Large Sending Offload: Design and Implementation for High-Speed Communications Rate up to 100 Gbps

Mohamed Elbesht1i[1] Mike Dixon Terry Koziniec

School of Engineering and Information Technology, Murdoch University, Perth Australia, 6160

m.elbeshti@murdoch.edu.au

ABSTRACT

With the release of P802.3ba IEEE, send and receive data at 40 Gbps to 100 Gbps could become possible. Offloading part of the processing protocol to the network card showed success in sending data at high speeds up to 10 Gbps. Eventually, by enhancing transmission processing and taking advantage of the high speed, a potential network interface structure and performance needs to be addressed and implemented in order to send packets quickly corresponding with 40 Gbps and 100 Gbps. In this paper, such a network interface that supports high speeds after 10 Gbps has been designed. An alternative Large Sending Offload algorithm for sending TCP/IP and UDP/IP packets has also been implemented. A cost-effective RISC processor and a simple DMA for data transfer have also been implemented and simulated and tested for high speed. The behavior model shows that, a 450 MHz RISC core can support the sending-side processing up to 100 Gbps transmission speed for the TCP/IP and UDP/IP protocol when the MTU is 512 bytes or larger. A DMA with 2115 MHz is required to eliminate the idle cycles while transferring data over the 64-bit local bus.

KEYWORDS

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

1 INRODUCTION

The structure and data processing of the Network Interface (NI) needs to be enhanced to support the new network speed at 100 Gbps. In the very early stage of its development, the 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. For this reason it is thus possible to integrate all the discrete components needed for network interface (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 has also reduced the cost of design and improved reliability and performance. Using ASIC to design NIs however provides a greater energy efficiency and better integration than those that are 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 NI, such as the use of a fully customized logic-based network interface, can be expected for the following reasons: The new trend of designing the network interface is to have all the network interface functions implemented in a single chip; secondly, there is the license requirements while using a commercial General Purpose Processor (GPP) as a core engine in an NI; and finally, the size of these embedded GPP is large enough to be accommodated in the NI chip, since it has not been designed for the NI's functions.

There are many off-the-shelf cost effective embedded cores that have become available and can be ported to an NI chip, since 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]. However, these processors are not optimized for LSO since these processors are designed to support general functions, follow complex instructions and maintain long and variable execution time. These GPPs also have a long number of pipeline stages, and large registers to accommodate all its possible uses. These

advances in the GPP have directed this research in investigating the use of specialized RISC embedded cores for the LSO.
A scalable network interface (sending-side) that is supported with a simple data path of RISC core has been designed and implemented in Xilinx [16] simulator. A Behavior model performance for the developed algorithm of the LSO for TCP and UDP protocols is tested for high speed communication rate up to 100 Gbps. The designed RSIC core's performance and evaluation for high speed communication rate have been recorded in this study.
The rest of this paper is organized as follows: Section 2 discusses the sending side structure in the NI model of the sending side processing; Section 3 analyses sending side processing methodology; The processing analysis is presented in section 4; the core design has been highlighted in section 5 and the behavior model simulation results are discussed in the section 6. The conclusion of this study is discussed in the last section.

2 SENDING-SIDE BLOCK DIAGRAM

It is a known fact that the shifting part of transport and data link layer processing to the network interface enhances the end node performance [10] whereas a host CPU sends only one large packet (up to 64 KB) instead of sending a multiple of small packet to the network interface and generating interrupts for sending after each packet. This certainly reduces the number of the interrupts that the host CPU sends to network interface after completing the task of generating one packet. In addition, on the sending side, one large packet reduces the impact of protocol overheads on network throughput [9] and enhances the bus utilizations [8]. However, offloading the entire transport layer processing [7] can cause more complications to the network interface design and needs more consideration for the flow of protocol [6]. Today significant challenges are faced by server platforms while performing TCP/IP or UDP/IP protocol processing. For instance, the speed of networks now exceeds the 10 Gbps. As a result, the design and implementations of high-performance Network Interfaces have become very challenging since the budget time for calculating and generating the packet header remains extremely short (only 123 ns when the line speed is 100 Gbps).

In this type of network, interface structure that supports the stateless Large Sending Offload (LSO) and one that processes the TCP/IP segmentation and UDP/IP fragmentation applications has been designed. The proposed network interface (sending-side) that can support the high speed is divided into three sections; the host interface (HI), Line Interface (LI) and the packet processing unit (Figure 1). The DMA is used to transfer data inside the network interface. The Sending Embedded Processor (SEP) and the DMA are attached to 64-bit bus.

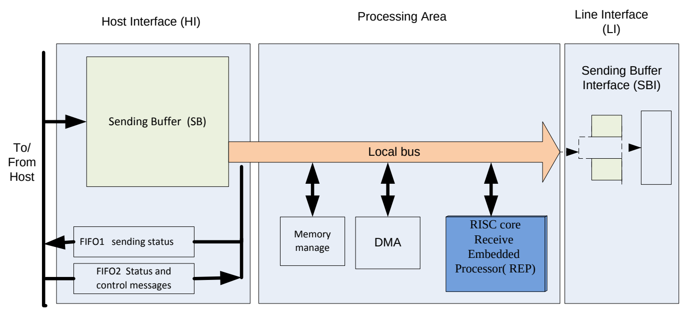


Figure 1: The Sending side model

After receiving an interrupt that a packet is stored in the Sending Buffer SB, the core engine at the network interface starts processing the larger packets. The processing, however, is dependent on the protocol type. For example, the TCP protocol supports the end-to-end reliability segmentation, where each packet has two identifiers: the Sequence Number (SN) and the Acknowledgment Number (AKN). The beginning segment carries the start sequence number of the TCP segment. It also carries an AKN, which is the SN of the next expected data portion of the transmission [4]. The SN and AKN need to be updated in each outgoing segment.
The aim of this work is to provide an alternative method for sending data faster to the MAC and physical unit. This includes the packet header process using a specialized RISC processor. For header processing, providing a new algorithm could enhance the flow packets processing. There are two possible approaches that can be implemented to complete processing the TCP or UDP packet. The first is the network interface' s engine that transfers the TCP/IP header of the original message from the Host Interface (HI) buffer to the local core buffer as a template header. (e.g., Inside the core' s registers) Whenever there is a segment, a copy of the template header to be sent to the Line

Interface (LI) after updating the essential fields, such as the SN inside the TCP and the datagram total length inside the IP header of the copied headers. Since the packet header is stored inside its register, the core engine then sends the packet herder from its register to HI. The second approach of processing the packet headers is updating the original large packet headers inside the SB and then initiating the DMA to move the packet from the HI to LI.

3. PROCESSING METHODOLOGY

The protocol process aims to establish and maintain the LSO of the TCP/IP and UDP/IP protocols. The process includes identifying the packet type, the total length of the packet, generating the packet header and sending a complete packet from the HI to LI (Figure 2). This processing can be started after a host CPU specifies packet to be sent to a network. This packet then is stored in SB for the segmentation or fragmentation. The host CPU also sends the necessary information to assist the core engine at the network interface in segmenting the large packet, such as the MSS. The SEP starts processing the large packet inside the HI to send to MAC unit as MTU. The SEP reads the length of the moved packet by extracting the packet length from the IP header. If the size of the packet is equal to or smaller than the maximum transport unit (MTU), then this packet is processed as Single Segment Message (SSM) and then it is passed to MAC and physical layer. If the packet size is larger than the MTU, the SEP starts calculating the first Maximum Segment Size (MSS) to be encapsulated with the packet header (e.g., TCP and IP) . which is then sent to LI. The TCP/IP packet can carry up to 1460 bytes when the MTU is 1500 bytes, whereas the UDP/IP packet can carry up to 1472 bytes in the payload part.
The processor uses several pointers to enable it to continue sending data to the LI: the Start Header Address Pointer (SHAP), the End-Header Address Pointer (EHAP), the Start Payload Pointer (SPP) and the End-payload Pointer (EPP) (Figure 3). The core engine is responsible for updating the packet headers for each outgoing segment from the HI to LI. The SHAP is pointed at the start-address of the large packet inside the HI. The EHAP is pointed to the end of the packet headers. The SPP pointer helps the SEP to locate the start application data. The last pointer is EPP, which points to the end of the first segment. The SEP updates the SPP and EPP pointer during the data movements of the first packet (the BOM). The processor core requires incrementing the SPP and EPP in order to transfer the entire application data to HI.

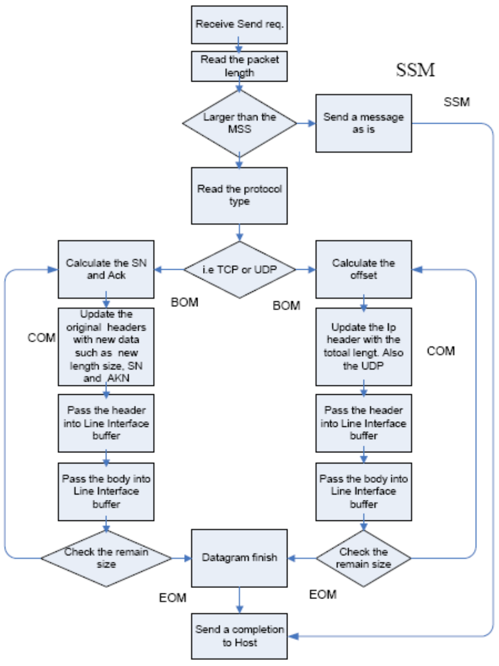

Receive Send req.
Read the packet length
SSM
SSM
Larger than the MSS
Send a message as is
Read the protocol type
Calculate the SN and Ack
i.e TCP or UDP
Calculate the offset
BOM
BOM
COM
Update the original headers with new data such as new length size, SN and AKN
Update the Ip header with the totoal lengt. Also the UDP
COM
Pass the header into Line Interface buffer
Pass the header into Line Interface buffer
Pass the body into Line Interface buffer
Pass the body into Line Interface buffer
Check the remain size
Datagram finish
Check the remain size
EOM
EOM
Send a completion to Host

Figure 2: illustrates inter-packets processing fro TCP and UDP

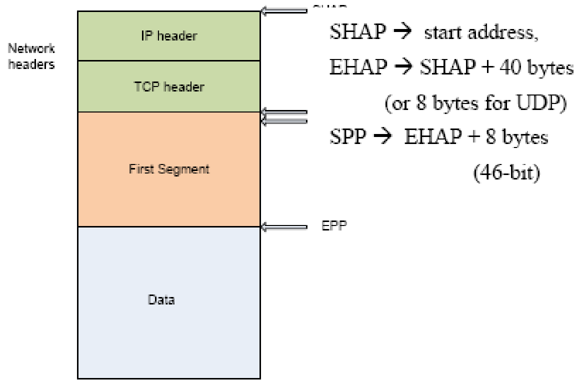


Figure 3: A new approach for sending packets over the local bus

3.1 UDP processing

With UDP processing, there is no guarantee that a packet will ever reach its destination. In addition, UDP has no flow control processing. However, careful flow processing is required to make sure that each piece of data is sent. The UDP packet processing starts after the protocol type is distinguished (when the protocol type is equal to "17"). The protocol type can be read from the first 32-bit of the IP header. The second 32-bit is then fetched from the IP header, which is the Identification field, and Fragment Offset field along with 'Do not Fragment' and 'More Fragment' flags. These three fields are used for fragment processing [5]. The Identification ID (16 bits), a datagram ID set by the source and the Fragmentation Offsets (13 bits), are required to distinguish the location of the datagram.

The application data of the long packet is then divided into portions of 8 bytes (64 bits) boundary in such a manner that the first packet, the Beginning of Message (BOM), carries the first 1472 bytes. The 'More fragments' flag (MF) in the first packet is set to one (to indicate that more fragments of this packet follow) Figure 4.

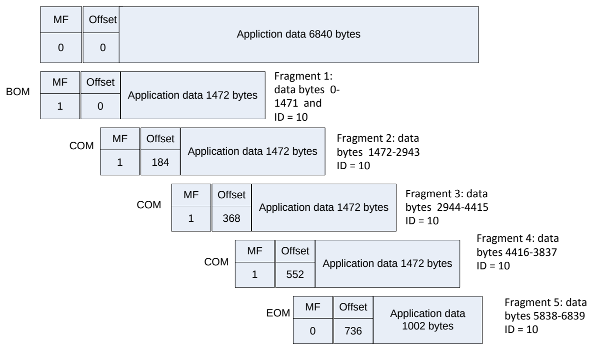

MF
Offset
0
0
Appliction data 6840 bytes
BOM
MF
Offset
1
0
Application data 1472 bytes
Fragment 1: data bytes 0-1471 and ID = 10
COM
MF
Offset
1
184
Application data 1472 bytes
Fragment 2: data bytes 1472-2943 ID = 10
COM
MF
Offset
1
368
Application data 1472 bytes
Fragment 3: data bytes 2944-4415 ID = 10
COM
MF
Offset
1
552
Application data 1472 bytes
Fragment 4: data bytes 4416-3837 ID = 10
EOM
MF
Offset
0
736
Application data 1002 bytes
Fragment 5: data bytes 5838-6839 ID = 10

Figure 4: Procedure of sending a UDP user data application

The original packet's header has the identification ID = "10", M-bit is equal to "0" (since no packets follow) and the offset is equal to "0" (the packet is a single packet). The MTU is 1500 bytes. This means that the packet can carry 1472 bytes in the payload part. The SEP divides the original packet into several fragments. The BOM has the first fragment of data (1472 bytes). The header information is as following: the packet ID is equal to "10". The M-bit is equal to "1" indicating more packets to follow with the same ID. The offset is "0". The COM is already following packet where the packet ID is equal to "10" and the offset of 184 (1472/8) because it is located at a relative location of 184 bytes. The M-bit is equal to "1". However, since the last packet is the end packet of the message (EOM), which holds, holding the rest of the data, the M-bit is equal to "0". The offset is 736 (5888/8).

3.2 TCP Processing

If the stored packet is TCP, the SEP starts calculating the first part of the TCP data application, the Beginning of Message (BOM), which is the first part of the application data, the IP values (sender and destination address) of the BOM are set based on the initial value. Conceptually, each TCP packet requires appropriate Sequence Number (SN) and Acknowledgement Number (AKN) inside the TCP header (Figure 5). The total length of the BOM is 1500 bytes (the default of the MTU), unless the two networks ends specify different sizes of the MTU when the TCP connection is being set up. The payload part of the BOM is 1460 bytes. After completing the sending the BOM to the HI, the core engine examines the remaining application data in case more packets need to be sent to HI. The continuation of message (COM) are the subsequent packets of the stream. With the COM, changing the SN and ACK is essential, where the length of the datagram remains as it is (e.g., 1500 bytes). The length of the datagram may change the End Of message (EOM) in the last sent packet, which bears the latter part of the application data.

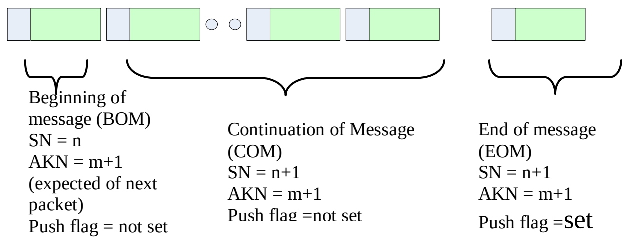


Figure 5: procedure of sending a TCP user data application

3.3 Data Movements

Since there is no analysis of the data movements inside the network interface, this research provides the analysis of packet movement between the network units (from the HI to LI). The DMA is used for transferring data between the HI and the LI. The SEP core initiates the DMA. Since the local bus is shared between the DMA and the SEP core, the SEP core requires the releasing of the local bus to let the DMA perform the data transfer. The local bus in this work is 64-bit. Each of the 64-bit transfered data consumes two DMA cycles. In the 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 the incrementing of the address counter. The use of the DMA for transferring data reduces the SEP processor instructions‟ cycles. The benefit is that the core processor is kept busy while transferring data (Figure 6). For example, the core calculates the remaining datagram size inside the NI's buffer in order to figure out which subroutine code should follow either the COM or EOM. This increases both the reliability, and the speed at which data can travel across a network.

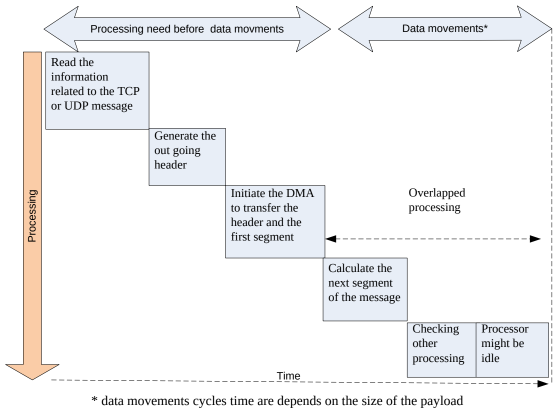

Processing need before data movments
Data movements*
Processing
Read the information related to the TCP or UDP message
Generate the out going header
Initiate the DMA to transfer the header and the first segment
Overlapped processing
Calculate the next segment of the message
Checking other processing
Processor might be idle
Time
* data movements cycles time are depends on the size of the payload

Figure 6: Pipeline processing at the sending-side

3. CYCLE PROCESSING

The model simulation processing started by delivering different sized valid packets to receiving-side can keep the Receiver Embedded Processor (REP) engaged with header processing and multiplexing the incoming packets to related linked-list while the DMA transfer cycle is in operation. There are three general steps of TCP processing (Figure 7). In the first, header processing is done to get the protocol type and the connection information, for example, the IP address and Sequence Number (SN). A search is made to find the match within the CAM entry. In the second, the type of incoming packet is determined, to see whether it is BOM, COM, EOM or SSM. The last step is to move a data from the Receiver Buffer Interface (RBI) to RB. RISC core becomes in idle cycle until the DMA releases the bus.

The TCP payloads vary from 6 to 1640 bytes [1]. From the SB to SBI the DMA requires a moving data (i.e MSS is 1460 bytes) of 366 cycles (183 cycles to read payload data over the 64-bit bus to the DMA's data register and 183 cycles to

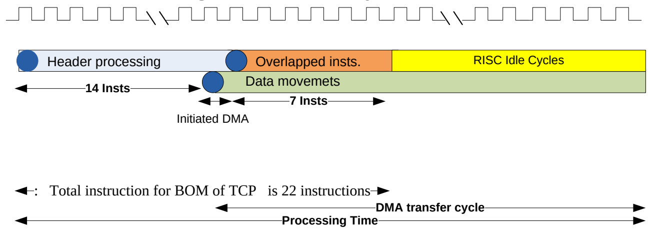


Figure 7: Large Receive Offload Processing cycles Characteristics

store it to RB). Clearly, the RISC core will be in idle mode until the DMA completes moving the data (Figure 8). The RISC can execute 6 instructions during the data moments and becomes idle with MSS at about 359 instructions. The idle cycle time affects the performance of the network card and its capabilities to deal with high speed networks.

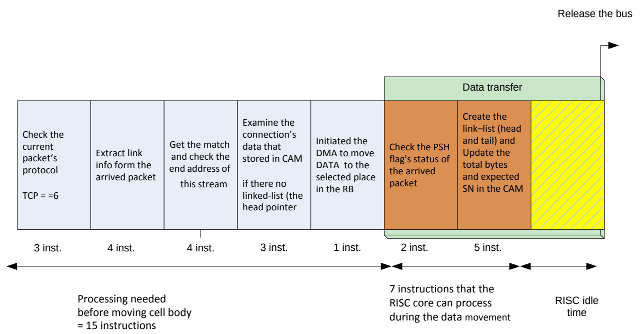

Release the bus
Data transfer
Check the current packet's protocol TCP = =6
Extract link info form the arrived packet
Get the match and check the end address of this stream
Examine the connection's data that stored in CAM if there no linked-list (the head pointer
Initiated the DMA to move DATA to the selected place in the RB
Check the PSH flag's status of the arrived packet
Create the link–list (head and tail) and Update the total bytes and expected SN in the CAM
3 inst.
4 inst.
4 inst.
3 inst.
1 inst.
2 inst.
5 inst.
Processing needed before moving cell body = 15 instructions
7 instructions that the RISC core can process during the data movement
RISC idle time

Figure 8: Large Receive Offload Processing cycles characteristics

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 sized 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

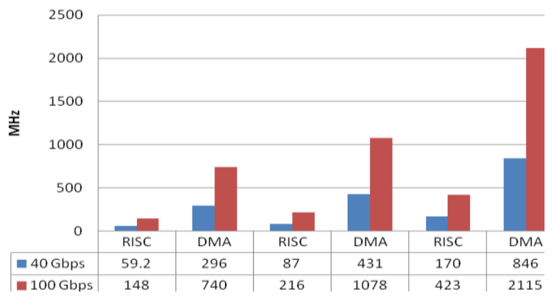
can access the multiport memories while the DMA controller moves data. The other approach is to use a DMA that runs at a higher clock rate than the RISC. We have adapted the way of using it that we have presented in Figure 1, since it is a straightforward data path scenario and easily implemented without 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 send different large packets to the sending side. Each time we increase the DMA's clock to reduce the idle cycles, we notice that the RISC core and DMA controller are working quickly to complete each message and transfering 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). The 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.

Table 1: 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

The DMA clock rate was measured while performing different packet sizes (Figure 9). Figure 9 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 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 2". This is natural because the number of messages that the NI sends is less than in the case of 512, which is only 81274382 packets per second when the packet size is 1500 bytes [1]. The cycle accurate simulations and the RISC core cycles for LSO function were recorded; we found that using a DMA controller five faster than the RISC core would improve the performance.



	RISC	DMA	RISC	DMA	RISC	DMA
🟦 40 Gbps	59.2	296	87	431	170	846
🟥 100 Gbps	148	740	216	1078	423	2115

Figure 9: The RISC and DMA clock rate in MHz for TCP Segmentation and UDP fragmentation (when the DMA has five RISC's clock rate)

Table 2: Total RSIC instructions to complete processing the LSO when the DMA becomes 2115 MH

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

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.

5. RISC CORE

The simulation results demonstrate 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 helped the RISC cores to manage and process the LSO at a low clock rate. Further, it 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 support only certain functions.

Designing a RISC core for specialized application, namely NI control and data path, is simpler than using 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 core simplifies the process of development of NI and reduces the size and cost.

We also noticed that the RISC performs only a few of the instructions to complete processing the LSO. These instructions are ‘Load’, ‘Store’, and related to other similar arithmetic and logic operation and conditional branches. The minimum type of instructions set used in the LSO function would make the control unit design simple and faster. 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. The NI RISC core has been designed to execute one instruction in three-pipeline stage (Figure 10) namely :

a) Fetch an instruction from local memory (Fetch stage).

b) Decode/execute the instruction and registers read (Decode/Execute stage).

c) Store results back into the destination register (write back, or W/B, stage).

The RISC fetches instruction are used to run the TCP/IP and UDP/IP protocol program from local memory. Decoding and Executing stage is done to execute the running instruction that has been fetched by the first stage of the pipeline. The last stage of the RISC's pipeline is W/B, in which the data is written to the RISC's register. Some instructions such as Store instruction terminates at the Decode/Execute stage.

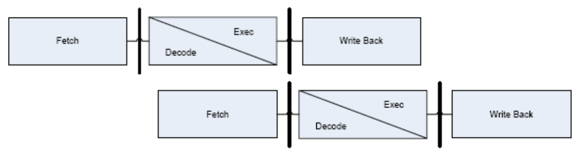
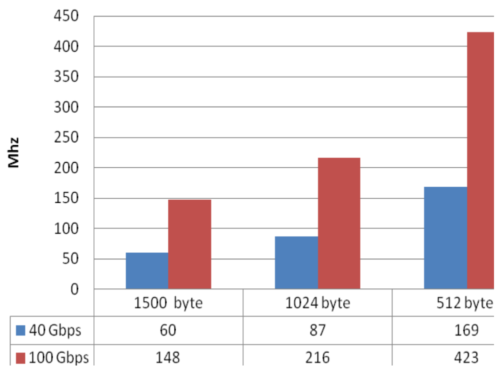


Figure 10: Structure of RISC instruction Pipeline

7. 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 been eliminated. We monitored the highest clock rate of the RISC core during processing different packets. We found the VHDL behavior model for the sending unit of the network interface having 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 11”. A RISC core with 423 MHz can be used to process the LSO at 100 Gbps when the packet size is



	1500 byte	1024 byte	512 byte
🟦 40 Gbps	60	87	169
🟥 100 Gbps	148	216	423

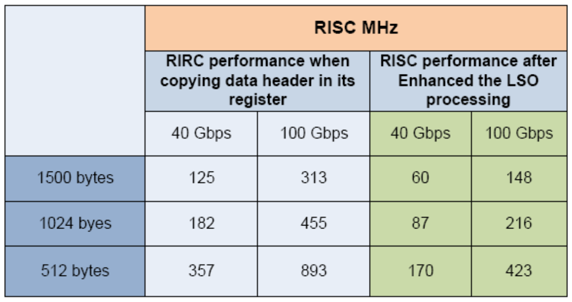
Figure 11: LSO for TCP/IP using DMA for data transfer (when the DMA 2115MHz)

A comparison of the RISC performance with the two approaches has been implemented. The First is copying the TCP and IP headers (that Host CPUT sent) from the SB to RISC's internal register. The second is to updating the packet headers inside the SB. After generating the packet header for each segment, the RISC needs to send the TCP and IP header to SBI for further processing. As a result, with the first LSO processing the RISC with 893 MHz is needed for 100 Gbps “Table 3”. It is clear that the RISC could spend more cycles than second approach since it needs to transfer the packet header from the internal buffer to SBI; it needs to modify the original header packet within the SB and to enhance and improve the process LSO and finally, to initiate the DMA to transfer the packet header from the SB to SBI

512bytes.

reduces the power of the RSIC to 423 MHz. The second approach of processing LSO also gives the RISC core more space to execute other functions that do not need to use local bus, such as calculating the next header's fields or checking the remaining size of the application data inside RB.

Table 3. Table captions should be placed above the table



	RISC MHz			
	RIRC performance when copying data header in its register		RISC performance after Enhanced the LSO processing	
	40 Gbps	100 Gbps	40 Gbps	100 Gbps
1500 bytes	125	313	60	148
1024 byes	182	455	87	216
512 bytes	357	893	170	423

8. 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 is high because of local bus's size (64-bit). The DMA clock rate decreases significantly if the local bus becomes wider (i.e 128 –bit [17]). We have also identified that 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 to add or modify a protocol. ASICs based solutions could provide better performance but are not flexible enough to add new or modify features.

9. REFERENCES

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

[2]. Postel, J., (August 1980). RFC 768: User Datagram Protocol. Internet Engineering Task Force. Retrieved from http://tools.ietf.org/html/rfc768

[3]. Attia, B. et al. "Modular Network Interface Design and Synthesis". IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 3, No 2, May 2012.

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

[5]. Postel, J. B, RFC 768: User Datagram Protocol. Internet Engineering Task Force. Retrieved from http://tools.ietf.org/html/rfc768.

[6]. Mogul, J., "TCP Offload Is a Dumb Idea Whose Time Has Come," Proc. 9th Workshop on Hot Topics in Operating Systems (HotOS IX), Usenix Assoc., 2003;

[7]. Earls, "TCP Offload Engines Finally Arrive," Storage Magazine, March 2002.

[8]. Intel Inc, "Small Packet Traffic Performance Optimization for 8255x and 8254x Ethernet Controllers", Application Note (AP-453), 2003.

[9]. Jin, H. and Yoo, C. "Impact of protocol overheads on network throughput over high-speed interconnects: measurement, analysis, and improvement." Journal of Supercomputing, Volume 41, Number 1 / July, 2007.

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

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

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

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

[16]. Xilinx ISE 13.1 release notes http://www.xilinx.com/support/documentation/dt_ise13-1.htm.

[17]. Altera."40- and 100-Gbps Ethernet MAC and PHY" June 2012 http://www.altera.com/literature/ug/ug_40_100gbe.pdf