

A FPGA-based Network stack with a reduced number of layers

Josue P. J. de Freitas, Gustavo F. Dessbesell, Joao Baptista dos Santos Martins
UNIVERSIDADE FEDERAL DE SANTA MARIA (UFSM)
josue.freitas@mail.ufsm.br, gfd@mail.ufsm.br, batista@inf.ufsm.br

ABSTRACT

This paper presents a proposal regarding a Network stack in FPGA with reduced number of layers. The network stack uses is composed by a Gigabit Ethernet Mac, connected to an Application Layer by a Middle Layer. The Middle layer implements some network features, like connection management and buffering. The Middle layer aim to be responsible for all network processing this way letting the Application layer free to process application information. Our proposed network node show a maximum 1311.63 Mb/s full-duplex throughput. Besides, considering just the internal 32 bits data bus and operation frequency of 125 MHz it's possible to reach around 8000Mb/s of full-duplex internal throughput. Our architecture presents a throughput 4.7 times higher in the best case (2.5 times higher on average) when compared to a software implementation running over personal computer. Also, comparing with an embedded network stack our architecture, using a microprocessor in Application layer, shows in the best case 44 times more throughput.

1. INTRODUCTION

The available bandwidth is increasing very quickly nowadays. According to Gilder's law [4] the available bandwidth doubles every six months in average. Considering Moore's law, which says that the number of transistors in an integrated circuit doubles approximately every two years.

A common network stack, developed in operating system software, shows a ratio around of 1 MHz to 1 Mb/s of full-duplex throughput [7] [6]. On an RFC regarding RDMA (Remote Direct Memory Access) [6] is shown that, in 2001, a 1.2 GHz Athlon could handle a maximum theoretical 2.7Gb/s of full-duplex throughput when using 100% of CPU capacity, i.e. the machine have too few CPU cycles to process user applications if we consider the Gigabit Ethernet technology available at that time. Copy through memory is considered a huge consumer of CPU cycles in software network stack implementations [2], this fact stimulated the development of hardware based network stacks and industry

standard network stacks with a reduced number of layers [5] [1].

Although the relation between general purpose processor and available bandwidth wasn't a huge problem in the past, this problem is increasing since network bandwidth is increasing much faster than processors capacity, even considering the multi-core architectures. The graphic in figure 1 shows processors frequency and the available bandwidth through the years. The graphic was assembled based on Intel processors released through the years available at <http://www.intel.com/pressroom/kits/quickreffam.htm> and Ethernet 802.3 standards releasing dates, and projections, found at <http://www.ieee802.org/>. In this graphic is possible to notice that this relation wasn't problematic before 1998, when Gigabit Ethernet standard was released. After this year the available bandwidth kept steady above processors capacity and it might keep this way in a long future since 100Gb/s Ethernet, 802.3ba standard, is already in experimental state in 2009.

Considering that embedded system use to have simple microprocessors, with few cache memory and lower frequency, this even more significant. So, this paper addresses these issues by proposing an FPGA-based network stack with a Middle layer responsible with network processing connected to a Gigabit Ethernet MAC and to an Application layer, in order to avoid the microprocessor getting involved in network stack processing.

The paper is organized as follows: Section 2 explains the network node architecture and relevant information. Section 3 presents results where our network stack is compared with pure software running over a PC and with an embedded an embedded TCP/IP network stack, lwIP[3]. Section 4 shows conclusions.

2. PROPOSED NETWORK NODE ARCHITECTURE

The Network node is composed by three main parts: a Gigabit Ethernet Mac (Gemac), a Middle layer, and an Application 2 . The Gemac block communicates with the buffer and Receiver and Sender blocks read and write data from and to the buffer. The Middle layer, situated between Application and Gemac, is where some network features are implemented. Furthermore, the Application is where data is processed, it communicates with the Middle layer by writing and reading to and from Sender and Receiver blocks.

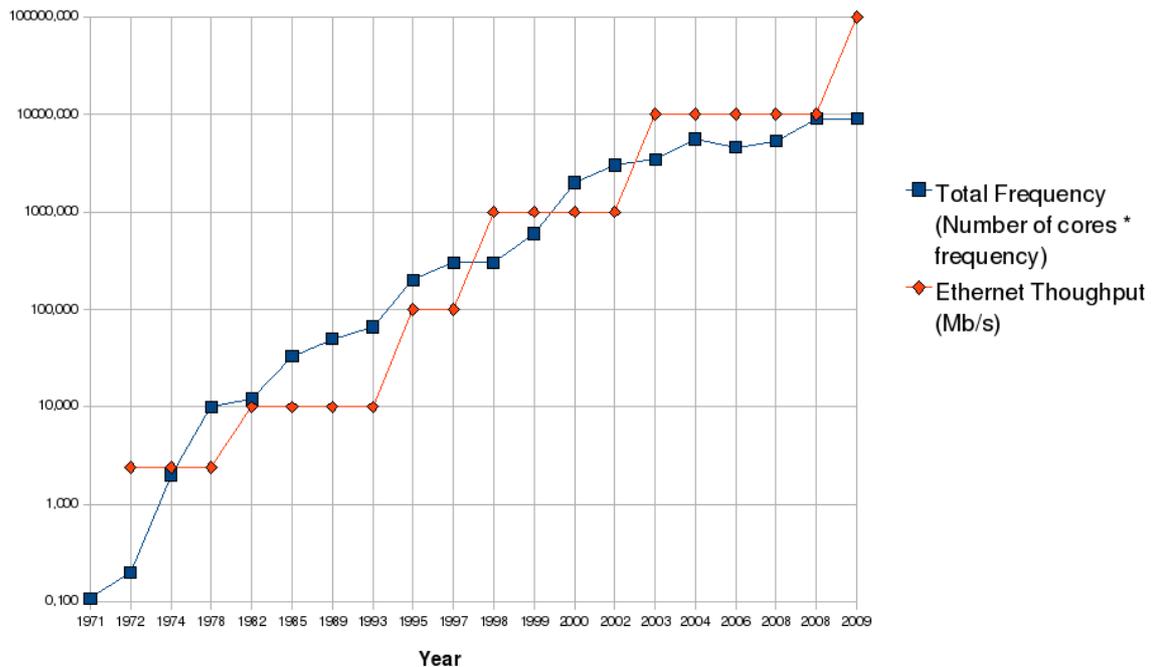


Figure 1: Available bandwidth versus Processors frequency

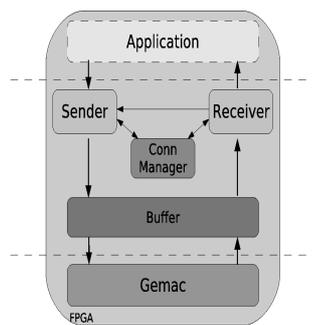


Figure 2: Proposal Network Node Architecture

A buffer is needed in order to support different applications that may have very different time to process the incoming data, this way avoiding discarding frames.

The Middle layer is composed by the following blocks: Sender, Receiver, Connection Manager and Buffer. The Receiver block checks some values in the control header and decides whether the inbound frame is correct or not. The control header is composed by 5 bytes after the Length/Type field of the Ethernet Frame and is shown in Figure 3. The fields First, Ack, Action, UIApp, Last, Seq were grouped into a single byte to make the bit manipulation easier in software, allowing the use of an unsigned char type to represent them.

- CID: Connection IDentification. An eight bit number that is chosen by the host that starts the communication;
- First: A '1' value indicates the communication first frame. The Receiver block checks if there is no other

active connection with the same key [SOURCE_MAC,CID]. If not, the connection is inserted on the Active Connection List through Connection Manager block;

- Ack: A '1' value indicates that source host requests an acknowledgement frame;
- Action: Together with other field indicates the type of the frame. A '00' means data transfer operation and '01' means an Acknowledgment (ACK) response;
- UIApp: Indicates to which Upper Level Application the frame is destined to;
- Last: A '1' value indicates communication end, causing the key [SOURCE_MAC,CID] to be removed from Active Connections List. If both First='1' and Last='1' the connection is not added to the Active Connection List;
- Seq: The Sequence field. It is used to check if the frames are sent correctly and without any loss.
- Reserved: Reserved for future use.
- Size: Size of Data in bytes. It was chosen to be 14 bits long since this way is possible to represent 16384 bytes of data, which is enough for the maximum Jumbo frame size that our architecture will support.

A finite state machine on the Receiver block realize all verifications. After the verifications from the Receiving algorithm, the header is stripped and stored in an entry on Connection Manager block and the data is forwarded to the Application layer. The Sender blocks communicates with the Connection Manager, getting information about the connection, and, in ACK frame sending cases, it communicates



Figure 3: Control header

Table 1: Device utilization for the proposed network stack architecture

Resource	Used	Available	Used(%)
Number of Slice Flip Flops	3,529	30,720	11%
DCM autocalibration logic	7	3,529	<1%
Number of 4 input LUTs	4,141	30,720	3%
Number of occupied Slices	4,080	15,360	26%
Number of RAMB16s (BRAM)	16	192	8%
Number of DCM_ADVs	1	8	12%

directly with the Receiver block. The Sender block communicates with the Application in order to read data processed by the Application.

3. IMPLEMENTATION

Our network node has been implemented in a Xilinx ML402 evaluation board. This board has Gigabit Ethernet PHY and a Xilinx Virtex 4 SX35 FPGA. Our system achieves a maximum frequency of 128 MHz, which satisfies Gigabit Ethernet requirements, where design must reach at least 125 MHz of maximum frequency. Block RAM (BRAM) were used in the Buffer implementation. The buffer size is 32 KB, where one half stores receiving data from Gemac and the other half stores data to be sent.

Resource utilization is available on table 1. By looking at this table is possible to notice that there are enough resources to improve our design in many ways, for example, increasing buffer size and improving parallelization.

Although the bus size used to communicate with Gemac must be 8 bits wide, our proposed architecture uses an 32-bit internal bus. This way, considering the clock frequency employed in our system (125 MHz), it is possible to reach an internal 4 Gb/s throughput in each way, i.e. an 8 Gb/s full-duplex internal throughput. The buffer block also chops and assembles the 32 bits to 8 bits and vice-versa in order to communicate with Gemac.

The software implementation, developed in C language using raw ethernet frames, was build using two threads through library Pthreads. One thread handles the data sending process and the other the data receiving process. This way, the comparison between software version and our FPGA architecture can be considered more fair than using a single thread implementation.

4. RESULTS

The results have been obtained by sending frames of different sizes to our proposed network node and for the software version implemented in two PCs. Both PCs where the software version run were a Pentium IV HT with 3.0GHz over a Linux operational system with a SMP (Symmetric Multi-

Processing) kernel, which allows the software application to use both threads in parallel.

The measurements have been made by using a PC connected through an Ethernet cable to the ML402 board. The same client software used in pure software measurements was used on the PC side to connect with our network stack on ML402 board. The ML402 board was configured with all architectures listed below, excepting PC Software running under Linux operational system. The Application layer on each architecture implements a pong application, which means that each application should response with the same data it was received.

- Proposal Arch (HW App): Our proposal architecture using an Application layer developed in Hardware Description Language.
- Proposal Arch (MB App): The same proposal architecture using a Microblaze@microprocessor, so this architecture have a software based Application layer.
- PC Software (Linux): A software developed in C language running over Linux operational system. This software implements the same features that our proposal architecture implements.
- lwIP TCP: A Gigabit Ethernet Mac connected with a Microblaze@microprocessor where lwIP embedded network stack was implemented, in this case using TCP sockets.
- lwIP UDP: The same as above instead using UDP sockets in communication.

The charts on Figures 4 and 5 show the latency and throughput results for ping-pong tests for our architecture and for other implementations. The latencies results show almost an equivalence between our proposal architecture and PC Software. The main reason behind these latency results is how latency test were made. One frame with specific size is sent (T0 time) and then the application keeps waiting until it comes back (T1 time) (see figure 6). Since just one frame is sent on this test, our architecture pipeline is empty, thus not using hardware resources like its inherent parallelism.

Although, comparing our architecture with lwIP embedded stack we achieve results more than 51 times better when using 6020 byte frame size. On average our latencies results were around 32 times better than TCP embedded software and around 15 times better than UDP.

On the other hand, throughput shows, on average, 2.5 times better results than the software implementation. The best result, for 120-byte frames, allows 4.7 times more throughput than in the software implementation, although it represents just 111.54 Mb/s more throughput. Considering the

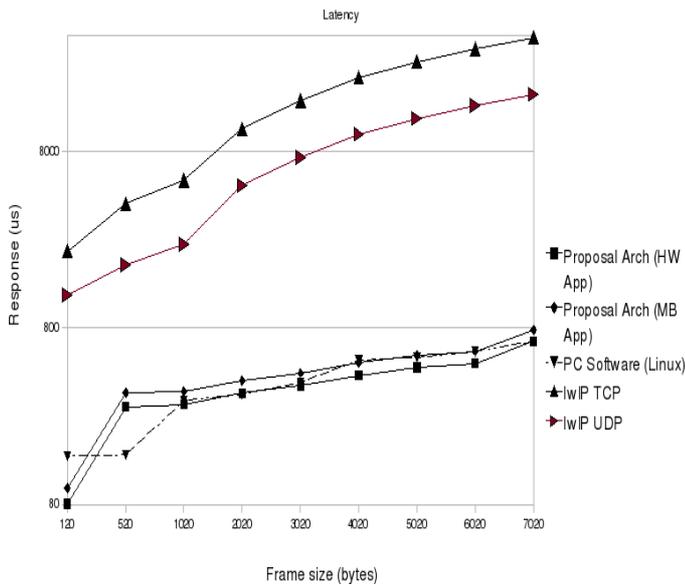


Figure 4: Proposal network stack architecture versus different network stacks' latencies

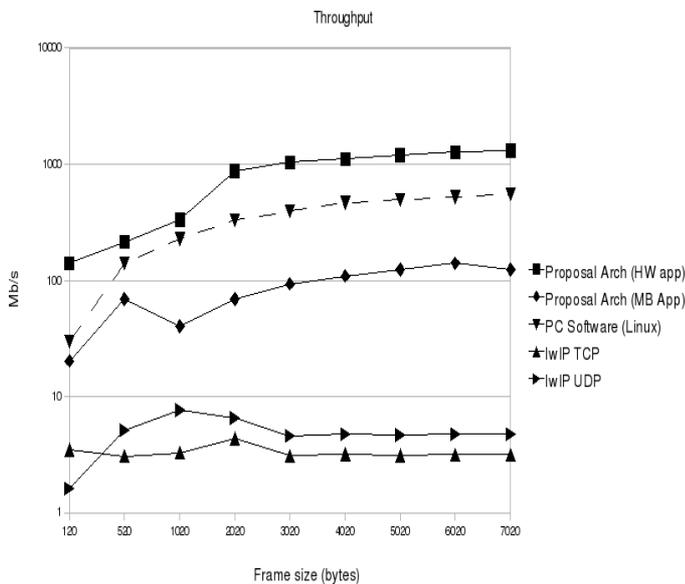


Figure 5: Proposal network stack architecture versus different network stacks' throughput results

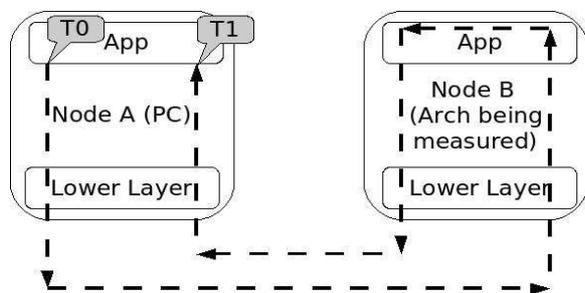


Figure 6: Benchmark environment

throughput, the best result provided by our architecture has been obtained using 7020-byte jumbo frames, when it is possible to transfer more 759.77 Mb/s than software implementation. In this configuration, our hardware is 2.38 times better than the software implementation. Also, comparing our architecture with lwIP implementation we had 27 times more throughput in TCP and 18 times on UDP.

This behavior, where throughput results are much better than latency results, is due our architecture pipeline is being fully functional most part of the time. Also, the buffer utilization allows an almost constant data rate between buffer and Gemac since we fill the buffer with 32 bits words at each 125MHz cycle and the buffer outputs 8 bits words to Gemac at the same frequency.

5. CONCLUSIONS

This paper presented a proposal of a network stack implemented in FPGA. The architecture has been implemented on an ML402 board and it's using around 13% (4,203 LUTs) of a Xilinx Virtex 4 SX35 FPGA. Its maximum frequency of operation is 128MHz, which satisfies the requirements for Gigabit Ethernet communication.

Our architecture showed, in the best case, a throughput performance of 4.7 times better in the best case and 2.5 times better in average compared with pure software implementation. Also, comparing with lwIP network stack it shows 44 times more throughput in our best case. Considering that in all frame sizes our architecture shows very best results than lwIP we could consider our proposal network stack a good solution to provide high bandwidth to embedded systems that pursue simple microprocessors.

6. ACKNOWLEDGEMENT

The authors would like to thank CNPq and the FINEP/SEBRAE program for supporting the project VoIPWIFI and this work.

7. REFERENCES

- [1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet – a gigabit-per-second local-area network. 1995.
- [2] D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An analysis of tcp processing overhead. 40(5):94–101, 2002.
- [3] A. e. a. Dunkels. lwip: A lightweigh tcp/ip stack available at <http://savannah.nongnu.org/projects/lwip/>. 2009.
- [4] G. Gilder. TELECOSM: How Infinite Bandwidth will Revolutionize Our World. 2001.
- [5] D. Goldenberg, M. Kagan, R. Ravid, and M. Tsirkin. Zero copy sockets direct protocol over infiniband-preliminary implementation and performance analysis. In *Proc. 13th Symposium on High Performance Interconnects*, pages 128–137, 2005.
- [6] A. Romanow, J. Mogul, T. Talpey, and S. Bailey. Rfc 4297 - remote direct memory access (rdma) over ip problem statement. IETF, 2005.
- [7] M. J. S. Smith. Gigabit ethernet and transport offload: transport offload engines help relieve tcp processing burden for gigabit ethernet. *Computer Technology Review*, 2002.