro compatível com o *Wireshark*. Esta abordagem possibilita o uso de ferramentas standard para analisar o tráfego. Complementarmente foram também desenvolvidas ferramentas específicas para execução automática de operações de análise temporal comuns em protocolos de tempo-real, nomeadamente a geração de gráficos e a extracção de dados estatísticos. O artigo inclui resultados experimentais obtidos de uma implementação piloto, os quais permitem concluir que a ferramenta apresenta um elevado rigor, com uma resolução de $10ns$ e um erro máximo de $100ns$, correspondendo a um melhoramento superior a uma ordem de grandeza em relação às ferramentas standard baseadas exclusivamente em software.**

Keywords **– Communication Networks, *Ethernet*, Real-Time, Distributed Systems, Sniffer, Network Analyzer, Time-Stamping, Reconfigurable Computing, FPGAs.**

Palavras chave **– Redes de Comunicação, *Ethernet*, Tempo-Real, Sistemas Distribuídos, *Sniffer*, Analisador de redes, Marcação Temporal, Computação Reconfigurável, FPGAs.**

I. Introduction

Nowadays, communication networks are used in many distributed applications that need to share information between the different nodes that compose the system. *Ethernet* has emerged as the *de facto* standard in generic local area networks, being used in a wide scope of applications, ranging from complex systems as banks, where massive amounts of data flows are supported, to the network used at home to access internet or share a printer. A common property shared by these "classical" *Ethernet* applications is that they are inherently soft-real time. In this context the network is required to provide high throughput and good average response times, but occasional abnormal traffic delays, due to e.g. overloads, have no catastrophic consequences.

In the last years *Ethernet* has established itself also as one of the most important networking technologies even for systems with extra functional requirements on timing, raising the so-called *Real-Time Ethernet* (RTE) protocols. Switched *Ethernet* architectures, in particular, have been highly regarded for this class of applications since they alleviate the impact of the non-determinism inherent to *Ethernet's* CSMA/CD medium access control (MAC). Nowadays RTE protocols are found in industrial environments, military systems, avionics, body networks in automotive applications, etc. Many of these application classes have strict timeliness requirements and thus, computational results must be logically correct but also be obtained during restrict and predictable time intervals. For instance, in the closed-loop control of the motors of mobile robots the commands have to be sent at precise time intervals or the performance degrades, potentially causing catastrophic consequences in material terms or even jeopardizing human lives, depending on the particular environment. Clearly, for this kind of applications the timeliness aspects of the protocols are of

Work partially supported by the FCT project PTDC/EEA-ACR/73307/2006 (HaRTES - Hard Real-Time Ethernet Switching)

paramount application, and thus the network performance has to be analyzed also in this dimension.

The validation of the proper operation of a network is typically done by means of the so-called traffic analyzer tools, also known as sniffers. The purpose of using these tools is to capture messages and the time instants in which they flow on the network. There are many applications available for this purpose, both commercial and open source. One of the best known applications is *Wireshark* [**?**] (formerly *EtheReal*). The problem of these solutions is that they are mainly software-based, relying on standard PC-class hardware to observe the traffic. Despite being simple and inexpensive, in this approach the timestamps are taken at the application layer, and thus suffer from a set of cumulative interference sources that inevitably compromise the accuracy of the measurements, resulting in temporal resolutions of the order of 1μs and an uncertainty of the order of milliseconds. Many real-time applications have cycle times of the order of milliseconds, thus these tools become clearly unsuitable for verifying the RTE timeliness aspects. Some of the commercial tools available for *Ethernet* networks are based on specialized hardware. In this case the timestamps are generated by customized network interface cards, exhibit improvements of one order of magnitude compared with software-only approaches. The main problem with these approaches is that they are proprietary, closed and extremely expensive.

The usage of programmable logic devices is a low cost and flexible solution to produce dedicated circuits to perform specific tasks. The existence of intellectual property (IP) cores allows increasing the productivity by reducing the prototype time and the costs associated to the manufacturing of Application Specific Integrated Circuit (ASICs). This observation fostered the development of a low-cost, high-resolution, open and flexible *Ethernet* analyzer. The analyzer framework is based on standard FPGA technology and uses open file formats, facilitating the use of commonly available tools (e.g WireShark). A set of analysis tools, specific to RTE protocols and not commonly available in standard analyzers, has also been developed in the scope of this work.

The remaining of this paper is organized as follows. Section II describes the motivation of the work and introduces some related applications and studies. Section III describes architecture and some relevant details of the sniffer internal structure. Section IV shows the results of implementation on FPGA. Section V describes the tests done and the results obtained. Finally, section VI presents the conclusions and some points to be improved and considered in the future work.

II. Related Work

Many protocol analyzer tools are available in the market. Some of them are pure software approaches, relying solely on the hardware of the host computers (typically PC-based), to carry out the packet capture, timestamping and analysis. These tools are normally extremely complete and powerful, being able to handle most of the available communication protocols. One of the best known applications belonging to this class is *Wireshark* [?] (formerly *EtheReal*). A common feature to this class of applications is that the timestamps are generated at the application-level, and thus the accuracy with which the events are recorded is affected by diverse cumulative sources of interference, such as the PCI bus, DMA, higher-priority interrupts, non-preemptive OS sections, scheduling jitter, etc. The net result of all these sources of indeterminism is that the time elapsed between the actual reception of a message and its timestamping is essentially unpredictable and highly variable, depending, among other things, from the processor utilization, device-drivers installed, sources of interrupts, locks by the access to shared resources and the hardware characteristics (direct memory access, cache memory and pipelined architectures). The accuracy with which the timestamp of the network events is generated is gross, typically with a temporal resolution of 1μs and an uncertainty of the order of milliseconds. Since many real-time applications have cycle times of the order of milliseconds this class of tools, despite inexpensive and powerful, become clearly unsuitable for use in RTE protocols analysis.

Many academic and commercially available tools improve the degree of accuracy provided by pure software-based by over one order of magnitude, thus reaching a level suitable to test the compliance of RTE protocols. These approaches are based on specialized hardware support. In [**?**] a multi-probe measurement instrument for real-time *Ethernet* networks is presented. It is composed by many probes that can be placed among the network and each one is responsible for monitoring a single full-duplex link. A secondary or auxiliary measurement network has been created to convey results of logging and timestamping. This measurement network is also used for distributed clock synchronization in order to allow the comparison of timestamps from different probes. Each probe associates a timestamp to every frame that transit in the monitored link and then, the original frame is encapsulated in a new longer frame containing the timestamp and additional information. The measurement network is a *Gigabit Ethernet* network and the captured packets are encapsulated in jumbo frames. These frames are then sent to a monitor station that de-encapsulate incoming packets retrieving original frames and timestamps and then the information is displayed using *Wireshark*.

Other tools for *Ethernet* networks have capabilities to generate traffic and, comparing it with captured traffic, they can measure latency or frame losses [**?**]. AE5501 [**?**] is commercialized by *Yokogawa*, costs about 3000€and it has a resolution of 1μs (with a maximum error of 3μs) for 10Mbps connections and a resolution of 100ns (with a maximum error of 300ns) for 100Mbps and 1Gbps connections. These tools are usually remotely controlled for configuration and for

a later analysis of the results and it is possible to connect various equipments among the network. Advanced tools can have higher processing and storage capacities (15T Bytes), with prices that can go up to 40000€. Besides being extremely costly, these tools are also closed and inflexible. They are based on custom hardware and use proprietary file formats. Thus, using standard tools or developing a new one for analyzing the traffic network is hardly accomplishable and, when possible, inefficient due to the poor integration.

The above discussion permits concluding that the protocol analyzers available are either unsuitable due to gross temporal accuracy, or expensive, closed and inflexible. Thus, there is room for the development of a low-cost, high-resolution, open, flexible and extensible *Ethernet* analyzer, which is the objective of this work.

III. Protocol Analyzer Architecture

The protocol analyzer acquisition unit was designed with the intention of making accurate time measurements and minimize as much as possible the perturbation in the network. The corresponding block diagram is represented in figure 1. The acquisition unit is inserted in the link under analysis using an *Ethernet* Test Access Point (TAP) that duplicates the data flows. There are many commercial TAPs [**?**], many of them with the capability of regenerate the signals. However, in our case, due to the objective of building a low cost tool, it was used a home-built passive TAP, using only a resistor-based adaptation circuit. Besides being considerably cheaper, this approach does not introduce additional latencies, contrarily to the regenerative TAPs. On the other hand the passive TAP introduces a perturbation at the physical level that in practical terms will reduce the maximum length of the link.

The interface between the programmable logic device responsible for processing the captured data and the transmission medium is done via 2 *Ethernet* PHYs, i.e. the *Ethernet* physical layer implementation chips, one for each traffic direction, allowing capturing simultaneously the traffic that flows in both directions of a full-duplex connection. The FPGA receives data from PHYs, registers the instant in which each message is received (timestamping) and then forwards them with all associated information to a personal computer, via an USB connection. The captured data is stored in the personal computer and can be analyzed later. Finally, the board also includes two push-buttons that allow starting and finishing the capture.

Internal Structure

The FPGA implements two distinct tasks, one related with reception and processing of the data captured from *Ethernet* network and the second one associated with the need to feed the data to the host computer. The tool is required to permit the analysis of full-duplex connections, thus two independent MAC IP cores have been used, allowing the reception

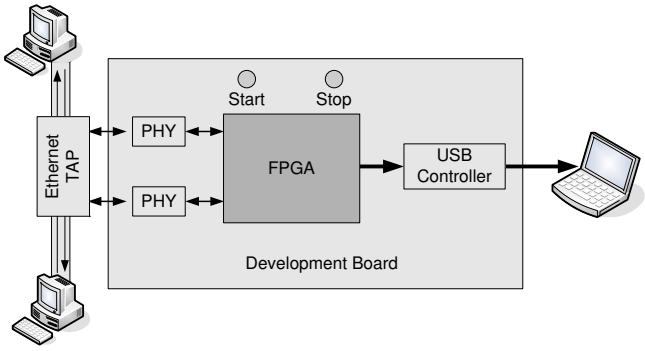


Fig. 1 - Architecture of the sniffer acquisition unit.

of two independent messages concurrently. Once received, the raw packet contents is associated with a control header that conveys additional control information associated with the packet, namely the instant in which the message has been received, its length and the number of message bytes stored. The format used to store the file internally in the FPGA is the same used on the host computer side, in order to minimize the processing time.

The memory used for message buffering has been divided in two independent FIFO (First In, First Out) memories. The *Data FIFO* is responsible for storing the raw content of each message, while the *Control FIFO* is responsible for storing the associated control information. This memory isolation facilitates the concurrent access to both memories. Finally, the messages are sent to the host computer via an USB connection. Since the protocol conversions are already done the process resumes itself to get one entry from the *Control FIFO*, compose it with the corresponding entry of the *Data FIFO* and send the whole info via the USB channel.

The medium access control layer is based in the *Tri-Mode Ethernet Media Access Controller* (TEMAC) [**?**]. It is a parameterizable core built by Xilinx and basically provides to the user three types of signals: clock, data and validation or control. Data is released in a 8 bit bus width and can have a latency variation of three clock cycles. Considering *Fast Ethernet* it corresponds to a maximum delay variation or jitter of 120ns.

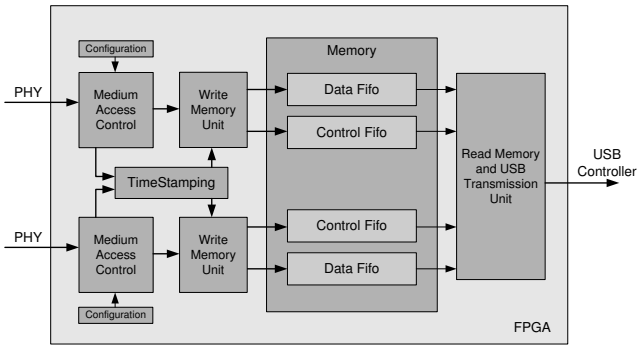


Fig. 2 - Internal structure of the sniffer hardware.

When the reception of a message starts (first eight

bytes) and the corresponding data valid signal is detected, the message is timestamped and this value is stored in the control memory. Simultaneously, data starts being stored into data memory. When the reception ends, the message length and the number of bytes captured are stored in control memory and thus the message becomes ready to be uploaded via the USB connection. This process is illustrated in figure 2.

It is possible to bound the maximum number of data bytes stored for each message. This option is useful since in many cases only the initial few bytes of the packets are relevant and thus it is possible to minimize the buffer sizes, upload time and log file size. Besides speeding up the operation, in heavily loaded scenarios this feature turns out to be particularly relevant since the throughput of the USB link is lower than the aggregated bandwidth of the full-duplex *Ethernet* link, and thus recording all the data leads inevitably to memory overflows. However, if the amount of stored data is reduced to appropriate levels, compatible with the USB channel bandwidth, memory overflows can be prevented. In many cases the relevant data is the message reception instant and a few control bytes stored at the head of the *Ethernet* packets, and thus stripping of the data packets has no relevant impact on the tool performance and accuracy of the information provided.

Interface with the Host PC

The interface between the host PC and the FPGA is based on a *EZ-USB FX2 USB* module from *Cypress* (*CY7C68013-100AC*), which supports USB 2.0 connections in high-speed mode (480Mbps). It incorporates an 8051 microcontroller, although only for configuration and management purposes, having no interference in the actual transmission. From the FPGA point of view the device is seen as a FIFO memory which the firmware accesses for configuration and management. The device has been configured to use bulk mode transmission, providing high bandwidth and error correction.

On the host computer side, the software has been developed for the Linux operating system. The connection is established and controlled using the functions available in the *libusb* library. The data captured by the FPGA is stored in a format compatible with the *libpcap* file format [**?**] used by *Wireshark*. Since the data is transmitted via USB in the same format as it is stored in the FPGA memory, the processing required on the PC side consists only in writing the received data into a file. Then, the captured data file can be opened using standard tools, in particular *Wireshark* (figure 3) and taking advantage of its exporting function, new files can be created in CSV (Coma Separated Values) or other arbitrary text format. These files can be opened with data processing applications (e.g. *Matlab*, *Octave* or *Scilab*). In the scope of this work a few *Octave* scripts have been developed to automate the extraction of commonly needed statistical data and graphs. Some examples can be seen in tables I and II or in figures 5, 6 and 7.

IV. Modeling and Synthesis

The sniffer hardware components were modeled using the VHDL hardware description language. The model was synthesized and implemented in a Xilinx XC3S1500 Spartan-3 low cost FPGA [**?**]. The synthesis final report is shown on figure 4.

```
Final Synthesis Report
=================================================
Selected Device :xc3s1500-4fg676
Device utilization summary:
Nr. of Slices: 2789 out of 13312 (20%)
Nr. of Slice Flip-Flops: 3587 out of 26624 (13%)
Nr. of 4 input LUTs: 4174 out of 26624 (15%)
Nr. of Bonded IOBs: 54 out of 487 (11%)
Nr. of Global CLKs: 7 out of 8 (87%)
Nr. of Block RAMs: 32 out of 32 (100%)
Timing Summary:
Speed Grade: -4
Min. period: 17.521ns (Max. Frequency: 57.075MHz)
Min. input arrival time before clock: 9.337ns
Max. output required time after clock: 9.922ns
Maximum combinational path delay: 12.338ns
```

Fig. 4 - Sniffer hardware synthesis report.

The implementation can operate up to 57Mhz and occupies 2789 FPGA logic cells, corresponding approximately to 20% of the available slices. Please note that a 100 Mbit/sec *Ethernet* network requires the MAC circuits operating at 25 MHz. To implement the sniffer in ASIC technology about 2,120,985 logic gates are needed. It should also be noted that all available block RAMs (576 Kbits) are in use for implementing the *Control* and *Data FIFOs*.

V. Test and Performance Analysis

In order to evaluate the performance of the developed tool, a set of tests have been carried out. The tests intended mainly to examine the temporal precision of the tool, the interference of the tool in the network and the capacity of the USB connection. In this context, a comparison was done between the results obtained using a software-only based sniffer application (*Wireshark*) and sniffer using the hardware built.

Interference in the Network

To analyze the interference of the tool in the network, two computers were connected. One was responsible for generating traffic, using the *packETH* traffic generator, and the other PC captured the traffic using *Wireshark*. For each test 500000 packets, with different periods and message lengths, were generated. The test has been carried out both with a direct connection between the PCs and with the TAP/FPGA tool. In the second scenario the traffic was captured both with the FPGA and by *Wireshark*, simultaneously.

The obtained results show that in the software based tool some packets are dropped without the user being alerted about it. On the other hand, the number of packets captured by the hardware tool is exactly coincident with the number of packets generated at the

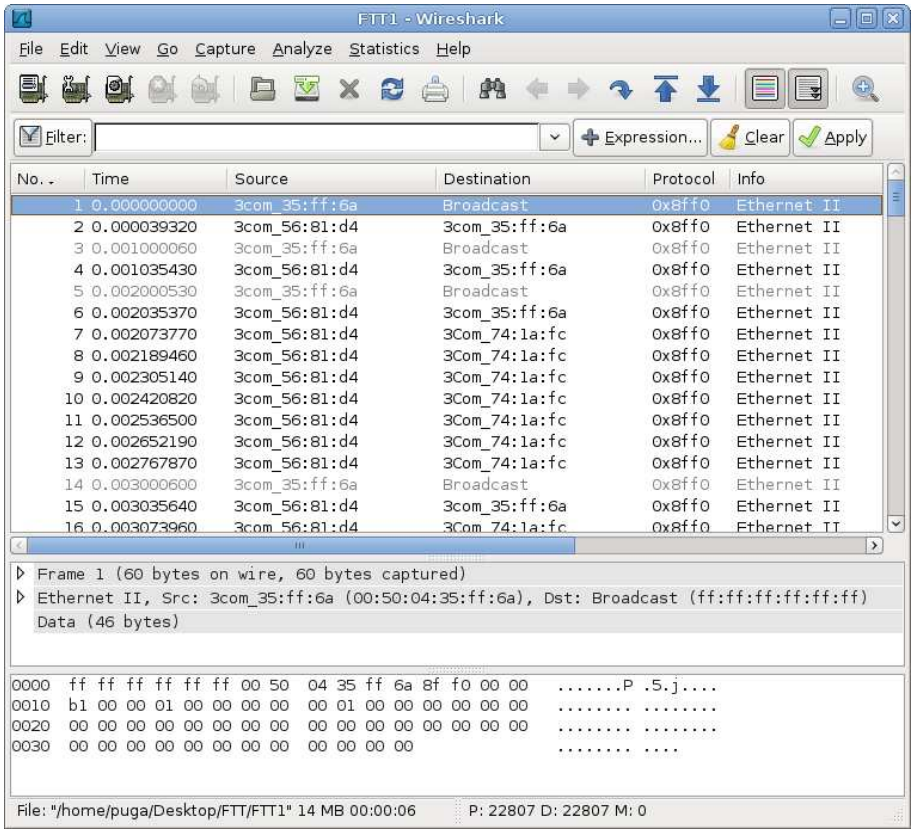


Fig. 3 - Appearance of a capture analyzed with *Wireshark*.

source node. Apart from that, it is also possible to observe that the percentage of dropped packets remains the same and it is independent of the presence of the hardware based tool. For this reason it is possible to infer that attenuation caused by TAP has no relevant impact in logical behavior of connections used in laboratory. The connections used have about one dozen of meters, however, the usage of longer links or worse quality cables can in some situations be problematic.

A closer look also allows to observe that packet losses using software applications increases with the increase of the length of messages and with the decrease of the time between messages, thus showing a dependency with the link bandwidth.

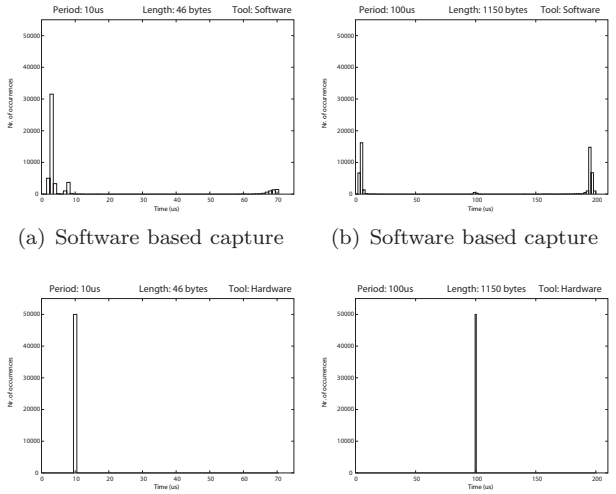
Capacity of the USB Connection

Another parameter that is important to evaluate is the capacity of USB connection. Theoretically, the USB protocol has enough bandwidth to support the *Fast Ethernet* data, even for full-duplex scenarios. Nevertheless, in practice some problems were experienced. The problems have been track down to the writing of the received data into the hard disk of the host PC. This operation is slow and inhibits the reception of data sent by FPGA. Due to this blocking time, the FPGA memory can become full, thus breaking the capture process. To ensure that this is the cause of the problem, a situation in which the data was not stored but only analyzed to detect any incoherency, was tested. In this case the capture was never interrupted.

Temporal Performance

To evaluate the temporal performance of the developed tool it is important to know the exact moment in which each packet is captured. For this reason, a FPGA was used to generate traffic. It is important to note that, according to the TEMAC specifications, it can have a latency variation up to 120ns (due to the synchronization of distinct clock domains). However, this value is much lower than the associated jitter of any personal computer. Once again *Wireshark* and the developed tool were used to capture packets and many tests were done using different period values and messages of different lengths. Then, the captured data was processed using *Octave* based scripts and some tables and graphs were generated.

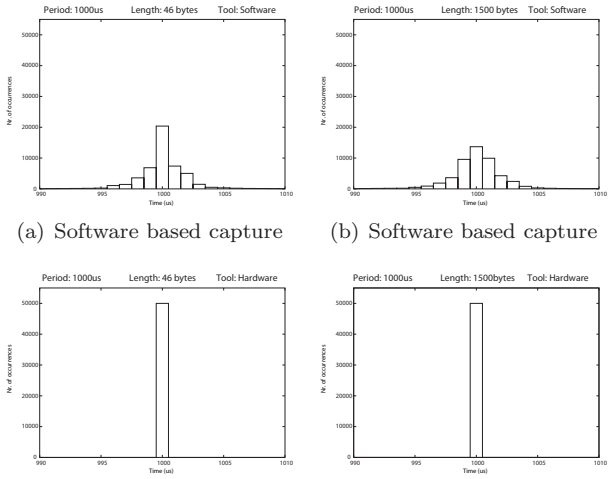
Tables I and II present the results of a software and hardware based capture, respectively. The results of three tests were represented graphically, with the x-axis representing the interval of time between two consecutive messages and the y-axis representing the number of occurrences of each interval. The FPGA-based tool produced similar results in all situations. On the other hand, the usage of software-only tools originated two distinct situations according to the time interval between two consecutive messages. When the inter-arrival time between two consecutive packets falls below 200μs messages start to be dropped. Furthermore, many messages accumulated in the NIC (Network In-



(a) Software based capture (b) Software based capture

(c) Hardware based capture (d) Hardware based capture

Fig. 5 - Results of hardware *versus* software timestamping.



(a) Software based capture (b) Software based capture

(c) Hardware based capture (d) Hardware based capture

Fig. 6 - Impact of the length of messages in the performance of the capture tools.

Period (μs)	Length ($Bytes$)	Max (μs)	Min (μs)	Avg (μs)	Std (μs)	Absolute Jitter (μs)
10	46	48872	2	15.39	335.62	48870
10	80	67542	2	13.78	373.52	67540
20	46	24786	2	21.76	158.31	24784
20	200	54927	2	22.78	276.42	54925
50	46	39159	2	50.90	188.11	39157
50	200	45161	2	56.32	283.38	45159
50	550	25654	2	56.43	249.81	25652
100	46	71817	2	101.46	334.54	71815
100	550	40805	2	105.51	280.21	40803
100	1150	117525	3	112.11	767.69	117522
200	46	280	121	200.03	2.37	159
200	1500	57007	80	202.78	291.44	56927
500	46	589	415	500.04	2.98	174
500	1500	53502	362	501.10	237.11	53140
1000	46	2969	6	1000.06	15.12	2963
1000	1500	50003	722	1001.82	272.29	49281

TABLE I

PERFORMANCE OF THE SOFTWARE-BASED TOOLS.

Period (μs)	Length ($Bytes$)	Max (μs)	Min (μs)	Avg (μs)	Std (μs)	Absolute Jitter (μs)
10	46	10.09	9.99	10.02	0.035	0.10
10	80	10.09	9.99	10.02	0.035	0.10
20	46	20.09	20.00	20.02	0.034	0.09
20	200	20.09	19.99	20.02	0.034	0.10
50	46	50.09	50.00	50.02	0.034	0.09
50	200	50.09	50.00	50.02	0.034	0.09
50	550	50.09	50.00	50.02	0.034	0.09
100	46	100.09	100.00	100.02	0.033	0.09
100	550	100.09	100.00	100.02	0.033	0.09
100	1150	100.09	100.00	100.02	0.033	0.09
200	46	200.09	200.00	200.02	0.031	0.09
200	1500	200.09	200.00	200.02	0.031	0.09
500	46	500.09	500.00	500.02	0.022	0.09
500	1500	500.09	500.00	500.02	0.022	0.09
1000	46	1000.02	999.93	1000.01	0.022	0.09
1000	1500	1000.02	999.93	1000.01	0.022	0.09

TABLE II

PERFORMANCE OF THE DEVELOPED SNIFFER.

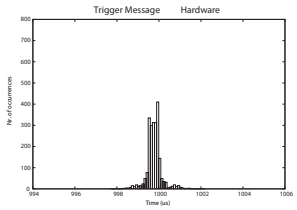
terface Card) internal memory, being transferred and processed by the device-driver in block, together. In this case, the difference between timestamps of messages is related with the processing time of each message and not with the receiving instant, as desired. In spite of being expected that the average time would be the same as the period of messages, in some situations messages are drooped and this value is obviously higher. This situation is represented in figures 5(a) and 5(b). To compare the results obtained with the results expected, in figures 5(c) and 5(d) the same situation is shown but using the FPGA-based tool.

Another situation, with a period higher than 200μs, in which the software based tools performs better is represented in figure 6. It should be remarked that the length of messages has impact in the performance of software tools, with the jitter value increasing when the frame size increases. In figure 6, messages with same period but different lengths were captured. Nevertheless, the performance of hardware based tools is still much better.

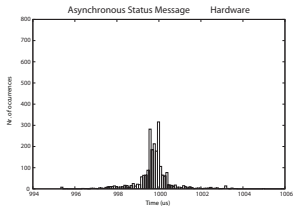
Finally, the FPGA-based sniffer was also successfully tested in a specific situation involving a real-time communication protocol, incidentally the FTT-SE protocol [**?**] developed at the IEETA/LSE laboratory, where this work was hosted.

Periodic messages were captured and analyzed (figure 7) and it was possible to observe that jitter value obtained by using software-based tools can be up to 20 times the value obtained by using dedicated hardware. It is also possible to observe that software based tools have a maximum resolution of 1μs. More than that, it is important to refer that the developed tool can capture messages in full-duplex connections, being able to correlate the master and slave messages, something than is out of reach of the software-based tools, which can only analyze half-duplex connections.

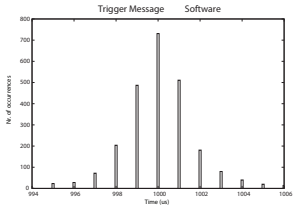
The results obtained (resolution of 10ns with a maximum error of 100ns) are very similar to the specification of some comercial tools. In the same conditions, AE5501 can achieve a latency resolution of 100ns with a maximum error of 300ns. Thus, most of the times these equipments are characterized in a pessimistic way and results correspond to the worst case situation.



(a) Trigger Message



(b) Async. Status Message



(c) Trigger Message

Fig. 7 - Jitter of periodic messages using FTT-SE protocol.

VI. Conclusion and Future Work

This paper has presented an hardware-assisted *Ethernet* protocol analyzer tool, with capacity to analyze real-time ethernet networks. It has been implemented using FPGA technology and the interface with the network is done using a passive *Ethernet* TAP. The FPGA itself has capacity to capture traffic in full-duplex connections. Timestamping and control information is associated to captured packets at the hardware level, thus permitting an high degree of resolution and accuracy. The captured data is afterwards sent to an host PC for further analysis, via an USB link. Data is saved in a open file format, that can be read by standard applications. This approach permits taking full advantage of all the functionalities supported by applications such as *Wireshark*. Some mechanisms to export data into other formats were also created. Captured traffic can also be analyzed using calculation tools such as *Matlab*/*Octave*/*Scilab* or spreadsheets. A set of tools for automating the extraction of common statistical data and graphs have also been developed.

Concerning the accuracy of the tool, it is possible to infer that the timestamps have a precision of $100ns$ and a resolution of $10ns$, independent of the traffic load, a value similar to the ones specified by commercially available hardware-based tools, which have costs orders of magnitude higher. On the contrary, software-based tools had a performance extremely dependent of the traffic load, with precisions that can be worse by several orders of magnitude, turning out these tools completely useless for carrying out measurements related with real-time *Ethernet* protocols.

Despite the positive results achieved, there are some points that can be improved and will be addressed in the future. The issue that can be pointed out to the current implementation is related with USB transmission that can limit the amount of captured data and the duration of capture. One possibility to alleviate the blocking time imposed by the host computer during the storage of the received data is increasing the memory capacity. It can be done using a FPGA with more block RAMs available, such as the Virtex-4 (XC4VFX140) that has about 20 times more capacity. Alternatively, an external memory can be used, e.g. the usage of a 64Mbytes DDR memory allows an increase of the capacity up to 512x. Nevertheless these solutions are only valid if the average *Ethernet* transmission rate is lower than the USB transmission rate. Otherwise the USB connection has to be replaced. Some alternatives that can be considered are *Gigabit Ethernet* (1000Mbps), FireWire (800Mbps) or a SATA connection (Serial Advanced Technology Attachment) (2400Mbps), being the later a potential solution for connecting to a host PC a *Gigabit Ethernet* sniffer with full-duplex traffic capture capabilities.

Another point that can be improved is the user interface, which should be more friendly. In this context it would be interesting the development of a plug-in to *Wireshark* to support and control all the functionalities of the developed tool. Namely it would be interesting to permit controlling the beginning of a capture, show the number of packets captured, the duration and then terminate the capture. It should also have a field to configure the maximum number of bytes captured from each message, avoiding having to synthesize the tool when this value is changed. In addition it would also be interesting to add new dissectors to *Wireshark* for supporting RTE protocols, in particular the FTT-SE protocol. It would also be desirable to have the possibility of generating graphs and tables inside the *Wireshark*, without having to explicitly export the data to other tools such as *Octave*.

Due to the constant evolution of the real-time protocols and the hardware capacities it should be considered the possibility of the future improvement of the timestamping precision. Currently, the main limitation is imposed by the latency variation or jitter introduced by the *Ethernet MAC* core used. A possible solution consists in the implementation of a timestamping unit working in parallel with the TEMAC. In this solution two parallel paths will process the received *Ethernet* frame: one will be responsible for data reception and validation and the other for timestamping based on the signals directly provided by the *Ethernet* PHY. Consequently, the timestamping will be performed closer to the physical layer, before the frame passes through the TEMAC, decreasing the delay between the actual frame reception at the network interface and its timestamping.

VII. Acknowledgments

The authors would like to thank Xilinx Inc. for the donation of the Tri-mode Ethernet MAC soft IP core, as well as ISE and ChipScope Pro design tools.

References

[1] "Wireshark", Jan. 2008, [Online] URL: http://www.wireshark.org/.

[2] A. Depari, P. Ferrari, A. Flammini, D. Marioli, and A. Ta-

roni, "Multi-probe measurement instrument for real-time Ethernet networks", *Proc. of IEEE WFCS2006*, pp. 313–320, June 2006.

[3] Test Equipment Connection, "Network analyzer : Anritsu md1231a", 2008, [Online] URL: http://www.testequipmentconnection.com/products/37104.

[4] Yokogawa, "Traffic Tester Mini AE5501", Feb. 2007, [online] URL: http://www.yokogawa.com/tm/data/ae5501/tm-ae5501_01.htm.

[5] Net Optics, "Network taps", Mar. 2008, [Online] URL: http://www. netoptics.com/products/default.asp.

[6] Inc Xilinx, "Tri-mode ethernet media access controller (TEMAC)", Mar. 2008, [Online] URL: http://www.xilinx.com/products/ipcenter/TEMAC.htm.

[7] "Libpcap file format", [Online] URL: http://wiki.wireshark.org/Development /LibpcapFileFormat, Dec. 2007.

[8] Inc Xilinx, "Xilinx webpage", Jan. 2008, [Online] URL: http:// www.xilinx.com.

[9] Ricardo Marau, Paulo Pedreiras, and Luís Almeida, "Enhanced ethernet switching for flexible hard real-time communication", *Proceedings of the 5th International Workshop on Real Time Networks (RTN'06)*, pp. 45–48, July 2006.

[10] P. Pedreiras, P. Gai, L. Almeida, and G.C. Buttazzo, "FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems", *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 3, pp. 162–172, Aug. 2005.

[11] Paulo Pedreiras, Luís Almeida, and Paolo Gai, "The ftt-ethernet protocol: Merging flexibility, timeliness and efficiency", *Proceedings of the 14 th Euromicro Conference on Real-Time Systems (ECRTS.02)*, pp. 134–142, 2002.

[12] J-D Decotignie, "A perspective on Ethernet-TCP/IP as a fieldbus", *Proceedings of the 4th FeT'2001 - International Conference on Fieldbus Systems and their Applications*, pp. 138–143, Nov. 2001.

[13] Hans Weibel and Dominic Béchaz, "IEEE 1588 Implementation and Performance of Time Stamping Techniques", *Conference on IEEE 1588*, Sept. 2004.

[14] P. Arlos and M. Fiedler, "A Comparation of Measurement Accuracy for DAG, Tcpdump and Windump", 2005, Blekinge Institute of Technology (Sweden) [online] www.its.bth.se/staff/pca.

[15] NetScout, "Sniffer products", 2008, [Online] URL: http://www.netscout.com/products/.