Hardware/Software Implementation Factors Influencing Ethernet Latency

Tomás P. Corrêa

Luís Almeida*

Emilio Bueno Peña

*CISTER Research Centre

CISTER-TR-181115

2018/07/18

Hardware/Software Implementation Factors Influencing Ethernet Latency

Tomás P. Corrêa, Luís Almeida*, Emilio Bueno Peña

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: lda@fe.up.pt

http://www.cister.isep.ipp.pt

Abstract

Minimum Cycle Time is a common performance indicator adopted to compare Real-Time Ethernet protocols. Though serving its purpose, Minimum Cycle Time excludes the delays inside the sending and receiving nodes, so it is insufficient to estimate the end-to-end latency. In this work, we describe some implementation possibilities of an Ethernet node in a System-on-Chip and present measurements of the delay to send/receive packets from/to the application layer. We chose different points in the software to make the measurement, so the results cover more use-cases. We found the Ethernet Lite Media Access Controller (MAC) to be faster than the hard MAC (GEM) and the Lightweight IP stack to add less than 2.2 ;cs. Finally, we show how a hardware accelerator can reduce the delay of high-priority packets by 1.4 ;cs.

Hardware/Software Implementation Factors Influencing Ethernet Latency

Tomás P. Corrêa
Instituto de Telecomunicações, Portugal
Universidad de Alcalá, Spain
E-mail: tomaspcorrea@gmail.com

Luis Almeida
CISTER / IT - FEUP
Universidade do Porto, Portugal
E-mail: lda@fe.up.pt

Emilio Bueno Peña
Departamento de Electrónica
Universidad de Alcalá, Spain
E-mail: emilio.bueno@uah.es

***Abstract*—Minimum Cycle Time is a common performance indicator adopted to compare Real-Time Ethernet protocols. Though serving its purpose, Minimum Cycle Time excludes the delays inside the sending and receiving nodes, so it is insufficient to estimate the end-to-end latency. In this work, we describe some implementation possibilities of an Ethernet node in a System-on-Chip and present measurements of the delay to send/receive packets from/to the application layer. We chose different points in the software to make the measurement, so the results cover more use-cases. We found the Ethernet Lite Media Access Controller (MAC) to be faster than the hard MAC (GEM) and the Lightweight IP stack to add less than 2.2 µs. Finally, we show how a hardware accelerator can reduce the delay of high-priority packets by 1.4 µs.**

***Index Terms*—Industrial communication systems, real-time Ethernet, end-to-end latency.**

I. Introduction

Industrial Internet of Things and Industry 4.0 are hot topics in the academia and industrial world nowadays, but sometimes industry is slow in adopting new technologies. Ethernet, for instance, already supports transfer rates up to 100 Gbps, but the Real-Time Ethernet (RTE) deployed in Factory Automation are mainly based on Fast Ethernet (100 Mbps), with CC-Link IE being an exception [1].

Since the early 2000s, several RTE protocols emerged for Factory Automation, e.g. EtherCAT, POWERLINK, CC-link IE, Sercos III, and PROFINET. To help matching application requirements with the network capabilities, the standard IEC 61784-2 defines several performance indicators, such as *throughput RTE*, *non-RTE bandwidth*, and *delivery time* [2]. Though not included in the standard, *minimum cycle time*, the minimum time between two consecutive data updates on a node, is also a popular performance indicator [3], [4], [5], [6], [7].

However, the minimum cycle time typically ignores the latency inside the nodes. In critical, high performance applications, like drives [8] or the control of Modular Multilevel Converters [9], [10], not only the cycle time has to be low but also the end-to-end latency must be as short as possible, because it adds to the loop delay and degrades the control performance [11].

In our research, we found few information on the time the nodes need to make incoming data available to the application layer or the time necessary to effectively start transmitting data through the link. As RTE inexorably moves to higher data rates, these delays will become more relevant, so it is important to know their magnitude and what are the main aspects influencing it. *Orfanus et al.* [12] list some strategies they adopted to optimize the implementation of an EtherCAT master, such as zero-copy buffers, memory pre-allocation, and mapping of application variables directly onto EtherCAT telegrams, but they omit timing figures. We wanted to characterize the delay inside the nodes and understand how both hardware and software implementations affect it. For that, we run experiments on Xilinx Zynq System-on-Chip (SoC), adopting different MAC implementations, data copy strategies and using or not the UDP/IP protocol stack. Curiously, we observed intriguing results in which an FPGA implemented MAC option (Ethernet Lite) outperformed a hardware-based one (Gigabit Ethernet MAC) in several scenarios. The results show not only how to reduce the delay in this specific device but also give hints on the key aspects to observe when choosing a hardware platform.

This work is organized as follows: in Section II we describe the options available in the Zynq SoC and introduce the protocol stack employed; in Section III we show measurement results of the delay to receive a packet till the data reaches the application layer, and the delay to send a packet, from the application layer command until the link becomes active. To make the results more useful, we present also delays from intermediate points, so they cover several use-cases and designs strategies.

II. Implementation details and possibilities

The Zynq System-on-Chip combines a single or dual core ARM processor with an FPGA fabric. It includes up to two hard Media Access Controller (MAC), named Gigabit Ethernet Mac (GEM). The GEM interface with the Physical Layer (PHY) is either RGMII or GMII and with the core and memory is a 32-bit AHP bus. A Direct Memory Access (DMA) engine, operating at a maximum frequency of 150 MHz (IC speed grade -2) [13], controls the flow of data to/from the main memory.

Besides the GEM, Xilinx provides two types of MAC as Intellectual Properties (IP) for synthesis and implementation inside the FPGA: the Ethernet Lite, free of charge but limited to 100 Mbps, and the soft Tri-Speed MAC (TEMAC), supporting up to 2.5 Gbps.

The Ethernet Lite soft MAC connects to the main memory via either AXI4-lite or AXI4 slave interfaces. The former does single transactions, only, what limits performance, and has a maximum clock of 150 MHz. The latter supports burst transactions of 256 words with a single addressing phase [14]. The AXI Master, though, does not use data bursts when running the demo echo server application, so the AXI4 performance is similar to AXI4-lite, with a minor gain due to the higher clock rate (up to 180 MHz). In both cases, the bandwidth can be increased by configuring the Ethernet Lite memory address region as *Device Memory*[1] instead of the standard *strong-ordered* (see [15], chapter 3): the number of clocks between valid write responses reduces from 18 to 3 and between read transactions from 17 to 5, according to our measurements. The impact in the delay is large: 35 µs against 7.73 µs to receive an 1024 B UDP packet. We tried using DMA to accelerate the data transfer, but its rate was the same as before and the total delay increased, due to the time spent configuring and triggering the DMA.

The other type of MAC available, TEMAC, supports AXI4-stream. AXI4-stream removes the addressing phase altogether [14] and, combined with a AXI DMA, can connect to the main memory using either of the high bandwidth interfaces AXI ACP or AXI HP. In both cases, the maximum clock is also 180 MHz, but the bus width is 64-bits, so one can expect twice the data transfer rate. However, experiments with this MAC are on-going work and are not included in this paper.

A. Lightweight IP

Lightweight IP (lwip) is an open-source TCP/IP stack developed from the beginning to be modular and use little RAM, so even small processors are able to run it. Adam Dunkels started lwip at the Swedish Institute of Computer Science in the early 2000s [16] and today a worldwide group of programmers maintains and further develops it. Several processor manufacturers ported it to their devices and Xilinx is no exception. During our work, we found lwip to be a good starting basis, not only due to the several protocol stacks themselves but also because it implements the drivers for the MAC and the routine to configure the PHY.

Lwip main feature to reduce RAM footprint is avoid copying data as it moves up and down the protocol layers. For that, it defines a data structure called PBUF that can be allocated dynamically, but to improve performance, lwip pre-allocates PBUFs for incoming packets, only trimming the size according to the amount of incoming data. The PBUFs make lwip efficient, as the experimental results show, even though it was not developed with real-time applications in mind.

When receiving a packet, the initial steps differ according to the MAC type: the Ethernet Lite MAC checks data integrity and immediately calls the interrupt service routine (ISR); the GEM and TEMAC first transfer the packet to an intermediate position inside the main memory and, upon completion, calls the ISR. Then, in all cases, the MAC driver identifies the origin of the interrupt (send, receive or error) and calls the corresponding handler. The receive handler copies the data to a PBUF structure, puts it into the receive queue, and exits (see Fig. 1a). The processing of the packet then happens outside the interrupt context, by pooling the receive queue for new data in a routine inside the infinite loop (Fig. 1b).

When sending a packet, the following processing sequence takes place (Fig. 2): after the user moves data to a PBUF and calls the routine *UDP_sendto()*, the stack adds the UDP header, selects an interface to send from, includes the IP and Ethernet headers, resolves the destination MAC address based on the destination IP address, and copies the packet to an intermediate memory location. The last steps depend on the MAC type: if using Ethernet Lite, the driver copies the packet to the MAC buffer and handles control to the MAC hardware to start transmission; with the GEM or TEMAC, the driver passes the control to the MAC hardware that transfers data to the internal buffer using DMA and starts transmission.

From the description above, the reader could identify that Xilinx implementation of lwip does copies that could be avoid if the user needs to reduce the delay. In the next section, where we show experimental results, we modify the implementation to assess the influence of these options in the total sending and receiving delays, as well as the performance of the different MACs.

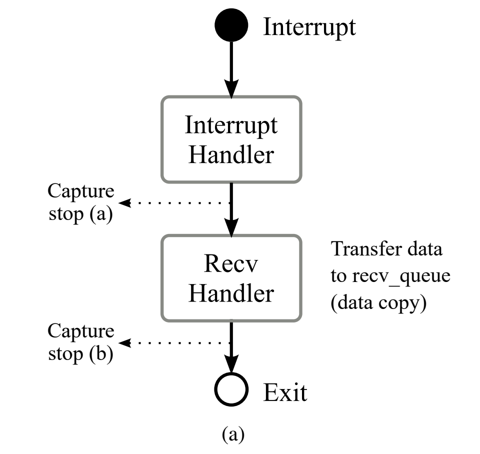
III. Experimental results

The employed experimental platform was a 7020 Zynq-based board with a Microchip Fast Ethernet PHY connected to a host PC. The host PC sends packets with total length of 64 bytes, 256 bytes and 1024 bytes, without considering preamble, start of frame delimiter and frame check sequence. The embedded processor runs the echo server demo application configured to echo UDP packets, i.e. it sends back packets received in a given port.

To measure the different delays, we designed a capture unit in VHDL to count the number of clock cycles between the positive edges of the start and stop ports. The time resolution was 10 ns. When measuring the incoming delays, the inverted RX_DV signal triggered the capture unit and a software-controlled output stopped it. When measuring outgoing delays, the software-controlled output triggered the start and the positive edge of the TX_EN stopped it. We verified the capture unit measurement by connecting the start and stop signals to an oscilloscope. For every test run, the program logged 512 measurements and sent them to the host PC for processing.

For incoming packets, we measured the time delay between when the MAC finished receiving a packet and three points in the software: when the processor entered the interrupt service routine (Fig. 3); when the processor finished copying the data to the PBUF structure and queued it (Fig. 4); and when entering the user defined UDP receive callback (Fig. 5).

[1]Unlike strong-ordered memory, a write to device memory is allowed to complete before it reaches the peripheral accessed by the write.



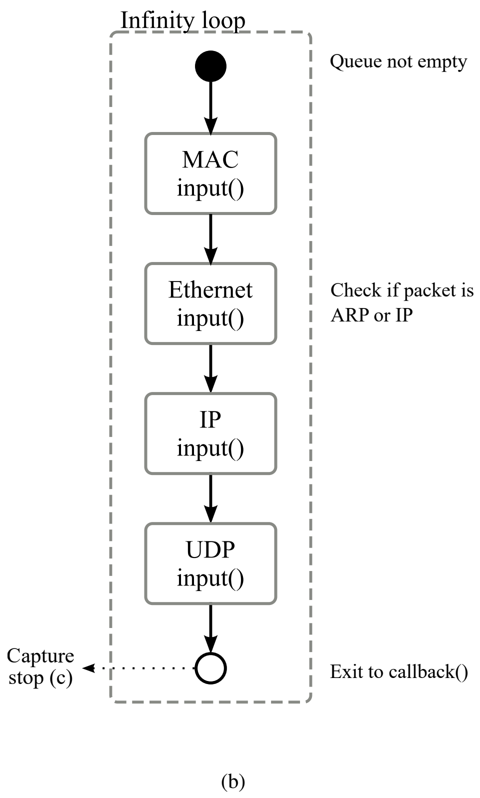


Figure 1: Incoming packet processing.

For outgoing packets, we measured the time delay from three points in the software to when the MAC starts sending data to the PHY. They are: when the user calls the *UDP_sendto()* routine (Fig. 6); when the processor starts copying the packet from the original PBUF (Fig. 7); and when the software triggers the MAC to send the packet (Fig. 8). Table I summarizes the delays obtained in terms of mean value and standard deviation σ.

A. Results Discussion

The results show a shorter latency to enter the ISR when using Ethernet Lite (Fig. 3). This MAC requests an interrupt just after the packet is received, because it saves the data

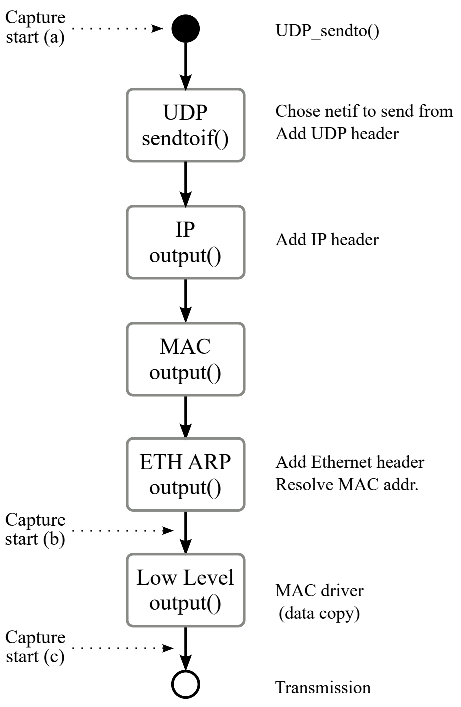
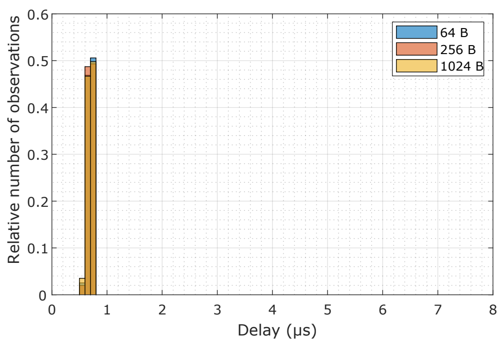
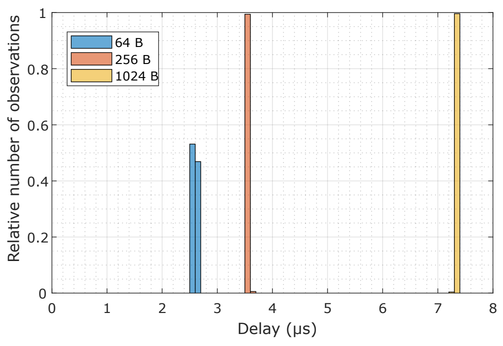


Figure 2: Outgoing packet sending.

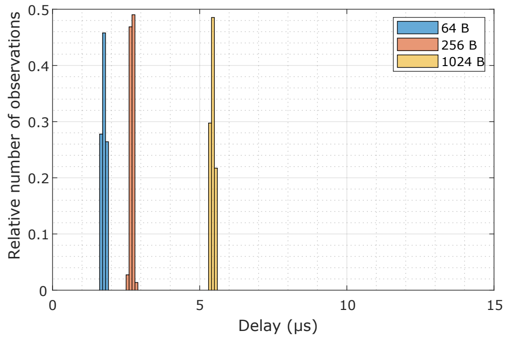


(a) Ethernet Lite.

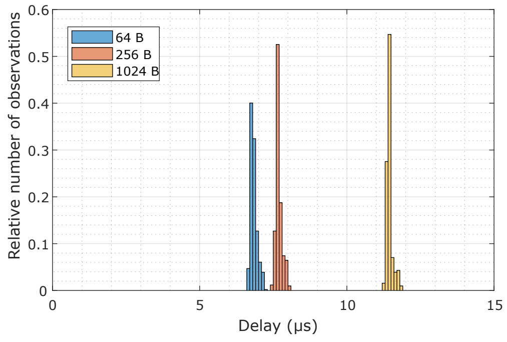


(b) GEM.

Figure 3: Incoming packet: delay to enter ISR after receiving packet (cap. stop (a) in Fig. 1). In Fig. 3a, the results are overlapping.



(a) Ethernet Lite, device memory.



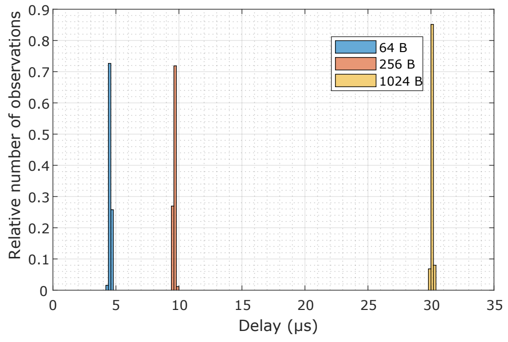
(b) GEM.

Figure 4: Incoming packet: delay to transfer data to PBUF after receiving packet (cap. stop (b) in Fig. 1).

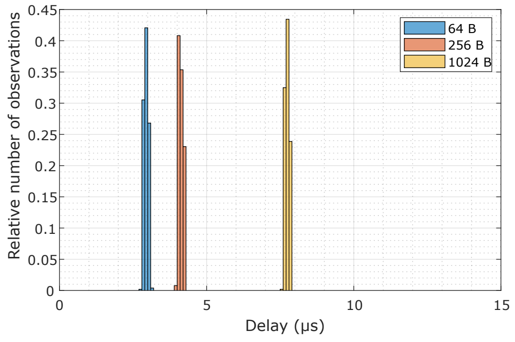
internally and henceforth further data movement needs the participation of the core, i.e., the core utilization for sending and receiving data is the total latency less the ISR one. The delay around 680 ns is expected, because the minimum interrupt latency of the A9 core is 360 ns [17] and the MAC takes around 150 ns to flag the interrupt. In contrast, the GEM has longer latency, depending on the packet size, because the DMA transfers the packet from the MAC FIFO to the main memory before flagging the interrupt.

When triggering the MAC to send a packet, again the Ethernet Lite shows a more predictable behavior (Fig. 8), for the same reason (absence of DMA transfer). In applications where accurate timing and low jitter are desired, e.g. when implementing a master stack or using time-triggered protocols, this is a considerable advantage of the soft MAC and the jitter is fairly low if considering the pure software implementation (Table I).

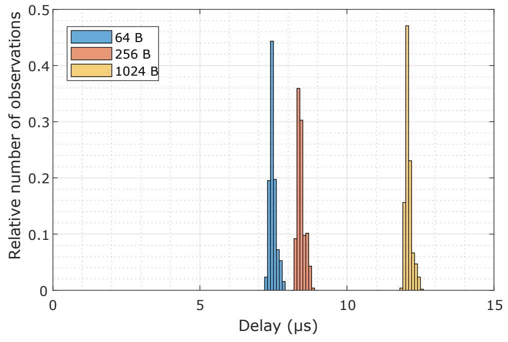
As explained in Section II, the configuration of Ethernet Lite memory as *device* has a significant impact in the delays, both for receiving (Fig. 5) and sending data (Fig. 6). The results show Ethernet Lite outperforming GEM independently of the packet size and direction when using delay as figure of merit. This result is surprising, as one would expect a dedicated hard peripheral, integrated to the processor, to be faster. The price to pay for Ethernet Lite shorter delays is higher utilization of the CPU.



(a) Ethernet Lite, strong-ordered.



(b) Ethernet Lite, device memory.

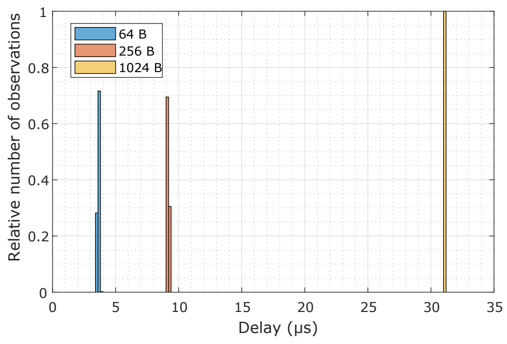


(c) GEM.

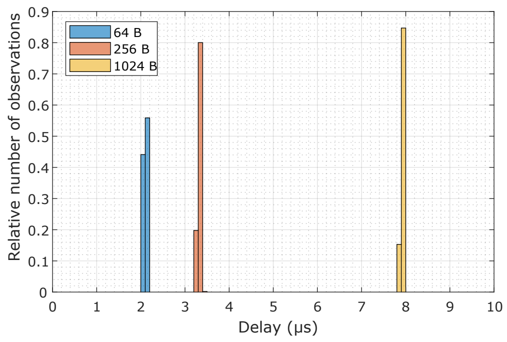
Figure 5: Incoming packet: delay to enter UDP callback after receiving packet (cap. stop (c) in Fig. 1).

The experiments conducted allow quantifying the costs of using IP and UDP protocols by computing the difference between the values after copying the packet to the PBUF and the UDP callback (see Table I). Besides the additional payload to accommodate the protocol headers, less than 2.2 μs are necessary to process the packet and call the user defined routine.

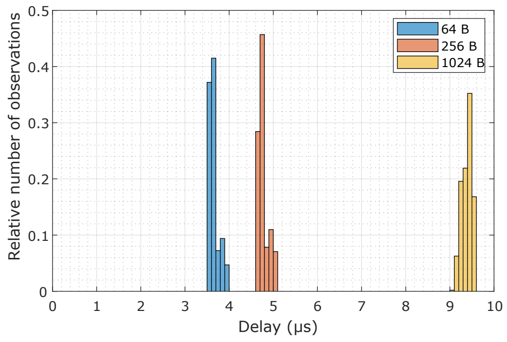
When the traffic is mixed, certain packets have higher priority and must be processed faster. For this, we can implement a packet identifier accelerator in the FPGA that will read all incoming packets and check their headers for some predefined characteristics, like being of IP type, being targeted to the node IP address, being an UDP packet with a given



(a) Ethernet Lite, strong-ordered.



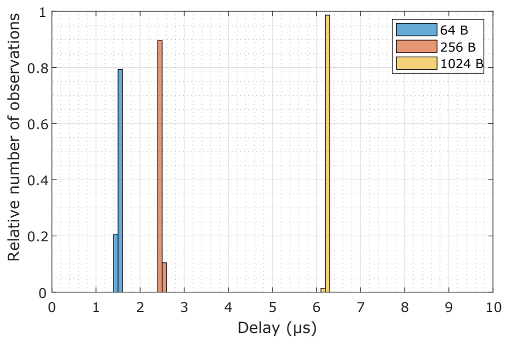
(b) Ethernet Lite, device memory.



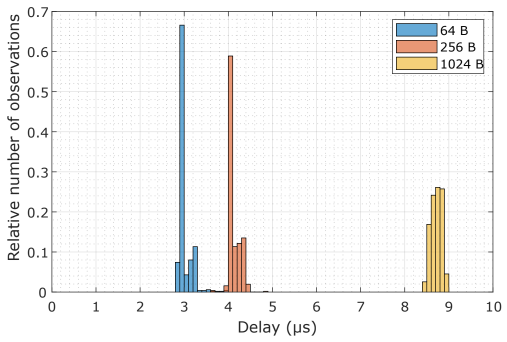
(c) GEM.

Figure 6: Outgoing packet: delay to start sending packet after UDP command (cap. start (a) in Fig. 2).

port number, etc. The receive handler can verify the register where the accelerator puts its findings and, if all "high priority" characteristics are met, it fast-tracks the payload directly to the application. Adopting this strategy we could reduce the reception delay of 64 Bytes and 256 Bytes to 1.51 µs and 2.85 µs , respectively (Fig. 9). Unexpectedly, the delay of 1024 Byte packets stayed the same, i.e. 7.74 µs. We verified the reason for such result and found that the CPU implements the read transactions over the AXI bus differently, with a slower data transfer when the fast-track strategy is adopted. This slower data transfer counterbalances the "jump" to the application layer as the payload increases.



(a) Ethernet Lite, device memory.



(b) GEM.

Figure 7: Outgoing packet: delay to send packet after starting copying from PBUFs (cap. start (b) in Fig. 2).

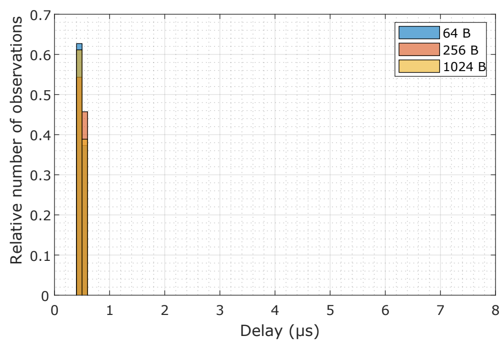
Table I: Incoming and outgoing packet mean delay and deviation, in µs.

	Packet	GEM		EL, DM		EL, SO	
	size	$\overline{x}$	3σ	$\overline{x}$	3σ	$\overline{x}$	3σ
Rx PBUF	64 B	6.84	0.33	1.75	0.17	3.32	0.19
	256 B	7.69	0.32	2.70	0.18	8.22	0.18
	1024 B	11.45	0.31	5.44	0.19	27.80	0.18
Rx UDP	64 B	7.48	0.35	2.95	0.20	4.53	0.23
	256 B	8.44	0.38	4.13	0.21	9.65	0.2
	1024 B	12.09	0.34	7.74	0.20	30.09	0.21
Tx PBUF	64 B	3.01	0.45	1.52	0.05	2.74	0.06
	256 B	4.13	0.38	2.48	0.05	8.02	0.05
	1024 B	8.71	0.35	6.23	0.05	29.05	0.06
Tx UDP	64 B	3.65	0.32	2.10	0.07	3.62	0.08
	256 B	4.77	0.34	3.31	0.07	9.18	0.08
	1024 B	9.38	0.34	7.92	0.08	31.07	0.08

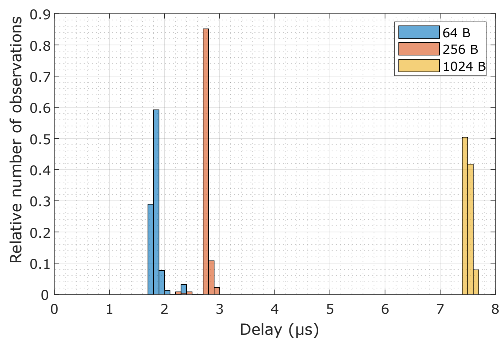
IV. Conclusions

In this work we investigated the delay to receive and send Ethernet packets with different MAC architectures and communication stack implementation details. We run extensive tests and included several results to help the designer estimate more accurately the total end-to-end latency when using Ethernet. The results show effective bandwidth of the MAC interface to the processor memory as a critical aspect. Others are minimizing copying data and pre-allocating memory.

The platform chosen to do this investigation was a Zynq-



(a) Ethernet Lite.



(b) GEM.

Figure 8: Outgoing packet: delay to send packet after triggering the MAC hardware (cap. start (a) in Fig. 2). In Fig. 8a, the results are overlapping.

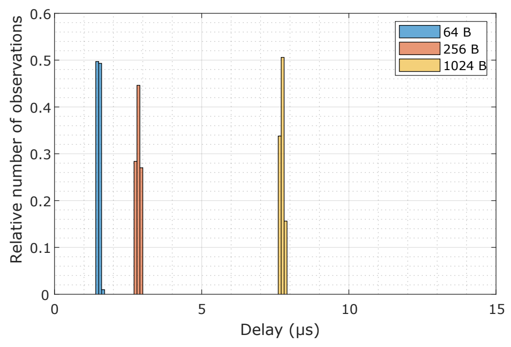


Figure 9: Fast-track for incoming UDP port 1026 packets.

based board using the Lightweight IP stack. First, we describe the different MACs available in the selected platform and the packet processing implementation. Second, we measured the delays between the packets being received by the MAC and selected points in the software to characterize the delay when adopting different strategies. We did the same for outgoing packets, but measuring the delay from selected points in the software to when the MAC effectively starts sending data.

The results show Ethernet Lite to be faster than the hard MAC available in this device, for any payload and both incoming and outgoing packets, when we change the memory model from the default *Strong-Ordered* to *Device Memory*. Short packets can be received and sent using the full UDP/IP stack with 3.3 µs and 2.1 µs, respectively. Additionally, we discussed how to reduce the delay by simplifying the protocol or adopting a packet identifier accelerator to fast-track high priority data to the application, avoiding unnecessary copies and the delay of the protocol stack.

Acknowledgment

This work was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, Brazil, and the Portuguese Government through FCT grant UID/EEA/50008/2013 - Instituto de Telecomunicações).

References

[1] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *IEEE Int. Conf. Emerging Technologies & Factory Automation (ETFA)*, Sep. 2014, pp. 1–8.

[2] L. Winkel, "Real-time ethernet in IEC 61784-2 and IEC 61158 series," in *Proc. IEEE Int. Conf. Industrial Informatics*, Aug. 2006, pp. 246–250.

[3] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial ethernet protocols," in *IEEE Int. Conf. Emerging Technologies & Factory Automation (ETFA)*, Sep. 2007, pp. 17–24.

[4] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," in *IEEE Int. Conf. Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2008, pp. 408–415.

[5] L. Seno, S. Vitturi, and C. Zunino, "Real time ethernet networks evaluation using performance indicators," in *IEEE Int. Conf. Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2009, pp. 1–8.

[6] J. Robert, J.-P. Georges, E. Rondeau, and T. Divoux, "Minimum cycle time analysis of ethernet-based real-time protocols," *International Journal of Computers, Communications and Control*, vol. 7, no. 4, pp. 743–757, 2012.

[7] R. Schlesinger, A. Springer, and T. Sauter, "New approach for improvements and comparison of high performance real-time ethernet networks," in *IEEE Int. Conf. Emerging Technologies & Factory Automation (ETFA)*, Sep. 2014, pp. 1–4.

[8] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto, and C. Zunino, "Real-time Ethernet networks for motion control," *Computer Standards & Interfaces*, vol. 33, no. 5, pp. 465–476, 2011.

[9] L. Mathe, P. D. Burlacu, and R. Teodorescu, "Control of a modular multilevel converter with reduced internal data exchange," *IEEE Trans. on Ind. Informat.*, vol. 13, no. 1, pp. 248–257, 2017.

[10] C. L. Toh and L. E. Norum, "Implementation of high speed control network with fail-safe control and communication cable redundancy in modular multilevel converter," in *European Conf. Power Electronics and Applications (EPE)*. IEEE, 2013, pp. 1–10.

[11] T. P. Corrêa., E. J. Bueno, and F. J. Rodriguez, "Communication network latency compensation in a modular multilevel converter," in *IEEE Energy Conversion Congress and Exposition (ECCE)*. IEEE, 2017.

[12] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, "EtherCAT-based platform for distributed control in high-performance industrial applications," in *IEEE Int. Conf. Emerging Technologies & Factory Automation (ETFA)*, Sep. 2013, pp. 1–8.

[13] Xilinx Inc., *Zynq-7000 All Programmable SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics (DS187)*, Jun. 2017.

[14] ——, *Vivado AXI Reference (UG1037)*, Jul. 2017.

[15] ——, *Zynq-7000 All Programmable SoC: Technical Reference Manual (UG585)*, Dec. 2017.

[16] A. Dunkels, "Full TCP/IP for 8 Bit Architectures," in *Proc. ACM/Usenix Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, San Francisco, May 2003. [Online]. Available: http://dunkels.com/adam/mobisys2003.pdf

[17] JBLopen. Arm cortex-a interrupt latency. Date last accessed Mar 8, 2018. [Online]. Available: https://www.jblopen.com/arm-cortex-a-interrupt-latency/