Cost effective RISC core supporting the Large Sending Offload

For high communication rate up to 100 Gbps

Mohamed Elbeshti[1]
School of IT, Murdoch Uni
Perth, Australia
M.elbeshti@murdoch.edu.au

Mike Dixon
School of IT, Murdoch Uni
Perth, Australia
Mike.dixon@murdoch.edu.au

Terry Koziniec
School of IT, Murdoch Uni
Perth, Australia
T.koziniec@murdoch.edu.au

Abstract— **The Ethernet speed has increased sending and receiving frames from 40 to 100 Gbps after the IEEE P802.3ba released. The industry and academia have focused scaling up the TCP/IP protocol processing for 40-100 Gbps. LSO is a *de facto* standard, which is offloaded to network interface for sending packets up to 10 Gbps. It not clears whether a network interface can support such function for new 40-100 Gbps. The widely use of the hardware-based NIC such as the use of a fully customized logic based network interface can be due to the following reasons; Still it is not clear whether the General Purpose Processor (GPP) can provide the processing required for high-speed line beyond the 10 Gbps. Also, the limit of the GPP's clock in supporting the processing of network interfaces. However, using a RISC core engine for offloading the LSO function can deliver some important features to network interfaces design, such as simplicity, scalability, shorter developing cycle time. In this paper, we have investigated using a specialized RISC core to process the LSO functions for TCP/IP and UDP/IP for high-speed communications rate up to 100 Gbps. To achieve this, we have enhanced the LSO algorithm to scale it to 100 Gbps. A fast DMA is used to support transferring data in the network interface. The LSO processing methodology on the network has presented. In addition, the RISC's performance and data movements for high communication rate up to 100 Gbps have been measured. A 148 MHz RISC core can support the sending-side processing for up to 100 Gbps transmission speed for the TCP/IP and UDP/IP protocol when the MTU is applied (1500 bytes). A DMA with 3759 MHz is required to eliminate the idle cycles while transferring data over the 64-bit local bus.**

Keywords-component; Large Sending Offload (LSO); RISC core; TCP/IP; VHDL simulator; Cycle-accurate performance evaluations.

I. INTRODUCTION

The early stage of development of high-speed network cards used off-the-shelf components and integrated these components on a Printed Circuit Board for the implementation of the network card. It is possible to integrate all the discrete components needed for NI in a single chip [3, 11] in ASIC. Moreover, specialized engine from the sequential machines and desecrates logic to address the need for a protocol reduces the cost of design and improve reliability and performance [11]. Using ASIC to design NI could provide a greater energy efficiency and better integration than programmable-based. However, ASIC also limits flexibility, limits upgradability, and makes NI design tailoring difficult in changing the algorithm of the protocol or supporting a new version of protocols. The wide use of the hardware-based NIC, such as the use of a fully customized logic-based network interface, can be expected for the following reasons: There is no clear indication whether the GPP speed provides adequate processing to deal with protocol functions such as the LSO and the data transfer rate, such as 40Gbps, or 100Gbps. The new trend of designing the network interface is to have all the network interface functions implemented in a single chip. Also, use of commercially available GPP as a core engine in an NI is expensive and difficult to integrate within the network interface chip. Furthermore, the size of these embedded cores is large enough to be accommodated in the NI chip, since it has not been designed for the NI's functions.

The recent advances in the area of CAD tools and Hardware Description Languages (HDL) have made the design of embedded processors for the performance of certain functions is possible. Furthermore, with HDL, it is possible to adapt these processors in order to comply with the functions that they are designed for. System-on-chip technology has enhanced the possibility of integrating the hardware blocks required in the NI and the GPP to be carried on one chip [12].

Many cost effective embedded cores have become available and can be ported to an ENI chip. However, these processors are not optimized for LSO. Since these processors are designed supporting general functions, such as the control unit has to support general functions, complex instructions and long and variable execution time. These GPP also has a large number of registers to accommodate all the possible use. These features of the GPP might not need in NI. These advances in the GPP have directed this research in investigating the use of specialized RISC embedded cores in the high-speed scalable NI design.

The rest of this paper is organized as follows: Section 2 discusses the sending side processing. In section 3, the sending side structure in the ENI model. The processing anaylsis presents in section 4. The core design has been highlighted in section 5. The VHDL-based simulation results are discussed in the next section. Conclusion is in the last section.

II. NI CONSIDERATION

In designing the NI, we have avoided using multiprocessing cores as processing cores at the NIC to serve a single function, such as the ones developed at Rice University and Purdue University which have proposed strengthening the network card with six processors to enable it to perform the 10 Gbps [2, 6]. The idea behind the multiprocessor is to divide the processing required for each incoming or outgoing packet. A 166 MHz controller with six processors can achieve 99 percent of theoretical throughput of 10Gbps. The designers Hyong, Vijay and Scott developed the Tigo programmable NI which was released in 1997 [5]. This depends on two 88 MHz MIPS R4000-based processors for the completion of data processing. Both processors perform either inbound or outbound functions. Despite these approaches achieving the goal of these networks, there are several concerns. The complexity of these proposed models is in the sharing of the main resources in the NI between the embedded processors. This is causes some of these processors to become idle after a given period. In addition, there are increases in the idle time between these processors since all of these processors intend to operate the network protocol processing over a single bus. Furthermore, accessing local memories, such as the instruction memory, also allows only one processor at a time. Another concern is that these processors occupy a large space in the NIC and can increase the development time and cost. Besides all that, Amdahl's law states that the speedup achievable on a parallel computer can be materially limited by the existence of a small fraction of inherently sequential code which cannot be parallelized. In this case of parallelization, Amdahl's law [7, 9] explains the in-depth processing inside a design. The implementation program is usually divided into two portions: the first, part "P", is the amount of the protocol processing program that can be made parallel, and the "1- P" is the other portion of the processing that cannot be parallelized; it remains serial. The maximum achievement on the NIC by using the N processors is :

$$\boldsymbol{S(N) = \frac{1}{(1-P)+\frac{P}{N}}}$$

Assuming the P is 90%, Then 1- P = 10 %. With this high assumption of parallelized code (90%), the problem is sped up by a maximum of a factor of 10, no matter how large value of *N* used. Accordingly, using a number of processors to support the LSO can be accomplished only if this design has extraordinarily high values of *P*: This is known as the embarrassingly parallel problem. The complicity could increase when LSO Software migrations will most likely start from serial code bases. Therefore, the target software design needs to identify the solution to meet the migration requirements. Furthermore, the programming model should be Symmetric Multiprocessing (SMP) or Asymmetric Multiprocessing (AMP). However, CPU intensive code for parallel processing using SMP is difficult to redesign.

This study aims to investigate the possibility of improving the structure of the NI by placing a specialized RISC as a core unit for the 100 Gbps for LSO to support the TCP/IP and UDP/IP.

We designed a single-issue RISC cores supported with three pipeline stages and forward engine to process the LSO functions. Since the receiving-side and the sending-side operation are completely independent, the NI is designed to handle both operations in parallel. In this paper, we have focused on the sending side only. The RISC cores performance is measured while processing the LSO function. The wide use of TCP and UDP applications over the Ethernet amounts to roughly 82% of the protocol usage [8], which has directed this research to evaluate these protocols. Other protocol types also can be studied and evaluated within this model. The scalable NI, therefore, can be used as an open guide for protocol processing and future use.

The analysis of the packet processing of TCP/IP and UDP in the NI will be addressed. The processing time in the NI is also measured to adjust the DMA and RISC's clock rates for 100 Gbps.

The rest of this paper is organized as follows: Section 2 discusses the sending side processing. In section 3, the sending side structure in the ENI model. The processing anaylsis presents in section 4. The core design has been highlighted in section 5. The VHDL-based simulation results are discussed in the next section. Conclusion is in the last section.

III. LARGE SENDING OFFLOAD (LSO)

It is evident that the LSO feature is helpful only on the transmit side by freeing an OS from the task of segmenting the application's
Transmit data into MTU-size. Using LSO, the core engine in the NI divides the data into Maximum Segment Size (MSS) (1460 bytes for TCP segment or 1472 bytes for UDP fragment). The Sending Embedded Processor (SEP) also generates the packet header attaching it to the payload part and eventually sending it to the MAC layer to be sent to a network line.

Transmissions are according to the protocol type. For instance, TCP/IP protocol uses the two identifiers in each packet; the Sequence Number (SN) and the Acknowledgment Number (AKN). The beginning segment carries the start sequence number of data. Segments also carry an AKN, which is a SN of the next expected data portion of the transmission. [4]. We also tried to use the UDP to be fragmented. Fragmenting the UDP is based on information on the IP header, such as the MF field and the offset.

LSO has designed by shifting the higher layer transmits processing to a NIC. For example, the processing for production packets controls data for each outgoing packet. The core engine in the NI is responsible to handle these tasks related to transport layer. In the smart NIC [13, 15] implementation, the datagram was sent to a NIC's buffer. The datagram can be sized up to 64 K byte (the receiver's TCP window size is set to 64 KB). At the NI, the core engine after reading all the information related to the moved datagram, such the position of the message inside the NIC' buffer and the packet size, then SEP first examines the size of datagram. The core engine starts by generating the network header for each chunk of data. The smart NIC holds a TCP/IP header

template that has the IP total length, the initial SN. A copy of the template header whenever there is a segment data needs to be sent to a network. It updates the essential fields inside the TCP and the IP header of the copied headers, such as the SN and in the datagram total length before sending a packet. Attached this copy to the selected segment to create a complete packet then send it to the MAC layer. However, these processing scenarios in these implantations are successful in offloading the LSO to the NIC, but still cannot be scale it to express network since the header copy itself required at least 5 cycles over the 64-bit bus (40 bytes header). In addition, there was no explanation for the data movements methods inside the NIC.

The main embodiment of this work is to provide an alternative method for sending data faster to the physical and MAC layer. This approach has focused on the header process and data movements. For header processing, we have provided a new algorithm for enhancement of the flow of the packets processing. After a host CPU specifies data to be sent to a network (larger than the MTU) and send to NI's Buffer, a Beginning of Message (BOM) processing is the first step required by NIC's core. The IP identification values are set based on the initial value. Conceptually, each packet required to apportion its SN and AKN number inside the TCP header. Within the processing of the BOM, the IP total length is 1500 bytes (the default MTU) unless the two networks ends specifies different size during the connection setting up. At the stage of Continuation of Massage (COM), only needs to update some fields inside network packet. For instance, the core engine requires updating the SN and the AKN inside the TCP. In the COM processing, the total length of the data gram will remain the same. In addition, the previous AKN is a SN of the current packet. End of Message (EOM) is the last part of the message, which contains the remaining data of the sent datagram. When the message is small (less than MTU), a Single Segment Message (SSM), can be sent as is to the MAC layer. Instead of copy a template header as in the another scenario, the core processor at the NIC decides which method of processing need for each segment either the BOM, COM,EOM or SSM.

To reduce the processing the core processor required for generating the headers, we have added extra features to the LSO processing, which is overlapping in manner. Processor can benefit from the transfer of data period by implementing other processing required for the followed packets. We have adjusted the LSO assembly code to keep the RISC busy while transferring data. For example, the core calculates the remaining datagram size inside the NIC's buffer to figure out which subroutine code should follow either COM or EOM.

IV. NI MODEL FOR SENDING SIDE

We have structured the proposed NI into three parts: communication Line Interface (LI), kernel processing, and Host Interface (HI) “Figure 1”. The HI and LI are implemented in hardware. The processing unit in the NI, which commonly processed functions that are related to header processing, is an embedded specialized RISC. Since the receiving-side and the sending-side operations are completely independent, the NI is designed to handle both operations in parallel. Two RISC-cores are considered for being in the NIC: one for *Sending-side* and the other for *receiving-side.*

A. Communication with the host

The NI communicates with the host through five FIFO buffers. Two FIFOs were implemented as memory-based, and the pointer of each FIFO is stored in the RISC's register. The RISC reaches any FIFO after reading its address.

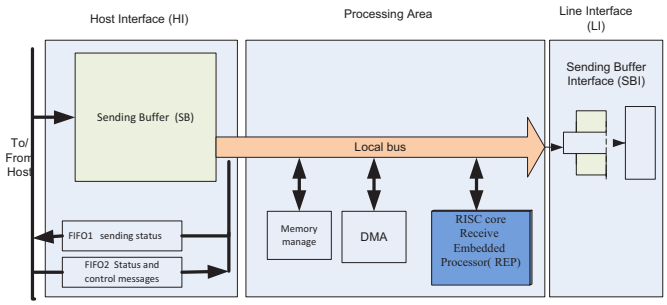

Figure 1: Sending block diagram

However, the interrupt mechanism that happens during the exchange of information may affect the overall performance of NI or the host CPU. Interrupting the host CPU or RISC cores (SEP) during their processing time will cost a certain amount of time in processing the interrupting task. These interrupts reduce the performance of the processing power of the Host CPU.

After the host CPU moves the message to the SB, the host required to notify the sender RISC by sending the location of the message inside the Sending Buffer (SB) and other necessary information needed for segmented the message, such as the MSS through FIFO 2. The SEP also sends the notification of the sending data through FIFO1

B. Buffers

The Sending Buffer Interface (SBI) VHDL based contains two buffers each of which hold one packet (1500 bytes). The sequential machine controls the SBI and allows only one buffer to be active and to receive data at a given time. The buffer will remain enabled until the complete packet has been stored.

The sequential machine will then allow the stored data to be sent out while it fills the other buffer.

SB is designed to be dual port memories. The SEP and the host can access the local memories simultaneously. 64-bit local bus is used to send data from the SB to SBI.

C. UDP processing

Since the receiving-side is entirely independent of the sending-side, the SEP, DMA and other devices are sharing a single bus to operate the LSO.

Fragmentation is required if a message over the MTU. The SEP is responsible to send this message over a multi of small packets to a network, where each fragments transfer as a separate packet, header and payload. The IP header has three fields that used for fragmentation processing [14]; Identification ID (16 bits). A datagram ID set by the source. Fragmentation offsets (13 bits), required to distinguish the location of the datagram. It also specified in multiple of 8 bytes. The flags filed (D-bit and M-bit) inside the IP header is used is need during the fragmentation procedure. The D-bit (do not fragment bit) prevents fragmentation. The M-bit specifies if this fragment is the last one in the original message or not.

Each packet carries a relevant data in mentioned fields. For example, if the SEP wants to send 6840 bytes of data to a network (Figure 2). The original packets' header has the identification ID = "10", M bit = "0" (since no packets following) and the offset = 0 (the packet is a single packet). The MTU is 1428 bytes. This means packet can carry 1400 bytes in the payload part. The SEP divides the original packet to several fragments. The BOM has the first fragment data (1400 bytes). The header information is as following; the packet ID = "10" , the M bit = "1" indicates more packets follows has the same ID. The offset is "0". The COM is the followed packet, which has ID = "10" and the offset is 175 (1400/8) because it is located at a relative location of 175 bytes. The M bit is "1". The EOM is the last packet of the message, which hold the rest of the data, has the M bit = "0". The offset is 700 (5600/8).

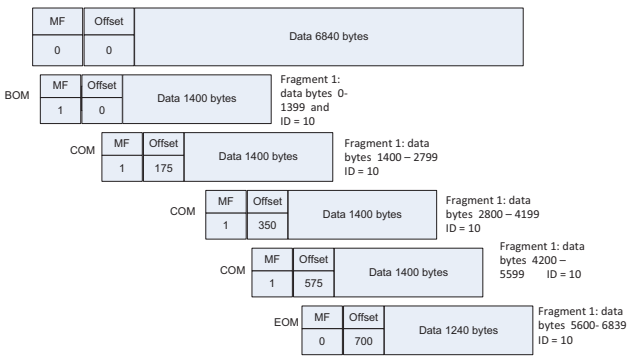

Figure 2: procedure of sending of 6840 bytes

D. TCP segmentation

Segmentation is the term used to describe the process of cutting streams of data into smaller packets, Maximum Segment Size MSS. The SEP has to divide the original message (larger than the MTU) into a number of segment data. Each packet has IP and TCP header. Each of these segment has its own TCP and IP header. TCP header has a SN the packet has to have and the AKN which the receiver expect to get. The segmentation processing starts with the REP reader the packet lent from the IP header, the SN and AKN from the TCP message that a host delivered to SB. SEP also reads the information related to this message such the MSS. Start processing the SN, AKN and total length of a datagram. Instead of copying the packet's header into SEP's registers, updating these fields inside the SB. Next, the SEP initiates the DMA to transfer the header and the appropriate segment data to the SBI.

E. Data movements

Using the Programmed I/O (PI/O) method for data movement makes the RISC core controlling the bus while data is moved. Where, the RISC processor associated with the transfer of data from one location to another, especially when moving a large amount of data (1640 bytes). The DMA is used for transferring data between the SB and the SBI. The RISC core initiates and controls the DMA. Since the local bus is shared between the DMA and the RISC core, the RISC core requires releasing the local bus to let the DMA performing the data transfer. Each transfer of 64-bits consumes two cycles. First cycle, the DMA reads the source buffer to get 64-bits to the DMA's register. During the second DMA cycle, the words move from the DMA's register to the destination buffer. The DMA state machine will then provide the read and write signals to the source and destination buffers. The state machine in the DMA is also responsible for incrementing of the address counter. The use of the DMA reduces the RISC processor instructions cycles.

We have simulated the Segmentation and fragmentation function for TCP and UDP respectively. The DMA controller is responsible for moving the packet header as well as the payload part from SB to SBI for both TCP and UDP protocol, where it needs 375 cycles. With pervious standard LSO, the core engine responsible to send the header from its register to SBI. In fact, this situation required additional instructions from the RISC core to transfer 40 bytes (TCP/IP headers) over the 64-bus. Further, the core has to wait for the DMA controller to release the local bus in order to deliver the headers from its registers to SBI. The core also requires, at least, 5 cycles to store the network header. In this simulator, the RISC core initiates the DMA to transfer data from SB to SBI. The core is responsible to update the packet headers for each segment inside the SB. The processor uses several pointers in order to continue updating data in the SB; the Start Header Address Pointer (SHAP), End-Header Address Pointer (EHAP), Start Payload Pointer (SPP) and End-payload Pointer (EPP).

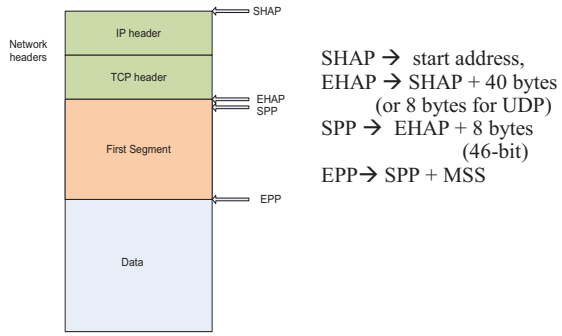

Figure 3: New approach of sending packets over the local bus.

The RISC core uses the SHAP pointer for reach the network headers in side the SB. The SPP pointer helps the RISC to locate the start. The EPP is used to point at the end of the first segment. The RISC updates this pointer during the data movements of the first packet (the BOM).

IV CYCLE PROCESSING

The TCP payloads vary from 6 to 1640 bytes [1]. The DMA required moving data (i.e MSS is 1460 bytes) from the SB to SBI is 366 cycles (183 cycles to read payload data over the 64-bit bus to the DMA's data register and 183 cycles to store it to RB). Clearly, the RISC core will be in idle mode until the DMA completes moving the data Figure 4. The RISC can execute 7 instructions during the data moments and becomes idle with MSS at about 359 instructions. The idle cycle's time affects the performance of the network card and its capabilities to deal with high speed networks.

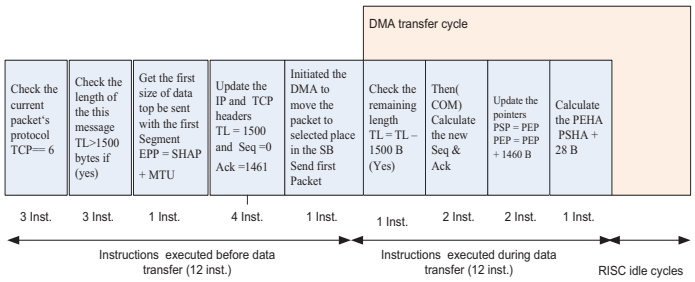

Figure 4: REP processing the BOM required 18 instructions

Small size packets, such as 64 until 256 bytes, may require less DMA cycles than other packets that have more payload bytes. However, using these small size packets could improve the NI's performance, yet it affects the end node's throughput [10]. We have focused on the 512 bytes packet to MTU packets (1500 bytes). The use of small packets can be studied on this Model, but they bear little payload data and may not be able to achieve 100Gbps.

We have studied the ways that can be used to reduce or to eliminate the idle cycles of the RISC. One of these solutions is the use of a multi-bus based on the sending side. The RISC can access the multiport memories while the DMA controller moves data. The other approach is to use a DMA that runs at a higher clock rate than the RISC. We have adapted the way of using it that we presented in "Figure 1", since it is a straightforward data path scenario and easily implemented without any changes in the NI's architecture. We have started adjusting the DMA's clock to reduce the idle cycles. In order to study and analyse the cycle –accurate NI simulator, we sent different large packets to the sending side. Each time we increased the DMA's clock to reduce the idle cycles, we have noticed that the RISC core and DMA controller were working quickly to complete each message and transfer it to RB.

When DMA's clock has five times the embedded processor core, the NI performance is increased significantly, where most, if not all, the idle cycles are reduced Table 1. Table 1 presents the total RISC cycles required for TCP/IP and UDP/IP packets when the DMA has five clock rates as RISC core. The DMA clock rate was measured while performing different packet sizes Figure 5. Figure 5 also shows the RISC clock rate in MHz for each packet before the idle cycles start.

When the packet size is 512 bytes, the idle cycles are reduced significantly. We have fixed the DMA clock rate to 2115 MHz and used this rate with other packet sizes (larger than 512 bytes). This rate of the DMA's clock helps to reduce the idle cycles in the other packets those are larger than the 512 bytes, such as 1500 bytes "Table II". This is natural because the number of messages that the NI send is less than in the case of 512, which is only 81274382 packets per second when the packet size is MTU [1].

In the relation to the cycle accurate simulations and the RISC core cycles for LSO function, we have found that using a DMA controller five faster than the RISC core will improve the performance.

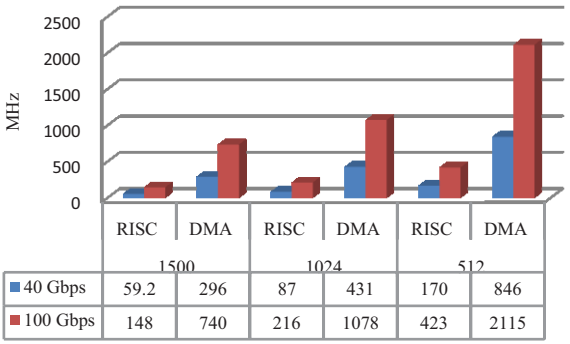

Figure 5: The RISC and DMA clock rate in MHz for TCP Segmentation and UDP fragmentation (when the DMA has five RISC's clock rate)

Because the RISC core can accomplish part of the processing while the DMA controller is moving the packets from the SB to SBI, the RISC core is forced to be idle for a few cycles until the DMA completes the payload transfer. Therefore, using a faster DMA will help to eliminate most of the idle cycles of the RISC core.

V. RISC CORE

The simulation results demonstrated that a RISC-based NI is scalable for a transmission line with a speed up to 100 Gbps. To reduce the design complexity, we presented a simple data path of the NI.

This simplicity helps the RISC cores to manage and process the LSO at a low clock rate. Further, it has made it possible to reduce the cost of development of RISC-based NIs. Such NIs can be flexible enough to support protocol changes or can even adapt new protocols, whereas, the customized logic-based NIs supports certain functions.

Design a RISC core for specialized application, namely NI control and data path, is simpler than using the off-the-shelf GPP processors. These general-purpose embedded processors are not optimized for a LSO function. Hence, some portions of GPP instructions that support general-purpose applications may not be required for the ENI design. For example, the Floating-Point Unit is not necessary for network interfaces. Also, we found that, using a data cache to store data is not required since it will not enhance the NI's performance or reduce the RISC' clock for this application. The elimination of these units in the design the core simplifies the process of development of NI and reduces the size and cost.

We have noticed also that the RISC performs a few of the instructions to complete processing the LSO. These instructions are load, store, arithmetic and logic operation and conditional branches. The minimum type the instructions set used in the LSO function would make the control unit design simple and fast. In addition, the limited number of instructions that are required to support the Ethernet interface processing can reduce the size and complexity of the control unit leading to an increased speed.

VI. Simulation Results

In the LSO function processing, it is clear that the RISC processing time becomes less when the DMA has a clock rate faster than the RISC core is (Five times faster the RISC's clock) where all the idle cycle associated with the RISC core processing has eliminated. We monitored the highest clock rate of the RISC core during processing the different packets. We have found the VHDL behavior model for the sending unit of the network interface has a 148 MHz RISC processor which can support 100 Gbps lines, when the DMA speed is 2115 MHz, and the packet size is 1500 bytes "Figure 6". A RISC core with 423 MHz can be used to process the LSO at 100 Gbps when the packet size is 512bytes.

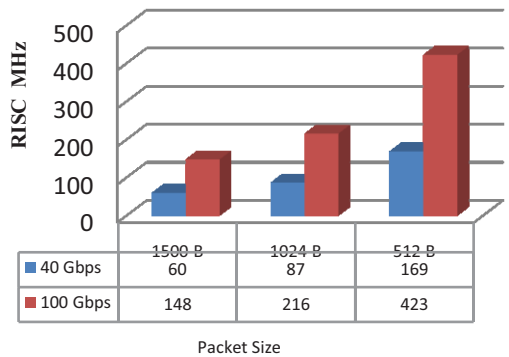

Figure 6: LSO for TCP/IP using DMA for data transfer (when the DMA 2115MHz)

VII. Conclusion

We have presented computer simulations results to measure the amount of processing required for LSO functions for TCP/IP. The simulation results have shown that a cost effective embedded RISC core can provide the required efficiency of the network interface to support a wide range of transmission line speed, up to 100 Gbps. A 423 MHz RISC core can support the sending side processing for up to 100 Gbps transmission speed for TCP/IP. A fast DMA (2115 MHz) is required to eliminate the RISC idle cycles. The DMA clock could be considered high, and this is because of small size of the local bus (64-bit). The DMA clock rate decreases significantly if the local bus becomes wider (i.e 128 –bit). Using cost effective RISC supporting higher speed network up to 100 Gbps for TCP/IP and UDP/IP is possible. The scalable NI based programmable could provide the flexibility needed for adding a new, or modify, protocol functions, while ASICs based solutions could provide better performance but are not flexible enough to add new or modify features.

TABLE I: TOTAL RISC INSTRUCTIONS FOR SEGMENTATION AND FRAGMENTATION WHEN THE DMA HAS FIVE CLOCK CYCLE OF THE RISC

	Packet Type	Packet Size					
		1500 bytes		1024 bytes		**512 bytes**	
		Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.
TCP	Single Segment Message	45	37	33	25	20	12
	Beginning Of Message	49	31	37	19	24	5
	Continuation Of Message	41	31	29	19	16	6
	End Of Message	41	37	30	25	17	11
UDP	Single Segment Message	45	37	33	25	22	13
	Beginning Of Message	49	31	37	19	25	7
	Continuation Of Message	40	32	28	20	16	8
	End Of Message	40	37	28	25	16	12

References

[1]. G. Held "Ethernet Networks (4th ed)," Design, Implantation, Operation and Management. John Wiley publisher LTD, 2003.

[2]. P. Willmann, H. Kim, S. Rixner and V. Pai," An Efficient Programmable 10 Gigabit Ethernet Network Interface Card," Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture November 2005.

[3]. O. Elkeelany, On chip novel video streaming system for bi-network multicasting protocols, Integration, the VLSI Journal, v.42 n.3, p.356-366, June, 2009.

[4]. J. B. Postel, "Transmission Control Protocol," NIC- RFC 793, Information Sciences Institute, Sept. 1981.

[5]. H. Kim, V. S. Pai and S.Rixner, " Exploiting task-level concurrency in a programmable network interface," Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp 61-72, 2003.

[6]. D. Schuff and S. Pai," Design Alternatives for a High-Performance Self-Securing Ethernet Network Interface," Parallel and Distributed Processing Symposium, IEEE International, pp 1 – 10, 2007.

[7]. Amdahl, G.M., "Validity of the single-processor approach to achieving large scale.computing capabilities," Proceedings of AFIPS Conference, 1967, pp. 483-485.

[8]. M. Allman and A. Falk," on the effective evaluation of TCP," ACM SIGCOMM Comput. Rev., vol 29, no 5, pp 59-70, Oct 1999.

[9]. M. Hill and M. Marty, "Amdahl's law in the multicore era," IEEE Comput., vol. 41, no. 7, pp. 33–38, Jul. 2008.

[10]. S. Makineni and R. Iyer," Measurement-based analysis of TCP/IP processing requirements," In 10th International Conference on High Performance Computing (HiPC 2003), Hyderabad, India, December 2003.

[11]. Y. Hoskote et al. A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS. IEEE Journal of Solid-State Circuits, 38(11):1866–1875, Nov. 2003.

[12]. C. Cranor *et al* .Architecture considerations for CPU and network interface integration IEEE Micro, January–February (2000), pp. 18–26.

[13]. G.Wiilium and W. Paul." ofload of TCP Segmentation to a Smart Adapter." U. S. Patent 5937169. 1999.

[14]. J. Postel RFC 791 Internet Protocol, protocol specification 1981

[15]. O. Cardona and J. B. Cunnlngham." System Load Based Dynamic Segmentation for Network Interface Card." U. S. Patent 0295098 A1. 2008

TABLE II: TOTAL RSIC INSTRUCTIONS TO COMPLETE PROCESSING THE LRO WHEN THE DMA BECOMES 2115 MHZ

	Packet Type	Packet Size					
		1500 bytes		1024 bytes		**512 bytes**	
		Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.
TCP	Single Segment Message	18	9	15	6	20	11
	Beginning Of Message	23	4	20	1	24	5
	Continuation Of Message	14	4	11	1	16	6
	End Of Message	15	9	11	5	17	11
UDP	Single Segment Message	8	0	8	0	22	13
	Beginning Of Message	23	5	20	2	25	7
	Continuation Of Message	13	5	11	3	16	8
	End Of Message	13	9	11	7	16	12